CASUP source code and tests

Generated with ROBODoc Version 4.99.43 (Jan 4 2017)

October 14, 2018

Contents

1 CASUP/ca_hx

[Modules]

NAME

ca_hx

SYNOPSIS

```
!$Id: ca_hx.f90 557 2018-10-10 16:29:20Z mexas $

module ca_hx
```

DESCRIPTION

Module with halo exchange for casup (3), the 2nd gen CA library for SUPercomputers. In this module halos are separate arrays from the central part of CA, which is not a coarray.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

CONTAINS

Public subroutines: ca_spalloc (1.19), ca_halloc (1.1), ca_hdalloc (1.2), ca_hx_all (1.4), ca_hx_glbar (1.7), ca_hx_check (1.5), ca_iter_dc (1.11), ca_iter_omp (1.12), ca_iter_tl (1.13), ca_run (1.17), ca_ising_energy (1.9), ca_ising_energy_col (1.10), ca_set_space_rnd (1.18), ca_mpi_halo_type_create (1.8.1), ca_mpi_halo_type_free (1.8.2), ca_mpi_hx_all (1.8.3), ca_mpi_ising_energy (1.8.10), ca_co_spalloc (1.6.7), ca_co_hx_all (1.6.1), ca_co_hx_check (1.6.2), ca_co_run (1.6.6), ca_co_ising_energy (1.6.3), ca_co_netcdf (1.6.5), ca_co_naive_io (1.6.4), ca_1D_halloc (1.3.1), ca_1D_hdalloc (1.3.2), ca_1D_hx_sall (1.3.4), ca_1D_hx_check (1.3.3). Public pure functions: ca_kernel_copy (1.14), ca_kernel_ising (1.15), ca_kernel_ising_ener (1.16). Private subroutines: in submodule ca_hx_mpi (1.8): ca_mpi_hxvn1m (1.8.4), ca_mpi_hxvn1p (1.8.5), ca_mpi_hxvn2m (1.8.6), ca_mpi_hxvn2p (1.8.7), ca_mpi_hxvn3m (1.8.8), ca_mpi_hxvn3p (1.8.9).

USES

cgca_m1co (9)

USED BY SOURCE

```
use cgca_m1co, only : ca_range, iarr, idef, ilrg
use mpi
implicit none

private
public :: ca_spalloc, ca_halloc, ca_hdalloc, ca_hx_all, ca_hx_glbar, &
  ca_hx_check, &
  ca_iter_dc, ca_iter_omp, ca_iter_tl, ca_run, ca_kernel_copy,       &
  ca_kernel_ising, ca_kernel_ising_ener,  &
  ca_ising_energy, ca_ising_energy_col, &
  ca_set_space_rnd, &
  ca_mpi_halo_type_create, ca_mpi_halo_type_free,  &
  ca_mpi_hx_all, ca_mpi_ising_energy, &
```

```
  ca_co_spalloc, ca_co_hx_all, ca_co_hx_check, ca_co_run, &
  ca_co_ising_energy, ca_co_netcdf, ca_co_naive_io, &
  ca_1D_halloc, ca_1D_hdalloc, ca_1D_hx_sall, ca_1D_hx_check

! These are halo coarrays. Parts of the CA array are designated
! "halo", but those are not coarrays.
!
! h1 - halo along dimension 1.
! h2 - halo along dimension 1.
! h3 - halo along dimension 1.
! "minu" - minus, "plus" - plus.
! "t" - tmp arrays.
integer( kind=iarr ), allocatable ::          &
  h1minu(:,:,:)[:,:,:], h1plus(:,:,:)[:,:,:], &
  h2minu(:,:,:)[:,:,:], h2plus(:,:,:)[:,:,:], &
  h3minu(:,:,:)[:,:,:], h3plus(:,:,:)[:,:,:], &
  tmp_space(:,:,:), mask_array(:,:,:)

integer :: hdepth,     & ! halo depth
  ci(3),         & ! coindex set of my image
  ucob(3),       & ! upper cobounds of halo coarrays
  ihsta(3),      & ! inner right halo start
  rhsta(3),      & ! outer right halo start
  rhend(3),      & ! outer right halo end
  lhsta(3),      & ! outer left halo start
  sub(3),        & ! upper bounds of space array (without halo)
  ierr,          & ! error variable
  total_cells,   & ! total number of cells in the global CA.
  nei_ci_L1(3),  & ! coindex set of left neighbour along 1
  nei_ci_R1(3),  & ! coindex set of right neighbour along 1
  nei_ci_L2(3),  & ! coindex set of left neighbour along 2
  nei_ci_R2(3),  & ! coindex set of right neighbour along 2
  nei_ci_L3(3),  & ! coindex set of left neighbour along 3
  nei_ci_R3(3),  & ! coindex set of right neighbour along 3
  nei_img_L(3),  & ! image indices for 3 left neighbours
  nei_img_R(3)     ! image indices for 3 right neighbours

! A scalar to calculate the total energy of CA
! These will not be needed when collectives can be used.
integer( kind=ilrg ) :: co_energy[*], co_magnet[*]

character( len=500 ) :: errmsg

abstract interface

!*****************************************************************
! Cray bug in 8.6.5!
! Should be able to use hdepth from the module via IMPORT!
! When fixed, remove "halo" and change back to use "hdepth"
! in this module, ca_hx_co and ca_hx_mpi!
!*****************************************************************
```

```
  !*********************************************************************72
  ! For non-coarray CA model
  !*********************************************************************72

  ! For a kernel function
  pure function kernel_proto( space, halo, coord )
    use cgca_m1co
!    import hdepth
    implicit none
    integer, intent(in) :: halo
    integer( kind=iarr ), intent(in), contiguous ::                    &
      space( 1-halo: , 1-halo: , 1-halo: )
    integer, intent(in) :: coord(3)
    integer( kind=iarr ) kernel_proto
  end function kernel_proto

  ! For a subroutine doing a single iteration
  subroutine iter_proto( space, halo, kernel )
    use cgca_m1co
    import :: kernel_proto
    implicit none
    integer, intent( in ) :: halo
    integer( kind=iarr ), intent(in), contiguous ::                    &
      space( 1-halo: , 1-halo: , 1-halo: )
    procedure( kernel_proto ) :: kernel
  end subroutine iter_proto

  ! For a HX routine
  subroutine hx_proto( space )
    use cgca_m1co
    implicit none
    integer( kind=iarr ), intent( inout ), allocatable :: space(:,:,:)
  end subroutine hx_proto

  !*********************************************************************72
  ! For coarray CA model
  !*********************************************************************72

  ! For a HX routine
  subroutine hx_co_proto( space )
    use cgca_m1co
    implicit none
    integer( kind=iarr ), intent( inout ), allocatable ::              &
      space(:,:,:)[:,:,:]
  end subroutine hx_co_proto

  !*********************************************************************72
  ! For MPI HX
  !*********************************************************************72

  subroutine hx_mpi_proto( space )
    use cgca_m1co
```

```
    implicit none
    integer( kind=iarr ), intent( inout ), allocatable :: space(:,:,:)
  end subroutine hx_mpi_proto

end interface

! Interfaces for submodule procedures.

interface

  !*********************************************************************72
  ! In submodule ca_hx_mpi
  !*********************************************************************72

  module subroutine ca_mpi_halo_type_create( space )
    integer( kind=iarr ), intent( inout ), allocatable :: space(:,:,:)
  end    subroutine ca_mpi_halo_type_create

  module subroutine ca_mpi_halo_type_free
  end    subroutine ca_mpi_halo_type_free

  module subroutine ca_mpi_hxvn1m( space )
    integer( kind=iarr ), intent(inout), allocatable :: space(:,:,:)
  end    subroutine ca_mpi_hxvn1m

  module subroutine ca_mpi_hxvn1p( space )
    integer( kind=iarr ), intent(inout), allocatable :: space(:,:,:)
  end    subroutine ca_mpi_hxvn1p

  module subroutine ca_mpi_hxvn2m( space )
    integer( kind=iarr ), intent(inout), allocatable :: space(:,:,:)
  end    subroutine ca_mpi_hxvn2m

  module subroutine ca_mpi_hxvn2p( space )
    integer( kind=iarr ), intent(inout), allocatable :: space(:,:,:)
  end    subroutine ca_mpi_hxvn2p

  module subroutine ca_mpi_hxvn3m( space )
    integer( kind=iarr ), intent(inout), allocatable :: space(:,:,:)
  end    subroutine ca_mpi_hxvn3m

  module subroutine ca_mpi_hxvn3p( space )
    integer( kind=iarr ), intent(inout), allocatable :: space(:,:,:)
  end    subroutine ca_mpi_hxvn3p

  module subroutine ca_mpi_hx_all( space )
    integer( kind=iarr ), intent(inout), allocatable :: space(:,:,:)
  end    subroutine ca_mpi_hx_all

  module subroutine ca_mpi_ising_energy( space, hx_sub, iter_sub,      &
                                         kernel, energy, magnet )
    integer( kind=iarr ), intent(inout), allocatable :: space(:,:,:)
```

```
  procedure( hx_mpi_proto ) :: hx_sub
  procedure( iter_proto ) :: iter_sub
  procedure( kernel_proto ) :: kernel
  integer( kind=ilrg ), intent(out) :: energy, magnet
end    subroutine ca_mpi_ising_energy

!*****************************************************************72
! In submodule ca_hx_co
!*****************************************************************72

module subroutine ca_co_spalloc( space, c, d, ir )
  integer( kind=iarr ), intent( inout ), allocatable ::              &
    space(:,:,:) [:,:,:]
  integer, intent(in) :: c(3), d, ir(3)
end subroutine ca_co_spalloc

module subroutine ca_co_hx_all( space )
  integer( kind=iarr ), intent( inout ), allocatable ::              &
    space(:,:,:) [:,:,:]
end subroutine ca_co_hx_all

module subroutine ca_co_hx_check( space, flag )
  integer( kind=iarr ), intent( in ), allocatable ::                 &
    space(:,:,:) [:,:,:]
  integer, intent( out ) :: flag
end subroutine ca_co_hx_check

module subroutine ca_co_run( space, hx_sub, iter_sub, kernel, niter )
  integer( kind=iarr ), intent( inout ), allocatable ::              &
    space(:,:,:) [:,:,:]
  procedure( hx_co_proto )  :: hx_sub
  procedure( iter_proto )   :: iter_sub
  procedure( kernel_proto ) :: kernel
  integer, intent(in) :: niter
end subroutine ca_co_run

module subroutine ca_co_ising_energy( space, hx_sub, iter_sub,       &
     kernel, energy, magnet )
  integer( kind=iarr ), intent( inout ), allocatable ::              &
    space(:,:,:)[:,:,:]
  procedure( hx_co_proto ) :: hx_sub
  procedure( iter_proto ) :: iter_sub
  procedure( kernel_proto ) :: kernel
  integer( kind=ilrg) , intent(out) :: energy, magnet
end subroutine ca_co_ising_energy

module subroutine ca_co_netcdf( space, fname )
  integer( kind=iarr ), intent( in ), allocatable ::                &
    space(:,:,:) [:,:,:]
  character( len=* ), intent( in ) :: fname
end subroutine ca_co_netcdf
```

```
  module subroutine ca_co_naive_io( coarray, fname )
    integer( kind=iarr ), intent( in ), allocatable ::                &
      coarray(:,:,:) [:,:,:]
    character( len=* ),intent( in ) :: fname
  end subroutine ca_co_naive_io

  !*********************************************************************72
  ! In submodule ca_hx_1D
  !*********************************************************************72

  module subroutine ca_1D_halloc
  end subroutine ca_1D_halloc

  module subroutine ca_1D_hdalloc
  end subroutine ca_1D_hdalloc

  module subroutine ca_1D_hx_sall( space )
    integer( kind=iarr ), intent(inout), allocatable :: space(:,:,:)
  end subroutine ca_1D_hx_sall

  module subroutine ca_1D_hx_check( space, flag )
    integer( kind=iarr ), intent(in), allocatable :: space(:,:,:)
    integer, intent( out ) :: flag
  end subroutine ca_1D_hx_check

end interface

contains
```

1.1 ca_hx/ca_halloc

[ca_hx] [Subroutines]

NAME

ca_halloc

SYNOPSIS

```
subroutine ca_halloc( ir )
```

INPUT

```
integer :: ir(3)

!    ir - codimensions
```

OUTPUT

```
!    none
```

SIDE EFFECTS

halo coarrays (module variables) are allocated

DESCRIPTION

This routine allocates 6 halo coarrays for von Neumann 6-neighbourhood. The coarrays are module variables. Coarray allocation is an implicit sync. Halos have depth hdepth. Halo coarrays have the same cobounds on all images.

NOTES

All images must call this routine!

USES USED BY SOURCE

```
integer :: i,j,k

main: associate( d => hdepth, c => sub )

if ( this_image() .eq. 1 ) write (*,*) "halloc: d:", d, "c:", c

allocate( h1minu( d, c(2), c(3) ) [ ir(1), ir(2), * ], stat=ierr,      &
          errmsg=errmsg )
if ( ierr .ne. 0 ) then
  write (*,*) "ERROR: ca_hx/ca_halloc: allocate( h1minu ). ierr: ",    &
              ierr, "errmsg:", trim(errmsg)
  error stop
end if

allocate( h1plus( d, c(2), c(3) ) [ ir(1), ir(2), * ], stat=ierr,      &
          errmsg=errmsg )
if ( ierr .ne. 0 ) then
  write (*,*) "ERROR: ca_hx/ca_halloc: allocate( h1plus ). ierr: ",    &
```

```
              ierr, "errmsg:", trim(errmsg)
  error stop
end if

allocate( h2minu( c(1), d, c(3) ) [ ir(1), ir(2), * ], stat=ierr,      &
          errmsg=errmsg )
if ( ierr .ne. 0 ) then
  write (*,*) "ERROR: ca_hx/ca_halloc: allocate( h2minu ). ierr: ",    &
              ierr, "errmsg:", trim(errmsg)
  error stop
end if

allocate( h2plus( c(1), d, c(3) ) [ ir(1), ir(2), * ], stat=ierr,      &
          errmsg=errmsg )
if ( ierr .ne. 0 ) then
  write (*,*) "ERROR: ca_hx/ca_halloc: allocate( h2plus ). ierr: ",    &
              ierr, "errmsg:", trim(errmsg)
  error stop
end if

allocate( h3minu( c(1), c(2), d ) [ ir(1), ir(2), * ], stat=ierr,      &
          errmsg=errmsg )
if ( ierr .ne. 0 ) then
  write (*,*) "ERROR: ca_hx/ca_halloc: allocate( h3minu ). ierr: ",    &
              ierr, "errmsg:", trim(errmsg)
  error stop
end if

allocate( h3plus( c(1), c(2), d ) [ ir(1), ir(2), * ], stat=ierr,      &
          errmsg=errmsg)
if ( ierr .ne. 0 ) then
  write (*,*) "ERROR: ca_hx/ca_halloc: allocate( h3plus ). ierr: ",    &
              ierr, "errmsg:", trim(errmsg)
  error stop
end if

! Calculate once and keep forever
  ci = this_image( h1minu )
ucob = ucobound(   h1minu )

! Now can set mask_array. The mask array must reflect
! the global CA space, i.e. must not be affected by partitioning
! of the model into images. This means the mask array must
! depend on coindex set of this image.
do concurrent( i=1:c(1), j=1:c(2), k=1:c(3) )
  mask_array(i,j,k) = int( mod( (i+j+k + ( ci(1)-1)*c(1) +             &
    (ci(2)-1)*c(2) + (ci(3)-1)*c(3) ) , 2 ), kind=iarr )
end do

end associate main

! Calculate coindex sets and image numbers for the 6 neighbours
```

```
! Coindex sets
  nei_ci_L1 = (/ ci(1)-1, ci(2), ci(3) /)
  nei_ci_R1 = (/ ci(1)+1, ci(2), ci(3) /)
  nei_ci_L2 = (/ ci(1), ci(2)-1, ci(3) /)
  nei_ci_R2 = (/ ci(1), ci(2)+1, ci(3) /)
  nei_ci_L3 = (/ ci(1), ci(2), ci(3)-1 /)
  nei_ci_R3 = (/ ci(1), ci(2), ci(3)+1 /)

! Image index
  nei_img_L(1) = image_index( h1plus, nei_ci_L1 )
  nei_img_R(1) = image_index( h1plus, nei_ci_R1 )
  nei_img_L(2) = image_index( h1plus, nei_ci_L2 )
  nei_img_R(2) = image_index( h1plus, nei_ci_R2 )
  nei_img_L(3) = image_index( h1plus, nei_ci_L3 )
  nei_img_R(3) = image_index( h1plus, nei_ci_R3 )

end subroutine ca_halloc
```

1.2 ca_hx/ca_hdalloc

[ca_hx] [Subroutines]

NAME

ca_hdalloc

SYNOPSIS

```
subroutine ca_hdalloc
```

INPUT

```
!    none
```

OUTPUT

```
!    none
```

SIDE EFFECTS

halo coarrays (module variables) are deallocated

DESCRIPTION

This routine deallocates 6 halo coarrays for von Neumann 6-neighbourhood. The coarrays are module variables. Coarray deallocation is an implicit sync. Halo coarrays have the same cobounds on all images.

NOTES

All images must call this routine!

USES USED BY SOURCE

```
deallocate( h1minu, stat=ierr, errmsg=errmsg )
if ( ierr .ne. 0 ) then
  write (*,*) "ERROR: ca_hx/ca_hdalloc: deallocate( h1minu ). ierr: ", &
              ierr, "errmsg:", trim(errmsg)
  error stop
end if

deallocate( h1plus, stat=ierr, errmsg=errmsg )
if ( ierr .ne. 0 ) then
  write (*,*) "ERROR: ca_hx/ca_hdalloc: deallocate( h1plus ). ierr: ", &
              ierr, "errmsg:", trim(errmsg)
  error stop
end if

deallocate( h2minu, stat=ierr, errmsg=errmsg )
if ( ierr .ne. 0 ) then
  write (*,*) "ERROR: ca_hx/ca_hdalloc: deallocate( h2minu ). ierr: ", &
              ierr, "errmsg:", trim(errmsg)
  error stop
end if
```

```
deallocate( h2plus, stat=ierr, errmsg=errmsg )
if ( ierr .ne. 0 ) then
  write (*,*) "ERROR: ca_hx/ca_hdalloc: deallocate( h2plus ). ierr: ", &
              ierr, "errmsg:", trim(errmsg)
  error stop
end if

deallocate( h3minu, stat=ierr, errmsg=errmsg )
if ( ierr .ne. 0 ) then
  write (*,*) "ERROR: ca_hx/ca_hdalloc: deallocate( h3minu ). ierr: ", &
              ierr, "errmsg:", trim(errmsg)
  error stop
end if

deallocate( h3plus, stat=ierr, errmsg=errmsg)
if ( ierr .ne. 0 ) then
  write (*,*) "ERROR: ca_hx/ca_hdalloc: deallocate( h3plus ). ierr: ", &
              ierr, "errmsg:", trim(errmsg)
  error stop
end if

end subroutine ca_hdalloc
```

1.3 ca_hx/ca_hx_1D

[ca_hx] [Submodules]

NAME

ca_hx_1D

SYNOPSIS

```
!$Id: ca_hx_1D.f90 558 2018-10-14 16:28:29Z mexas $

submodule ( ca_hx ) ca_hx_1D
```

DESCRIPTION

Submodule with routines for a single codimension CA coarrays. Whole array coarrays, not just halos.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

CONTAINS USES USED BY SOURCE

```
implicit none

! These are halo coarrays. Parts of the CA array are designated
! "halo", but those are not coarrays.
!
! "minu" - minus, "plus" - plus.
integer( kind=iarr ), allocatable ::          &
  h1Dminu(:,:,:)[:], h1Dplus(:,:,:)[:]

integer :: nei1D_L, nei1D_R

contains
```

1.3.1 ca_hx_1D/ca_1D_halloc

[ca_hx_1D] [Subroutines]

NAME

ca_1D_halloc

SYNOPSIS

```
module subroutine ca_1D_halloc
```

SIDE EFFECTS

halo coarrays (submodule variables) are allocated

DESCRIPTION

This routine allocates 2 halo coarrays for von Neumann 6-neighbourhood. The coarrays are submodule variables. Coarray allocation is an implicit sync. Halos have depth hdepth. Halo coarrays have the same cobounds on all images.

NOTES

All images must call this routine!

USES USED BY SOURCE

```
integer :: i,j,k

main: associate( d => hdepth, c => sub )

if ( this_image() .eq. 1 ) write (*,*) "1D_halloc: d:", d, "c:", c

allocate( h1Dminu( c(1), c(2), d ) [*], stat=ierr, errmsg=errmsg )
if ( ierr .ne. 0 ) then
  write (*,*) "ERROR: ca_hx_1D/ca_1D_halloc: allocate( h1Dminu )." //  &
              " ierr: ", ierr, "errmsg:", trim(errmsg)
  error stop
end if

allocate( h1Dplus( c(1), c(2), d ) [*], stat=ierr, errmsg=errmsg )
if ( ierr .ne. 0 ) then
  write (*,*) "ERROR: ca_hx_1D/ca_1D_halloc: allocate( h1Dplus )." //  &
              " ierr: ", ierr, "errmsg:", trim(errmsg)
  error stop
end if

! Calculate once and keep forever
! In 1D version space has a single codimension.
! To use the old code, we make the first 2 codimensions, and the
! first 2 cobounds are equal to 1.
    ci    = 1
  ucob    = 1
    ci(3) = this_image()
  ucob(3) = num_images()
```

```
! Now can set mask_array. The mask array must reflect
! the global CA space, i.e. must not be affected by partitioning
! of the model into images. This means the mask array must
! depend on coindex set of this image.
do concurrent( i=1:c(1), j=1:c(2), k=1:c(3) )
  mask_array(i,j,k) = int( mod( (i+j+k + ( ci(1)-1)*c(1) +             &
    (ci(2)-1)*c(2) + (ci(3)-1)*c(3) ) , 2 ), kind=iarr )
end do

end associate main

! Calculate image numbers for the 2 neighbours
  nei1D_L = this_image()-1
  nei1D_R = this_image()+1

end subroutine ca_1D_halloc
```

1.3.2 ca_hx_1D/ca_1D_hdalloc

[ca_hx_1D] [Subroutines]

NAME

ca_1D_hdalloc

SYNOPSIS

```
module subroutine ca_1D_hdalloc
```

INPUT

```
!    none
```

OUTPUT

```
!    none
```

SIDE EFFECTS

halo coarrays (module variables) are deallocated

DESCRIPTION

This routine deallocates 2 halo coarrays for von Neumann 6-neighbourhood. The coarrays are module variables. Coarray deallocation is an implicit sync. Halo coarrays have the same cobounds on all images.

NOTES

All images must call this routine!

USES USED BY SOURCE

```
deallocate( h1Dminu, stat=ierr, errmsg=errmsg )
if ( ierr .ne. 0 ) then
  write (*,*) "ERROR: ca_hx_1D/ca_1D_hdalloc: " //                    &
    "deallocate( h1Dminu ). ierr: ", ierr, "errmsg:", trim(errmsg)
  error stop
end if

deallocate( h1Dplus, stat=ierr, errmsg=errmsg )
if ( ierr .ne. 0 ) then
  write (*,*) "ERROR: ca_hx_1D/ca_1D_hdalloc: " //                    &
    "deallocate( h1Dplus ). ierr: ", ierr, "errmsg:", trim(errmsg)
  error stop
end if

end subroutine ca_1D_hdalloc
```

1.3.3 ca_hx_1D/ca_1D_hx_check

[ca_hx_1D] [Subroutines]

NAME

ca_1D_hx_check

SYNOPSIS

```
module subroutine ca_1D_hx_check( space, flag )
```

INPUT

```
integer( kind=iarr ), intent(in), allocatable :: space(:,:,:)

!    space - CA array
```

OUTPUT

```
integer, intent( out ) :: flag

!    flag .eq. 0 - check passed
!    flag .ne. 0 - check failed
```

SIDE EFFECTS

none

DESCRIPTION

This routine is of very limited use. It checks hx code for only a single case - all images set all their cell values to this_image() and then a single hx step is done. In this case I'm sure what's in my halos. So just check for this and either return flag=0 or flag > 0 indicating which halo failed. Fail values:

```
      0  - pass, no failures
      1  - test 1 failed
      2  - test 2 failed
      3  - tests 1,2 failed
```

USES USED BY SOURCE

```
flag = 0

! Test 1
if ( ci(3) .ne. 1       ) then
  if ( any( space( 1:sub(1), 1:sub(2), lhsta(3):0 )                    &
                    .ne. nei1D_L ) ) flag = 1
end if

! Test 2
if ( ci(3) .ne. ucob(3) ) then
  if ( any( space( 1:sub(1), 1:sub(2), rhsta(3):rhend(3) )             &
                    .ne. nei1D_R ) ) flag = 2
```

```
end if

end subroutine ca_1D_hx_check
```

1.3.4 ca_hx_1D/ca_1D_hx_sall

[ca_hx_1D] [Subroutines]

NAME

ca_1D_hx_sall

SYNOPSIS

```
module subroutine ca_1D_hx_sall( space )
```

INPUT

```
integer( kind=iarr ), intent(inout), allocatable :: space(:,:,:)

!    space - non coarray array with CA model
```

OUTPUT

```
!    none
```

SIDE EFFECTS

halo coarrays are updated

DESCRIPTION

HX is a 2-step process. In step 1 I copy parts of my space array, not space halos!, into my halo coarrays. Step 2 is remote comms - I pull neighbour halo coarrays into my local space array halo sections. sync all is used for synchronisation (sall in the name). Note

USES USED BY SOURCE

```
if ( ci(3) .ne. 1       ) then
  h1Dminu(:,:,:) = space( 1:sub(1) , 1:sub(2) ,        1 : hdepth )
end if
if ( ci(3) .ne. ucob(3) ) then
  h1Dplus(:,:,:) = space( 1:sub(1) , 1:sub(2) , ihsta(3) : sub(3) )
end if

sync all

if ( ci(3) .ne. 1 ) then
  space( 1:sub(1) , 1:sub(2) , lhsta(3) : 0 ) =                        &
    h1Dplus(:,:,:) [ nei1D_L ]
end if

if ( ci(3) .ne. ucob(3) ) then
  space( 1:sub(1) , 1:sub(2) , rhsta(3) : rhend(3) ) =                 &
    h1Dminu(:,:,:) [ nei1D_R ]
end if

end subroutine ca_1D_hx_sall
```

1.4 ca_hx/ca_hx_all

[ca_hx] [Subroutines]

NAME

ca_hx_all

SYNOPSIS

```
subroutine ca_hx_all( space )
```

INPUT

```
integer( kind=iarr ), intent(inout), allocatable :: space(:,:,:)

!    space - non coarray array with CA model
```

OUTPUT

```
!    none
```

SIDE EFFECTS

halo coarrays are updated

DESCRIPTION

HX is a 2-step process. In step 1 I copy parts of my space array, not space halos!, into my halo coarrays. Step 2 is remote comms - I pull neighbour halo coarrays into my local space array halo sections.

Important! the 6 routines must be called in the same order on all images to avoid deadlocks. So to make it fool proof I don't allow the user to call individual hx routines. These are private to this module. The user only calls this routine.

USES USED BY SOURCE

```
! HX is a 2-step process

! Step 1

!    I prepare my 6 coarray halos for use by my neighbours.
!    These are copies of the slabs of space array, i.e. top/bottom,
!    left/right and front/back.
!    Note that these are not space array halos!
!    Refer to the schematic in ca_spalloc for details.
!    When done, I declare that my 6 coarray halos "ready" to be read -
!    ready for remote calls.
!    This is a local routine, which must be run before any
!    of hx routines are called.
!    This routine is called when all cells on my image have been
!    processed, so doesn't make sense to separate separate assignments
!    into different routines - no performance gain at all.
if ( ci(1) .ne. 1       ) then
  h1minu(:,:,:) = space(        1 : hdepth , 1:sub(2) , 1:sub(3) )
end if
```

```
if ( ci(1) .ne. ucob(1) ) then
  h1plus(:,:,:) = space( ihsta(1) : sub(1) , 1:sub(2) , 1:sub(3) )
end if
if ( ci(2) .ne. 1       ) then
  h2minu(:,:,:) = space( 1:sub(1) ,        1 : hdepth , 1:sub(3) )
end if
if ( ci(2) .ne. ucob(2) ) then
  h2plus(:,:,:) = space( 1:sub(1) , ihsta(2) : sub(2) , 1:sub(3) )
end if
if ( ci(3) .ne. 1       ) then
  h3minu(:,:,:) = space( 1:sub(1) , 1:sub(2) ,        1 : hdepth )
end if
if ( ci(3) .ne. ucob(3) ) then
  h3plus(:,:,:) = space( 1:sub(1) , 1:sub(2) , ihsta(3) : sub(3) )
end if

! Step 2

!    An image updates its space array halo layer (left side)
!    along direction 1 from a coarray halo (h1plus) on an image
!    which is 1 lower along codimension 1.
if ( ci(1) .ne. 1 ) then
  sync images( nei_img_L(1) )
  space( lhsta(1):0, 1:sub(2), 1:sub(3) ) =                              &
    h1plus(:,:,:) [ nei_ci_L1(1), nei_ci_L1(2), nei_ci_L1(3) ]
end if

!    An image updates its space array halo layer (right side)
!    along direction 1 from a coarray halo (h1minu) on an image
!    which is 1 higher along codimension 1.
if ( ci(1) .ne. ucob(1) ) then
  sync images( nei_img_R(1) )
  space( rhsta(1) : rhend(1), 1:sub(2) , 1:sub(3) ) =                    &
    h1minu(:,:,:) [ nei_ci_R1(1), nei_ci_R1(2), nei_ci_R1(3) ]
end if

!    An image updates its space array halo layer (left side)
!    along direction 2 from a coarray halo (h1plus) on an image
!    which is 1 lower along codimension 2.
if ( ci(2) .ne. 1 ) then
  sync images( nei_img_L(2) )
  space( 1:sub(1) , lhsta(2) : 0, 1:sub(3) ) =                           &
    h2plus(:,:,:) [ nei_ci_L2(1), nei_ci_L2(2), nei_ci_L2(3) ]
end if

!    An image updates its space array halo layer (right side)
!    along direction 2 from a coarray halo (h2minu) on an image
!    which is 1 higher along codimension 1.
if ( ci(2) .ne. ucob(2) ) then
  sync images( nei_img_R(2) )
  space( 1:sub(1) , rhsta(2) : rhend(2) , 1:sub(3) ) =                   &
    h2minu(:,:,:) [ nei_ci_R2(1), nei_ci_R2(2), nei_ci_R2(3) ]
```

```
end if

!    An image updates its space array halo layer (left side)
!    along direction 3 from a coarray halo (h3plus) on an image
!    which is 1 lower along codimension 3.
if ( ci(3) .ne. 1 ) then
  sync images( nei_img_L(3) )
  space( 1:sub(1) , 1:sub(2) , lhsta(3) : 0 ) =                          &
    h3plus(:,:,:) [ nei_ci_L3(1), nei_ci_L3(2), nei_ci_L3(3) ]
end if

!    An image updates its space array halo layer (right side)
!    along direction 3 from a coarray halo (h3minu) on an image
!    which is 1 higher along codimension 3.
if ( ci(3) .ne. ucob(3) ) then
  sync images( nei_img_R(3) )
  space( 1:sub(1) , 1:sub(2) , rhsta(3) : rhend(3) ) =                   &
    h3minu(:,:,:) [ nei_ci_R3(1), nei_ci_R3(2), nei_ci_R3(3) ]
end if

end subroutine ca_hx_all
```

1.5 ca_hx/ca_hx_check

[ca_hx] [Subroutines]

NAME

ca_hx_check

SYNOPSIS

```
subroutine ca_hx_check( space, flag )
```

INPUT

```
integer( kind=iarr ), intent(in), allocatable :: space(:,:,:)

!    space - CA array
```

OUTPUT

```
integer, intent( out ) :: flag

!    flag .eq. 0 - check passed
!    flag .ne. 0 - check failed
```

SIDE EFFECTS

none

DESCRIPTION

This routine is of very limited use. It checks hx code for only a single case - all images set all their cell values to this_image() and then a single hx step is done. In this case I'm sure what's in my halos. So just check for this and either return flag=0 or flag > 0 indicating which halo failed. Fail values:

```
0  - pass, no failures
1  - test 1 failed
2  - test 2 failed
3  - tests 1,2 failed
4  - test 3 failed
5  - tests 1,3 failed
6  - tests 2,3 failed
7  - tests 1,2,3 failed
8  - test 4 failed
9  - tests 1,4 failed
10 - tests 2,4 failed
11 - tests 1,2,4 failed
12 - tests 3,4 failed
13 - tests 1,3,4 failed
14 - tests 2,3,4 failed
15 - tests 1,2,3,4 failed
16 - test 5 failed
```

and so on.

USES USED BY SOURCE

```
! coindex set and the image number of the neighbour
integer :: i(3), n

flag = 0

! Test 1
if ( ci(1) .ne. 1 ) then
  ! This is the neighbour
  i = (/ ci(1)-1, ci(2), ci(3) /) ! neighbour's coindex set
  n = image_index( h1plus, i )    ! neighbour image number
  if ( any( space( lhsta(1):0,        1:sub(2), 1:sub(3) ) .ne. n ) )  &
    flag = flag + 1
end if

! Test 2
if ( ci(1) .ne. ucob(1) ) then
  ! This is the neighbour
  i = (/ ci(1)+1, ci(2), ci(3) /) ! neighbour's coindex set
  n = image_index( h1plus, i )    ! neighbour image number
  if ( any( space( rhsta(1):rhend(1), 1:sub(2), 1:sub(3) ) .ne. n ) )  &
    flag = flag + 2
end if

! Test 3
if ( ci(2) .ne. 1 ) then
  ! This is the neighbour
  i = (/ ci(1), ci(2)-1, ci(3) /) ! neighbour's coindex set
  n = image_index( h1plus, i )    ! neighbour image number
  if ( any( space( 1:sub(1), lhsta(2):0,        1:sub(3) ) .ne. n ) )  &
    flag = flag + 4
end if

! Test 4
if ( ci(2) .ne. ucob(2) ) then
  ! This is the neighbour
  i = (/ ci(1), ci(2)+1, ci(3) /) ! neighbour's coindex set
  n = image_index( h1plus, i )    ! neighbour image number
  if ( any( space( 1:sub(1), rhsta(2):rhend(2), 1:sub(3) ) .ne. n ) )  &
    flag = flag + 8
end if

! Test 5
if ( ci(3) .ne. 1 ) then
  ! This is the neighbour
  i = (/ ci(1), ci(2), ci(3)-1 /) ! neighbour's coindex set
  n = image_index( h1plus, i )    ! neighbour image number
  if ( any( space( 1:sub(1), 1:sub(2), lhsta(3):0 )        .ne. n ) )  &
    flag = flag + 16
end if

! Test 6
if ( ci(3) .ne. ucob(3) ) then
```

```
  ! This is the neighbour
  i = (/ ci(1), ci(2), ci(3)+1 /) ! neighbour's coindex set
  n = image_index( h1plus, i )     ! neighbour image number
  if ( any( space( 1:sub(1), 1:sub(2), rhsta(3):rhend(3) ) .ne. n ) )  &
    flag = flag + 32
end if

end subroutine ca_hx_check
```

1.6 ca_hx/ca_hx_co

[ca_hx] [Submodules]

NAME

ca_hx_co

SYNOPSIS

```
!$Id: ca_hx_co.f90 560 2018-10-14 19:02:34Z mexas $

submodule ( ca_hx ) ca_hx_co
```

DESCRIPTION

Submodule with routines for whole CA implemented with coarrays, not just halos.

AUTHOR

Anton Shterenlikht

COPYRIGHT

CONTAINS USES USED BY SOURCE

```
implicit none

contains
```

1.6.1 ca_hx_co/ca_co_hx_all

[ca_hx_co] [Subroutines]

NAME

ca_co_hx_all

SYNOPSIS

```
module subroutine ca_co_hx_all( space )
```

INPUT

```
integer( kind=iarr ), intent(inout), allocatable :: space(:,:,:) [:,:,:]

!    space - coarray array with CA model
```

OUTPUT

```
!    space is updated, just the halo layers
```

SIDE EFFECTS

none

DESCRIPTION

There is pair-wise handshake sync between images. Each image, except those on the boundary, will send halos to and receive halos from 6 neighbouring images.

USES USED BY SOURCE

```
! An image updates its space coarray halo layer (left side)
! along direction 1 from the last layer in the coarray on an image
! which is 1 lower along codimension 1.
if ( ci(1) .ne. 1 ) then
  sync images( nei_img_L(1) )
  space( lhsta(1):0,           1:sub(2) , 1:sub(3) ) =                &
    space( ihsta(1) : sub(1) , 1:sub(2) , 1:sub(3) )                 &
      [ nei_ci_L1(1), nei_ci_L1(2), nei_ci_L1(3) ]
end if

! An image updates its space coarray halo layer (right side)
! along direction 1 from the 1st layer in the coarray on an image
! which is 1 higher along codimension 1.
if ( ci(1) .ne. ucob(1) ) then
  sync images( nei_img_R(1) )
  space( rhsta(1) : rhend(1) , 1:sub(2) , 1:sub(3) ) =                &
    space( 1 : hdepth        , 1:sub(2) , 1:sub(3) )                 &
      [ nei_ci_R1(1), nei_ci_R1(2), nei_ci_R1(3) ]
end if

! An image updates its space coarray halo layer (left side)
! along direction 2 from the last layer in the coarray on an image
```

```
! which is 1 lower along codimension 2.
if ( ci(2) .ne. 1 ) then
  sync images( nei_img_L(2) )
  space(   1:sub(1) , lhsta(2) : 0,       1:sub(3) ) =                &
    space( 1:sub(1) , ihsta(2) : sub(2) , 1:sub(3) )                  &
      [ nei_ci_L2(1), nei_ci_L2(2), nei_ci_L2(3) ]
end if

! An image updates its space coarray halo layer (right side)
! along direction 2 from the 1st layer in the coarray on an image
! which is 1 higher along codimension 1.
if ( ci(2) .ne. ucob(2) ) then
  sync images( nei_img_R(2) )
  space(   1:sub(1) , rhsta(2) : rhend(2) , 1:sub(3) ) =              &
    space( 1:sub(1) , 1 : hdepth          , 1:sub(3) )                &
      [ nei_ci_R2(1), nei_ci_R2(2), nei_ci_R2(3) ]
end if

! An image updates its space coarray halo layer (left side)
! along direction 3 from the last layer in the coarray on an image
! which is 1 lower along codimension 3.
if ( ci(3) .ne. 1 ) then
  sync images( nei_img_L(3) )
  space(   1:sub(1) , 1:sub(2) , lhsta(3) : 0 ) =                     &
    space( 1:sub(1) , 1:sub(2) , ihsta(3) : sub(3) )                  &
    [ nei_ci_L3(1), nei_ci_L3(2), nei_ci_L3(3) ]
end if

! An image updates its space coarray halo layer (right side)
! along direction 3 from the 1st layer in the coarray on an image
! which is 1 higher along codimension 3.
if ( ci(3) .ne. ucob(3) ) then
  sync images( nei_img_R(3) )
  space(   1:sub(1) , 1:sub(2) , rhsta(3) : rhend(3) ) =              &
    space( 1:sub(1) , 1:sub(2) , 1 : hdepth )                         &
      [ nei_ci_R3(1), nei_ci_R3(2), nei_ci_R3(3) ]
end if

end subroutine ca_co_hx_all
```

1.6.2 ca_hx_co/ca_co_hx_check

[ca_hx_co] [Subroutines]

NAME

ca_co_hx_check

SYNOPSIS

```
module subroutine ca_co_hx_check( space, flag )
```

INPUT

```
integer( kind=iarr ), intent( in ), allocatable :: space(:,:,:) [:,:,:]

!    space - CA array coarray
```

OUTPUT

```
integer, intent( out ) :: flag

!    flag .eq. 0 - check passed
!    flag .ne. 0 - check failed
```

SIDE EFFECTS

none

DESCRIPTION

This routine is of very limited use. It checks hx code for only a single case - all images set all their cell values to this_image() and then a single hx step is done. In this case I'm sure what's in my halos. So just check for this and either return flag=0 or flag > 0 indicating which halo failed. Fail values:

```
0  - pass, no failures
1  - test 1 failed
2  - test 2 failed
3  - tests 1,2 failed
4  - test 3 failed
5  - tests 1,3 failed
6  - tests 2,3 failed
7  - tests 1,2,3 failed
8  - test 4 failed
9  - tests 1,4 failed
10 - tests 2,4 failed
11 - tests 1,2,4 failed
12 - tests 3,4 failed
13 - tests 1,3,4 failed
14 - tests 2,3,4 failed
15 - tests 1,2,3,4 failed
16 - test 5 failed
```

and so on.

USES USED BY SOURCE

```
! coindex set and the image number of the neighbour
integer :: i(3), n

flag = 0

! Test 1
if ( ci(1) .ne. 1 ) then
  ! This is the neighbour
  i = (/ ci(1)-1, ci(2), ci(3) /) ! neighbour's coindex set
  n = image_index( space, i )    ! neighbour image number
  if ( any( space( lhsta(1):0,        1:sub(2), 1:sub(3) ) .ne. n ) )  &
    flag = flag + 1
end if

! Test 2
if ( ci(1) .ne. ucob(1) ) then
  ! This is the neighbour
  i = (/ ci(1)+1, ci(2), ci(3) /) ! neighbour's coindex set
  n = image_index( space, i )    ! neighbour image number
  if ( any( space( rhsta(1):rhend(1), 1:sub(2), 1:sub(3) ) .ne. n ) )  &
    flag = flag + 2
end if

! Test 3
if ( ci(2) .ne. 1 ) then
  ! This is the neighbour
  i = (/ ci(1), ci(2)-1, ci(3) /) ! neighbour's coindex set
  n = image_index( space, i )    ! neighbour image number
  if ( any( space( 1:sub(1), lhsta(2):0,        1:sub(3) ) .ne. n ) )  &
    flag = flag + 4
end if

! Test 4
if ( ci(2) .ne. ucob(2) ) then
  ! This is the neighbour
  i = (/ ci(1), ci(2)+1, ci(3) /) ! neighbour's coindex set
  n = image_index( space, i )    ! neighbour image number
  if ( any( space( 1:sub(1), rhsta(2):rhend(2), 1:sub(3) ) .ne. n ) )  &
    flag = flag + 8
end if

! Test 5
if ( ci(3) .ne. 1 ) then
  ! This is the neighbour
  i = (/ ci(1), ci(2), ci(3)-1 /) ! neighbour's coindex set
  n = image_index( space, i )    ! neighbour image number
  if ( any( space( 1:sub(1), 1:sub(2), lhsta(3):0 )        .ne. n ) )  &
    flag = flag + 16
end if

! Test 6
if ( ci(3) .ne. ucob(3) ) then
```

```
  ! This is the neighbour
  i = (/ ci(1), ci(2), ci(3)+1 /) ! neighbour's coindex set
  n = image_index( space, i )     ! neighbour image number
  if ( any( space( 1:sub(1), 1:sub(2), rhsta(3):rhend(3) ) .ne. n ) )  &
    flag = flag + 32
end if

end subroutine ca_co_hx_check
```

1.6.3 ca_hx_co/ca_co_ising_energy

[ca_hx_co] [Subroutines]

NAME

ca_co_ising_energy

SYNOPSIS

```
module subroutine ca_co_ising_energy( space, hx_sub, iter_sub, kernel, &
  energy, magnet )
```

INPUT

```
integer( kind=iarr ), intent(inout), allocatable :: space(:,:,:)[:,:,:]
procedure( hx_co_proto )  :: hx_sub
procedure( iter_proto )   :: iter_sub
procedure( kernel_proto ) :: kernel

!       space - space coarray before iterations start
!      hx_sub - HX routine
!             - ca_co_hx_all
!    iter_sub - the subroutine performing a single CA iteration, e.g.
!             - ca_co_iter_tl - triple nested loop
!             - ca_co_iter_dc - do concurrent
!             - ca_co_iter_omp - OpenMP
!      kernel - a function to be called for every cell inside the loop
```

OUTPUT

```
integer( kind=ilrg ) , intent(out) :: energy, magnet

!    energy - Total energy of CA system
!    magnet - Total magnetisation of the CA system
```

SIDE EFFECTS

module array tmp_space is updated

DESCRIPTION

Calculate the total energy and the total magnetisation of CA using ising (49) model. These integer values might be very large so I'm using a large integer kind (ilrg (9.28)). This routine uses collectives. Magnetisation is defined as the fraction of the 1 spins. The only valid kernel is ca_kernel_ising_ener (1.16).

USES USED BY SOURCE

```
call hx_sub( space )                    ! space updated, sync images
call iter_sub( space, hdepth, kernel ) ! tmp_space updated, local op
energy =                                                            &
  int( sum( tmp_space( 1:sub(1), 1:sub(2), 1:sub(3) ) ), kind=ilrg )
magnet =                                                            &
  int( sum(     space( 1:sub(1), 1:sub(2), 1:sub(3) ) ), kind=ilrg )
```

```
call co_sum( energy )
call co_sum( magnet )

end subroutine ca_co_ising_energy
```

1.6.4 ca_hx_co/ca_co_naive_io

[ca_hx_co] [Subroutines]

NAME

ca_co_naive_io

SYNOPSIS

```
module subroutine ca_co_naive_io( coarray, fname )
```

INPUTS

```
integer( kind=iarr ), allocatable, intent( in ) :: coarray(:,:,:)[:,:,:]
character( len=* ),intent( in ) :: fname

!    - coarray - what array to dump
!    - fname - what file name to use
```

SIDE EFFECTS

A single binary file is created on image 1 with contents of coarray.

DESCRIPTION

All images call this routine! However only image 1 does all the work. The other images are waiting.

USES

none

USED BY

none, end user.

SOURCE

```
integer :: ierr=0, coi1, coi2, coi3, i2, i3, funit=0,                &
  lb(3),   & ! lower bounds   of the coarray
  ub(3),   & ! upper bounds   of the coarray
  lcob(3), & ! lower cobounds of the coarray
  ucob(3)    ! upper cobounds of the coarray

! Only image1 does this. All other images do nothing.
! So sync all probably should be used after a call to
! this routine in the program.

main: if ( this_image() .eq. 1 ) then

  ! Assume the coarray has halos. Don't write those.
    lb = lbound( coarray ) + hdepth
    ub = ubound( coarray ) - hdepth
  lcob = lcobound( coarray )
  ucob = ucobound( coarray )

  open( newunit=funit, file=fname, form="unformatted",                &
```

```
       access="stream", status="replace", iostat=ierr )
  if ( ierr .ne. 0 ) then
    write (*,*) "ERROR: ca_hx/ca_co_naive_io: open:", ierr
    error stop
  end if

 ! nested loops for writing in correct order from all images
 do coi3 = lcob(3), ucob(3)
  do i3 = lb(3), ub(3)
   do coi2 = lcob(2), ucob(2)
    do i2 = lb(2), ub(2)
     do coi1 = lcob(1), ucob(1)
       write( unit=funit, iostat=ierr )                                &
            coarray( lb(1):ub(1), i2, i3 ) [ coi1, coi2, coi3 ]
     end do
    end do
   end do
  end do
 end do

 close( unit=funit, iostat=ierr )
 if ( ierr .ne. 0 ) then
   write (*,*) "ERROR: ca_hx/ca_co_naive_io: close:", ierr
  error stop
 end if

end if main

end subroutine ca_co_naive_io
```

1.6.5 ca_hx_co/ca_co_netcdf

[ca_hx_co] [Subroutines]

NAME

ca_co_netcdf

SYNOPSIS

```
module subroutine ca_co_netcdf( space, fname )
use netcdf
```

INPUTS

```
     integer( kind=iarr ), intent( in ), allocatable ::                &
       space(:,:,:) [:,:,:]
     character( len=* ), intent( in ) :: fname

!    coarray - what coarray to dump
!    fname - what file name to use
```

OUTPUTS

```
!    None
```

SIDE EFFECTS

A single binary file is created using netcdf with contents of coarray.

DESCRIPTION

Parallel Stream Write Coarray of Integers: All images must call this routine!

MPI must be initialised prior to calling this routine, most probably in the main program. Likewise MPI must be terminated only when no further MPI routines can be called. This will most likely be in the main program. There are some assumptions about the shape of the passed array.

The default integer is assumed for the array at present!

AUTHOR

Anton Shterenlikht, adapted from the code written by David Henty, Luis Cebamanos, EPCC

COPYRIGHT

Note that this routine has special Copyright conditions.

```
    !------------------------------------------------------------------------------!
    !                                                                              !
    !  netCDF routine for Fortran Coarrays                                         !
    !                                                                              !
    !  David Henty, EPCC; d.henty@epcc.ed.ac.uk                                    !
    !                                                                              !
    !  Copyright 2013 the University of Edinburgh                                  !
    !                                                                              !
    !  Licensed under the Apache LICENSE, Version 2.0 (the "LICENSE");             !
```

USES

MPI library, netCDF library

USED BY

none, end user.

SOURCE

```
integer, parameter :: totdim = 3, arrdim = totdim, coardim = 3

integer :: img, nimgs, comm, ierr=0, rank=0, mpisize=0

integer, dimension(totdim) :: asizehal
integer, dimension(arrdim) :: arrsize, arstart, artsize
integer, dimension(coardim) :: coarsize, copos

integer :: ncid=0, varid=0, dimids(arrdim)
integer :: x_dimid=0, y_dimid=0, z_dimid=0

  img = this_image()
nimgs = num_images()

asizehal(:) = shape( space )
   copos(:) = this_image( space )

! Subtract halos
 arrsize(:) = asizehal(1:arrdim) - 2*hdepth
coarsize(:) = ucobound(space) - lcobound(space) + 1

! Does the array fit exactly?
if ( product( coarsize ) .ne. nimgs) then
 write(*,*) 'ERROR: ca_hx/ca_co_netcdf: non-conforming coarray'
 error stop
end if

comm = MPI_COMM_WORLD
call MPI_Comm_size( comm, mpisize, ierr )
call MPI_Comm_rank( comm, rank, ierr )
```

```
! Sanity check
if ( mpisize .ne. nimgs .or. rank .ne. img-1 ) then
  write(*,*) 'ERROR: ca_hx/ca_co_netcdf: MPI/coarray mismatch'
  error stop
end if


! This is the global arrayy
artsize(:) = arrsize(:) * coarsize(:)

! Correspondent portion of this global array
arstart(:) = arrsize(:) * (copos(:)-1) + 1 ! Use Fortran indexing

! ! debug
! write (*,*) "netCDF-image",img, "asizehal", asizehal, "copos", copos,         &
!  "arrsize", arrsize, "coarsize", coarsize,                              &
!  "artsize", artsize, "arstart", arstart

! Create (i.e. open) the netCDF file. The NF90_NETCDF4 flag causes a
! HDF5/netCDF-4 type file to be created. The comm and info parameters
! cause parallel I/O to be enabled.
ierr = nf90_create( fname, ior(nf90_netcdf4,nf90_mpiio), ncid, &
                    comm=comm, info=MPI_INFO_NULL )
if ( ierr .ne. nf90_noerr) then
  write (*,*) "ERROR: ca_hx/ca_co_netcdf: nf90_create:",               &
              nf90_strerror( ierr )
  error stop
end if

! Define the dimensions. NetCDF returns an ID for each. Any
! metadata operations must take place on ALL processors
ierr = nf90_def_dim( ncid, "x", artsize(1), x_dimid )
if ( ierr .ne. nf90_noerr) then
  write (*,*) "ERROR: ca_hx/ca_co_netcdf: nf90_def_dim X:",            &
              nf90_strerror( ierr )
  error stop
end if

ierr = nf90_def_dim( ncid, "y", artsize(2), y_dimid )
if ( ierr .ne. nf90_noerr) then
  write (*,*) "ERROR: ca_hx/ca_co_netcdf: nf90_def_dim Y:",            &
              nf90_strerror( ierr )
  error stop
end if

ierr = nf90_def_dim( ncid, "z", artsize(3), z_dimid )
if ( ierr .ne. nf90_noerr) then
  write (*,*) "ERROR: ca_hx/ca_co_netcdf: nf90_def_dim Z:",            &
              nf90_strerror( ierr )
  error stop
end if
```

```
! The dimids array is used to pass the ID's of the dimensions of
! the variables.
dimids = (/ x_dimid , y_dimid, z_dimid /)

! Define the variable. The type of the variable in this case is
! NF90_INT (4-byte int).
ierr = nf90_def_var(ncid, "data", NF90_INT, dimids, varid)
if ( ierr .ne. nf90_noerr) then
  write (*,*) "ERROR: ca_hx/ca_co_netcdf: nf90_def_var:",             &
              nf90_strerror( ierr )
  error stop
end if

! Make sure file it not filled with default values
! which doubles wrote volume
ierr = nf90_def_var_fill(ncid, varid, 1, 1)
if ( ierr .ne. nf90_noerr) then
  write (*,*) "ERROR: ca_hx/ca_co_netcdf: nf90_def_var_fill:",        &
              nf90_strerror( ierr )
  error stop
end if

! End define mode. This tells netCDF we are done defining
! metadata. This operation is collective and all processors will
! write their metadata to disk.
ierr = nf90_enddef(ncid)
if ( ierr .ne. nf90_noerr ) then
  write (*,*) "ERROR: ca_hx/ca_co_netcdf: nf90_enddef:",              &
              nf90_strerror( ierr )
  error stop
end if

! Parallel access
ierr = nf90_var_par_access( ncid, varid, nf90_collective )
if ( ierr .ne. nf90_noerr ) then
  write (*,*) "ERROR: ca_hx/ca_co_netcdf: nf90_var_par_access:",      &
              nf90_strerror( ierr )
  error stop
end if

! Write the data to file, start will equal the displacement from the
! start of the file and count is the number of points each proc writes.
ierr = nf90_put_var( ncid, varid,                                     &
             space( 1:arrsize(1), 1:arrsize(2), 1:arrsize(3) ),       &
             start = arstart, count = arrsize )
if ( ierr .ne. nf90_noerr) then
  write (*,*) "ERROR: ca_hx/ca_co_netcdf: nf90_put_var:",             &
              nf90_strerror( ierr )
  error stop
end if

! Close the file. This frees up any internal netCDF resources
```

```
! associated with the file, and flushes any buffers.
ierr = nf90_close(ncid)
if ( ierr .ne. nf90_noerr) then
  write (*,*) "ERROR: ca_hx/ca_co_netcdf: nf90_close:",                &
              nf90_strerror( ierr )
  error stop
end if

end subroutine ca_co_netcdf
```

1.6.6 ca_hx_co/ca_co_run

[ca_hx_co] [Subroutines]

NAME

ca_co_run

SYNOPSIS

```
module subroutine ca_co_run( space, hx_sub, iter_sub, kernel, niter )
```

INPUT

```
integer( kind=iarr ), intent(inout), allocatable :: space(:,:,:) [:,:,:]
procedure( hx_co_proto )  :: hx_sub
procedure( iter_proto )   :: iter_sub
procedure( kernel_proto ) :: kernel
integer, intent(in) :: niter

!       space - space coarray before iterations start
!      hx_sub - HX routine, e.g.
!             - ca_co_hx_all
!    iter_sub - the subroutine performing a single CA iteration, e.g.
!             - ca_iter_tl - triple nested loop
!             - ca_iter_dc - do concurrent loop
!             - ca_iter_omp - OpenMP loop
!      kernel - a function to be called for every cell inside the loop
!        iter - number of iterations to do
```

OUTPUT

```
!    space - CA coarray at the end of niter iterations
```

SIDE EFFECTS

module array tmp_space is updated

DESCRIPTION

This is a driver routine for CA iterations. HX is done before each iteration. Then a given number of iterations is performed with a given routine and a given kernel. One iteration is really 2 iterations: odd and even, Hence the upper loop limit is 2*niter.

USES USED BY SOURCE

```
integer :: i

tmp_space = space

do i = 1, 2*niter
  call hx_sub( space )                    ! space updated, with HX
  call iter_sub( space, hdepth, kernel ) ! tmp_space updated, local op
  space = tmp_space                       ! local op
  mask_array = 1_iarr - mask_array       ! Flip the mask array
```

```
end do

end subroutine ca_co_run
```

1.6.7 ca_hx_co/ca_co_spalloc

[ca_hx_co] [Subroutines]

NAME

ca_co_spalloc

SYNOPSIS

```
module subroutine ca_co_spalloc( space, c, d, ir )
```

INPUT

```
integer( kind=iarr ), allocatable, intent(inout) :: space(:,:,:) [:,:,:]
integer, intent(in) :: c(3), d, ir(3)

!    space - CA array to allocate, with halos!
!        c - array with space dimensions
!        d - depth of the halo layer
!       ir - codimensions
```

OUTPUT

```
!    space is allocated and set to zero.
```

SIDE EFFECTS

none

DESCRIPTION

This routine allocates the CA coarray array, with halos of depth d. Also save some vars in this module for future. Also, on first call, allocate a module work local space array, not a coarray (tmp_space) of the same mold as space. This array is used in CA iterations later.

USES USED BY SOURCE

```
integer :: i,j,k

if ( allocated( space ) ) then
  write (*,*) "WARN: ca_hx_co/ca_co_spalloc: image:", this_image(),    &
    "space already allocated, deallocating!"
  deallocate( space, stat=ierr, errmsg=errmsg )
  if ( ierr .ne. 0 ) then
    write (*,*)                                                        &
      "ERROR: ca_hx_co/ca_co_spalloc: deallocate( space ), ierr:",     &
                ierr, "errmsg:", trim(errmsg)
    error stop
  end if
end if

allocate( space( 1-d:c(1)+d, 1-d:c(2)+d, 1-d:c(3)+d )[ir(1), ir(2), *],&
          source=0_iarr, stat=ierr, errmsg=errmsg )
if ( ierr .ne. 0 ) then
```

```
  write (*,*)                                                          &
    "ERROR: ca_co_hx/ca_co_spalloc: allocate( space ), ierr:",         &
              ierr, "errmsg:", trim(errmsg)
  error stop
end if

! Calculate once and keep forever
! Need a picture to explain these halo vars.
! The picture is only for dimension 1.
! The same diagram can be drawn for dimensions 2 and 3.
!
! hdepth
! |    |
! +----+---------------------+----+--
! |    |                     |    | ^ hdepth
! |    |                     |    | v
! +----+---------------------+----+--
! |<-lhsta  |           |    |    |
! |   0|    |           |    |    |
! |    |1   |           |    |    |
! | hdepth->|           |    |    |
! |    |    |           |<-ihsta  |
! |    |    |           sub->|    |
! |    |    |           |    |<-rhsta
! |    |    |           |  rhend->|
! |    |    |           |    |    |
! +----+---------------------+----+
! |    |                     |    |
! |    |                     |    |
! +----+---------------------+----+

hdepth = d
   sub = c
 ihsta = sub - hdepth + 1
 rhsta = sub + 1
 rhend = sub + hdepth
 lhsta = -hdepth + 1

! Total number of cells in the global CA
total_cells = sub(1)*sub(2)*sub(3)*num_images()

! Tmp space array for CA iterations, allocated on the first call
if ( .not. allocated( tmp_space ) ) then
  allocate( tmp_space( 1-d:c(1)+d, 1-d:c(2)+d, 1-d:c(3)+d ),           &
            source=0_iarr, stat=ierr, errmsg=errmsg )
  if ( ierr .ne. 0 ) then
    write (*,"(a,i0,a,a)") "ERROR: ca_hx_co/ca_co_spalloc: " //        &
      "allocate( tmp_space ), ierr: ", ierr, " errmsg: ", trim(errmsg)
    error stop
  end if
end if
```

```
! mask_array must be (re)allocated.
! Mask array, no halos, the values are set in ca_halloc.
if ( allocated( mask_array ) ) then
  deallocate( mask_array, stat=ierr, errmsg=errmsg )
  if ( ierr .ne. 0 ) then
    write (*,"(a,i0,a,a)") "ERROR: ca_hx_co/ca_spalloc: " //           &
      "deallocate( mask_array ), ierr: ", ierr, "errmsg:", trim(errmsg)
    error stop
  end if
end if

allocate( mask_array( c(1), c(2), c(3) ), stat=ierr, errmsg=errmsg )
if ( ierr .ne. 0 ) then
  write (*,"(a,i0,a,a)") "ERROR: ca_hx_co/ca_co_spalloc: " //          &
    "allocate( mask_array ) ", "ierr:", ierr, "errmsg:", trim(errmsg)
  error stop
end if

! Calculate once and keep forever
  ci = this_image( space )
ucob = ucobound(   space )

main: associate( d => hdepth, c => sub )

! Now can set mask_array. The mask array must reflect
! the global CA space, i.e. must not be affected by partitioning
! of the model into images. This means the mask array must
! depend on coindex set of this image.
do concurrent( i=1:c(1), j=1:c(2), k=1:c(3) )
  mask_array(i,j,k) = int( mod( (i+j+k + (ci(1)-1)*c(1) +              &
    (ci(2)-1)*c(2) + (ci(3)-1)*c(3) ) , 2 ), kind=iarr )
end do

end associate main

! Calculate coindex sets and image numbers for the 6 neighbours
! Coindex sets
  nei_ci_L1 = (/ ci(1)-1, ci(2), ci(3) /)
  nei_ci_R1 = (/ ci(1)+1, ci(2), ci(3) /)
  nei_ci_L2 = (/ ci(1), ci(2)-1, ci(3) /)
  nei_ci_R2 = (/ ci(1), ci(2)+1, ci(3) /)
  nei_ci_L3 = (/ ci(1), ci(2), ci(3)-1 /)
  nei_ci_R3 = (/ ci(1), ci(2), ci(3)+1 /)

! Image index
  nei_img_L(1) = image_index( space, nei_ci_L1 )
  nei_img_R(1) = image_index( space, nei_ci_R1 )
  nei_img_L(2) = image_index( space, nei_ci_L2 )
  nei_img_R(2) = image_index( space, nei_ci_R2 )
  nei_img_L(3) = image_index( space, nei_ci_L3 )
  nei_img_R(3) = image_index( space, nei_ci_R3 )
```

```
end subroutine ca_co_spalloc
```

1.7 ca_hx/ca_hx_glbar

[ca_hx] [Subroutines]

NAME

ca_hx_glbar

SYNOPSIS

```
subroutine ca_hx_glbar( space )
```

INPUT

```
integer( kind=iarr ), intent(inout), allocatable :: space(:,:,:)

!    space - non coarray array with CA model
```

OUTPUT

```
!    none
```

SIDE EFFECTS

halo coarrays are updated

DESCRIPTION

The only difference from ca_hx_all (1.4) is that here I use a global barrier (hence the name GL BAR), sync all.

USES USED BY SOURCE

```
if ( ci(1) .ne. 1       ) then
  h1minu(:,:,:) = space(        1 : hdepth , 1:sub(2) , 1:sub(3) )
end if
if ( ci(1) .ne. ucob(1) ) then
  h1plus(:,:,:) = space( ihsta(1) : sub(1) , 1:sub(2) , 1:sub(3) )
end if
if ( ci(2) .ne. 1       ) then
  h2minu(:,:,:) = space( 1:sub(1) ,        1 : hdepth , 1:sub(3) )
end if
if ( ci(2) .ne. ucob(2) ) then
  h2plus(:,:,:) = space( 1:sub(1) , ihsta(2) : sub(2) , 1:sub(3) )
end if
if ( ci(3) .ne. 1       ) then
  h3minu(:,:,:) = space( 1:sub(1) , 1:sub(2) ,        1 : hdepth )
end if
if ( ci(3) .ne. ucob(3) ) then
  h3plus(:,:,:) = space( 1:sub(1) , 1:sub(2) , ihsta(3) : sub(3) )
end if

sync all

if ( ci(1) .ne. 1 ) then
```

```
  space( lhsta(1):0, 1:sub(2), 1:sub(3) ) =                          &
    h1plus(:,:,:) [ nei_ci_L1(1), nei_ci_L1(2), nei_ci_L1(3) ]
end if

if ( ci(1) .ne. ucob(1) ) then
  space( rhsta(1) : rhend(1), 1:sub(2) , 1:sub(3) ) =                &
    h1minu(:,:,:) [ nei_ci_R1(1), nei_ci_R1(2), nei_ci_R1(3) ]
end if

if ( ci(2) .ne. 1 ) then
  space( 1:sub(1) , lhsta(2) : 0, 1:sub(3) ) =                       &
    h2plus(:,:,:) [ nei_ci_L2(1), nei_ci_L2(2), nei_ci_L2(3) ]
end if

if ( ci(2) .ne. ucob(2) ) then
  space( 1:sub(1) , rhsta(2) : rhend(2) , 1:sub(3) ) =               &
    h2minu(:,:,:) [ nei_ci_R2(1), nei_ci_R2(2), nei_ci_R2(3) ]
end if

if ( ci(3) .ne. 1 ) then
  space( 1:sub(1) , 1:sub(2) , lhsta(3) : 0 ) =                      &
    h3plus(:,:,:) [ nei_ci_L3(1), nei_ci_L3(2), nei_ci_L3(3) ]
end if

if ( ci(3) .ne. ucob(3) ) then
  space( 1:sub(1) , 1:sub(2) , rhsta(3) : rhend(3) ) =               &
    h3minu(:,:,:) [ nei_ci_R3(1), nei_ci_R3(2), nei_ci_R3(3) ]
end if

end subroutine ca_hx_glbar
```

1.8 ca_hx/ca_hx_mpi

[ca_hx] [Submodules]

NAME

ca_hx_mpi

SYNOPSIS

```
!$Id: ca_hx_mpi.f90 560 2018-10-14 19:02:34Z mexas $

submodule ( ca_hx ) ca_hx_mpi
```

DESCRIPTION

Submodule of module ca_hx (1) with MPI related routines. To aid portability, the module works only with default integer kind, i.e. MPI_integer. Other MPI integer kinds might not be widely available, meaning that other Fortran integer kinds might be less portable. So make sure that space array kind is the same as default integer. This is likely to be the case with

```
integer, parameter :: iarr = selected_int_kind( 8 )
```

iarr (9.26) is set in cgca_m1co (9).

Creation/release (free) of MPI types is left as the user's responsibility. This is because the user might want to change halo depth in the same program. This is hard/impossible to keep completely invisible to the user.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

CONTAINS

ca_mpi_halo_type_create (1.8.1), ca_mpi_halo_type_free (1.8.2), ca_mpi_hxvn1m (1.8.4), & ca_mpi_hxvn1p (1.8.5), ca_mpi_hxvn2m (1.8.6), ca_mpi_hxvn2p (1.8.7), ca_mpi_hxvn3m (1.8.8), & ca_mpi_hxvn3p (1.8.9), ca_mpi_hx_all (1.8.3)

USES USED BY SOURCE

```
! For reference
!
! MPI_SEND(  BUF, COUNT, DATATYPE, DEST,   TAG, COMM,          IERROR )
! MPI_RECV(  BUF, COUNT, DATATYPE, SOURCE, TAG, COMM, STATUS,  IERROR )
!
! MPI_ISEND( BUF, COUNT, DATATYPE, DEST,   TAG, COMM, REQUEST, IERROR )
! MPI_IRECV( BUF, COUNT, DATATYPE, SOURCE, TAG, COMM, REQUEST, IERROR )

implicit none

! Tags for sending messages in 6 directions.
integer, parameter :: TAG1L = 1, TAG1R = 2, TAG2L = 3, TAG2R = 4,      &
  TAG3L = 5, TAG3R = 6
```

```
integer, save :: &
  rank,          & ! MPI rank
  !status( MPI_STATUS_SIZE ), & ! used in MPI_RECV, etc.
  mpi_h1_LV,     & ! MPI halo, dim 1, left virtual
  mpi_h1_LR,     & ! MPI halo, dim 1, left real
  mpi_h1_RR,     & ! MPI halo, dim 1, right real
  mpi_h1_RV,     & ! MPI halo, dim 1, right virtual
  mpi_h2_LV,     & ! MPI halo, dim 2, left virtual
  mpi_h2_LR,     & ! MPI halo, dim 2, left real
  mpi_h2_RR,     & ! MPI halo, dim 2, right real
  mpi_h2_RV,     & ! MPI halo, dim 2, right virtual
  mpi_h3_LV,     & ! MPI halo, dim 3, left virtual
  mpi_h3_LR,     & ! MPI halo, dim 3, left real
  mpi_h3_RR,     & ! MPI halo, dim 3, right real
  mpi_h3_RV,     & ! MPI halo, dim 3, right virtual
  mpi_ca_integer,& ! MPI matching type for iarr
  errcode,       & ! Need to preserve ierr
  errlen           ! The length of the output error message

! A flag to track the state of MPI types for halos.
! Set initially to .false.
! Calling ca_mpi_halo_type_create sets it to .true.
! Calling ca_mpi_halo_type_free sets it to .false. again.

logical, save :: halo_type_created = .false.

contains
```

1.8.1 ca_hx_mpi/ca_mpi_halo_type_create

[ca_hx_mpi] [Subroutines]

NAME

ca_mpi_halo_type_create

SYNOPSIS

```
module subroutine ca_mpi_halo_type_create( space )
```

INPUT

```
integer( kind=iarr), intent( inout ), allocatable :: space(:,:,:)

!    space - the CA array
```

OUTPUT

```
!    none
```

SIDE EFFECTS

12 MPI halo types, module variables, are created.

DESCRIPTION

For each direction there are 4 MPI halo data types:

- array elements in the halo part of the array to the left of the real data,
- array elements of halo thickness inside the real part of the array on its left side,
- array elements of halo thickness inside the real part of the array on its right side,
- array elements in the hallp part of the array to the right of the real data.

Refer to the diagram in ca_hx (1)/ca_spalloc (1.19).

NOTES

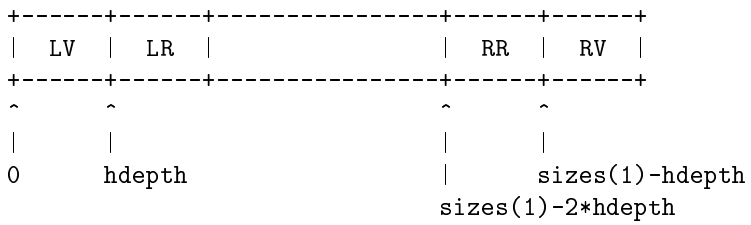
Call this routine after ca_halloc (1.1). All images must call this routine! Pay particular attention to the starts of all arrays. Refer to the details in e.g:

```
https://www.open-mpi.org/doc/v3.0/man3/MPI_Type_create_subarray.3.php
```

In particular:

```
In a Fortran program with arrays indexed starting from 1,
if the starting coordinate of a particular dimension
of the subarray is n, then the entry in array of starts
for that dimension is n-1.
```

A diagram is probably needed for starts, because it's different from that in ca_hx (1)/ca_spalloc (1.19). Using only a single dimension, e.g. 1.



starts for 4 halo arrays along dim 1

USES USED BY SOURCE

```
!MPI_TYPE_CREATE_SUBARRAY(NDIMS, ARRAY_OF_SIZES, ARRAY_OF_SUBSIZES,
!    ARRAY_OF_STARTS, ORDER, OLDTYPE, NEWTYPE, IERROR)
!
!    INTEGER    NDIMS, ARRAY_OF_SIZES(*), ARRAY_OF_SUBSIZES(*),
!    ARRAY_OF_STARTS(*), ORDER, OLDTYPE, NEWTYPE, IERROR

integer :: sizes(3), subsizes(3), starts(3)

! Set MPI rank, keep forever
call MPI_COMM_RANK( MPI_COMM_WORLD, rank, ierr)
!write (*,*) "my rank:", rank, "my img:", this_image()

! Set MPI matching type for iarr: mpi_ca_integer.
! Set once, keep forever.
call MPI_TYPE_CREATE_F90_INTEGER( ca_range, mpi_ca_integer, ierr )

! The sizes is just the shape of the space array, for all cases
sizes = shape( space )

! Dimension 1

subsizes = (/ hdepth, sub(2), sub(3) /)

! 1. dimension 1, left virtual (LV) type

starts = (/ 0, hdepth, hdepth /)

!write (*,"(3(a,3(i0,tr1)))") "sizes: ", sizes, " subsizes: ", subsizes, " starts: ", starts

call MPI_TYPE_CREATE_SUBARRAY( 3, sizes, subsizes, starts,            &
  MPI_ORDER_FORTRAN, mpi_ca_integer, mpi_h1_LV, ierr )
if ( ierr .ne. MPI_SUCCESS ) then
  errcode = ierr
  call MPI_ERROR_STRING( errcode, errmsg, errlen, ierr )
  write (*,"(a,i0,a)") "ERROR ca_hx_mpi/ca_mpi_halo_type_create: " //  &
    "MPI_TYPE_CREATE_SUBARRAY: dim 1: left virtual (LV): error: ",     &
    errcode, " error message: " // trim(errmsg)
  error stop
end if
```

```
call MPI_TYPE_COMMIT( mpi_h1_LV, ierr )
if ( ierr .ne. MPI_SUCCESS ) then
  write (*,"(a,i0)") "ERROR ca_hx_mpi/ca_mpi_halo_type_create: " //    &
    "MPI_TYPE_COMMIT: dim 1: left virtual (LV): ierr: ", ierr
  error stop
end if

! 2. dimension 1, left real (LR) type

starts = (/ hdepth, hdepth, hdepth /)

call MPI_TYPE_CREATE_SUBARRAY( 3, sizes, subsizes, starts,             &
  MPI_ORDER_FORTRAN, mpi_ca_integer, mpi_h1_LR, ierr )
if ( ierr .ne. MPI_SUCCESS ) then
  write (*,"(a,i0)") "ERROR ca_hx_mpi/ca_mpi_halo_type_create: " //    &
    "MPI_TYPE_CREATE_SUBARRAY: dim 1: left real (LR): ierr: ", ierr
  error stop
end if

call MPI_TYPE_COMMIT( mpi_h1_LR, ierr )
if ( ierr .ne. MPI_SUCCESS ) then
  write (*,"(a,i0)") "ERROR ca_hx_mpi/ca_mpi_halo_type_create: " //    &
    "MPI_TYPE_COMMIT: dim 1: left real (LR): ierr: ", ierr
  error stop
end if

! 3. dimension 1, right real (RR) type

starts = (/ sizes(1) - 2*hdepth, hdepth, hdepth /)

call MPI_TYPE_CREATE_SUBARRAY( 3, sizes, subsizes, starts,             &
  MPI_ORDER_FORTRAN, mpi_ca_integer, mpi_h1_RR, ierr )
if ( ierr .ne. MPI_SUCCESS ) then
  write (*,"(a,i0)") "ERROR ca_hx_mpi/ca_mpi_halo_type_create: " //    &
    "MPI_TYPE_CREATE_SUBARRAY: dim 1: right real (RR): ierr: ", ierr
  error stop
end if

call MPI_TYPE_COMMIT( mpi_h1_RR, ierr )
if ( ierr .ne. MPI_SUCCESS ) then
  write (*,"(a,i0)") "ERROR ca_hx_mpi/ca_mpi_halo_type_create: " //    &
    "MPI_TYPE_COMMIT: dim 1: right real (RR): ierr: ", ierr
  error stop
end if

! 4. dimension 1, right virtual (RV) type

starts = (/ sizes(1) - hdepth, hdepth, hdepth /)

call MPI_TYPE_CREATE_SUBARRAY( 3, sizes, subsizes, starts,             &
  MPI_ORDER_FORTRAN, mpi_ca_integer, mpi_h1_RV, ierr )
if ( ierr .ne. MPI_SUCCESS ) then
```

```
  write (*,"(a,i0)") "ERROR ca_hx_mpi/ca_mpi_halo_type_create: " //    &
    "MPI_TYPE_CREATE_SUBARRAY: dim 1: right virtual (RV): ierr: ", ierr
  error stop
end if

call MPI_TYPE_COMMIT( mpi_h1_RV, ierr )
if ( ierr .ne. MPI_SUCCESS ) then
  write (*,"(a,i0)") "ERROR ca_hx_mpi/ca_mpi_halo_type_create: " //    &
    "MPI_TYPE_COMMIT: dim 1: right virtual (RV): ierr: ", ierr
  error stop
end if

! Dimension 2

subsizes = (/ sub(1), hdepth, sub(3) /)

! 5. dimension 2, left virtual (LV) type

starts = (/ hdepth, 0, hdepth /)

call MPI_TYPE_CREATE_SUBARRAY( 3, sizes, subsizes, starts,             &
  MPI_ORDER_FORTRAN, mpi_ca_integer, mpi_h2_LV, ierr )
if ( ierr .ne. MPI_SUCCESS ) then
  write (*,"(a,i0)") "ERROR ca_hx_mpi/ca_mpi_halo_type: " //           &
    "MPI_TYPE_CREATE_SUBARRAY: dim 2: left virtual (LV): ierr: ", ierr
  error stop
end if

call MPI_TYPE_COMMIT( mpi_h2_LV, ierr )
if ( ierr .ne. MPI_SUCCESS ) then
  write (*,"(a,i0)") "ERROR ca_hx_mpi/ca_mpi_halo_type: " //           &
    "MPI_TYPE_COMMIT: dim 2: left virtual (LV): ierr: ", ierr
  error stop
end if

! 6. dimension 2, left real (LR) type

starts = (/ hdepth, hdepth, hdepth /)

call MPI_TYPE_CREATE_SUBARRAY( 3, sizes, subsizes, starts,             &
  MPI_ORDER_FORTRAN, mpi_ca_integer, mpi_h2_LR, ierr )
if ( ierr .ne. MPI_SUCCESS ) then
  write (*,"(a,i0)") "ERROR ca_hx_mpi/ca_mpi_halo_type: " //           &
    "MPI_TYPE_CREATE_SUBARRAY: dim 2: left real (LR): ierr: ", ierr
  error stop
end if

call MPI_TYPE_COMMIT( mpi_h2_LR, ierr )
if ( ierr .ne. MPI_SUCCESS ) then
  write (*,"(a,i0)") "ERROR ca_hx_mpi/ca_mpi_halo_type: " //           &
    "MPI_TYPE_COMMIT: dim 2: left real (LR): ierr: ", ierr
  error stop
```

```
end if

! 7. dimension 2, right real (RR) type

starts = (/ hdepth, sizes(2) - 2*hdepth, hdepth /)

call MPI_TYPE_CREATE_SUBARRAY( 3, sizes, subsizes, starts,             &
  MPI_ORDER_FORTRAN, mpi_ca_integer, mpi_h2_RR, ierr )
if ( ierr .ne. MPI_SUCCESS ) then
  write (*,"(a,i0)") "ERROR ca_hx_mpi/ca_mpi_halo_type: " //           &
    "MPI_TYPE_CREATE_SUBARRAY: dim 2: right real (RR): ierr: ", ierr
  error stop
end if

call MPI_TYPE_COMMIT( mpi_h2_RR, ierr )
if ( ierr .ne. MPI_SUCCESS ) then
  write (*,"(a,i0)") "ERROR ca_hx_mpi/ca_mpi_halo_type: " //           &
    "MPI_TYPE_COMMIT: dim 2: right real (RR): ierr: ", ierr
  error stop
end if

! 8. dimension 2, right virtual (RV) type

starts = (/ hdepth, sizes(2) - hdepth, hdepth /)

call MPI_TYPE_CREATE_SUBARRAY( 3, sizes, subsizes, starts,             &
  MPI_ORDER_FORTRAN, mpi_ca_integer, mpi_h2_RV, ierr )
if ( ierr .ne. MPI_SUCCESS ) then
  write (*,"(a,i0)") "ERROR ca_hx_mpi/ca_mpi_halo_type: " //           &
    "MPI_TYPE_CREATE_SUBARRAY: dim 2: right virtual (RV): ierr: ", ierr
  error stop
end if

call MPI_TYPE_COMMIT( mpi_h2_RV, ierr )
if ( ierr .ne. MPI_SUCCESS ) then
  write (*,"(a,i0)") "ERROR ca_hx_mpi/ca_mpi_halo_type: " //           &
    "MPI_TYPE_COMMIT: dim 2: right virtual (RV): ierr: ", ierr
  error stop
end if

! Dimension 3

subsizes = (/ sub(1), sub(2), hdepth /)

! 9. dimension 3, left virtual (LV) type

starts = (/ hdepth, hdepth, 0 /)

call MPI_TYPE_CREATE_SUBARRAY( 3, sizes, subsizes, starts,             &
  MPI_ORDER_FORTRAN, mpi_ca_integer, mpi_h3_LV, ierr )
if ( ierr .ne. MPI_SUCCESS ) then
  write (*,"(a,i0)") "ERROR ca_hx_mpi/ca_mpi_halo_type: " //           &
```

```
    "MPI_TYPE_CREATE_SUBARRAY: dim 3: left virtual (LV): ierr: ", ierr
  error stop
end if

call MPI_TYPE_COMMIT( mpi_h3_LV, ierr )
if ( ierr .ne. MPI_SUCCESS ) then
  write (*,"(a,i0)") "ERROR ca_hx_mpi/ca_mpi_halo_type: " //           &
    "MPI_TYPE_COMMIT: dim 3: left virtual (LV): ierr: ", ierr
  error stop
end if

! 10. dimension 3, left real (LR) type

starts = (/ hdepth, hdepth, hdepth /)

call MPI_TYPE_CREATE_SUBARRAY( 3, sizes, subsizes, starts,             &
  MPI_ORDER_FORTRAN, mpi_ca_integer, mpi_h3_LR, ierr )
if ( ierr .ne. MPI_SUCCESS ) then
  write (*,"(a,i0)") "ERROR ca_hx_mpi/ca_mpi_halo_type: " //           &
    "MPI_TYPE_CREATE_SUBARRAY: dim 3: left real (LR): ierr: ", ierr
  error stop
end if

call MPI_TYPE_COMMIT( mpi_h3_LR, ierr )
if ( ierr .ne. MPI_SUCCESS ) then
  write (*,"(a,i0)") "ERROR ca_hx_mpi/ca_mpi_halo_type: " //           &
    "MPI_TYPE_COMMIT: dim 3: left real (LR): ierr: ", ierr
  error stop
end if

! 11. dimension 3, right real (RR) type

starts = (/ hdepth, hdepth, sizes(3) - 2*hdepth /)

call MPI_TYPE_CREATE_SUBARRAY( 3, sizes, subsizes, starts,             &
  MPI_ORDER_FORTRAN, mpi_ca_integer, mpi_h3_RR, ierr )
if ( ierr .ne. MPI_SUCCESS ) then
  write (*,"(a,i0)") "ERROR ca_hx_mpi/ca_mpi_halo_type: " //           &
    "MPI_TYPE_CREATE_SUBARRAY: dim 3: right real (RR): ierr: ", ierr
  error stop
end if

call MPI_TYPE_COMMIT( mpi_h3_RR, ierr )
if ( ierr .ne. MPI_SUCCESS ) then
  write (*,"(a,i0)") "ERROR ca_hx_mpi/ca_mpi_halo_type: " //           &
    "MPI_TYPE_COMMIT: dim 3: right real (RR): ierr: ", ierr
  error stop
end if

! 12. dimension 3, right virtual (RV) type

starts = (/ hdepth, hdepth, sizes(3) - hdepth /)
```

```
call MPI_TYPE_CREATE_SUBARRAY( 3, sizes, subsizes, starts,             &
  MPI_ORDER_FORTRAN, mpi_ca_integer, mpi_h3_RV, ierr )
if ( ierr .ne. MPI_SUCCESS ) then
  write (*,"(a,i0)") "ERROR ca_hx_mpi/ca_mpi_halo_type: " //           &
    "MPI_TYPE_CREATE_SUBARRAY: dim 3: right virtual (RV): ierr: ", ierr
  error stop
end if

call MPI_TYPE_COMMIT( mpi_h3_RV, ierr )
if ( ierr .ne. MPI_SUCCESS ) then
  write (*,"(a,i0)") "ERROR ca_hx_mpi/ca_mpi_halo_type: " //           &
    "MPI_TYPE_COMMIT: dim 3: right virtual (RV): ierr: ", ierr
  error stop
end if

! MPI types for halos have been created.
! Set the corresponding flag to .true.
halo_type_created = .true.

end subroutine ca_mpi_halo_type_create
```

1.8.2 ca_hx_mpi/ca_mpi_halo_type_free

[ca_hx_mpi] [Subroutines]

NAME

ca_mpi_halo_type_free

SYNOPSIS

```
module subroutine ca_mpi_halo_type_free
```

INPUT

```
!    none
```

OUTPUT

```
!    none
```

SIDE EFFECTS

12 MPI halo types, module variables, are freed.

DESCRIPTION

Refer to ca_mpi_halo_type_create (1.8.1) for details of these 12 types. Need to call this routine if want to re-create the halo types, perhaps with different halo depth, or for a different space array.

NOTES

Will give an error if data types are not committed. All images must call this routine!

USES USED BY SOURCE

```
! Dimension 1

call MPI_TYPE_FREE( mpi_h1_LV, ierr )
if ( ierr .ne. MPI_SUCCESS ) then
  write (*,"(a,i0)") "ERROR ca_hx_mpi/ca_mpi_halo_type_free: " //      &
    "MPI_TYPE_FREE: dim 1: left virtual (LV): ierr: ", ierr
  error stop
end if

call MPI_TYPE_FREE( mpi_h1_LR, ierr )
if ( ierr .ne. MPI_SUCCESS ) then
  write (*,"(a,i0)") "ERROR ca_hx_mpi/ca_mpi_halo_type_free: " //      &
    "MPI_TYPE_FREE: dim 1: left real (LR): ierr: ", ierr
  error stop
end if

call MPI_TYPE_FREE( mpi_h1_RR, ierr )
if ( ierr .ne. MPI_SUCCESS ) then
  write (*,"(a,i0)") "ERROR ca_hx_mpi/ca_mpi_halo_type_free: " //      &
    "MPI_TYPE_FREE: dim 1: right real (RR): ierr: ", ierr
  error stop
```

```
end if

call MPI_TYPE_FREE( mpi_h1_RV, ierr )
if ( ierr .ne. MPI_SUCCESS ) then
  write (*,"(a,i0)") "ERROR ca_hx_mpi/ca_mpi_halo_type_free: " //      &
    "MPI_TYPE_FREE: dim 1: right virtual (RV): ierr: ", ierr
  error stop
end if

! Dimension 2

call MPI_TYPE_FREE( mpi_h2_LV, ierr )
if ( ierr .ne. MPI_SUCCESS ) then
  write (*,"(a,i0)") "ERROR ca_hx_mpi/ca_mpi_halo_type_free: " //      &
    "MPI_TYPE_FREE: dim 2: left virtual (LV): ierr: ", ierr
  error stop
end if

call MPI_TYPE_FREE( mpi_h2_LR, ierr )
if ( ierr .ne. MPI_SUCCESS ) then
  write (*,"(a,i0)") "ERROR ca_hx_mpi/ca_mpi_halo_type_free: " //      &
    "MPI_TYPE_FREE: dim 2: left real (LR): ierr: ", ierr
  error stop
end if

call MPI_TYPE_FREE( mpi_h2_RR, ierr )
if ( ierr .ne. MPI_SUCCESS ) then
  write (*,"(a,i0)") "ERROR ca_hx_mpi/ca_mpi_halo_type_free: " //      &
    "MPI_TYPE_FREE: dim 2: right real (RR): ierr: ", ierr
  error stop
end if

call MPI_TYPE_FREE( mpi_h2_RV, ierr )
if ( ierr .ne. MPI_SUCCESS ) then
  write (*,"(a,i0)") "ERROR ca_hx_mpi/ca_mpi_halo_type_free: " //      &
    "MPI_TYPE_FREE: dim 2: right virtual (RV): ierr: ", ierr
  error stop
end if

! Dimension 3

call MPI_TYPE_FREE( mpi_h3_LV, ierr )
if ( ierr .ne. MPI_SUCCESS ) then
  write (*,"(a,i0)") "ERROR ca_hx_mpi/ca_mpi_halo_type_free: " //      &
    "MPI_TYPE_FREE: dim 3: left virtual (LV): ierr: ", ierr
  error stop
end if

call MPI_TYPE_FREE( mpi_h3_LR, ierr )
if ( ierr .ne. MPI_SUCCESS ) then
  write (*,"(a,i0)") "ERROR ca_hx_mpi/ca_mpi_halo_type_free: " //      &
    "MPI_TYPE_FREE: dim 3: left real (LR): ierr: ", ierr
```

```
  error stop
end if

call MPI_TYPE_FREE( mpi_h3_RR, ierr )
if ( ierr .ne. MPI_SUCCESS ) then
  write (*,"(a,i0)") "ERROR ca_hx_mpi/ca_mpi_halo_type_free: " //      &
    "MPI_TYPE_FREE: dim 3: right real (RR): ierr: ", ierr
  error stop
end if

call MPI_TYPE_FREE( mpi_h3_RV, ierr )
if ( ierr .ne. MPI_SUCCESS ) then
  write (*,"(a,i0)") "ERROR ca_hx_mpi/ca_mpi_halo_type_free: " //      &
    "MPI_TYPE_FREE: dim 3: right virtual (RV): ierr: ", ierr
  error stop
end if

! MPI types have been freed.
! Reset the flag back to .false.
! Will need to re-create MPI types for halos *before* any HX.

halo_type_created = .false.

end subroutine ca_mpi_halo_type_free
```

1.8.3 ca_hx_mpi/ca_mpi_hx_all

[ca_hx_mpi] [Subroutines]

NAME

ca_mpi_hx_all

SYNOPSIS

```
module subroutine ca_mpi_hx_all( space )
```

INPUT

```
integer( kind=iarr ), intent(inout), allocatable :: space(:,:,:)

!    space - non coarray array with CA model
```

OUTPUT

```
!    space is changed
```

SIDE EFFECTS

none

DESCRIPTION

Do all MPI HX. To avoid problems I don't allow the user to call individual hx routines. These are private to this module. The user only calls this routine.

NOTE

ca_mpi_halo_type_create (1.8.1) must be called prior to calling this routine. Note! This routine will only work if iarr (9.26) is the *default* integer. This is because MPI_INTEGER is used for space, as other MPI integer kinds might not be implemented.

USES

ca_mpi_hxvn1m (1.8.4), ca_mpi_hxvn1p (1.8.5), ca_mpi_hxvn2m (1.8.6), ca_mpi_hxvn2p (1.8.7), ca_mpi_hxvn3m (1.8.8), ca_mpi_hxvn3p (1.8.9)

USED BY SOURCE

```
! Make sure (some) MPI halo types have been created.
if ( .not. halo_type_created ) then
  write (*,"(a)") "ERROR ca_hx_mpi/ca_mpi_hx_all: Need to create " //  &
    "MPI types. Call ca_mpi_halo_type_create first!"
  error stop
end if

call ca_mpi_hxvn1m( space )
call ca_mpi_hxvn1p( space )
call ca_mpi_hxvn2m( space )
call ca_mpi_hxvn2p( space )
call ca_mpi_hxvn3m( space )
call ca_mpi_hxvn3p( space )
```

```
end subroutine ca_mpi_hx_all
```

1.8.4 ca_hx_mpi/ca_mpi_hxvn1m

[ca_hx_mpi] [Subroutines]

NAME

ca_mpi_hxvn1m

SYNOPSIS

```
module subroutine ca_mpi_hxvn1m( space )
```

INPUT

```
integer( kind=iarr), intent(inout), allocatable :: space(:,:,:)

!    space - the CA array
```

OUTPUT

```
!    space is updated
```

SIDE EFFECTS

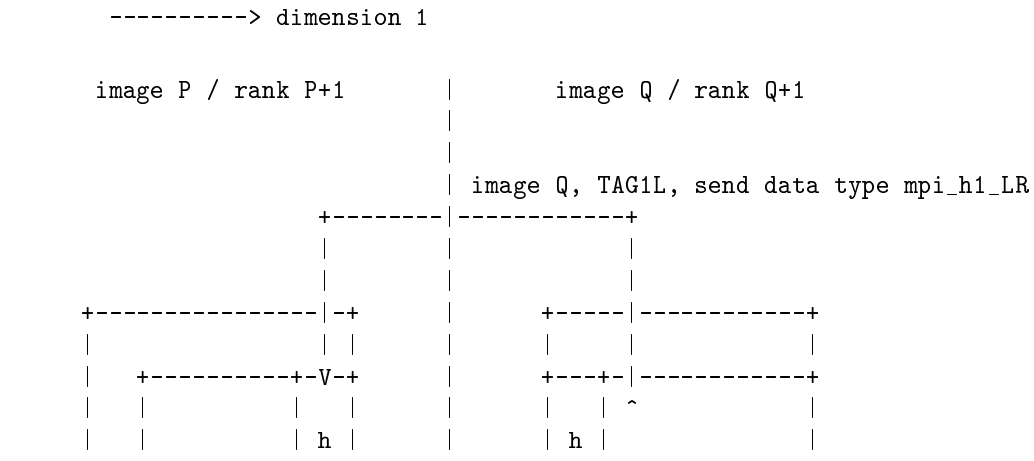
none

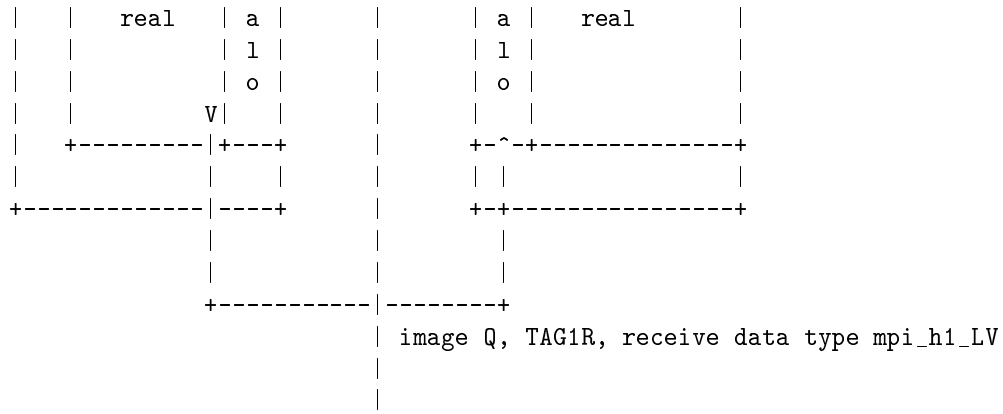
DESCRIPTION

Use non-blocking send/receive. An image does 2 remote ops:

- Send its space array real halo layer, left side (mpi_h1_LR) along dimension 1 into a virtual halo layer, right side (mpi_h1_RV) on an image which is 1 lower along codimension 1. Tag this message with TAG1L.
- Receive its space array virtual halo layer, left side (mpi_h1_LV) along dimension 1 from a real halo layer, right side (mpi_h1_RR) on an image which is 1 lower along codimension 1. Tag this message with TAG1R.

Schematic diagram, only showing what is relevant for HX along dimension 1:





USES USED BY

ca_mpi_hx_all (1.8.3)

SOURCE

```
integer :: reqs1m(2), stats(MPI_STATUS_SIZE, 2)

if ( ci(1) .ne. 1 ) then

  ! Rank is image number -1.

  ! Receive from the left neighbour, tag = TAG1R
  call MPI_IRECV( space, 1, mpi_h1_LV, nei_img_L(1)-1, TAG1R,          &
    MPI_COMM_WORLD, reqs1m(1), ierr )

  ! Send to the left neighbour, tag = TAG1L
  call MPI_ISEND( space, 1, mpi_h1_LR, nei_img_L(1)-1, TAG1L,          &
    MPI_COMM_WORLD, reqs1m(2), ierr )

  call MPI_WAITALL( 2, reqs1m, stats, ierr )

end if

end subroutine ca_mpi_hxvn1m
```

1.8.5 ca_hx_mpi/ca_mpi_hxvn1p

[ca_hx_mpi] [Subroutines]

NAME

ca_mpi_hxvn1p

SYNOPSIS

```
module subroutine ca_mpi_hxvn1p( space )
```

INPUT

```
integer( kind=iarr), intent(inout), allocatable :: space(:,:,:)

!    space - the CA array
```

OUTPUT

```
!    space is updated
```

SIDE EFFECTS

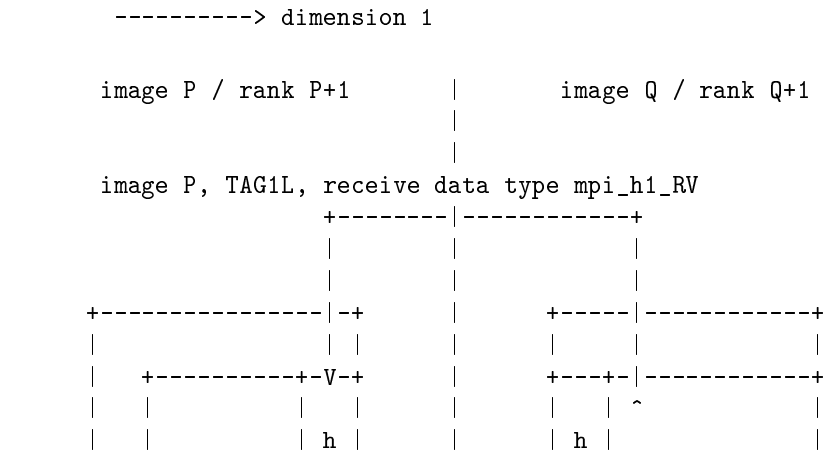
none

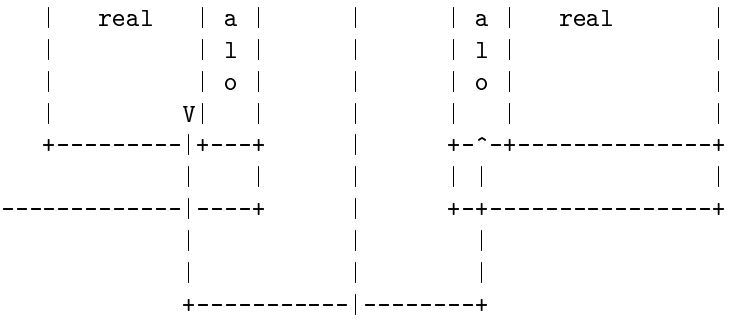
DESCRIPTION

Use non-blocking send/receive. An image does 2 remote ops:

- Send its space array real halo layer, right side (mpi_h1_RR) along dimension 1 into a virtual halo layer, left side (mpi_h1_LV) on an image which is 1 higher along codimension 1. Tag this message with TAG1R.
- Receive its space array virtual halo layer, right side (mpi_h1_RV) along dimension 1 from a real halo layer, right side (mpi_h1_LR) on an image which is 1 higher along codimension 1. Tag this message with TAG1L.

Schematic diagram, only showing what is relevant for HX along dimension 1:





USES USED BY

ca_mpi_hx_all (1.8.3)

SOURCE

```
integer :: reqs1p(2), stats(MPI_STATUS_SIZE, 2)

if ( ci(1) .ne. ucob(1) ) then

  ! Rank is image number -1.

  ! Receive from the right neighbour, tag = TAG1L
  call MPI_IRECV( space, 1, mpi_h1_RV, nei_img_R(1)-1, TAG1L,          &
    MPI_COMM_WORLD, reqs1p(1), ierr )

  ! Send to the right neighbour, tag = TAG1R
  call MPI_ISEND( space, 1, mpi_h1_RR, nei_img_R(1)-1, TAG1R,          &
    MPI_COMM_WORLD, reqs1p(2), ierr )

  call MPI_WAITALL( 2, reqs1p, stats, ierr )

end if

end subroutine ca_mpi_hxvn1p
```

1.8.6 ca_hx_mpi/ca_mpi_hxvn2m

[ca_hx_mpi] [Subroutines]

NAME

ca_mpi_hxvn2m

SYNOPSIS

```
module subroutine ca_mpi_hxvn2m( space )
```

INPUT

```
integer( kind=iarr ), intent(inout), allocatable :: space(:,:,:)

!    space - the CA array
```

OUTPUT

```
!    space is updated
```

SIDE EFFECTS

none

DESCRIPTION

HX along dimension 2. See ca_mpi_hxvn1m (1.8.4).

USES USED BY

ca_mpi_hx_all (1.8.3)

SOURCE

```
integer :: reqs2m(2), stats(MPI_STATUS_SIZE, 2)

if ( ci(2) .ne. 1 ) then

  ! Rank is image number -1.

  ! Receive from the left neighbour, tag = TAG2R
  call MPI_IRECV( space, 1, mpi_h2_LV, nei_img_L(2)-1, TAG2R,          &
    MPI_COMM_WORLD, reqs2m(1), ierr )

  ! Send to the left neighbour, tag = TAG2L
  call MPI_ISEND( space, 1, mpi_h2_LR, nei_img_L(2)-1, TAG2L,          &
    MPI_COMM_WORLD, reqs2m(2), ierr )

  call MPI_WAITALL( 2, reqs2m, stats, ierr )

end if

end subroutine ca_mpi_hxvn2m
```

1.8.7 ca_hx_mpi/ca_mpi_hxvn2p

[ca_hx_mpi] [Subroutines]

NAME

ca_mpi_hxvn2p

SYNOPSIS

```
module subroutine ca_mpi_hxvn2p( space )
```

INPUT

```
integer( kind=iarr ), intent(inout), allocatable :: space(:,:,:)

!    space - the CA array
```

OUTPUT

```
!    space is updated
```

SIDE EFFECTS

none

DESCRIPTION

HX along dimension 2. See ca_mpi_hxvn1p (1.8.5).

USES USED BY

ca_mpi_hx_all (1.8.3)

SOURCE

```
integer :: reqs2p(2), stats(MPI_STATUS_SIZE, 2)

if ( ci(2) .ne. ucob(2) ) then

  ! Rank is image number -1.

  ! Receive from the right neighbour, tag = TAG2L
  call MPI_IRECV( space, 1, mpi_h2_RV, nei_img_R(2)-1, TAG2L,          &
    MPI_COMM_WORLD, reqs2p(1), ierr )

  ! Send to the right neighbour, tag = TAG2R
  call MPI_ISEND( space, 1, mpi_h2_RR, nei_img_R(2)-1, TAG2R,          &
    MPI_COMM_WORLD, reqs2p(2), ierr )

  call MPI_WAITALL( 2, reqs2p, stats, ierr )

end if

end subroutine ca_mpi_hxvn2p
```

1.8.8 ca_hx_mpi/ca_mpi_hxvn3m

[ca_hx_mpi] [Subroutines]

NAME

ca_mpi_hxvn3m

SYNOPSIS

```
module subroutine ca_mpi_hxvn3m( space )
```

INPUT

```
integer( kind=iarr ), intent(inout), allocatable :: space(:,:,:)

!    space - the CA array
```

OUTPUT

```
!    space is updated
```

SIDE EFFECTS

none

DESCRIPTION

HX along dimension 3. See ca_mpi_hxvn1m (1.8.4).

USES USED BY

ca_mpi_hx_all (1.8.3)

SOURCE

```
integer :: reqs3m(2), stats(MPI_STATUS_SIZE, 2)

if ( ci(3) .ne. 1 ) then

  ! Rank is image number -1.

  ! Receive from the left neighbour, tag = TAG3R
  call MPI_IRECV( space, 1, mpi_h3_LV, nei_img_L(3)-1, TAG3R,          &
    MPI_COMM_WORLD, reqs3m(1), ierr )

  ! Send to the left neighbour, tag = TAG3L
  call MPI_ISEND( space, 1, mpi_h3_LR, nei_img_L(3)-1, TAG3L,          &
    MPI_COMM_WORLD, reqs3m(2), ierr )

  call MPI_WAITALL( 2, reqs3m, stats, ierr )

end if

end subroutine ca_mpi_hxvn3m
```

1.8.9 ca_hx_mpi/ca_mpi_hxvn3p

[ca_hx_mpi] [Subroutines]

NAME

ca_mpi_hxvn3p

SYNOPSIS

```
module subroutine ca_mpi_hxvn3p( space )
```

INPUT

```
integer( kind = iarr ), intent(inout), allocatable :: space(:,:,:)

!    space - the CA array
```

OUTPUT

```
!    space is updated
```

SIDE EFFECTS

none

DESCRIPTION

HX along dimension 3. See ca_mpi_hxvn1p (1.8.5).

USES USED BY

ca_mpi_hx_all (1.8.3)

SOURCE

```
integer :: reqs3p(2), stats(MPI_STATUS_SIZE, 2)

if ( ci(3) .ne. ucob(3) ) then

  ! Rank is image number -1.

  ! Receive from the right neighbour, tag = TAG3L
  call MPI_IRECV( space, 1, mpi_h3_RV, nei_img_R(3)-1, TAG3L,          &
    MPI_COMM_WORLD, reqs3p(1), ierr )

  ! Send to the right neighbour, tag = TAG3R
  call MPI_ISEND( space, 1, mpi_h3_RR, nei_img_R(3)-1, TAG3R,          &
    MPI_COMM_WORLD, reqs3p(2), ierr )

  call MPI_WAITALL( 2, reqs3p, stats, ierr )

end if

end subroutine ca_mpi_hxvn3p
```

1.8.10 ca_hx_mpi/ca_mpi_ising_energy

[ca_hx_mpi] [Subroutines]

NAME

ca_mpi_ising_energy

SYNOPSIS

```
module subroutine ca_mpi_ising_energy( space, hx_sub, iter_sub, kernel,&
  energy, magnet )
```

INPUT

```
integer( kind=iarr ), intent(inout), allocatable :: space(:,:,:)
procedure( hx_mpi_proto ) :: hx_sub
procedure( iter_proto ) :: iter_sub
procedure( kernel_proto ) :: kernel

!       space - space array before iterations start
!      hx_sub - ca_mpi_hx_all
!    iter_sub - the subroutine performing a single CA iteration, e.g.
!             - ca_iter_tl - triple nested loop
!             - ca_iter_dc - do concurrent
!             - ca_iter_omp - OpenMP
!      kernel - a function to be called for every cell inside the loop
```

OUTPUT

```
integer( kind=ilrg ), intent( out ) :: energy, magnet

!    energy - Total energy of CA system
!    magnet - Total magnetisation of the CA system
```

SIDE EFFECTS

module array tmp_space is updated

DESCRIPTION

Calculate the total energy and the total magnetisation of CA using ising (49) model. Note that I'm passing integers of kind ilrg (9.28) to MPI_INTEGER8. This should work as long as ilrg (9.28) is 8 bytes long. So set ilrg (9.28) to selected_int_kind(10). This routine uses MPI_ALLREDUCE with MPI_SUM. Magnetisation is defined as the fraction of the 1 spins. The only valid kernel is ca_kernel_ising_ener (1.16).

USES USED BY SOURCE

```
integer( kind=ilrg ) :: img_energy, img_magnet

call hx_sub( space )    ! space updated, sync images
! tmp_space updated, local op
call iter_sub( space=space, halo=hdepth, kernel=kernel )
img_energy = &
  int( sum( tmp_space( 1:sub(1), 1:sub(2), 1:sub(3) ) ), kind=ilrg )
```

```
img_magnet = &
  int( sum(     space( 1:sub(1), 1:sub(2), 1:sub(3) ) ), kind=ilrg )

! write (*,*) "img:", this_image(), "img_energy:", img_energy, "img_magnet:", img_magnet

call MPI_ALLREDUCE( img_energy, energy, 1, MPI_INTEGER8, MPI_SUM,      &
  MPI_COMM_WORLD, ierr)
call MPI_ALLREDUCE( img_magnet, magnet, 1, MPI_INTEGER8, MPI_SUM,      &
  MPI_COMM_WORLD, ierr)

end subroutine ca_mpi_ising_energy
```

1.9 ca_hx/ca_ising_energy

[ca_hx] [Subroutines]

NAME

ca_ising_energy

SYNOPSIS

```
subroutine ca_ising_energy( space, hx_sub, iter_sub, kernel, energy,   &
  magnet )
```

INPUT

```
integer( kind=iarr ), intent(inout), allocatable :: space(:,:,:)
procedure( hx_proto ) :: hx_sub
procedure( iter_proto ) :: iter_sub
procedure( kernel_proto ) :: kernel

!       space - space array before iterations start
!    iter_sub - the subroutine performing a single CA iteration, e.g.
!             - ca_iter_tl - triple nested loop
!             - ca_iter_dc - do concurrent
!             - ca_iter_omp - OpenMP
!      kernel - a function to be called for every cell inside the loop
```

OUTPUT

```
integer( kind=ilrg ), intent(out) :: energy, magnet

!    energy - Total energy of CA system
!    magnet - Total magnetisation of the CA system
```

SIDE EFFECTS

module array tmp_space is updated

DESCRIPTION

Calculate the total energy and the total magnetisation of CA using ising (49) model. This routine does not use collectives. Magnetisation is defined as the fraction of the 1 spins. The only valid kernel is ca_kernel_ising_ener (1.16).

USES USED BY SOURCE

```
integer( kind=ilrg) :: my_energy, my_magnet

co_energy = 0
co_magnet = 0

call hx_sub( space )                    ! space updated, sync images
call iter_sub( space, hdepth, kernel ) ! tmp_space updated, local op
my_energy =                                                          &
  int( sum( tmp_space( 1:sub(1), 1:sub(2), 1:sub(3) ) ), kind=ilrg )
```

```
my_magnet =                                                              &
  int( sum(     space( 1:sub(1), 1:sub(2), 1:sub(3) ) ), kind=ilrg )

!write (*,"(a,i0,a,999i3)") "img: ", this_image(), " tmp_space: ", tmp_space
!write (*,*) "img:", this_image(), "my energy:", my_energy

! This is a tmp version on systems with no CO_SUM!
! Change on Cray!
!
! Image 1 calculates the total values in a *coarray* variable!
critical
  co_energy[1] = co_energy[1] + my_energy
  co_magnet[1] = co_magnet[1] + my_magnet
end critical

! I read the total values from image 1
! Magnetisation is real value, scaled by the total number of
! cells in the global model.
sync all
energy = co_energy[1]
magnet = co_magnet[1]

! Better do the scaling in the end user program.
! I want the routine to return an integer magnetisation
! so that the results can be exactly reproducible.
!magnet = real( co_magnet[1] ) / real( total_cells )

end subroutine ca_ising_energy
```

1.10 ca_hx/ca_ising_energy_col

[ca_hx] [Subroutines]

NAME

ca_ising_energy_col

SYNOPSIS

```
subroutine ca_ising_energy_col( space, hx_sub, iter_sub, kernel,       &
  energy, magnet)
```

INPUT

```
integer( kind=iarr ), intent(inout), allocatable :: space(:,:,:)
procedure( hx_proto ) :: hx_sub
procedure( iter_proto ) :: iter_sub
procedure( kernel_proto ) :: kernel

!       space - space array before iterations start
!      hx_sub - ca_hx_all
!    iter_sub - the subroutine performing a single CA iteration, e.g.
!             - ca_iter_tl - triple nested loop
!             - ca_iter_dc - do concurrent
!             - ca_iter_omp - OpenMP
!      kernel - a function to be called for every cell inside the loop
```

OUTPUT

```
integer( kind=ilrg ) , intent(out) :: energy, magnet

!    energy - Total energy of CA system
!    magnet - Total magnetisation of the CA system
```

SIDE EFFECTS

module array tmp_space is updated

DESCRIPTION

Calculate the total energy and the total magnetisation of CA using ising (49) model. These integer values might be very large so I'm using a large integer kind (ilrg (9.28)). This routine uses collectives. Magnetisation is defined as the fraction of the 1 spins. The only valid kernel is ca_kernel_ising_ener (1.16).

USES USED BY SOURCE

```
call hx_sub( space )                    ! space updated, sync images
call iter_sub( space, hdepth, kernel ) ! tmp_space updated, local op
energy =                                                              &
  int( sum( tmp_space( 1:sub(1), 1:sub(2), 1:sub(3) ) ), kind=ilrg )
magnet =                                                              &
  int( sum(     space( 1:sub(1), 1:sub(2), 1:sub(3) ) ), kind=ilrg )
call co_sum( energy )
call co_sum( magnet )
```

```
end subroutine ca_ising_energy_col
```

1.11 ca_hx/ca_iter_dc

[ca_hx] [Subroutines]

NAME

ca_iter_dc

SYNOPSIS

```
subroutine ca_iter_dc( space, halo, kernel )
```

INPUT

```
integer, intent( in ) :: halo
integer( kind=iarr ), intent( in ), contiguous ::                      &
  space( 1-halo: , 1-halo: , 1-halo: )
procedure( kernel_proto ) :: kernel

!    space - CA array at the end of the previous iteration
```

OUTPUT

```
!    none
```

SIDE EFFECTS

module array tmp_space, CA state at the end of this iteration, is updated

DESCRIPTION

This routine does a single CA iteration with DO CONCURRENT. The space array from the previous iteration is read only. The space array at the end of this iteration, tmp_space, from this module, is written only.

USES USED BY

ca_run (1.17), ca_ising_energy (1.9)

SOURCE

```
integer :: i, j, k

! Do not include halo! So start at 1 and end at sub.
do concurrent( k = 1:sub(3), j = 1:sub(2), i = 1:sub(1) )
  tmp_space( i,j,k ) = kernel( space = space, halo = hdepth,           &
    coord = (/ i , j , k /) )
end do

end subroutine ca_iter_dc
```

1.12 ca_hx/ca_iter_omp

[ca_hx] [Subroutines]

NAME

ca_iter_omp

SYNOPSIS

```
subroutine ca_iter_omp( space, halo, kernel )
```

INPUT

```
integer, intent( in ) :: halo
integer( kind=iarr ), intent( in ), contiguous ::                      &
  space( 1-halo: , 1-halo: , 1-halo: )
procedure( kernel_proto ) :: kernel

!    space - CA array at the end of the previous iteration
```

OUTPUT

```
!    none
```

SIDE EFFECTS

module array tmp_space, CA state at the end of this iteration, is updated

DESCRIPTION

This routine does a single CA iteration with triple nested loops. The space array from the previous iteration is read only. The space array at the end of this iteration, tmp_space, from this module, is written only.

USES USED BY

ca_run (1.17), ca_ising_energy (1.9)

SOURCE

```
integer :: i, j, k

! Do not include halo! So start at 1 and end at sub.

!$omp parallel do default(none)                                         &
!$omp private(i,j,k) shared(sub,space,hdepth,tmp_space)
! !$omp collapse(3)
do k = 1, sub(3)
do j = 1, sub(2)
do i = 1, sub(1)
  tmp_space( i,j,k ) = kernel( space = space, halo = hdepth,           &
    coord = (/ i , j , k /) )
end do
end do
end do
```

```
!$omp end parallel do

end subroutine ca_iter_omp
```

1.13 ca_hx/ca_iter_tl

[ca_hx] [Subroutines]

NAME

ca_iter_tl

SYNOPSIS

```
subroutine ca_iter_tl( space, halo, kernel )
```

INPUT

```
integer, intent( in ) :: halo
integer( kind=iarr ), intent( in ), contiguous ::                      &
  space( 1-halo: , 1-halo: , 1-halo: )
procedure( kernel_proto ) :: kernel

!    space - CA array at the end of the previous iteration
```

OUTPUT

```
!    none
```

SIDE EFFECTS

module array tmp_space, CA state at the end of this iteration, is updated

DESCRIPTION

This routine does a single CA iteration with triple nested loops. The space array from the previous iteration is read only. The space array at the end of this iteration, tmp_space, from this module, is written only.

USES USED BY

ca_run (1.17), ca_co_run (1.6.6), ca_ising_energy (1.9)

SOURCE

```
integer :: i, j, k

! Do not include halo! So start at 1 and end at sub.
do k = 1, sub(3)
do j = 1, sub(2)
do i = 1, sub(1)

  tmp_space( i,j,k ) = kernel( space = space, halo = hdepth,           &
    coord = (/ i , j , k /) )

!  write (*, "(a,i0,a,4(i0,tr1))") &
!    "img: ", this_image(), " i,j,k,energy: ", i, j, k, tmp_space(i,j,k)

end do
end do
```

```
end do

end subroutine ca_iter_tl
```

1.14 ca_hx/ca_kernel_copy

[ca_hx] [Subroutines]

NAME

ca_kernel_copy

SYNOPSIS

```
pure function ca_kernel_copy( space, halo, coord )
```

INPUTS

```
integer, intent( in ) :: halo
integer( kind=iarr ), intent( in ), contiguous ::                      &
  space( 1-halo: , 1-halo: , 1-halo: )
integer, intent(in) :: coord(3)
```

OUTPUT

```
integer( kind=iarr ) ca_kernel_copy
```

SIDE EFFECTS

None DESCIPTION This is a simplest CA kernel function. Simply copy the previous state of each cell. This function is used only to test the library.

USES USED BY

ca_iter_tl (1.13), ca_run (1.17)

SOURCE

```
ca_kernel_copy = space( coord(1), coord(2), coord(3) )

end function ca_kernel_copy
```

1.15 ca_hx/ca_kernel_ising

[ca_hx] [Subroutines]

NAME

ca_kernel_ising

SYNOPSIS

```
pure function ca_kernel_ising( space, halo, coord )
```

INPUTS

```
integer, intent( in ) :: halo
integer( kind=iarr ), intent( in ), contiguous ::                    &
  space( 1-halo: , 1-halo: , 1-halo: )
integer, intent( in ) :: coord(3)
```

OUTPUT

```
integer( kind=iarr ) ca_kernel_ising
```

SIDE EFFECTS

None DESCIPTION ising (49) magnetisation CA kernel function. The CA state is magnetic spin, either 0 (down) or 1 (up). The sign of the spin (CA cell state) is changed if and only if this spin has the same number of parallel and anti-parallel neighbours, i.e. the sum of 6 neighbours is exactly 3 - three neighbours of spin up (3*1=3) and three neighbours of spin down (0). The multiplier alternates between 0 and 1 from one cell to the next. This preserves the energy. For more details see Sec. 2.2.3 "The Q2R rule" in: B. Chopard, M. Droz "Cellular Automata Modeling of Physical Systems", Cambridge, 1998.

USES USED BY

ca_iter_tl (1.13), ca_iter_dc (1.11), ca_iter_omp (1.12), ca_run (1.17), ca_co_run (1.6.6)

SOURCE

```
integer( kind=iarr ) :: n

associate( s => space, i => coord(1), j => coord(2), k => coord(3) )

n = s(i-1,j,k) + s(i+1,j,k) + s(i,j-1,k) + s(i,j+1,k) + s(i,j,k-1) +   &
    s(i,j,k+1)

if ( n .eq. 3 .and. mask_array(i,j,k) .eq. 1 ) then
  ! If the sum of 6 neighbours is exactly 3 and the mask value is 1
  ! then flip the state.
  ca_kernel_ising = 1_iarr - s(i,j,k)
else
  ! Otherwise no change
  ca_kernel_ising = s(i,j,k)
end if

end associate
```

```
end function ca_kernel_ising
```

1.16 ca_hx/ca_kernel_ising_ener

[ca_hx] [Subroutines]

NAME

ca_kernel_ising_ener

SYNOPSIS

```
pure function ca_kernel_ising_ener( space, halo, coord )
```

INPUTS

```
integer, intent( in ) :: halo
integer( kind=iarr ), intent( in ), contiguous ::                      &
  space( 1-halo: , 1-halo: , 1-halo: )
integer, intent( in ) :: coord(3)
```

OUTPUT

```
integer( kind=iarr ) ca_kernel_ising_ener
```

SIDE EFFECTS

None DESCIPTION ising (49) magnetisation CA kernel function for energy calculation. Each neighbour of the same spin adds -1 to the energy. Each neighbour of the opposite spin adds +1 to the energy. For more details see Sec. 2.2.3 "The Q2R rule" in: B. Chopard, M. Droz, "Cellular Automata Modeling of Physical Systems", Cambridge, 1998.

To avoid double counting, each cell only checks links with cells to its *left* in all 3 dimensions. This leaves the last cells in the global CA space along each dimension, which must also look to the right, i.e. to the global right halo. This is not satisfactory because not all cells are processed in the same way. However, I cannot think of a better algorithm.

USES USED BY

ca_iter_tl (1.13), ca_iter_dc (1.11), ca_iter_omp (1.12), ca_ising_energy (1.9)

SOURCE

```
integer( kind=iarr ) :: count(6)

count=0

associate( s => space, i => coord(1), j => coord(2), k => coord(3) )

! s(i,j,k) - s(i-1,j,k)                => 0, -1, 1
! abs( s(i,j,k) - s(i-1,j,k) )         => 0, 1
! 2 * abs( s(i,j,k) - s(i-1,j,k) )     => 0, 2
! 2 * abs( s(i,j,k) - s(i-1,j,k) ) - 1  => -1, 1

! Neighbours to the left along 3 directions
count(1) = 2_iarr * abs( s(i,j,k) - s(i-1,j,k) ) - 1_iarr
count(2) = 2_iarr * abs( s(i,j,k) - s(i,j-1,k) ) - 1_iarr
count(3) = 2_iarr * abs( s(i,j,k) - s(i,j,k-1) ) - 1_iarr
```

```
! Neighbours to the right, only for the globally last cells
if ( ci(1) .eq. ucob(1) .and. i .eq. sub(1) ) then
  count(4) = 2_iarr * abs( s(i,j,k) - s(i+1,j,k) ) - 1_iarr
end if
if ( ci(2) .eq. ucob(2) .and. j .eq. sub(2) ) then
  count(5) = 2_iarr * abs( s(i,j,k) - s(i,j+1,k) ) - 1_iarr
end if
if ( ci(3) .eq. ucob(3) .and. k .eq. sub(3) ) then
  count(6) = 2_iarr * abs( s(i,j,k) - s(i,j,k+1) ) - 1_iarr
end if

ca_kernel_ising_ener = sum( count )

! write (*,"(a,4(i0,tr1))") "i,j,k,energy:", i,j,k,ca_kernel_ising_ener

end associate

end function ca_kernel_ising_ener
```

1.17 ca_hx/ca_run

[ca_hx] [Subroutines]

NAME

ca_run

SYNOPSIS

```
subroutine ca_run( space, hx_sub, iter_sub, kernel, niter )
```

INPUT

```
integer( kind=iarr ), intent(inout), allocatable :: space(:,:,:)
procedure( hx_proto ) :: hx_sub
procedure( iter_proto ) :: iter_sub
procedure( kernel_proto ) :: kernel
integer, intent(in) :: niter

!       space - space array before iterations start
!      hx_sub - HX routine, e.g.
!             - ca_hx_all
!             _ ca_mpi_hx_all
!    iter_sub - the subroutine performing a single CA iteration, e.g.
!             - ca_iter_tl - triple nested loop
!             - ca_iter_dc - do concurrent
!             - ca_iter_omp - OpenMP
!      kernel - a function to be called for every cell inside the loop
!        iter - number of iterations to do
```

OUTPUT

```
!    space - CA array at the end of niter iterations
```

SIDE EFFECTS

module array tmp_space is updated

DESCRIPTION

This is a driver routine for CA iterations. HX is done before each iteration. Then a given number of iterations is performed with a given routine and a given kernel. One iteration is really 2 iterations: odd and even, Hence the upper loop limit is 2*niter.

USES USED BY SOURCE

```
integer :: i

tmp_space = space

do i = 1, 2*niter
  call hx_sub( space )                ! space updated, with HX
  ! tmp_space updated, local op
  call iter_sub( space=space, halo=hdepth, kernel=kernel )
```

```
  space = tmp_space                   ! local op
  mask_array = 1_iarr - mask_array    ! Flip the mask array
end do

end subroutine ca_run
```

1.18 ca_hx/ca_set_space_rnd

[ca_hx] [Subroutines]

NAME

ca_set_space_rnd

SYNOPSIS

```
subroutine ca_set_space_rnd( seed, frac1, space )
```

INPUT

```
integer :: seed(:)
real :: frac1

!    seed - RND seed array
!   frac1 - fraction of "1" cells, spin up cells, [0..1].
```

OUTPUT

```
integer( kind=iarr ), intent( inout ), contiguous ::                    &
  space( 1-hdepth: , 1-hdepth:, 1-hdepth: )

!   space - CA array is set to reproducible RND values, and
!           in a way that the global CA array is not dependent
!           on the partition or the number of images
```

SIDE EFFECTS

A tmp external file is created from image 1 to store RND values. It is deleted at the exit from this routine, also by image 1.

DESCRIPTION

Must call this routine after ca_halloc (1.1). Generate a 1D array or RND from a given seed. The length of the array is the size of the global CA model, i.e. don't generate values for the halos. Image 1 writes this array to file. Then, inside a critical region each image reads its own data. This is a slow routine.

SOURCE

```
integer :: co_k, co_j, co_i, k, j, i, funit

integer( kind=iarr ) :: value

integer( kind=iarr ), allocatable :: rnd_arr_int(:)

real, allocatable :: rnd_arr(:)

character(:), allocatable :: fname

fname = "tmp_rnd_file"

! Image 1 sets the file
```

```
make_file: if ( this_image() .eq. 1 ) then

  ! Sanity check
  if ( frac1 .lt. 0.0 .or. frac1 .gt. 1.0 ) then
    write (*,*) "ERROR: ca_hx/ca_set_space_rnd: frac1 outside of" //   &
      " admissable range [0..1]: frac1:", frac1
    error stop
  end if

    write (*,*)
  ! total_cells would have been set already in ca_spalloc
  ! No halo cells are included in this number!
  allocate( rnd_arr(     total_cells ) )
  allocate( rnd_arr_int( total_cells ) )

  call random_seed( put = seed )
  call random_number( rnd_arr )
!  rnd_arr_int = nint( rnd_arr )

  rnd_arr_int = 0
  where ( rnd_arr .lt. frac1 )
    rnd_arr_int = 1
  end where

  open( newunit=funit, file=fname, status="replace", access="stream",  &
        form="unformatted", iostat=ierr )
  if ( ierr .ne. 0 ) then
    write (*,*) "ERROR: ca_hx/ca_set_space_rnd: " //                   &
      "open( fname ), ierr:", ierr
    error stop
  end if

  write (funit) rnd_arr_int

  close(funit, status="keep", iostat=ierr )
  if ( ierr .ne. 0 ) then
    write (*,*) "ERROR: ca_hx/ca_set_space_rnd: " //                   &
      "close( fname ), ierr:", ierr
    error stop
  end if

  deallocate( rnd_arr )
  deallocate( rnd_arr_int )

end if make_file

! All images wait for img1 to write data to file and close it.
sync all

! Each image in turn reads the data from the file
critical
```

```
  open( newunit=funit, file=fname, status="old", access="stream",      &
        form="unformatted", iostat=ierr )
  if ( ierr .ne. 0 ) then
    write (*,*) "ERROR: ca_hx/ca_set_space_rnd: " //                   &
      "open( fname ), ierr:", ierr
  end if

  do co_k = 1, ucob(3)
  do    k = 1,  sub(3)
  do co_j = 1, ucob(2)
  do    j = 1,  sub(2)
  do co_i = 1, ucob(1)
  do    i = 1,  sub(1)
    read( funit ) value
    if ( co_k .eq. ci(3) .and. co_j .eq. ci(2) .and.                   &
         co_i .eq. ci(1) ) then
       ! ===>>> NOTE <<<===
       ! Only real space cells are assigned value!
       ! The halo cells are set to 0 in ca_spalloc!
       space(i,j,k) = value
    end if
  end do
  end do
  end do
  end do
  end do
  end do

  close(funit, status="keep", iostat=ierr )
  if ( ierr .ne. 0 ) then
    write (*,*) "ERROR: ca_hx/ca_set_space_rnd: " //                   &
      "close( fname ), ierr:", ierr
  end if

end critical

! All images must read their data before image 1 deletes the file
sync all

! Image 1 deletes the file
file_delete: if ( this_image() .eq. 1 ) then

  open( newunit=funit, file=fname, status="old", iostat=ierr )
  if ( ierr .ne. 0 ) then
    write (*,*) "ERROR: ca_hx/ca_set_space_rnd: " //                   &
      "open( fname ), ierr:", ierr
    error stop
  end if

  close(funit, status="delete", iostat=ierr )
  if ( ierr .ne. 0 ) then
    write (*,*) "ERROR: ca_hx/ca_set_space_rnd: " //                   &
```

```
      "close( fname ), ierr:", ierr
    error stop
  end if

end if file_delete

end subroutine ca_set_space_rnd
```

1.19 ca_hx/ca_spalloc

[ca_hx] [Subroutines]

NAME

ca_spalloc

SYNOPSIS

```
subroutine ca_spalloc( space, c, d )
```

INPUT

```
integer( kind=iarr ), allocatable, intent(inout) :: space(:,:,:)
integer, intent(in) :: c(3), d

!    space - CA array to allocate, with halos!
!        c - array with space dimensions
!        d - depth of the halo layer
```

OUTPUT

```
!    space is allocated and set to zero.
```

SIDE EFFECTS

none

DESCRIPTION

This routine allocates the CA array space, with halos of depth d. Also save some vars in this module for future. Also, on first call, allocate a module work space array (tmp_space) of the same mold as space. This array is used in CA iterations later.

USES USED BY SOURCE

```
if ( allocated( space ) ) then
  write (*,*) "WARN: ca_hx/ca_spalloc: image:", this_image(), "space", &
              "already allocated, deallocating!"
  deallocate( space, stat=ierr, errmsg=errmsg )
  if ( ierr .ne. 0 ) then
    write (*,*) "ERROR: ca_hx/ca_spalloc: deallocate( space ), ierr:", &
                ierr, "errmsg:", trim(errmsg)
    error stop
  end if
end if

! Note that halo cells are set to 0 here too.
allocate( space( 1-d:c(1)+d, 1-d:c(2)+d, 1-d:c(3)+d ), source=0_iarr,  &
          stat=ierr, errmsg=errmsg )
if ( ierr .ne. 0 ) then
  write (*,*) "ERROR: ca_hx/ca_spalloc: allocate( space ), ierr:",     &
              ierr, "errmsg:", trim(errmsg)
  error stop
```

```
end if

! Calculate once and keep forever
! Need a picture to explain these halo vars.
! The picture is only for dimension 1.
! The same diagram can be drawn for dimensions 2 and 3.
!
! hdepth
! |    |
! +----+---------------------+----+--
! |    |                     |    | ^ hdepth
! |    |                     |    | v
! +----+---------------------+----+--
! |<-lhsta  |           |    |    |
! |   0|    |           |    |    |
! |    |1   |           |    |    |
! | hdepth->|           |    |    |
! |    |    |           |<-ihsta  |
! |    |    |           sub->|    |
! |    |    |           |    |<-rhsta
! |    |    |           |  rhend->|
! |    |    |           |    |    |
! +----+---------------------+----+
! |    |                     |    |
! |    |                     |    |
! +----+---------------------+----+

hdepth = d
   sub = c
 ihsta = sub - hdepth + 1
 rhsta = sub + 1
 rhend = sub + hdepth
 lhsta = -hdepth + 1

if ( this_image() .eq. 1 ) write (*,*) "spalloc: hdepth:", hdepth,     &
 "sub:", sub, "ihsta:", ihsta, "rhsta:", rhsta, "rhend:", rhend,       &
 "lhsta:", lhsta

! Total number of cells in the global CA
total_cells = sub(1)*sub(2)*sub(3)*num_images()

! tmp_space must be (re)allocated.
if ( allocated( tmp_space ) ) then
  deallocate( tmp_space, stat=ierr, errmsg=errmsg )
  if ( ierr .ne. 0 ) then
    write (*,"(a,i0,a,a)") "ERROR: ca_hx/ca_spalloc: " //              &
      "deallocate( tmp_space ), ierr: ", ierr, "errmsg:", trim(errmsg)
    error stop
  end if
end if

! Note that halo cells are set to 0 here too.
```

```
allocate( tmp_space, source=space, stat=ierr, errmsg=errmsg )
if ( ierr .ne. 0 ) then
  write (*,"(a,i0,a,a)") "ERROR: ca_hx/ca_spalloc: " //                &
    "allocate( tmp_space ), ierr: ", ierr, " errmsg: ", trim(errmsg)
  error stop
end if

! mask_array must be (re)allocated.
! Mask array, no halos, the values are set in ca_halloc.
if ( allocated( mask_array ) ) then
  deallocate( mask_array, stat=ierr, errmsg=errmsg )
  if ( ierr .ne. 0 ) then
    write (*,"(a,i0,a,a)") "ERROR: ca_hx/ca_spalloc: " //              &
      "deallocate( mask_array ), ierr: ", ierr, "errmsg:", trim(errmsg)
    error stop
  end if
end if

allocate( mask_array( c(1), c(2), c(3) ), stat=ierr, errmsg=errmsg )
if ( ierr .ne. 0 ) then
  write (*,"(a,i0,a,a)") "ERROR: ca_hx/ca_spalloc: " //                &
    "allocate( mask_array ) ", "ierr:", ierr, "errmsg:", trim(errmsg)
  error stop
end if

end subroutine ca_spalloc
```

2 CASUP/ca_hx_input

[Modules]

NAME

ca_hx_input

SYNOPSIS

```
!$Id: ca_hx_input.f90 561 2018-10-14 20:48:19Z mexas $

module ca_hx_input
```

DESCRIPTION

Module with routines to input command line data into programs.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

CONTAINS

Public subroutines: ca_cmd_real (2.1)

USES USED BY SOURCE

```
use cgca_m1co, only : rdef
implicit none

private
public :: ca_cmd_real

contains
```

2.1 ca_hx_input/ca_cmd_real

[ca_hx_input] [Subroutines]

NAME

ca_cmd_real

SYNOPSIS

```
subroutine ca_cmd_real( n, data )
```

INPUTS

```
integer, intent( in ) :: n

!    n - number of command line arguments to read
```

OUTPUT

```
real(kind=rdef) :: data(n)

!      data - array of real of size n, which contains the N command
!    line arguments.
```

SIDE EFFECTS

None DESCIPTION Read all command line arguments, assume these are integer or floating point, and save them into a real array "data". Some simple checks are done, but possible other special cases will be missed. Use with caution!

USES USED BY

ca_iter_tl (1.13), ca_run (1.17)

SOURCE

```
character(len=100) :: value
character(len=10) :: fmt
integer i, arglen, ierr
do i = 1, n
   call get_command_argument( i, value, arglen, ierr )

   if (ierr .gt. 0) then
     write (*,'(a,i0,a)') "ERROR: ca_hx_input/ca_cmd_input: argument ",&
                          i, " cannot be retrieved"
     error stop
   elseif (ierr .eq. -1) then
     write (*,'(a,i0,a)') "ERROR: ca_hx_input/ca_cmd_input: argument ",&
                i, " length is longer than the string to store it"
     error stop
   elseif (ierr .lt. -1) then
     write (*,'(3(a,i0))') "ERROR: ca_hx_input/ca_cmd_input:" //       &
                " get_command_argument( ", i, " ) returned ", ierr,    &
                "unknown error, should never end up here"
```

```
     error stop
   end if

   write(fmt,"(a,i0,a)") "(f", arglen, ")"
   read (value,fmt) data(i)

end do

end subroutine ca_cmd_real
```

3 CASUP/casup

[Modules]

NAME

casup

SYNOPSIS

```
!$Id: casup.f90 561 2018-10-14 20:48:19Z mexas $

module casup
```

DESCRIPTION

The top level module for casup.

NOTES

The lowest level modules are level 1, e.g. cgca_m1co (9). Level 1 modules use no other modules. Level 2 modules use only level 1 modules. Level 3 modules use some level 2 modules and possibly also level 1 modules. And so on. All modules except the top level, this one, are named

```
cgca_mX<name>
```

where X is the level number, starting from 1, and name is the module name.

The modules group routines dealing with a particular data structure or a problem, e.g. cgca_m3clvg (26) deals with cleavage propagation, cgca_m2gb (11) deals with the grain connectivity array.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

CONTAINS

All modules in the library, which are not used by other modules directly

USED BY

none, top level module

SOURCE

```
use cgca_m1co
use cgca_m1clock

use ca_hx
use ca_hx_input
use cgca_m2alloc
use cgca_m2gb
use cgca_m2geom
use cgca_m2glm
use cgca_m2hdf5
```

```
use cgca_m2hx
use cgca_m2lnklst
use cgca_m2mpiio
use cgca_m2netcdf
use cgca_m2out
use cgca_m2pck
use cgca_m2phys
use cgca_m2red
use cgca_m2rnd
use cgca_m2rot
use cgca_m2stat

use cgca_m3clvg
! use cgca_m3clvgt - does not build
use cgca_m3gbf
use cgca_m3nucl
!use cgca_m3pfem
use cgca_m3sld

!use cgca_m4fr

end module casup
```

4 CASUP/Makefile-Cray

[Make files]

NAME

Makefile-Cray

SYNOPSIS

```
#$Id: Makefile-Cray 561 2018-10-14 20:48:19Z mexas $

FC=             ftn
```

PURPOSE

Build/install casup (3) with Cray.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

SOURCE

```
#FFLAGS=                -c -dm -eacFn -m3 -rl -g -O0 -h bounds
FFLAGS=         -c -dm -eacFn -m3 -rl # -g -O0 -h bounds
#FFLAGS=                -c -dm -eacFn -m3 -rl -O3,cache3,fp4,ipa5 # -g -O0 -h bounds
casup=          casup
MYLIB=          lib$(casup).a
LIBDIR=         $(HOME)/lib

SRC=            cgca_m1clock.f90 cgca_m1co.f90 \
                ca_hx.f90 ca_hx_mpi.f90 ca_hx_co.f90 ca_hx_1D.f90 \
                ca_hx_input.f90 \
                cgca_m2alloc.f90 cgca_m2gb.f90 \
                cgca_m2geom.f90 cgca_m2glm.f90 cgca_m2hdf5.f90 cgca_m2hx.f90 \
                cgca_m2lnklst.f90 cgca_m2mpiio.f90 cgca_m2netcdf.f90 cgca_m2out.f90 \
                cgca_m2pck.f90 cgca_m2phys.f90 cgca_m2red.f90 cgca_m2rnd.f90 \
                cgca_m2rot.f90 cgca_m2stat.f90 cgca_m3clvg.f90 cgca_m3gbf.f90 \
                cgca_m3nucl.f90 cgca_m3pfem.f90 cgca_m3sld.f90 cgca_m4fr.f90 \
                m2hx_hxic.f90 m2hx_hxir.f90 m2out_sm1.f90 \
                m2out_sm2_mpi.f90 m3clvg_sm1.f90 m3clvg_sm2.f90 m3clvg_sm3.f90 \
                m3sld_sm1.f90 m3sld_hc.f90 m3pfem_sm1.f90 \
                casup.f90
                # cgca_m3clvgt.f90 # - broken, does not build
                # m3clvgt_sm1.f90 # - broken, does not build
OBJ=            $(SRC:.f90=.o)
LST=            $(SRC:.f90=.lst)
CLEAN+=         $(OBJ) $(LST)

CLEAN+=         $(MYLIB)
```

```
.SUFFIXES:
.SUFFIXES:      .f90 .o

all:            $(OBJ) $(MYLIB)

.f90.o:
                $(FC) $(FFLAGS) $<

$(MYLIB):       $(OBJ)
                ar -r $(MYLIB) $(OBJ)

install:        $(MYLIB)
                cp $(MYLIB) $(LIBDIR)

deinstall:
                cd $(LIBDIR) && rm $(MYLIB)

clean:
                rm -rf $(CLEAN)
```

5 CASUP/Makefile-Cray-wp

[Make files]

NAME

Makefile-Cray-wp

SYNOPSIS

```
#$Id: Makefile-Cray-wp 533 2018-03-30 14:31:26Z mexas $

FC=             ftn
```

PURPOSE

Build/install casup (3) with Cray compiler. Whole program optimisation

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

SOURCE

```
HPL_DIR=        $(HOME)/cray_pl
CLEAN+=         $(HPL_DIR)

FFLAGS=         -c -dm -eacFn -m3 -rl -hwp -hpl=$(HPL_DIR)
casup=          casup
MYLIB=          lib$(casup).a
LIBDIR=         $(HOME)/lib

SRC=            cgca_m1clock.f90 cgca_m1co.f90 \
                ca_hx.f90 ca_hx_mpi.f90 ca_hx_co.f90 \
                cgca_m2alloc.f90 cgca_m2gb.f90 \
                cgca_m2geom.f90 cgca_m2glm.f90 cgca_m2hdf5.f90 cgca_m2hx.f90 \
                cgca_m2lnklst.f90 cgca_m2mpiio.f90 cgca_m2netcdf.f90 cgca_m2out.f90 \
                cgca_m2pck.f90 cgca_m2phys.f90 cgca_m2red.f90 cgca_m2rnd.f90 \
                cgca_m2rot.f90 cgca_m2stat.f90 cgca_m3clvg.f90 cgca_m3gbf.f90 \
                cgca_m3nucl.f90 cgca_m3pfem.f90 cgca_m3sld.f90 cgca_m4fr.f90 \
                m2hx_hxic.f90 m2hx_hxir.f90 m2out_sm1.f90 \
                m2out_sm2_mpi.f90 m3clvg_sm1.f90 m3clvg_sm2.f90 m3clvg_sm3.f90 \
                m3sld_sm1.f90 m3sld_hc.f90 m3pfem_sm1.f90 \
                casup.f90
                # cgca_m3clvgt.f90 # - broken, does not build
                # m3clvgt_sm1.f90 # - broken, does not build
OBJ=            $(SRC:.f90=.o)
LST=            $(SRC:.f90=.lst)
CLEAN+=         $(OBJ) $(LST)

CLEAN+=         $(MYLIB)
```

```
.SUFFIXES:
.SUFFIXES:      .f90 .o

all:            $(OBJ) $(MYLIB)

.f90.o:
                $(FC) $(FFLAGS) $<

$(MYLIB):       $(OBJ)
                ar -r $(MYLIB) $(OBJ)

install:        $(MYLIB)
                cp $(MYLIB) $(LIBDIR)

deinstall:
                cd $(LIBDIR) && rm $(MYLIB)

clean:
                rm -rf $(CLEAN)
```

6 CASUP/Makefile-FreeBSD

[Make files]

NAME

Makefile-FreeBSD

SYNOPSIS

```
#$Id: Makefile-FreeBSD 549 2018-04-27 14:56:15Z mexas $

FC=             caf
```

PURPOSE

Build/install casup (3) on FreeBSD with GCC/OpenCoarrays.

NOTES

Some parts of casup (3), in particular coarrays of derived type with pointer or allocatable components, require gcc7+. So need to build lang/opencoarrays with gcc7+. lang/opencoarrays can use either of 3 MPI ports:

```
net/mpich net/openmpi net/openmpi2
```

net/mpich is the default. The 2 openmpi ports have not been integrated fully with opencoarrays yet. Also need to rebuild a number of other ports:

```
science/hdf5 science/netcdf science/netcdf-fortran
```

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

SOURCE

```
FFLAGS=         -c -O2 -Wall -fPIC -g -fbacktrace -fall-intrinsics\
                -fcheck-array-temporaries -fopenmp\
                -fcheck=bounds\
                -I/usr/local/include

casup=          casup
MYLIB=          lib$(casup).a
CLEAN+=         $(MYLIB)
LIBDIR=         $(HOME)/lib
MODDIR=         $(HOME)/include

SRC=            cgca_m1clock.f90 cgca_m1co.f90 \
                ca_hx.f90 ca_hx_mpi.f90 ca_hx_co.f90 \
                cgca_m2alloc.f90 cgca_m2gb.f90 \
                cgca_m2geom.f90 cgca_m2glm.f90 cgca_m2hdf5.f90 cgca_m2hx.f90 \
```

```
                cgca_m2lnklst.f90 cgca_m2mpiio.f90 cgca_m2netcdf.f90 cgca_m2out.f90 \
                cgca_m2pck.f90 cgca_m2phys.f90 cgca_m2red.f90 cgca_m2rnd.f90 \
                cgca_m2rot.f90 cgca_m2stat.f90 \
                m2hx_hxic.f90 m2hx_hxir.f90 m2out_sm2_mpi.f90 \
                cgca_m3clvg.f90 cgca_m3gbf.f90 \
                cgca_m3nucl.f90 cgca_m3sld.f90 \
                m3clvg_sm1.f90 m3clvg_sm2.f90 m3clvg_sm3.f90 \
                m3sld_hc.f90 m3sld_sm1.f90 \
                casup.f90
#               cgca_m3pfem.f90
MOD=            $(SRC:.f90=.mod)
OBJ=            $(SRC:.f90=.o)
CLEAN+=         $(MOD) $(OBJ)

                # cgca_m4fr.f90 - not ready
                # cgca_m3clvgt.f90 # - broken, does not build
                # m2out_sm1.f90 # - Cray only
                # m3clvgt_sm1.f90 # - broken, does not build
                # m3pfem_sm1.f90 - gcc7 ICE

SMOD=           cgca*.smod ca*.smod
CLEAN+=         $(SMOD)

.SUFFIXES:
.SUFFIXES:      .f90 .o

all:            $(OBJ) $(MYLIB)

.f90.o:
                $(FC) $(FFLAGS) $<

$(MYLIB):       $(OBJ)
                ar -r $(MYLIB) $(OBJ)

install:        $(MYLIB)
                cp $(MYLIB) $(LIBDIR)/
                cp casup.mod $(MODDIR)/

deinstall:
                cd $(LIBDIR) && rm $(MYLIB)
                cd $(MODDIR) && rm casup.mod

clean:
                rm -f $(CLEAN)
```

7 CASUP/stats

[Code statistics]

NAME

stats

SOURCE

```
      42 text files.
classified 42 files
      42 unique files.
       0 files ignored.

github.com/AlDanial/cloc v 1.78  T=0.07 s (604.9 files/s, 261020.6 lines/s)
-------------------------------------------------------------------------------
File                            blank        comment           code
-------------------------------------------------------------------------------
ca_hx.f90                         307            608            686
cgca_m3clvg.f90                   280            575            629
cgca_m2gb.f90                     203            431            458
cgca_m3clvgt.f90                  190            436            434
cgca_m2hx.f90                     133            200            424
cgca_m3pfem.f90                   290            711            406
cgca_m3sld.f90                    152            274            339
ca_hx_mpi.f90                     217            459            331
ca_hx_co.f90                      134            342            314
m2hx_hxic.f90                      51            115            265
cgca_m2rot.f90                    146            237            247
m2hx_hxir.f90                      62             80            157
cgca_m2glm.f90                     92            127            157
cgca_m2stat.f90                    75            111            157
m3clvg_sm3.f90                     63            124            127
cgca_m2hdf5.f90                    50            107            116
cgca_m2out.f90                     51            102            116
cgca_m3gbf.f90                     55             86            114
m3clvgt_sm1.f90                    57             99            112
cgca_m2phys.f90                    76            187            112
cgca_m2alloc.f90                   79            143            109
cgca_m2mpiio.f90                   48            141             95
m2out_sm2_mpi.f90                  47            148             93
m3sld_hc.f90                       43            100             91
cgca_m3nucl.f90                    45             75             89
m3pfem_sm1.f90                     60            134             87
m3sld_sm1.f90                      41             91             81
cgca_m2lnklst.f90                  78            182             80
ca_hx_1D.f90                       57            118             79
m3clvg_sm1.f90                     52            114             76
cgca_m2pck.f90                     15             40             54
cgca_m2netcdf.f90                  42            121             53
cgca_m4fr.f90                      27             37             50
m2out_sm1.f90                      25             77             50
```

cgca_m1co.f90	80	231	48
cgca_m2rnd.f90	38	70	48
cgca_m2geom.f90	23	43	35
ca_hx_input.f90	22	41	30
m3clvg_sm2.f90	18	40	28
casup.f90	9	36	26
cgca_m1clock.f90	28	27	26
cgca_m2red.f90	27	62	25
SUM:	3588	7482	7054

8 CGPACK/cgca_m1clock

[Modules]

NAME

cgca_m1clock

SYNOPSIS

```
!$Id: cgca_m1clock.f90 379 2017-03-22 09:57:10Z mexas $

module cgca_m1clock
```

DESCRIPTION

Module with timing clocks

AUTHOR

Luis Cebamanos, modified by Anton Shterenlikht

COPYRIGHT

CONTAINS

Public functions: cgca_benchtime (8.1)

SOURCE

```
  implicit none
  private
  public :: cgca_benchtime

  integer, parameter :: dp = kind(1.0d0),                              &
    int32kind = selected_int_kind( 9),                                 &
    int64kind = selected_int_kind(18),                                 &
    intkind = int64kind

  logical,       save :: firstcall = .true.
  real(kind=dp), save :: ticktime = 0.0_dp

!
!  Select high resolution clock
!

  integer(kind = intkind) :: count, rate

contains

real(kind=dp) function benchtick()

  if (firstcall) then

     firstcall = .false.
```

```
     call system_clock(count, rate)
     ticktime = 1.0_dp / real(rate, kind=dp)

  end if

  benchtick = ticktime

end function benchtick
```

8.1 cgca_m1clock/cgca_benchtime

[cgca_m1clock] [Subroutines]

NAME

cgca_benchtime

SYNOPSIS

```
real(kind=dp) function cgca_benchtime()
```

SOURCE

```
  real(kind=dp) :: dummy

! Ensure clock is initialised

  if (firstcall) dummy = benchtick()

  call system_clock(count)

  cgca_benchtime  = real(count, kind=dp)*ticktime

end function cgca_benchtime
```

9 CGPACK/cgca_m1co

[Modules]

NAME

cgca_m1co

SYNOPSIS

```
!$Id: cgca_m1co.f90 537 2018-04-03 13:57:55Z mexas $

module cgca_m1co
```

DESCRIPTION

Lowest level module, contains named global parameters and variables. Contains routines which do not use modules (level 1 routines).

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

CONTAINS

Various global parameters and variables

USED BY

Most/all higher level modules.

SOURCE

```
use iso_fortran_env
implicit none
```

9.1 cgca_m1co/ca_range

[cgca_m1co] [Parameters]

PARAMETER

```
integer, parameter :: ca_range = 8
```

DESCRIPTION

Range argument for CA data, used in SELECTED_INT_KIND and in MPI MPI_Type_create_f90_integer. The use of the same range parameter will ensure that the integer data kinds (MPI types) are matching. Typical values (no guarantee) are:

```
ca_range     integer size, byte
 10            8
 8             4, default integer
 4             2
 2             1
```

9.2 cgca_m1co/cgca_clvg_lowest_state

[cgca_m1co] [Parameters]

PARAMETER

```
integer( kind=iarr ), parameter ::                                    &
 cgca_clvg_lowest_state = cgca_clvg_state_111_edge
```

DESCRIPTION

The the lowest cleavage state, used for sizing the lower bound of e.g. the grain volume array. Cell state of type cgca_state_type_frac (9.23).

9.3 cgca_m1co/cgca_clvg_state_100_edge

[cgca_m1co] [Parameters]

PARAMETER

```
integer( kind=iarr ), parameter :: cgca_clvg_state_100_edge = -2_iarr
```

DESCRIPTION

Edges of a cleavage crack of {100} family. Cell state of type cgca_state_type_frac (9.23).

9.4 cgca_m1co/cgca_clvg_state_100_flank

[cgca_m1co] [Parameters]

PARAMETER

```
integer( kind=iarr ), parameter :: cgca_clvg_state_100_flank = -1_iarr
```

DESCRIPTION

Flanks of a cleavage crack of {100} family. Cell state of type cgca_state_type_frac (9.23).

9.5 cgca_m1co/cgca_clvg_state_110_edge

[cgca_m1co] [Parameters]

PARAMETER

```
integer( kind=iarr ), parameter :: cgca_clvg_state_110_edge = -4_iarr
```

DESCRIPTION

Edges of a cleavage crack of {110} family. Cell state of type cgca_state_type_frac (9.23).

9.6 cgca_m1co/cgca_clvg_state_110_flank

[cgca_m1co] [Parameters]

PARAMETER

```
integer( kind=iarr ), parameter :: cgca_clvg_state_110_flank = -3_iarr
```

DESCRIPTION

Flanks of a cleavage crack of {110} family. Cell state of type cgca_state_type_frac (9.23).

9.7 cgca_m1co/cgca_clvg_state_111_edge

[cgca_m1co] [Parameters]

PARAMETER

```
integer( kind=iarr ), parameter :: cgca_clvg_state_111_edge = -6_iarr
```

DESCRIPTION

Edges of a cleavage crack of {111} family. Cell state of type cgca_state_type_frac (9.23).

9.8 cgca_m1co/cgca_clvg_state_111_flank

[cgca_m1co] [Parameters]

PARAMETER

```
integer( kind=iarr ), parameter :: cgca_clvg_state_111_flank = -5_iarr
```

DESCRIPTION

Flanks of a cleavage crack of {111} family. Cell state of type cgca_state_type_frac (9.23).

9.9 cgca_m1co/cgca_clvg_states

[cgca_m1co] [Parameters]

PARAMETER

```
integer(kind=iarr), parameter ::                                    &
 cgca_clvg_states( size(cgca_clvg_states_flank) +                   &
                   size(cgca_clvg_states_edge) ) =                  &
  (/ cgca_clvg_states_flank, cgca_clvg_states_edge /)
```

DESCRIPTION

Array to store all cleavage states, flanks and edges. Cell state of type cgca_state_type_frac (9.23).

9.10 cgca_m1co/cgca_clvg_states_edge

[cgca_m1co] [Parameters]

PARAMETER

```
integer(kind=iarr), parameter :: cgca_clvg_states_edge(3)  =           &
  (/ cgca_clvg_state_100_edge,                                         &
     cgca_clvg_state_110_edge,                                         &
     cgca_clvg_state_111_edge /)
```

DESCRIPTION

Array to store all edge cleavage states. Cell state of type cgca_state_type_frac (9.23).

9.11 cgca_m1co/cgca_clvg_states_flank

[cgca_m1co] [Parameters]

PARAMETER

```
integer(kind=iarr), parameter :: cgca_clvg_states_flank(3) =           &
  (/ cgca_clvg_state_100_flank,                                        &
     cgca_clvg_state_110_flank,                                        &
     cgca_clvg_state_111_flank /)
```

DESCRIPTION

Array to store all flank cleavage states. Cell state of type cgca_state_type_frac (9.23).

9.12 cgca_m1co/cgca_frac_states

[cgca_m1co] [Parameters]

PARAMETER

```
integer( kind=iarr ), parameter ::                                    &
 cgca_frac_states( size(cgca_clvg_states) + 1 ) =                     &
  (/ cgca_gb_state_fractured, cgca_clvg_states /)
```

DESCRIPTION

Array to store all fracture states: cleavage, fractured GB, etc. Cell state of type cgca_state_type_frac (9.23).

9.13 cgca_m1co/cgca_gb_state_fractured

[cgca_m1co] [Parameters]

PARAMETER

```
integer( kind=iarr ), parameter :: cgca_gb_state_fractured = 1_iarr
```

DESCRIPTION

Fractured grain boundary, cell state of type cgca_state_type_frac (9.23).

9.14 cgca_m1co/cgca_gb_state_intact

[cgca_m1co] [Parameters]

PARAMETER

```
integer( kind=iarr ), parameter :: cgca_gb_state_intact = 2_iarr
```

DESCRIPTION

Intact grain boundary, cell state of type cgca_state_type_frac (9.23).

9.15 cgca_m1co/cgca_gcupd_size1

[cgca_m1co] [Parameters]

PARAMETER

```
integer( kind=idef ), parameter :: cgca_gcupd_size1 = 100_idef
```

DESCRIPTION

This is size1 for the allocatable array coarray gcupd_arr in module cgca_m3clvg (26). The value is the maximum number of fractured grain boundaries on an image in a single CA iteration. 100 is probably an overkill. Perhaps 2-3 will be enough, but it's not too big a waste.

9.16 cgca_m1co/cgca_gcupd_size2

[cgca_m1co] [Parameters]

PARAMETER

```
integer( kind=idef ), parameter :: cgca_gcupd_size2 = 3_idef
```

DESCRIPTION

This is size2 for the allocatable array coarray gcupd_arr in module cgca_m3clvg (26). The value is the number of useful fields for each entry. At present only 3 entries planned: (1) grain 1, (2) grain 2, (3) state of grain boundary between grains 1 and 2.

9.17 cgca_m1co/cgca_intact_state

[cgca_m1co] [Parameters]

PARAMETER

```
integer( kind=iarr ), parameter :: cgca_intact_state = 0_iarr
```

DESCRIPTION

Intact state for fracture array, cell state of type cgca_state_type_frac (9.23).

9.18 cgca_m1co/cgca_liquid_state

[cgca_m1co] [Parameters]

PARAMETER

```
integer( kind=iarr ), parameter :: cgca_liquid_state = 0_iarr
```

DESCRIPTION

Liquid phase, cell state of type cgca_state_type_grain (9.24). All states of the same type must be unique.

NOTE

All grains must be greater than the liquid state, i.e. all grains must be positive.

9.19 cgca_m1co/cgca_lowest_state

[cgca_m1co] [Parameters]

PARAMETER

```
integer(kind=iarr), parameter :: cgca_lowest_state = -huge(0_iarr)
```

DESCRIPTION

Lowest possible state in the model

9.20 cgca_m1co/cgca_scodim

[cgca_m1co] [Parameters]

PARAMETER

```
integer( kind=idef ), parameter :: cgca_scodim = 3
```

DESCRIPTION

Number of spatial codimensions for the main space coarray and other coarrays with more that 1 codimensions.

9.21 cgca_m1co/cgca_slcob

[cgca_m1co] [Variables]

PARAMETER

```
integer( kind=idef ) :: cgca_slcob( cgca_scodim )
```

DESCRIPTION

Lower cobounds of the space coarray and other coarrays with more that 1 codimensions.

9.22 cgca_m1co/cgca_state_null

[cgca_m1co] [Parameters]

PARAMETER

```
integer( kind=iarr ), parameter :: cgca_state_null = huge( 0_iarr )
```

DESCRIPTION

An inactive (null, void, nonexistent) state of a cell in the fracture layer, cell state of type cgca_state_type_frac (9.23). This state is given to cells which are outside of the FE model. These cells are not analysed at all in any fracture routines.

9.23 cgca_m1co/cgca_state_type_frac

[cgca_m1co] [Parameters]

PARAMETER

```
integer( kind=idef ), parameter :: cgca_state_type_frac = 2_idef
```

DESCRIPTION

Cell state type for fractures

9.24 cgca_m1co/cgca_state_type_grain

[cgca_m1co] [Parameters]

PARAMETER

```
integer( kind=idef ), parameter :: cgca_state_type_grain = 1_idef
```

DESCRIPTION

Cell state type for grains

9.25 cgca_m1co/cgca_sucob

[cgca_m1co] [Variables]

PARAMETER

```
integer( kind=idef ) :: cgca_sucob( cgca_scodim )
```

DESCRIPTION

Upper cobounds of the space coarray and other coarrays with more that 1 codimensions.

9.26 cgca_m1co/iarr

[cgca_m1co] [Parameters]

PARAMETER

```
integer, parameter :: iarr = selected_int_kind( ca_range )
```

DESCRIPTION

Integer kind for CA arrays

9.27 cgca_m1co/idef

[cgca_m1co] [Parameters]

PARAMETER

```
integer, parameter :: idef = selected_int_kind( 8 )
```

DESCRIPTION

Default integer kind

9.28 cgca_m1co/ilrg

[cgca_m1co] [Parameters]

PARAMETER

```
integer, parameter :: ilrg = selected_int_kind( 10 )
```

DESCRIPTION

Integer kind for large numbers, e.g. volumes, total number of cells, etc.

9.29 cgca_m1co/ldef

[cgca_m1co] [Parameters]

PARAMETER

```
integer, parameter :: ldef = kind( .true. )
```

DESCRIPTION

Default logical kind

9.30 cgca_m1co/pi

[cgca_m1co] [Parameters]

PARAMETER

```
real( kind=rdef ), parameter :: cgca_pi = 3.14159265358979323846264338_rdef
```

DESCRIPTION

pi

9.31 cgca_m1co/rdef

[cgca_m1co] [Parameters]

PARAMETER

```
integer, parameter :: rdef = selected_real_kind( 6, 30 )
```

DESCRIPTION

Default real kind

9.32 cgca_m1co/rlrg

[cgca_m1co] [Parameters]

PARAMETER

```
integer, parameter :: rlrg = selected_real_kind( 15, 300 )
```

DESCRIPTION

High precision real kind, most likely will be double precision

10 CGPACK/cgca_m2alloc

[Modules]

NAME

cgca_m2alloc

SYNOPSIS

```
!$Id: cgca_m2alloc.f90 381 2017-03-22 11:29:44Z mexas $

module cgca_m2alloc
```

DESCRIPTION

Module dealing with the allocation and deallocation of various arrays. Several routines are used because they allocate arrays of different dimensionality, i.e.

```
    (:)[:,:,:]
(:,:,:)[:,:,:]
```

(:,:,:,:)[:,:,:]

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

CONTAINS

cgca_as (10.2), cgca_ds (10.5), cgca_av (10.3), cgca_dv (10.6), cgca_art (10.1), cgca_drt (10.4)

USES

cgca_m1co (9)

USED BY

cgca

SOURCE

```
use cgca_m1co
implicit none

private
public :: cgca_as, cgca_ds, cgca_av, cgca_dv, cgca_art, cgca_drt

contains
```

10.1 cgca_m2alloc/cgca_art

[cgca_m2alloc] [Subroutines]

NAME

cgca_art

SYNOPSIS

```
subroutine cgca_art(l,u,col1,cou1,col2,cou2,col3,coarray)
```

INPUTS

```
integer(kind=idef),intent(in) :: l,u,col1,cou1,col2,cou2,col3
real(kind=rdef),allocatable,intent(inout) :: coarray(:,:,:)[:,:,:]
```

SIDE EFFECTS

coarray becomes allocated

DESCRIPTION

Allocate rotation tensor array. This is an array (l:u,3,3) defined on every image. Note that the first index is the grain number and the next two are the rotation tensor. So that the array element (153,3,1) is the rotation tensor component R31 for grain number 153. (87,:,:) is a 3x3 matrix defining the full (non-symmetric, but orthogonal!!!) rotation tensor for grain 87.

NOTES

This routine must be called prior to calling cgca_rt (24.6).

USES

none

USED BY

cgca_m2alloc (10)

SOURCE

```
integer :: errstat

errstat=0

if (.not. allocated(coarray))                                &
  allocate(coarray(l:u,3,3) [col1:cou1,col2:cou2,col3:*],  &
            source = 0.0_rdef, stat=errstat)

if (errstat .ne. 0) then
  write (*,'(a,i0)') "ERROR: cgca_art: image: ", this_image()
  write (*,'(a)') "ERROR: cgca_art: cannot allocate coarray"
  error stop
end if

end subroutine cgca_art
```

10.2 cgca_m2alloc/cgca_as

[cgca_m2alloc] [Subroutines]

NAME

cgca_as

SYNOPSIS

```
subroutine cgca_as( l1, u1, l2, u2, l3, u3, col1, cou1, col2, cou2,    &
 col3, props, coarray )
```

INPUTS

```
integer(kind=idef),intent(in) ::                                       &
 l1,   & ! Lower and uppper bounds of the coarray along
 u1,   & ! dimensions 1,2, and 3.
 l2,   &
 u2,   &
 l3,   &
 u3,   &
 col1, & ! Lower and uppper cobounds of the coarray along
 cou1, & ! codimensions 1, 2 and 3. Note the last upper
 col2, & ! cobound is not specified. In allocate it is
 cou2, & ! given as * to allow for arbitrary number of
 col3, & ! images at run time.
 props   ! Number of cell state properties to use.

integer(kind=iarr),allocatable,intent(inout) ::                        &
 coarray(:,:,:,:)[:,:,:]  ! coarray to allocate
```

SIDE EFFECTS

coarray becomes allocated with all values assigned cgca_liquid_state (9.18).

Global array variables cgca_slcob (9.21), cgca_sucob (9.25) are assigned values.

DESCRIPTION

This routine allocates a 4D arrays on each image. The first 3 dimensions define a cell, and the last, 4th dimension defines the number or the cell state. The routine increases extent by 2 in each direction, thus creating space for storing halos. No check for the validity of the coarray dimension is made here. The user should make sure the coarray dimension values passed to the routine make sense.

NOTES

We want the coarray to have halos to exchange data between processors. So need to increase the extent by 2 in each dimension.

USES

none

USED BY

cgca_m2alloc (10)

SOURCE

```
integer, parameter :: halo=1
integer :: errstat = 0

if ( .not. allocated(coarray) ) allocate( coarray(                    &
  l1-halo:u1+halo, l2-halo:u2+halo, l3-halo:u3+halo, props)           &
  [col1:cou1, col2:cou2, col3:*], source=cgca_liquid_state,           &
    stat=errstat)
if ( errstat .ne. 0 ) then
  write (*,'(2(a,i0))') 'ERROR: cgca_m2alloc/cgca_as:&
    & allocate( coarray ), image: ', this_image(), " err. code: ",    &
    errstat
  error stop
end if

! Assign the cobounds to the global variables for use by other routines
cgca_slcob = lcobound( coarray )
cgca_sucob = ucobound( coarray )

end subroutine cgca_as
```

10.3 cgca_m2alloc/cgca_av

[cgca_m2alloc] [Subroutines]

NAME

cgca_av

SYNOPSIS

```
subroutine cgca_av(l,u,col1,cou1,col2,cou2,col3,coarray)
```

INPUTS

```
integer(kind=idef),intent(in) :: l,u,col1,cou1,col2,cou2,col3
integer(kind=ilrg),allocatable,intent(inout) :: coarray(:)[:,:,:]
```

SIDE EFFECTS

None

DESCRIPTION

This routine allocates a 1D coarray of length l:u. Coarray variable "coarray" becomes allocated, with all values assigned to zero.

USES

None

USED BY

cgca_m2alloc (10)

SOURCE

```
integer :: errstat

errstat=0

if (.not. allocated(coarray))                          &
  allocate(coarray(l:u) [col1:cou1,col2:cou2,col3:*],  &
            source = 0_ilrg, stat=errstat)

if (errstat .ne. 0) then
 write (*,'(a,i0)') "ERROR: cgca_av: image: ", this_image()
 write (*,'(a)') "ERROR: cgca_av: cannot allocate coarray"
 write (*,'(a,i0)') "ERROR: cgca_av: error code: ", errstat
 error stop
end if

end subroutine cgca_av
```

10.4 cgca_m2alloc/cgca_drt

[cgca_m2alloc] [Subroutines]

NAME

cgca_drt

SYNOPSIS

```
subroutine cgca_drt(coarray)
```

INPUT

```
real(kind=rdef),allocatable,intent(inout) :: coarray(:,:,:)[:,:,:]
```

SIDE EFFECTS

coarray becomes deallocated

DESCRIPTION

Deallocate rotation tensor array.

USES

none

USED BY

cgca_m2alloc (10)

SOURCE

```
integer :: errstat

errstat=0

if (allocated(coarray)) then
  deallocate(coarray,stat=errstat)
  if (errstat .ne. 0) then
    write (*,'(a,i0)') "ERROR: cgca_drt: image: ", this_image()
    write (*,'(a)') "ERROR: cgca_drt: cannot deallocate coarray"
    error stop
  end if
end if

! if coarray is not allocated, do nothing

end subroutine cgca_drt
```

10.5 cgca_m2alloc/cgca_ds

[cgca_m2alloc] [Subroutines]

NAME

cgca_ds

SYNOPSIS

```
subroutine cgca_ds(coarray)
```

INPUT

```
integer(kind=iarr),allocatable,intent(inout) :: coarray(:,:,:,:)[:,:,:]
```

SIDE EFFECTS

coarray becomes deallocated

DESCRIPTION

This routine deallocates a 3D coarray. It first checks whether the array is allocated. If the array is not allocated, a warning is issued, but the program is *not* terminated.

USES

none

USED BY

cgca_m2alloc (10)

SOURCE

```
integer :: errstat

errstat=0

if (allocated(coarray)) then
 deallocate(coarray,stat=errstat)
 if (errstat .ne. 0) then
  write (*,'(a,i0)') "ERROR: cgca_ds: image: ", this_image()
  write (*,'(a)')    "ERROR: cgca_ds: cannot deallocate coarray"
  write (*,'(a,i0)') "ERROR: cgca_ds: error code: ", errstat
  error stop
 end if
else
 write (*,'(a,i0,a)') "WARNING: cgca_ds: image: ", this_image(), &
  ", coarray is not allocated, cannot deallocate"
end if

end subroutine cgca_ds
```

10.6 cgca_m2alloc/cgca_dv

[cgca_m2alloc] [Subroutines]

NAME

cgca_dv

SYNOPSIS

```
subroutine cgca_dv(coarray)
```

INPUT

```
integer(kind=ilrg),allocatable,intent(inout) :: coarray(:)[:,:,:]
```

SIDE EFFECTS

coarray becomes deallocated

DESCRIPTION

deallocate volume coarray

USES

none

USED BY

cgca_m2alloc (10)

SOURCE

```
integer :: errstat

errstat=0

if (allocated(coarray)) then
 deallocate(coarray,stat=errstat)
 if (errstat .ne. 0) then
  write (*,'(a,i0)') "ERROR: cgca_dv: image: ", this_image()
  write (*,'(a)') "ERROR: cgca_dv: cannot deallocate coarray"
  write (*,'(a,i0)') "ERROR: cgca_dv: error code: ", errstat
  error stop
 end if
end if

! if coarray is not allocated, do nothing

end subroutine cgca_dv
```

11 CGPACK/cgca_m2gb

[Modules]

NAME

cgca_m2gb

SYNOPSIS

```
!$Id: cgca_m2gb.f90 529 2018-03-26 11:25:45Z mexas $

module cgca_m2gb
```

DESCRIPTION

Module dealing with grain boundaries. Most routines in this module are concerned with creating, updating and printing of the *local* array gc (11.11).

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

CONTAINS

cgca_gcu (11.9), cgca_gcp (11.6), cgca_dgc (11.2), cgca_gcf (11.4), cgca_gcr (11.7), cgca_gbs (11.3), cgca_agc (11.1) (private to this module), cgca_gcr_pure (11.8), cgca_gcf_pure (11.5)

USES

cgca_m1co (9)

USED BY

cgca_m3clvg (26)

SOURCE

```
use cgca_m1co
implicit none

private
public :: cgca_dgc, cgca_gbs, cgca_gcf, cgca_gcp, cgca_gcr,           &
          cgca_gcu, cgca_igb, &
! Do not use! NOt READY!
 cgca_gcr_pure, cgca_gcf_pure
```

11.1 cgca_m2gb/cgca_agc

[cgca_m2gb] [Subroutines]

NAME

cgca_agc

SYNOPSIS

```
subroutine cgca_agc( len )
```

INPUT

```
integer, intent( in ) :: len
```

SIDE EFFECTS

Allocate gc (11.11) array

DESCRIPTION

Allocate the grain connectivity (gc (11.11)) array, and set to zero. The first dimension is given by len. The second dimension is 3, because:

```
1 - grain number
2 - grain neighbour number
3 - grain boundary state - fractured or intact
```

USES

gc (11.11)

USED BY

module cgca_m2gb (11): cgca_gcu (11.9)

SOURCE

```
integer :: errstat=0

allocate( gc(len,3), source=0_iarr, stat=errstat )
if ( errstat .ne. 0 ) then
  write (*,'(2(a,i0))') "ERROR: cgca_agc: img: ", this_image(),        &
                        " cannot allocate gc, error code: ", errstat
  error stop
end if

end subroutine cgca_agc
```

11.2 cgca_m2gb/cgca_dgc

[cgca_m2gb] [Subroutines]

NAME

cgca_dgc

SYNOPSIS

```
subroutine cgca_dgc
```

SIDE EFFECTS

deallocate gc (11.11) array

USES

gc (11.11)

USED BY

module cgca_m2gb (11): cgca_gcu (11.9)

SOURCE

```
integer :: errstat

deallocate( gc, stat=errstat )
if (errstat .ne. 0) then
  write (*,'(2(a,i0))') "ERROR: cgca_dgc: img: ", this_image(),        &
                        " cannot deallocate gc, err code: ", errstat
  error stop
end if

end subroutine cgca_dgc
```

11.3 cgca_m2gb/cgca_gbs

[cgca_m2gb] [Subroutines]

NAME

cgca_gbs

SYNOPSIS

```
subroutine cgca_gbs( coarray )
```

INPUT

```
integer( kind=iarr ), allocatable, intent(inout) :: &
 coarray(:,:,:,:)[:,:,:]
```

SIDE EFFECTS

state of coarray changes

DESCRIPTION

This routines does Grain Boundary Smoothing (GBS). It works only with cgca_state_type_grain (9.24) layer. For each cell that has neighbours from other grains the routine substites the cell value (grain number) by that of the grain that has most cells in the (3,3,3) neighbourhood. Example:

```
      -> 2        5 5 5
    / |           5 5 5
   3  V           5 5 5
      1      5 1 1
             5 1 1
             5 5 5
       1 1 1
       8 1 1
       8 1 1
```

The central cell is grain 1. In the (3,3,3) neighbourhood, including the central cell, there are 14 cells with grain 5, 11 cells with grain 1, and 2 cells with grain 8. The highest number of cells belong to grain 5, hence the central cell state in changed to 5.

NOTE

There are no remote comms in this routine.

SOURCE

```
integer( kind=iarr ), allocatable :: array(:,:,:)
integer( kind=iarr ) :: neigh(27,2)=0_iarr, newgrain
integer( kind=idef ) :: &
 lbv(4),   & ! lower bounds of the complete (plus virtual) coarray
 ubv(4),   & ! upper bounds of the complete (plus virtual) coarray
 lbr(4),   & ! lower bounds of the "real" coarray, lower virtual+1
 ubr(4),   & ! upper bounds of the "real" coarray, upper virtual-1
 x1,x2,x3, & ! local coordinates in an array, which are also
 n1,n2,n3    ! local coord. of the neighbours [-1,0,1]
```

```
integer :: errstat,i

! determine the extents
lbv = lbound(coarray)
ubv = ubound(coarray)
lbr = lbv+1
ubr = ubv-1

! allocate the temp array
allocate( array( lbv(1):ubv(1), lbv(2):ubv(2), lbv(3):ubv(3) ), &
          stat=errstat )
if (errstat .ne. 0) then
  write (*,'(2(a,i0))') "ERROR: cgca_gbs/cgca_m2gb: image: ",          &
    this_image(), " allocate( array ): err. code: ", errstat
  error stop
end if

! copy the grain layer to the temp array
array = coarray(:,:,:,cgca_state_type_grain)

! loop over all cells
do x3 = lbr(3), ubr(3)
do x2 = lbr(2), ubr(2)
do x1 = lbr(1), ubr(1)

  ! Construct the array with all neighbour numbers.
  ! Note, at this stage the array might contain non-grain
  ! phase, e.g. liquid, if the cell is at the model
  ! boundary, with non-grain halo cell neighbours.
  ! It is easier to ignore these cases on calculation.
  ! We'll simply forbid changing into non-grain state later.
  i=0
  do n3 = -1,1
  do n2 = -1,1
  do n1 = -1,1
    i=i+1
    ! The first column of neigh array contains grain numbers
    ! for all 27 cells in the (3,3,3) array
    neigh(i,1) =  array( x1+n1, x2+n2, x3+n3 )
  end do
  end do
  end do

  ! Count the number of neighbours of each grain, and
  ! assign to the second column of array neigh.
  do i=1,27
    neigh(i,2) = int( count( neigh(:,1) .eq. neigh(i,1) ), kind=iarr )
  end do

  ! Get the grain number that has most cells in the neigh array
  newgrain = neigh( maxloc( neigh(:,2), dim=1 ), 1 )
```

```
  if ( newgrain .ne. cgca_liquid_state )               &
    coarray(x1,x2,x3,cgca_state_type_grain) = newgrain

end do
end do
end do

! deallocate the temp array
! The intention is that this routine is called only
! once, or, at best, only a few times, so keeping the
! array allocated for the duration of the execution is a waste.
deallocate( array, stat=errstat )
if (errstat .ne. 0) then
  write (*,'(2(a,i0))') "ERROR: cgca_gbs/cgca_m2gb: image: ",          &
    this_image(), " deallocate( array): err. code: ", errstat
  error stop
end if

end subroutine cgca_gbs
```

11.4 cgca_m2gb/cgca_gcf

[cgca_m2gb] [Subroutines]

NAME

cgca_gcf

SYNOPSIS

```
subroutine cgca_gcf( g1, g2 )
```

INPUTS

```
integer( kind=iarr ), intent( in ) :: g1, g2
```

SIDE EFFECTS

gc (11.11) array modified

DESCRIPTION

This routine changes the state of the grain boundary integrity between the two provided grains to cgca_gb_state_fractured (9.13).

USES

gc (11.11)

USED BY

cgca_clvgsd (26.5), cgca_clvgsp (26.7), cgca_gcupdn (26.15.2)

SOURCE

```
integer( kind=iarr ) :: tmp, grain1, grain2
integer :: matches, img, i, gclen, match1

img = this_image()

!*********************************************************************72
! sanity checks
!*********************************************************************72

! grains must be different. However, if there is no sync between images
! it's possible that 2 different images write to gcupd simultaneously,
! thus creating impossible combinations of GB pairs, including
! identical grain numbers for both grains. Let's issue a warning, not
! error for this case, and return, i.e. don't add this impossible pair.
if ( g1 .eq. g2 ) then
 write (*,"(a,i0,tr1,i0,a,i0)")                                          &
   "WARN: cgca_gcf: identical grain numbers: ", g1, g2, " image ", img
 return
 ! error stop
end if

!*********************************************************************72
```

```
! end of sanity checks
!*********************************************************************72

! Use the local variables
grain1 = g1
grain2 = g2

! Make grain1 < grain2
if ( grain2 .lt. grain1 ) then
  tmp = grain1
  grain1 = grain2
  grain2 = tmp
end if

! Look for matches
matches = 0
 match1 = 0_iarr
  gclen = ubound(gc,1)

! First match
do i = 1, gclen
  if ( gc(i,1) .eq. grain1 .and. gc(i,2) .eq. grain2 ) then
    gc(i,3) = cgca_gb_state_fractured
    matches = matches+1
    ! line number of the first match
    match1 = i
    exit
  end if
end do

! Second match, start from the line with the first match
do i = match1 + 1, gclen
  if ( gc(i,1) .eq. grain2 .and. gc(i,2) .eq. grain1 ) then
    gc(i,3) = cgca_gb_state_fractured
    matches = matches+1
    exit
  end if
end do

! Sanity check
if ( matches .eq. 0 ) then
 ! don't issue the INFO message, just swamps the stdout
 ! perhaps better logic should be built in future
 ! write (*,'(2(a,i0),tr1,i0,a)') "INFO: cgca_gcf: image ",              &
 ! img, " pair ", g1, g2, " does not exist."
else if ( matches .ne. 2 ) then
 write (*,'(3(a,i0),",",tr1,i0,".")') "ERROR: cgca_gcf: image ",       &
  img, ": found ", matches, " matches for pair: ", g1, g2
 write (*,'(a,i0,a)') "ERROR: cgca_gcf: image ", img,                  &
  ": Should have found exactly two matches or none!"
 write (*,'(a,i0,a)') "ERROR: cgca_gcf: image ", img,                  &
  ": This means the gc array is corrupted. Aborting!"
```

```
 error stop
end if

end subroutine cgca_gcf
```

11.5 cgca_m2gb/cgca_gcf_pure

[cgca_m2gb] [Subroutines]

NAME

cgca_gcf_pure

SYNOPSIS

```
subroutine cgca_gcf_pure( g1, g2, iflag )

!    Despite the name, this subroutine is actually not PURE!
!    I overlooked a restriction that a pure subroutine cannot
!    modify a variable accissible via host association, which
!    is gc in this case.
```

INPUTS

```
integer( kind=iarr ), intent(in) :: g1, g2
```

OUTPUTS

```
integer, intent(out) :: iflag
```

SIDE EFFECTS

gc (11.11) array modified

DESCRIPTION

This routine changes the state of the grain boundary integrity between the two provided grains to cgca_gb_state_fractured (9.13). IFLAG:

```
    0 - successful termination
    1 - the 2 grains are identical - probably fatal error
    2 - there is no such pair in the array - probably a warning
    3 - the number of grain pairs is not 2 - probably a fatal error
```

USES

gc (11.11)

USED BY

cgca_clvgsd (26.5), cgca_clvgsp (26.7), cgca_gcupdn (26.15.2)

SOURCE

```
integer( kind=iarr ) :: tmp, grain1, grain2
integer :: matches, img, i, gclen, match1

img = this_image()

! The two grains must be different. If not set iflag to 1 and return
! immediately.
```

```
if ( g1 .eq. g2 ) then
  iflag = 1
  return
end if

! Use the local variables
grain1 = g1
grain2 = g2

! Make grain1 < grain2
if ( grain2 .lt. grain1 ) then
  tmp = grain1
  grain1 = grain2
  grain2 = tmp
end if

! Look for matches
matches = 0
 match1 = 0_iarr
  gclen = ubound(gc,1)

! First match
do i = 1, gclen
  if ( gc(i,1) .eq. grain1 .and. gc(i,2) .eq. grain2 ) then

! CANNOT DO THIS in a PURE routine!!!
! READ the pure rules and fix!!!
!    gc(i,3) = cgca_gb_state_fractured

    matches = matches+1
    ! line number of the first match
    match1 = i
    exit
  end if
end do

! Second match, start from the line with the first match
do i = match1 + 1, gclen
  if ( gc(i,1) .eq. grain2 .and. gc(i,2) .eq. grain1 ) then

! CANNOT DO THIS in a PURE routine!!!
! READ the pure rules and fix!!!
!    gc(i,3) = cgca_gb_state_fractured

    matches = matches+1
    exit
  end if
end do

! Sanity check
if ( matches .eq. 0 ) then
 ! Set iflag to 2 and return immediately
```

```
 iflag = 2
 return
else if ( matches .ne. 2 ) then
 ! Set iflag to 3 and return immediately
 iflag = 3
 return
end if

end subroutine cgca_gcf_pure
```

11.6 cgca_m2gb/cgca_gcp

[cgca_m2gb] [Subroutines]

NAME

cgca_gcp

SYNOPSIS

```
subroutine cgca_gcp( ounit, fname )
```

INPUTS

```
integer( kind=idef ), intent(in) :: ounit
character( len=* ), intent(in) :: fname
```

SIDE EFFECTS

create output file on each processor and dump gc (11.11) array to it

DESCRIPTION

Print grain connectivity (gc (11.11)) array to file. Remember that gc (11.11) is a *local* array, and hence this routine will write only the gc (11.11) from the processor which called this routine. It is indended to be called from all images with file names supplied linked to the processor/image number via this_image.

USES

gc (11.11)

USED BY

none

SOURCE

```
integer :: i, errstat=0, img
character( len=10) :: state

img = this_image()

! open file
open( unit=ounit, file=trim(fname), form="formatted",                  &
      status="replace", iostat=errstat )
if ( errstat .ne. 0 ) then
  write (*,'(2(a,i0))') "ERROR: cgca_gcp: img: ", img,                 &
                        " cannot open file, error code: ", errstat
  error stop
end if

! write data, one line at a time
do i = 1, ubound( gc, dim=1 )

  ! convert numerical values of intact and fractured into words
  if ( gc(i,3) .eq. cgca_gb_state_intact) then
    state = "intact"
```

```
  else if ( gc(i,3) .eq. cgca_gb_state_fractured) then
    state = "fractured"
  else
    write (*,'(2(a,i0))') "ERROR: cgca_gcp: image: ", img,             &
                  " state is neither intact nor fractured: ", gc(i,3)
    error stop
  end if

  ! actual write
  write ( unit=ounit, fmt="(2(i0,tr1),a10)", iostat=errstat)           &
   gc(i,1:2), state

  if ( errstat .ne. 0 ) then
    write (*,'(2(a,i0))') "ERROR: cgca_gcp: image: ", img,             &
                     " cannot write to file, error code: ", errstat
    error stop
  end if

end do

! close file
close( ounit, iostat=errstat )
if ( errstat .ne. 0 ) then
  write (*,'(2(a,i0))') "ERROR: cgca_gcp: image: ", img,               &
                        " cannot close file, error code: ", errstat
  error stop
end if

end subroutine cgca_gcp
```

11.7 cgca_m2gb/cgca_gcr

[cgca_m2gb] [Subroutines]

NAME

cgca_gcr

SYNOPSIS

```
subroutine cgca_gcr( g1, g2, intact )
```

INPUTS

```
integer( kind=iarr ), intent( in ) :: g1, g2
```

OUTPUT

```
logical( kind=ldef ), intent(out) :: intact
```

SIDE EFFECTS

none

DESCRIPTION

Find and return the boundary integrity state between 2 grains. Returns .true. if the boundary is intact, .false. if it is fractured. The routine does some simple checks and stops with errors where appropriate.

USES

gc (11.11)

USED BY

module cgca_m3clvg (26): cgca_clvgsd (26.5)

SOURCE

```
integer( kind=iarr ) :: tmp, grain1, grain2
integer :: img, i
logical :: match

img = this_image()

!*********************************************************************73
! sanity checks
!*********************************************************************73

! grains must be different
if ( g1 .eq. g2 ) then
  write (*,'(2(a,i0),tr1,i0,".")') "ERROR: cgca_gcr: image ", img, &
    ": The two grain numbers must differ: ", g1, g2
  error stop
end if

!*********************************************************************73
```

```
! end of sanity checks
!*********************************************************************73

! Use the local variables
grain1 = g1
grain2 = g2

! Make grain1 < grain2
if ( grain2 .lt. grain1 ) then
     tmp = grain1
  grain1 = grain2
  grain2 = tmp
end if

! Look for matches
match=.false.
do i=1, ubound( gc, 1 )
 mtch: if ( gc(i,1) .eq. grain1 .and. gc(i,2) .eq. grain2 ) then
  match = .true.
  if ( gc(i,3) .eq. cgca_gb_state_intact) then
   intact = .true.
   exit
  else if ( gc(i,3) .eq. cgca_gb_state_fractured) then
   intact = .false.
   exit
  else
   ! Must never end up here. This is an error
   write (*,'(a,i0)')                                                  &
     "ERROR: cgca_gcr: the state is neither intact nor fractured. &
        &Integrity of the gc array is broken. img: ", img
   error stop
  end if
 end if mtch
end do

! Sanity check
if (.not. match) then
  write (*,'(a,i0,a,i0,tr1,i0,".")') "WARN: cgca_gcr: image ", &
    img, ": No match found for given pair: ", g1, g2
  write (*,'(a,i0,a)') "WARN: cgca_gcr: image ", img,      &
    ": Returning .true. in intact!"
  intact = .true.
end if

end subroutine cgca_gcr
```

11.8 cgca_m2gb/cgca_gcr_pure

[cgca_m2gb] [Subroutines]

NAME

cgca_gcr_pure

SYNOPSIS

```
pure subroutine cgca_gcr_pure( g1, g2, intact, iflag )
```

INPUTS

```
integer( kind=iarr ), intent(in) :: g1, g2
```

OUTPUT

```
logical( kind=ldef ), intent(out) :: intact
integer, intent(out) :: iflag
```

SIDE EFFECTS

None, this is a PURE subroutine

DESCRIPTION

Find and return the boundary integrity state between 2 grains. Returns .true. if the boundary is intact, .false. if it is fractured. The routine does some simple checks and stops with errors where appropriate.

NOTES

Cannot have any external IO in a PURE subroutine. Hence all error conditions are returned back via integer flag, IFLAG. IFLAG meanings:

```
0 - successful completion
1 - The 2 grains are identical. This is a fatal condition,
    the caller routine should probably abort.
```

USES

gc (11.11)

USED BY

module cgca_m3clvg (26): cgca_clvgsd (26.5)

SOURCE

```
integer( kind=iarr ) :: tmp, grain1, grain2
integer :: img, i
logical :: match

img = this_image()

! The 2 grains must be different. If not, set iflag=1, and
! return immediately.
```

```
if ( g1 .eq. g2 ) then
  iflag = 1
  return
end if

! Use local variables
grain1 = g1
grain2 = g2

! Make grain1 < grain2
if ( grain2 .lt. grain1 ) then
     tmp = grain1
  grain1 = grain2
  grain2 = tmp
end if

! Look for matches
match=.false.
do i=1, ubound( gc, 1 )
  mtch: if ( gc(i,1) .eq. grain1 .and. gc(i,2) .eq. grain2 ) then
    match = .true.
    if ( gc(i,3) .eq. cgca_gb_state_intact) then
      intact = .true.
      exit
    else if ( gc(i,3) .eq. cgca_gb_state_fractured) then
      intact = .false.
      exit
    else
      ! Must never end up here. This is an error. Set iflag=2 and
      ! return immediately.
      iflag = 2
      return
    end if
  end if mtch
end do

! No match probably means something went wrong. This is probably
! not a fatal condition, but watch out! Set iflag=3 and return
! .true. in inact.
if ( .not. match ) then
  iflag = 3
  intact = .true.
end if

end subroutine cgca_gcr_pure
```

11.9 cgca_m2gb/cgca_gcu

[cgca_m2gb] [Subroutines]

NAME

cgca_gcu

SYNOPSIS

```
subroutine cgca_gcu( coarray )
```

INPUT

```
integer( kind=iarr ), intent( in ), allocatable ::                    &
  coarray(:,:,:,:) [:,:,:]
```

SIDE EFFECTS

Updated state of gc (11.11) array

DESCRIPTION

Update a local grain connectivity (gc (11.11)) array. This means: scan the whole of the local real (no halos) model coarray. For each cell check all 26 neighbours. When a cell has a neighbour of a different number, this is understood as a grain boundary between the grains denoted by the states of both cells. If this pair is not already in gc (11.11), then it is added to it in the right place. gc (11.11) is sorted, first by the first column, then by the second.

state(26,2) - 26 possible pairs of different states, and 2 states. In practice, 26 is a stupid value. This can only happen if a grain is only 1 cell, and if each of the 26 neighbouring cells has a unique different value. Anyway, defining the array of this size seems fool proof.

The 3rd entry in each row of gc (11.11) is the grain boundary integrity, either cgca_gb_state_intact (9.14) - intact or cgca_gb_state_fractured (9.13) - fractured.

NOTES

Only the (:,:,:,cgca_state_type_grain (9.24)) values are used in this routine. The cgca_state_type_frac (9.23) values are *not* used.

USES

gc (11.11), cgca_agc (11.1), cgca_dgc (11.2)

USED BY

via module cgca_m2gb (11)

SOURCE

```
integer( kind=iarr ), allocatable :: tmp(:,:)
integer( kind=iarr ) :: state(26,2) = 0_iarr
integer :: errstat=0, x1, x2, x3, n1, n2, n3, lbr(4), ubr(4), pair,    &
 con, img, gclen, i, j

img = this_image()

!*********************************************************************72
```

```
! checks
!*********************************************************************72

if ( .not. allocated(coarray) ) then
 write (*,'(a,i0)') "ERROR: cgca_gc: coarray not allocated, img: ", img
 error stop
end if

!*********************************************************************72
! end of checks
!*********************************************************************72

! if gc not already allocated, allocate to zero length!
if ( .not.  allocated( gc ) ) call cgca_agc(0)

! Assume the coarray has halos. Ignore those
lbr = lbound( coarray ) + 1
ubr = ubound( coarray ) - 1

       do x3 = lbr(3), ubr(3)
       do x2 = lbr(2), ubr(2)
inner: do x1 = lbr(1), ubr(1)

  ! Loop over 26 neighbours, find all neighbours with different states
  pair = 0
  do n3 = x3-1, x3+1
  do n2 = x2-1, x2+1
  do n1 = x1-1, x1+1

    ! if the states differ but no liquid
    if ( ( coarray(x1,x2,x3,cgca_state_type_grain) .ne.                &
           coarray(n1,n2,n3,cgca_state_type_grain) )                   &
         .and.                                                         &
         (coarray(n1,n2,n3,cgca_state_type_grain) .ne.                 &
          cgca_liquid_state) ) then

      ! this is another pair
      pair = pair + 1

      ! add this pair to state array
      state( pair, 1 ) = coarray( x1, x2, x3, cgca_state_type_grain )
      state( pair, 2 ) = coarray( n1, n2, n3, cgca_state_type_grain )

      ! Make state(pair,1) < state(pair,2)
      if ( coarray(n1,n2,n3,cgca_state_type_grain) .lt.                &
           coarray(x1,x2,x3,cgca_state_type_grain) ) then

        ! swap them
        state(pair,1) = coarray(n1,n2,n3,cgca_state_type_grain)
        state(pair,2) = coarray(x1,x2,x3,cgca_state_type_grain)
      end if
    end if
```

```
  end do
  end do
  end do

  ! At the end of the this loop, "state" array will have
  ! all pairs of different grains containg the central cell.
  ! Now all of these pair, which are not already in gc, need
  ! to be added there.

  ! For all identified pairs of states
  newcon: do con = 1, pair

    ! Check if this connectivity is already in the gc array.
    ! If yes, cycle to the next pair.
    gclen = ubound( gc, dim=1 )

    do i = 1, gclen
      ! each pair repeats twice, so the order doesn't matter
      ! If the pair is already in gc, then cycle
      if ( gc(i,1) .eq. state(con,1) .and. &
           gc(i,2) .eq. state(con,2) ) cycle newcon
    end do

    ! If this connectivity is not yet in the gc array, then extend
    ! the gc array by 2 rows.
    gclen = gclen + 2

! this should be replaced by MOVE_ALLOC!!!

    ! Allocate a temp array with length 2 more than gc, and set
    ! to intact.
    allocate( tmp(gclen, 3), source=0_iarr, stat=errstat )
    if (errstat .ne. 0) then
      write (*,'(2(a,i0))') "ERROR: cgca_gc: img: ", img,              &
                        " cannot allocate tmp, error code: ", errstat
      error stop
    end if

    ! Copy gc to the beginning of the temp array
    tmp(1:gclen-2,:) = gc

    ! Can replace this with move_alloc, but I make use of the
    ! temp array later, so this is better!
    call cgca_dgc
    call cgca_agc( gclen )
    gc = tmp

!*********************************************************
! debug output
!
!if (this_image() .eq. 1) then
```

```
! do k=1,gclen
!  write (*,*) gc(k,:)
! end do
! write (*,*) "state1,state2",state(con,1),state(con,2)
!end if
!*********************************************************

    ! find where to insert the first pair: state(con,1), state(con,2)
    i1: do i=1,gclen

      ! If state(con,1) doesn't exist in the first row at all, but
      ! is smaller that some existing value
      if (state(con,1) .lt. gc(i,1)) then
        tmp(1:gclen-i-1,:) = gc(i:gclen-2,:)
        gc(i,1) = state(con,1)
        gc(i,2) = state(con,2)
        gc(i,3) = cgca_gb_state_intact
        gc(i+1:gclen-1,:) = tmp(1:gclen-i-1,:)
        exit i1
      else if (state(con,1) .eq. gc(i,1)) then

        ! If state(con,1) already exists in the first column, then
        ! need to sort by the second column
        do j=i,gclen
          if (state(con,2) .lt. gc(j,2)) then
            tmp(1:gclen-j-1,:) = gc(j:gclen-2,:)
            gc(j,1) = state(con,1)
            gc(j,2) = state(con,2)
            gc(j,3) = cgca_gb_state_intact
            gc(j+1:gclen-1,:) = tmp(1:gclen-j-1,:)
            exit i1
          else if (state(con,2) .gt. gc(j,2) .and.    &
                   gc(j+1,1) .ne. gc(j,1) ) then

            ! If state(con,2) is greater than all column 2 values,
            ! for the same column 1 value, just add it *after* this
            ! row.
            tmp(1:gclen-j-2,:) = gc(j+1:gclen-2,:)
            gc(j+1,1) = state(con,1)
            gc(j+1,2) = state(con,2)
            gc(j+1,3) = cgca_gb_state_intact
            gc(j+2:gclen-1,:) = tmp(1:gclen-j-2,:)
            exit i1
          end if
        end do
      else if (i .gt. gclen-2) then

        ! If state(con,1) is greater than all column 1 values, just
        ! add it at the end.
        gc(i,1) = state(con,1)
        gc(i,2) = state(con,2)
        gc(i,3) = cgca_gb_state_intact
```

```
      exit i1
    end if
  end do i1

 ! Find where to insert the second pair:
 ! ( state(con,2), state(con,1) ).
 ! Note that as the first pair has been inserted already,
 ! this loop is not exactly as the previous one.
 i2: do i=1,gclen
   ! If state(con,2) doesn't exist in the first row at all, but
   ! is smaller that some existing value
   if (state(con,2) .lt. gc(i,1)) then
     tmp(1:gclen-i,:) = gc(i:gclen-1,:)
     gc(i,1) = state(con,2)
     gc(i,2) = state(con,1)
     gc(i,3) = cgca_gb_state_intact
     gc(i+1:gclen,:) = tmp(1:gclen-i,:)
     exit i2
   else if (state(con,2) .eq. gc(i,1)) then
     ! If state(con,2) already exist in the first column, then
     ! need to sort by the second column by state(con,1)

     do j=i,gclen
       if (state(con,1) .lt. gc(j,2)) then
         tmp(1:gclen-j,:) = gc(j:gclen-1,:)
         gc(j,1) = state(con,2)
         gc(j,2) = state(con,1)
         gc(j,3) = cgca_gb_state_intact
         gc(j+1:gclen,:) = tmp(1:gclen-j,:)
         exit i2
       else if (state(con,1) .gt. gc(j,2) .and.   &
                gc(j+1,1) .ne. gc(j,1) ) then
         ! If state(con,1) is greater than all column 2 values, just
         ! add it *after* this row.
         tmp(1:gclen-j-1,:) = gc(j+1:gclen-1,:)
         gc(j+1,1) = state(con,2)
         gc(j+1,2) = state(con,1)
         gc(j+1,3) = cgca_gb_state_intact
         gc(j+2:gclen,:) = tmp(1:gclen-j-1,:)
         exit i2
       end if
     end do
   else if (i .gt. gclen-1) then
     ! If state(con,2) is greater than all column 1 values, just
     ! add it at the end.
     gc(i,1) = state(con,2)
     gc(i,2) = state(con,1)
     gc(i,3) = cgca_gb_state_intact
     exit i2
   end if
 end do i2
```

```
  deallocate(tmp, stat=errstat )
  if (errstat .ne. 0) then
    write (*,'(2(a,i0))') "ERROR: cgca_gcu: img: ", img,               &
                          " cannot deallocate tmp, err code: ", errstat
    error stop
  end if

end do newcon
end do inner
end do
end do

end subroutine cgca_gcu
```

11.10 cgca_m2gb/cgca_igb

[cgca_m2gb] [Subroutines]

NAME

cgca_igb

SYNOPSIS

```
subroutine cgca_igb ( coarray )
```

INPUT

```
integer( kind=iarr ), allocatable, intent(inout) :: &
 coarray(:,:,:,:)[:,:,:]
```

SIDE EFFECTS

state of coarray changes

DESCRIPTION

Initialise Grain Boundary (IGB) cells. Simply scan through the (:,:,:,cgca_state_type_grain (9.24)) array and mark all cells which have a neighbour of a different state as cgca_gb_state_intact (9.14) in (:,:,:,cgca_state_type_frac (9.23)) array. Clearly this routine must be called before any fracture is simulated. Possibly a halo exchange should be called before and/or after calling this routine.

NOTES

All images must call this routine. No remote comms in this routine.

SOURCE

```
integer( kind=idef ) :: &
 lbv(4),   & ! lower bounds of the complete (plus virtual) coarray
 ubv(4),   & ! upper bounds of the complete (plus virtual) coarray
 lbr(4),   & ! lower bounds of the "real" coarray, lower virtual+1
 ubr(4),   & ! upper bounds of the "real" coarray, upper virtual-1
 x1,x2,x3, & ! local coordinates in an array, which are also
 n1,n2,n3    ! local coord. of the neighbours [-1,0,1]

! determine the extents
lbv = lbound(coarray)
ubv = ubound(coarray)
lbr = lbv+1
ubr = ubv-1

! scan over all cells
outer: do x3 = lbr(3), ubr(3)
       do x2 = lbr(2), ubr(2)
       do x1 = lbr(1), ubr(1)

  ! choose all neighbourhood cells
  inner: do n3 = x3-1, x3+1
         do n2 = x2-1, x2+1
```

```
          do n1 = x1-1, x1+1

    ! Ignore the global halo cells
    if ( n1 .eq. lbv(1) .or. n1 .eq. ubv(1) .or.                      &
         n2 .eq. lbv(2) .or. n2 .eq. ubv(2) .or.                      &
         n3 .eq. lbv(3) .or. n3 .eq. ubv(3) ) cycle

    ! If the neighbouring grain .ne. the state of the central
    ! cell, then mark this central cell as GB in fracture
    ! array and exit
    if ( coarray(n1, n2, n3, cgca_state_type_grain) .ne.              &
         coarray(x1, x2, x3, cgca_state_type_grain) ) then
      coarray(x1, x2, x3, cgca_state_type_frac) = cgca_gb_state_intact
      exit inner
    end if

  end do
  end do
  end do inner

end do
end do
end do outer

end subroutine cgca_igb
```

11.11 cgca_m2gb/gc

[cgca_m2gb] [Data structures]

NAME

gc

SYNOPSIS

```
integer( kind=iarr ), allocatable, save :: gc(:,:)
```

DESCRIPTION

Local, *not* coarray, grain connectivity (gc) array. This array must be SAVEd.

NOTES

gc is a private array. It is not accessible from outside of cgca_m2gb (11) module, hence we can use a simple name, with no cgca_ prefix. gc is zero initially!

USED BY

All routines of module cgca_m2gb (11): cgca_gcu (11.9), cgca_gcp (11.6), cgca_agc (11.1), cgca_dgc (11.2), cgca_gcf (11.4), cgca_gcr (11.7).

12 CGPACK/cgca_m2geom

[Modules]

NAME

cgca_m2geom

SYNOPSIS

```
!$Id: cgca_m2geom.f90 379 2017-03-22 09:57:10Z mexas $

module cgca_m2geom
```

DESCRIPTION

Module dealing with various 3D geometrical problems

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

CONTAINS

cgca_boxsplit (12.1)

USES

cgca_m1co (9)

USED BY

cgca_m3pfem (30)

SOURCE

```
use cgca_m1co, only : idef
implicit none

private
public :: cgca_boxsplit

contains
```

12.1 cgca_m2geom/cgca_boxsplit

[cgca_m2geom] [Subroutines]

NAME

cgca_boxsplit

SYNOPSIS

```
subroutine cgca_boxsplit( lwr, upr, lwr1, upr1, lwr2, upr2 )
```

INPUTS

```
!    lwr(3) - integer, lower corner of the box
!    upr(3) - integer, upper corner of the box

integer( kind=idef ), intent( in ) :: lwr(3), upr(3)
```

OUTPUTS

```
!    lwr1(3) - integer, lower corner of new box 1
!    upr1(3) - integer, upper corner of new box 1
!    lwr2(3) - integer, lower corner of new box 2
!    upr2(3) - integer, upper corner of new box 2

integer( kind=idef ), intent( out ) :: lwr1(3), upr1(3), lwr2(3),      &
  upr2(3)
```

DESCRIPTION

This routine splits the box, specified by two corner coordinates into two smaller boxes, along the biggest dimension of the original box.

SOURCE

```
integer( kind=idef ) :: boxsize(3), splitdim

! If the box is only a single cell, return immediately
if ( all( lwr .eq. upr ) ) then
  lwr1 = lwr
  lwr2 = lwr
  upr1 = upr
  upr2 = upr
  return
end if

! Find the biggest dimension of the box.
boxsize = upr - lwr + 1
splitdim = maxloc( boxsize, dim=1 ) ! 1, 2 or 3 only

! Set the dimensions of each new box initially equal to
! the old box
  lwr1 = lwr
```

```
  upr1 = upr
  lwr2 = lwr
  upr2 = upr

! Change only relevant dimensions
if ( splitdim .eq. 1 ) then
  upr1(1) = ( lwr(1) + upr(1) ) / 2 ! new box 1
  lwr2(1) = upr1(1) + 1             ! new box 2
else if ( splitdim .eq. 2 ) then
  upr1(2) = ( lwr(2) + upr(2) ) / 2 ! new box 1
  lwr2(2) = upr1(2) + 1             ! new box 2
else if ( splitdim .eq. 3 ) then
  upr1(3) = ( lwr(3) + upr(3) ) / 2 ! new box 1
  lwr2(3) = upr1(3) + 1             ! new box 2
end if

end subroutine cgca_boxsplit
```

13 CGPACK/cgca_m2glm

[Modules]

NAME

cgca_m2glm

SYNOPSIS

```
!$Id: cgca_m2glm.f90 379 2017-03-22 09:57:10Z mexas $

module cgca_m2glm
```

DESCRIPTION

Module dealing with Global to Local Mapping (glm) and vice versa

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

CONTAINS

cgca_gl (13.1), cgca_lg (13.4), cgca_ico (13.2)

USES

cgca_m1co (9)

USED BY

cgca_m3nucl (29)

SOURCE

```
use cgca_m1co
implicit none

private
public :: cgca_gl, cgca_lg, cgca_ico, cgca_ico2

contains
```

13.1 cgca_m2glm/cgca_gl

[cgca_m2glm] [Subroutines]

NAME

cgca_gl

SYNOPSIS

```
subroutine cgca_gl(super,coarray,imgpos,local)
```

INPUTS

```
integer(kind=idef),intent(in) :: super(3)
integer(kind=iarr),allocatable,intent(inout) :: coarray(:,:,:,:)[:,:,:]
```

OUTPUT

```
integer(kind=idef),intent(out) :: imgpos(3),local(3)
```

DESCRIPTION

This routine converts a cell coordinate from a global, super, array to the image coordinates in the coarray grid and the local cell coordinates in this image :

- super(3) are cell coordinates in a super array
- coarray is the model
- imgpos(3) is the image position in the grid
- local(3) are cell coordinates in that image's array

NOTES

The global coordinates must start from 1!

Any image can call this routine

USES USED BY SOURCE

```
integer :: &
 lbr(4)   ,& ! lower bounds of the "real" coarray, lbv+1
 ubr(4)   ,& ! upper bounds of the "real" coarray, ubv-1
 szr(3)   ,& ! size or the "real" coarray, ubr-lbr+1
 lcob(3)  ,& ! lower cobounds of the coarray
 ucob(3)  ,& ! upper cobounds of the coarray
 usup(3)  ,& ! upper bound of the super array, szr*(ucob-lcob+1)
 thisimage

thisimage = this_image()

! check for allocated
```

```
if (.not. allocated(coarray)) then
 write (*,'(a,i0)') "ERROR: cgca_gl: image", thisimage
 write (*,'(a)') "ERROR: cgca_gl: coarray is not allocated"
 error stop
end if

lbr=lbound(coarray)+1
ubr=ubound(coarray)-1

! the 4th dimension is to do with the number of cell state
! types. This is not relevant here.
szr=ubr(1:3)-lbr(1:3)+1

lcob=lcobound(coarray)
ucob=ucobound(coarray)
usup=szr*(ucob-lcob+1)

! check for bounds

if (any(super .gt. usup) .or. any(super .lt. 1)) then
 write (*,'(a,i0)') "ERROR: cgca_gl: image", thisimage
 write (*,'(a)') "ERROR: cgca_gl: one or more super array&
                  & coordinate(s) are ouside the bounds"
 write (*,'(a,3(i0,tr1))') "ERROR: cgca_gl: super array coord: ",super
 write (*,'(a)') "ERROR: cgca_gl: lower bound must be 1"
 write (*,'(a,3(i0,tr1))') "ERROR: cgca_gl: upper bounds: ", usup
 error stop
end if

! actual calculation

imgpos = lcob + (super-1)/szr
local = lbr(1:3) + super-szr*(imgpos-lcob) - 1

! checks after

if (any(imgpos .gt. ucob) .or. any(imgpos .lt. lcob)) then
 write (*,'(a,i0)') "ERROR: cgca_gl: image", thisimage
 write (*,'(a)') "ERROR: cgca_lg: one or more image positions&
                  & are ouside the bounds"
 write (*,'(a,3(i0,tr1))') &
  "ERROR: cgca_gl: image positions: ",imgpos
 write (*,'(a,3(i0,tr1))') &
  "ERROR: cgca_gl: lower image grid bounds: ", lcob
 write (*,'(a,3(i0,tr1))') &
  "ERROR: cgca_gl: upper image grid bounds: ", ucob
 error stop
end if

if (any(local .gt. ubr(1:3)) .or. any(local .lt. lbr(1:3))) then
 write (*,'(a,i0)') "ERROR: cgca_gl: image", thisimage
 write (*,'(a)') "ERROR: cgca_lg: one or more local coordinates &
```

```
                  & are ouside the bounds"
 write (*,'(a,3(i0,tr1))') "ERROR: cgca_gl: local coordinates: ",local
 write (*,'(a,3(i0,tr1))') "ERROR: cgca_gl: lower bounds: ", lbr
 write (*,'(a,3(i0,tr1))') "ERROR: cgca_gl: upper bounds: ", ubr
 error stop
end if

end subroutine cgca_gl
```

13.2 cgca_m2glm/cgca_ico

[cgca_m2glm] [Subroutines]

NAME

cgca_ico

SYNOPSIS

```
subroutine cgca_ico( ind, cosub, flag )
```

INPUTS

```
integer( kind=idef ), intent(in) :: ind
```

OUTPUT

```
integer( kind=idef ), intent(out) :: cosub( cgca_scodim ), flag
```

DESCRIPTION

This routine converts image index of coarray into its set of cosubscripts.

NOTES

This is a serial routine, just computation, no inter-image communication is involved. Any and all images can call this routine. flag is set to 0 on normal exit. flag is 1 if the coarray index .lt. 0.

SOURCE

```
integer( kind=idef ) :: codim( cgca_scodim ), step, step2, rem, rem2

! Set as default
flag = 0

! Sanity check
if ( ind .le. 0 ) then

  ! Set the flag and return immediately.
  flag = 1
  return
end if

! codimensions
codim = cgca_sucob - cgca_slcob + 1

! along 1
step = mod( ind, codim(1) )
if ( step .eq. 0 ) step = codim(1)
cosub(1) = cgca_slcob(1) + step - 1

! along 2
! number of full layers
step = ind / ( codim(1) * codim(2) )
```

```
! number of images in the last unfilled layer
rem = mod( ind , codim(1) * codim(2) )

! if all layers ar filled, take step2 as codim(2)
if ( rem .eq. 0 ) then
  step2 = codim(2)
else
  ! number of full columns in the last unfilled layer
  step2 = rem / codim(1)
end if

! number of images in the last unfilled column
rem2 = mod( rem, codim(1) )

! if it's not zero, increment the step
if ( rem2 .ne. 0 ) step2 = step2 + 1

cosub(2) = cgca_slcob(2) + step2 -1

! along 3
if ( rem .ne. 0 ) step = step + 1
cosub(3) = cgca_slcob(3) + step - 1

end subroutine cgca_ico
```

13.3 cgca_m2glm/cgca_ico2

[cgca_m2glm] [Subroutines]

NAME

cgca_ico2

SYNOPSIS

```
subroutine cgca_ico2( lcob, ucob, ind, cosub )
```

INPUTS

```
integer( kind=idef ), intent(in) :: lcob(:), ucob(:), ind
```

OUTPUT

```
integer( kind=idef ), intent(out) :: cosub( size(lcob) )
```

DESCRIPTION

This routine converts image index of coarray into its set of cosubscripts. It borrows the code from the f2008 standard,

```
http://j3-fortran.org/doc/year/10/10-007r1.pdf
```

Section C.10.1.

NOTES

This is a serial routine, just computation, no inter-image

SOURCE

```
integer :: n, i, m, ml, extent

n = size( cosub )
m = ind - 1
do i = 1, n-1
  extent = ucob(i) - lcob(i) + 1
  ml = m
  m = m / extent
  cosub( i ) = ml - m * extent + lcob(i)
end do
cosub( n ) = m + lcob( n )

end subroutine cgca_ico2
```

13.4 cgca_m2glm/cgca_lg

[cgca_m2glm] [Subroutines]

NAME

cgca_lg

SYNOPSIS

```
subroutine cgca_lg(imgpos,local,coarray,super)
```

INPUTS

```
integer(kind=idef),intent(in) :: imgpos(3),local(3)
integer(kind=iarr),allocatable,intent(inout) :: coarray(:,:,:,:)[:,:,:]
```

OUTPUT

```
integer(kind=idef),intent(out) :: super(3)
```

DESCRIPTION

This routine converts the image coordinates in the grid and the local cell coordinates in this image into the global cell coordinates in the super array:

```
   - imgpos(3) is the image position in the grid
   - local(3) are cell coordinates in that image's array
   - coarray is the model
   - super(3) are cell coordinates in a super array
```

NOTES

The global, super, coordinates must start from 1!

Any image can call this routine

USES USED BY

cgca_gbf1f (28.1)

SOURCE

```
integer :: &
 lbr(4)   ,& ! lower bounds of the "real" coarray, lbv+1
 ubr(4)   ,& ! upper bounds of the "real" coarray, ubv-1
 szr(3)   ,& ! size or the "real" coarray, ubr-lbr+1
 lcob(3)  ,& ! lower cobounds of the coarray
 ucob(3)  ,& ! upper cobounds of the coarray
 usup(3)  ,& ! upper bound of the super array, szr*(ucob-lcob+1)
 thisimage

thisimage = this_image()

! check for allocated
```

```
if (.not. allocated(coarray)) then
 write (*,'(a,i0)') "ERROR: cgca_lg: image", thisimage
 write (*,'(a)') "ERROR: cgca_lg: coarray is not allocated"
 error stop
end if

lbr=lbound(coarray)+1
ubr=ubound(coarray)-1

! the 4th dimension is to do with the number of cell state
! types. This is not relevant here.
szr=ubr(1:3)-lbr(1:3)+1

lcob=lcobound(coarray)
ucob=ucobound(coarray)
usup=szr*(ucob-lcob+1)

! check for bounds

if (any(imgpos .gt. ucob) .or. any(imgpos .lt. lcob)) then
 write (*,'(a,i0)') "ERROR: cgca_lg: image", thisimage
 write (*,'(a)') "ERROR: cgca_lg: one or more image positions&
                  & are ouside the bounds"
 write (*,'(a,3(i0,tr1))') "ERROR: cgca_lg: image positions: ",imgpos
 write (*,'(a,3(i0,tr1))') &
  "ERROR: cgca_lg: lower image grid bounds: ", lcob
 write (*,'(a,3(i0,tr1))') &
  "ERROR: cgca_lg: upper image grid bounds: ", ucob
 error stop
end if

if (any(local .gt. ubr(1:3)) .or. any(local .lt. lbr(1:3))) then
 write (*,'(a,i0)') "ERROR: cgca_lg: image", thisimage
 write (*,'(a)') "ERROR: cgca_lg: one or more local coordinates&
                  & are ouside the bounds"
 write (*,'(a,3(i0,tr1))') &
  "ERROR: cgca_lg: local coordinates: ", local
 write (*,'(a,3(i0,tr1))') &
  "ERROR: cgca_lg: lower bounds: ", lbr
 write (*,'(a,3(i0,tr1))') &
  "ERROR: cgca_lg: upper bounds: ", ubr
 error stop
end if

! actual calculation

super = szr*(imgpos-lcob) + local-lbr(1:3)+1

! check for bounds

if (any(super .gt. usup) .or. any(super .lt. 1)) then
 write (*,'(a,i0)') "ERROR: cgca_lg: image", thisimage
```

```
 write (*,'(a)') "ERROR: cgca_lg: one or more super array &
                  & coordinates are ouside the bounds"
 write (*,'(a,3(i0,tr1))') &
  "ERROR: cgca_lg: super array coord: ",super
 write (*,'(a)') "ERROR: cgca_lg: lower bound must be 1"
 write (*,'(a,3(i0,tr1))') "ERROR: cgca_lg: upper bounds: ", usup
 error stop
end if

end subroutine cgca_lg
```

14 CGPACK/cgca_m2hdf5

[Modules]

NAME

cgca_m2hdf5

SYNOPSIS

```
!$Id: cgca_m2hdf5.f90 414 2017-05-18 12:55:39Z mexas $

module cgca_m2hdf5
```

DESCRIPTION

Module with hdf5 related routines

AUTHOR

Luis Cebamanos

COPYRIGHT

See LICENSE (33)

CONTAINS

Subroutines: cgca_pswci4 (14.1)

USES

cgca_m1co (9)

USED BY

cgca

SOURCE

```
use cgca_m1co
use mpi
use hdf5
implicit none
private
public :: cgca_pswci4

contains
```

14.1 cgca_m2hdf5/cgca_pswci4

[cgca_m2hdf5] [Subroutines]

NAME

cgca_pswci4

SYNOPSIS

```
subroutine cgca_pswci4( coarray, stype, fname )
```

INPUTS

```
    integer( kind=iarr ), allocatable, intent( in ) ::                &
      coarray(:,:,:,:)[:,:,:]
    integer( kind=idef ),intent( in ) :: stype
    character( len=* ), intent( in ) :: fname
```

OUTPUTS

```
!     None
```

SIDE EFFECTS

A single binary file is created using hdf5 with contents of coarray.

DESCRIPTION

Parallel Stream Write Coarray of Integers (PSWC), number 4. Dump the coarray to file in a binary file in HDF5 format:

- coarray - what array to dump
- stype - what cell state type to dump
- fname - what file name to use

NOTES

All images must call this routine!

MPI must be initialised prior to calling this routine, most probably in the main program. Likewise MPI must be terminated only when no further MPI routines can be called. This will most likely be in the main program. There are some assumptions about the shape of the passed array.

The default integer is assumed for the array at present!

AUTHOR

Luis Cebamanos, adapted from the code written by David Henty, EPCC

COPYRIGHT

Note that this routine has special Copyright conditions.

```
    !-----------------------------------------------------------------------------!
    !                                                                             !
```

```
!  hdf5 routine for Fortran Coarrays                                          !
!                                                                             !
!  David Henty, EPCC; d.henty@epcc.ed.ac.uk                                   !
!                                                                             !
!  Copyright 2013 the University of Edinburgh                                 !
!                                                                             !
!  Licensed under the Apache LICENSE, Version 2.0 (the "LICENSE");            !
!  you may not use this file except in compliance with the LICENSE.           !
!  You may obtain a copy of the LICENSE at                                    !
!                                                                             !
!      http://www.apache.org/licenses/LICENSE-2.0                             !
!                                                                             !
!  Unless required by applicable law or agreed to in writing, software        !
!  distributed under the LICENSE is distributed on an "AS IS" BASIS,          !
!  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.   !
!  See the LICENSE for the specific language governing permissions and        !
!  limitations under the LICENSE.                                             !
!                                                                             !
!-----------------------------------------------------------------------------!
```

USES

cgca_m1co (9), MPI library, hdf5 library

USED BY

none, end user.

SOURCE

```
   integer, parameter :: totdim = 4, arrdim = totdim-1, coardim = 3

   integer :: img, nimgs, comm, ierr=0, rank=0, mpisize=0

   integer, dimension(totdim) :: asizehal
   integer, dimension(arrdim) :: arrsize, arstart, artsize
   integer, dimension(coardim) :: coarsize, copos

   integer :: info = MPI_INFO_NULL
   integer(hsize_t), dimension(coardim) :: dimsf  ! dataset dimensions.

   character(len=8), parameter :: dsetname = "IntArray" ! Dataset name

   integer(hid_t) :: file_id       ! file identifier
   integer(hid_t) :: dset_id       ! dataset identifier
   integer(hid_t) :: filespace     ! dataspace identifier in file
   integer(hid_t) :: memspace      ! dataspace identifier in memory
   integer(hid_t) :: plist_id      ! property list identifier

   integer(hsize_t), dimension(coardim) :: count
   integer(hssize_t), dimension(coardim) :: offset

!   integer :: ncid, varid, oldmode, dimids(coardim)
!   integer :: x_dimid, y_dimid, z_dimid
```

```
   img = this_image()
 nimgs = num_images()

   comm = MPI_COMM_WORLD
   call MPI_Comm_size( comm, mpisize, ierr )
   call MPI_Comm_rank( comm, rank, ierr )

   ! Sanity check
   if ( mpisize .ne. nimgs .or. rank .ne. img-1 ) then
      write(*,*) 'ERROR: cgca_m2hdf5/cgca_pswci4: MPI/coarray mismatch'
      error stop
   end if

   asizehal(:) = shape( coarray )
   copos(:) = this_image( coarray )

   ! Subtract halos
   arrsize(:) = asizehal(1:arrdim) - 2
   coarsize(:) = ucobound(coarray) - lcobound(coarray) + 1



   ! Does the array fit exactly?
   if ( product( coarsize ) .ne. nimgs) then
      write(*,*) 'ERROR: cgca_m2hdf5/cgca_pswci4: non-conforming coarray'
      error stop
   end if


   ! This is the global arrayy
   artsize(:) = arrsize(:) * coarsize(:)

   ! Correspondent portion of this global array
   arstart(:) = arrsize(:) * (copos(:)-1) + 1 ! Use Fortran indexing

   dimsf(:) = artsize(:)

   count(1) = arrsize(1)     ! Defines the number of values each proc dumps to
   count(2) = arrsize(2)
   count(3) = arrsize(3)                                  ! the HDF5 file.

! ! debug
! write (*,*) "hdf5 - image",img, "asizehal", asizehal, "copos", copos,         &
!  "arrsize", arrsize, "coarsize", coarsize,                              &
!  "artsize", artsize, "arstart", arstart, "stype", stype

  offset(:) = (copos(:)-1) * count(:)  ! Defines the offset used in the HDF5 file

  ! Initialize FORTRAN predefined datatypes
  CALL h5open_f(ierr)
```

```
! Setup file access property list with parallel I/O access.
CALL h5pcreate_f(H5P_FILE_ACCESS_F, plist_id, ierr)
CALL h5pset_fapl_mpio_f(plist_id, comm, info, ierr)
! debug
if (img .eq. 1) then
   write (*,*) "Setup file access property list"
end if

! Create the file collectively.
CALL h5fcreate_f(fname, H5F_ACC_TRUNC_F, file_id, ierr, &
    access_prp = plist_id)
if (ierr .ne. 0) then
   write(0,*) 'Unable to open: ', trim(fname), ': ', ierr
   call mpi_abort(MPI_COMM_WORLD, 1, ierr)
endif
if (img .eq. 1) then
   write (*,*) "Created a file collectively"
end if
CALL h5pclose_f(plist_id, ierr)
if (img .eq. 1) then
   write (*,*) "Close property list"
end if
! Create the data space for the  dataset.
CALL h5screate_simple_f(coardim, dimsf, filespace, ierr)
if (img .eq. 1) then
   write (*,*) "Create data space for the dataset"
end if
! Create the dataset with default properties.
CALL h5dcreate_f(file_id, dsetname, H5T_NATIVE_INTEGER, filespace, &
                    dset_id, ierr)
  if (img .eq. 1) then
     write(*,*) "Create dataset with default properties"
  end if
  CALL h5sclose_f(filespace, ierr)
  if (img .eq. 1) then
     write(*,*) "Close the dataset"
  end if

! Each process defines dataset in memory and writes it to the hyperslab
! in the file.
CALL h5screate_simple_f(coardim, count, memspace, ierr)
if (img .eq. 1) then
   write(*,*) "Each process defines dataset in mem"
end if
! Select hyperslab in the file.
CALL h5dget_space_f(dset_id, filespace, ierr)
CALL h5sselect_hyperslab_f (filespace, H5S_SELECT_SET_F, offset, &
    count, ierr)
if (img .eq. 1) then
   write(*,*) "Selects a hyperslab in the file"
end if
```

```
  ! Create property list for collective dataset write
  CALL h5pcreate_f(H5P_DATASET_XFER_F, plist_id, ierr)
  CALL h5pset_dxpl_mpio_f(plist_id, H5FD_MPIO_COLLECTIVE_F, ierr)
  if (img .eq. 1) then
     write(*,*) "Crete property list for coll dataset"
  end if

  ! Write the dataset collectively.
  CALL h5dwrite_f(dset_id, H5T_NATIVE_INTEGER, coarray(1:arrsize(1),    &
       1:arrsize(2), 1:arrsize(3), stype), dimsf, ierr,             &
       file_space_id = filespace, mem_space_id = memspace,          &
       xfer_prp = plist_id)
  if (img .eq. 1) then
     write(*,*) "Write dataset colectively"
  end if

  ! Close dataspaces.
  CALL h5sclose_f(filespace, ierr)
  CALL h5sclose_f(memspace, ierr)
  if (img .eq. 1) then
     write(*,*) "Close stuff"
  end if
  ! Close the dataset and property list.
  CALL h5dclose_f(dset_id, ierr)
  CALL h5pclose_f(plist_id, ierr)

  ! Close the file.
  CALL h5fclose_f(file_id, ierr)

  ! Close FORTRAN predefined datatypes.
  CALL h5close_f(ierr)


end subroutine cgca_pswci4
```

15 CGPACK/cgca_m2hx

[Modules]

NAME

cgca_m2hx

SYNOPSIS

```
!$Id: cgca_m2hx.f90 431 2017-06-30 13:13:49Z mexas $

module cgca_m2hx
```

DESCRIPTION

Module dealing with halo exchange

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

CONTAINS

cgca_hxi (15.2), cgca_hxg (15.1)

USES

cgca_m1co (9)

USED BY

cgca_m3clvg (26) cgca_m3sld (31)

SOURCE

```
use cgca_m1co
implicit none

private
public :: cgca_hxi,  &
          cgca_hxir, & ! In submodule m2hx_hxir
          cgca_hxic, & ! In submodule m2hx_hxic
          cgca_hxg

interface
  ! In submodule m2hx_hxir
  module subroutine cgca_hxir( coarray )
    ! State of coarray will change!
    integer( kind=iarr ), allocatable, intent( inout ) ::              &
       coarray(:,:,:,:)[:,:,:]
  end subroutine cgca_hxir

  ! In submodule m2hx_hxic
  module subroutine cgca_hxic( coarray, flag )
```

```
    ! State of coarray does not change!
    integer( kind=iarr ), allocatable, intent( inout ) ::              &
       coarray(:,:,:,:)[:,:,:]
    ! flag is 0 if and only if hx was done correctly, i.e. if
    ! the state of all real cells on the boundaries is consistent with
    ! the states of the corresponding halo cells.
    integer, intent( out ) :: flag
  end subroutine cgca_hxic
end interface

contains
```

15.1 cgca_m2hx/cgca_hxg

[cgca_m2hx] [Subroutines]

NAME

cgca_hxg

SYNOPSIS

```
subroutine cgca_hxg(coarray)
```

INPUT

```
integer(kind=iarr),allocatable,intent(inout) :: coarray(:,:,:,:)[:,:,:]
```

SIDE EFFECTS

coarray is changed

DESCRIPTION

This routine does the global halo exchange, i.e. on the boundaries of the whole model. In other wordsl it imposes self-similar boundary conditions on the global (super) array. The routine exchanges halos on *all* cell state types. This is an overkill, as it is likely that only one cell state type needs to be halo exchanged at a time. However, it makes for an easier code, and there is virtually no performance penaly, so we do it this way.

NOTES

All images must call this routine!

USES

none

USED BY

module cgca_m3clvg (26): cgca_clvgp (26.17.1)

SOURCE

```
integer(kind=idef) :: &
  lbv(4)      , & ! lower bounds of the "virtual" coarray
  ubv(4)      , & ! upper bounds of the "virtual" coarray
  lbr(4)      , & ! lower bounds of the "real" coarray, lbv+1
  ubr(4)      , & ! upper bounds of the "real" coarray, ubv-1
  lcob(3)     , & ! lower cobounds of the coarray
  ucob(3)     , & ! upper cobounds of the coarray
  imgpos(3)   , & ! position of the image in a coarray grid
  imgpos1mns1 , & ! positions of the neighbouring images
  imgpos1pls1 , & ! along 3 directions
  imgpos2mns1 , & !
  imgpos2pls1 , & !
  imgpos3mns1 , & !
  imgpos3pls1 , & !
  z1,z2,z3        ! temp vars to store coarray grid coordinates
```

```
! check for allocated
if (.not. allocated(coarray)) &
  error stop "ERROR: cgca_hlg: coarray is not allocated"

lbv=lbound(coarray)
ubv=ubound(coarray)
lbr=lbv+1
ubr=ubv-1
lcob=lcobound(coarray)
ucob=ucobound(coarray)

imgpos=this_image(coarray)
imgpos1mns1=imgpos(1)-1
imgpos1pls1=imgpos(1)+1
imgpos2mns1=imgpos(2)-1
imgpos2pls1=imgpos(2)+1
imgpos3mns1=imgpos(3)-1
imgpos3pls1=imgpos(3)+1

! Make sure only the virtual (halo) arrays are assigned to.
! The real array values must never appear on the left
! hand side of the assignment expressions.
! The halo exchange process is copying real array values
! into halos. There must not ever be copying of real values
! to real, or halo to halo, or halo to real.

!*************************************
! 2D halos
!*************************************

! exchange 2D halos in direction 1

if (imgpos(1) .eq. lcob(1))                          &
  coarray(lbv(1),lbr(2):ubr(2),lbr(3):ubr(3),:) =    &
  coarray(ubr(1),lbr(2):ubr(2),lbr(3):ubr(3),:)      &
    [ucob(1),imgpos(2),imgpos(3)]

if (imgpos(1) .eq. ucob(1))                          &
  coarray(ubv(1),lbr(2):ubr(2),lbr(3):ubr(3),:) =    &
  coarray(lbr(1),lbr(2):ubr(2),lbr(3):ubr(3),:)      &
    [lcob(1),imgpos(2),imgpos(3)]

! exchange 2D halos in direction 2

if (imgpos(2) .eq. lcob(2))                          &
  coarray(lbr(1):ubr(1),lbv(2),lbr(3):ubr(3),:) =    &
  coarray(lbr(1):ubr(1),ubr(2),lbr(3):ubr(3),:)      &
    [imgpos(1),ucob(2),imgpos(3)]

if (imgpos(2) .eq. ucob(2))                          &
  coarray(lbr(1):ubr(1),ubv(2),lbr(3):ubr(3),:) =    &
  coarray(lbr(1):ubr(1),lbr(2),lbr(3):ubr(3),:)      &
```

```
    [imgpos(1),lcob(2),imgpos(3)]

! exchange 2D halos in direction 3

if (imgpos(3) .eq. lcob(3))                          &
  coarray(lbr(1):ubr(1),lbr(2):ubr(2),lbv(3),:) =    &
  coarray(lbr(1):ubr(1),lbr(2):ubr(2),ubr(3),:)      &
    [imgpos(1),imgpos(2),ucob(3)]

if (imgpos(3) .eq. ucob(3))                          &
  coarray(lbr(1):ubr(1),lbr(2):ubr(2),ubv(3),:) =    &
  coarray(lbr(1):ubr(1),lbr(2):ubr(2),lbr(3),:)      &
    [imgpos(1),imgpos(2),lcob(3)]

!************************************
! 1D halos
!************************************

! exchange 1D halos parallel to direction 3
! the 4 edges of the super array

! operation 1
if (imgpos(1) .eq. lcob(1) .and. imgpos(2) .eq. lcob(2))               &
  coarray(lbv(1),lbv(2),lbr(3):ubr(3),:) =                             &
  coarray(ubr(1),ubr(2),lbr(3):ubr(3),:) [ucob(1),ucob(2),imgpos(3)]

! operation 2
if (imgpos(1) .eq. ucob(1) .and. imgpos(2) .eq. lcob(2))               &
  coarray(ubv(1),lbv(2),lbr(3):ubr(3),:) =                             &
  coarray(lbr(1),ubr(2),lbr(3):ubr(3),:) [lcob(1),ucob(2),imgpos(3)]

! operation 3
if (imgpos(1) .eq. ucob(1) .and. imgpos(2) .eq. ucob(2))               &
  coarray(ubv(1),ubv(2),lbr(3):ubr(3),:) =                             &
  coarray(lbr(1),lbr(2),lbr(3):ubr(3),:) [lcob(1),lcob(2),imgpos(3)]

! operation 4
if (imgpos(1) .eq. lcob(1) .and. imgpos(2) .eq. ucob(2))               &
  coarray(lbv(1),ubv(2),lbr(3):ubr(3),:) =                             &
  coarray(ubr(1),lbr(2),lbr(3):ubr(3),:) [ucob(1),lcob(2),imgpos(3)]

! exchange 1D halos parallel to direction 3
! all intermediate edges, self-similarity along 1

! operation 5
if (imgpos(1) .eq. lcob(1) .and. imgpos(2) .ne. ucob(2))               &
  coarray(lbv(1),ubv(2),lbr(3):ubr(3),:) =                             &
  coarray(ubr(1),lbr(2),lbr(3):ubr(3),:) [ucob(1),imgpos2pls1,imgpos(3)]

! operation 6
if (imgpos(1) .eq. lcob(1) .and. imgpos(2) .ne. lcob(2))               &
  coarray(lbv(1),lbv(2),lbr(3):ubr(3),:) =                             &
```

```
  coarray(ubr(1),ubr(2),lbr(3):ubr(3),:) [ucob(1),imgpos2mns1,imgpos(3)]

! operation 7
if (imgpos(1) .eq. ucob(1) .and. imgpos(2) .ne. ucob(2))               &
  coarray(ubv(1),ubv(2),lbr(3):ubr(3),:) =                             &
  coarray(lbr(1),lbr(2),lbr(3):ubr(3),:) [lcob(1),imgpos2pls1,imgpos(3)]

! operation 8
if (imgpos(1) .eq. ucob(1) .and. imgpos(2) .ne. lcob(2))               &
  coarray(ubv(1),lbv(2),lbr(3):ubr(3),:) =                             &
  coarray(lbr(1),ubr(2),lbr(3):ubr(3),:) [lcob(1),imgpos2mns1,imgpos(3)]

! exchange 1D halos parallel to direction 3
! all intermediate edges, self-similarity along 2

! operation 9
if (imgpos(1) .ne. ucob(1) .and. imgpos(2) .eq. lcob(2))               &
  coarray(ubv(1),lbv(2),lbr(3):ubr(3),:) =                             &
  coarray(lbr(1),ubr(2),lbr(3):ubr(3),:) [imgpos1pls1,ucob(2),imgpos(3)]

! operation 10
if (imgpos(1) .ne. lcob(1) .and. imgpos(2) .eq. lcob(2))               &
  coarray(lbv(1),lbv(2),lbr(3):ubr(3),:) =                             &
  coarray(ubr(1),ubr(2),lbr(3):ubr(3),:) [imgpos1mns1,ucob(2),imgpos(3)]

! operation 11
if (imgpos(1) .ne. ucob(1) .and. imgpos(2) .eq. ucob(2))               &
  coarray(ubv(1),ubv(2),lbr(3):ubr(3),:) =                             &
  coarray(lbr(1),lbr(2),lbr(3):ubr(3),:) [imgpos1pls1,lcob(2),imgpos(3)]

! operation 12
if (imgpos(1) .ne. lcob(1) .and. imgpos(2) .eq. ucob(2))               &
  coarray(lbv(1),ubv(2),lbr(3):ubr(3),:) =                             &
  coarray(ubr(1),lbr(2),lbr(3):ubr(3),:) [imgpos1mns1,lcob(2),imgpos(3)]

! exchange 1D halos parallel to direction 1
! the 4 edges of the super array

! operation 1
if (imgpos(2) .eq. lcob(2) .and. imgpos(3) .eq. lcob(3))               &
  coarray(lbr(1):ubr(1),lbv(2),lbv(3),:) =                             &
  coarray(lbr(1):ubr(1),ubr(2),ubr(3),:) [imgpos(1),ucob(2),ucob(3)]

! operation 2
if (imgpos(2) .eq. ucob(2) .and. imgpos(3) .eq. lcob(3))               &
  coarray(lbr(1):ubr(1),ubv(2),lbv(3),:) =                             &
  coarray(lbr(1):ubr(1),lbr(2),ubr(3),:) [imgpos(1),lcob(2),ucob(3)]

! operation 3
if (imgpos(2) .eq. ucob(2) .and. imgpos(3) .eq. ucob(3))               &
  coarray(lbr(1):ubr(1),ubv(2),ubv(3),:) =                             &
  coarray(lbr(1):ubr(1),lbr(2),lbr(3),:) [imgpos(1),lcob(2),lcob(3)]
```

```
! operation 4
if (imgpos(2) .eq. lcob(2) .and. imgpos(3) .eq. ucob(3))               &
  coarray(lbr(1):ubr(1),lbv(2),ubv(3),:) =                             &
  coarray(lbr(1):ubr(1),ubr(2),lbr(3),:) [imgpos(1),ucob(2),lcob(3)]

! exchange 1D halos parallel to direction 1
! intermediate edges, self-similarity along 2

! operation 5
if (imgpos(2) .eq. lcob(2) .and. imgpos(3) .ne. ucob(3))               &
  coarray(lbr(1):ubr(1),lbv(2),ubv(3),:) =                             &
  coarray(lbr(1):ubr(1),ubr(2),lbr(3),:) [imgpos(1),ucob(2),imgpos3pls1]

! operation 6
if (imgpos(2) .eq. lcob(2) .and. imgpos(3) .ne. lcob(3))               &
  coarray(lbr(1):ubr(1),lbv(2),lbv(3),:) =                             &
  coarray(lbr(1):ubr(1),ubr(2),ubr(3),:) [imgpos(1),ucob(2),imgpos3mns1]

! operation 7
if (imgpos(2) .eq. ucob(2) .and. imgpos(3) .ne. ucob(3))               &
  coarray(lbr(1):ubr(1),ubv(2),ubv(3),:) =                             &
  coarray(lbr(1):ubr(1),lbr(2),lbr(3),:) [imgpos(1),lcob(2),imgpos3pls1]

! operation 8
if (imgpos(2) .eq. ucob(2) .and. imgpos(3) .ne. lcob(3))               &
  coarray(lbr(1):ubr(1),ubv(2),lbv(3),:) =                             &
  coarray(lbr(1):ubr(1),lbr(2),ubr(3),:) [imgpos(1),lcob(2),imgpos3mns1]

! exchange 1D halos parallel to direction 1
! intermediate edges, self-similarity along 3

! operation 9
if (imgpos(2) .ne. ucob(2) .and. imgpos(3) .eq. lcob(3))               &
  coarray(lbr(1):ubr(1),ubv(2),lbv(3),:) =                             &
  coarray(lbr(1):ubr(1),lbr(2),ubr(3),:) [imgpos(1),imgpos2pls1,ucob(3)]

! operation 10
if (imgpos(2) .ne. lcob(2) .and. imgpos(3) .eq. lcob(3))               &
  coarray(lbr(1):ubr(1),lbv(2),lbv(3),:) =                             &
  coarray(lbr(1):ubr(1),ubr(2),ubr(3),:) [imgpos(1),imgpos2mns1,ucob(3)]

! operation 11
if (imgpos(2) .ne. ucob(2) .and. imgpos(3) .eq. ucob(3))               &
  coarray(lbr(1):ubr(1),ubv(2),ubv(3),:) =                             &
  coarray(lbr(1):ubr(1),lbr(2),lbr(3),:) [imgpos(1),imgpos2pls1,lcob(3)]

! operation 12
if (imgpos(2) .ne. lcob(2) .and. imgpos(3) .eq. ucob(3))               &
  coarray(lbr(1):ubr(1),lbv(2),ubv(3),:) =                             &
  coarray(lbr(1):ubr(1),ubr(2),lbr(3),:) [imgpos(1),imgpos2mns1,lcob(3)]
```

```
! exchange 1D halos parallel to direction 2
! the 4 edges of the super array

! operation 1
if (imgpos(1) .eq. lcob(1) .and. imgpos(3) .eq. lcob(3))               &
  coarray(lbv(1),lbr(2):ubr(2),lbv(3),:) =                             &
  coarray(ubr(1),lbr(2):ubr(2),ubr(3),:) [ucob(1),imgpos(2),ucob(3)]

! operation 2
if (imgpos(1) .eq. lcob(1) .and. imgpos(3) .eq. ucob(3))               &
  coarray(lbv(1),lbr(2):ubr(2),ubv(3),:) =                             &
  coarray(ubr(1),lbr(2):ubr(2),lbr(3),:) [ucob(1),imgpos(2),lcob(3)]

! operation 3
if (imgpos(1) .eq. ucob(1) .and. imgpos(3) .eq. ucob(3))               &
  coarray(ubv(1),lbr(2):ubr(2),ubv(3),:) =                             &
  coarray(lbr(1),lbr(2):ubr(2),lbr(3),:) [lcob(1),imgpos(2),lcob(3)]

! operation 4
if (imgpos(1) .eq. ucob(1) .and. imgpos(3) .eq. lcob(3))               &
  coarray(ubv(1),lbr(2):ubr(2),lbv(3),:) =                             &
  coarray(lbr(1),lbr(2):ubr(2),ubr(3),:) [lcob(1),imgpos(2),ucob(3)]

! exchange 1D halos parallel to direction 2
! intermediate edges, self-similarity along 3

! operation 5
if (imgpos(1) .ne. ucob(1) .and. imgpos(3) .eq. lcob(3))               &
  coarray(ubv(1),lbr(2):ubr(2),lbv(3),:) =                             &
  coarray(lbr(1),lbr(2):ubr(2),ubr(3),:) [imgpos1pls1,imgpos(2),ucob(3)]

! operation 6
if (imgpos(1) .ne. lcob(1) .and. imgpos(3) .eq. lcob(3))               &
  coarray(lbv(1),lbr(2):ubr(2),lbv(3),:) =                             &
  coarray(ubr(1),lbr(2):ubr(2),ubr(3),:) [imgpos1mns1,imgpos(2),ucob(3)]

! operation 7
if (imgpos(1) .ne. ucob(1) .and. imgpos(3) .eq. ucob(3))               &
  coarray(ubv(1),lbr(2):ubr(2),ubv(3),:) =                             &
  coarray(lbr(1),lbr(2):ubr(2),lbr(3),:) [imgpos1pls1,imgpos(2),lcob(3)]

! operation 8
if (imgpos(1) .ne. lcob(1) .and. imgpos(3) .eq. ucob(3))               &
  coarray(lbv(1),lbr(2):ubr(2),ubv(3),:) =                             &
  coarray(ubr(1),lbr(2):ubr(2),lbr(3),:) [imgpos1mns1,imgpos(2),lcob(3)]

! exchange 1D halos parallel to direction 2
! intermediate edges, self-similarity along 1

! operation 9
if (imgpos(1) .eq. lcob(1) .and. imgpos(3) .ne. ucob(3))               &
  coarray(lbv(1),lbr(2):ubr(2),ubv(3),:) =                             &
```

```
  coarray(ubr(1),lbr(2):ubr(2),lbr(3),:) [ucob(1),imgpos(2),imgpos3pls1]

! operation 10
if (imgpos(1) .eq. lcob(1) .and. imgpos(3) .ne. lcob(3))               &
  coarray(lbv(1),lbr(2):ubr(2),lbv(3),:) =                             &
  coarray(ubr(1),lbr(2):ubr(2),ubr(3),:) [ucob(1),imgpos(2),imgpos3mns1]

! operation 11
if (imgpos(1) .eq. ucob(1) .and. imgpos(3) .ne. ucob(3))               &
  coarray(ubv(1),lbr(2):ubr(2),ubv(3),:) =                             &
  coarray(lbr(1),lbr(2):ubr(2),lbr(3),:) [lcob(1),imgpos(2),imgpos3pls1]

! operation 12
if (imgpos(1) .eq. ucob(1) .and. imgpos(3) .ne. lcob(3))               &
  coarray(ubv(1),lbr(2):ubr(2),lbv(3),:) =                             &
  coarray(lbr(1),lbr(2):ubr(2),ubr(3),:) [lcob(1),imgpos(2),imgpos3mns1]

!************************************
! corner halos
!************************************

! corner 1
if (imgpos(1) .eq. lcob(1) .or.             &
    imgpos(2) .eq. lcob(2) .or.             &
    imgpos(3) .eq. lcob(3))                 &
then
  z1 = imgpos1mns1
  z2 = imgpos2mns1
  z3 = imgpos3mns1
  if (z1 .lt. lcob(1)) z1 = ucob(1)
  if (z2 .lt. lcob(2)) z2 = ucob(2)
  if (z3 .lt. lcob(3)) z3 = ucob(3)
  coarray(lbv(1),lbv(2),lbv(3),:) =         &
  coarray(ubr(1),ubr(2),ubr(3),:) [z1,z2,z3]
end if

! corner 2
if (imgpos(1) .eq. ucob(1) .or.             &
    imgpos(2) .eq. lcob(2) .or.             &
    imgpos(3) .eq. lcob(3))                 &
then
  z1 = imgpos1pls1
  z2 = imgpos2mns1
  z3 = imgpos3mns1
  if (z1 .gt. ucob(1)) z1 = lcob(1)
  if (z2 .lt. lcob(2)) z2 = ucob(2)
  if (z3 .lt. lcob(3)) z3 = ucob(3)
  coarray(ubv(1),lbv(2),lbv(3),:) =         &
  coarray(lbr(1),ubr(2),ubr(3),:) [z1,z2,z3]
end if

! corner 3
```

```
if (imgpos(1) .eq. lcob(1) .or.             &
    imgpos(2) .eq. ucob(2) .or.             &
    imgpos(3) .eq. lcob(3))                 &
then
  z1 = imgpos1mns1
  z2 = imgpos2pls1
  z3 = imgpos3mns1
  if (z1 .lt. lcob(1)) z1 = ucob(1)
  if (z2 .gt. ucob(2)) z2 = lcob(2)
  if (z3 .lt. lcob(3)) z3 = ucob(3)
  coarray(lbv(1),ubv(2),lbv(3),:) =         &
  coarray(ubr(1),lbr(2),ubr(3),:) [z1,z2,z3]
end if

! corner 4
if (imgpos(1) .eq. ucob(1) .or.             &
    imgpos(2) .eq. ucob(2) .or.             &
    imgpos(3) .eq. lcob(3))                 &
then
  z1 = imgpos1pls1
  z2 = imgpos2pls1
  z3 = imgpos3mns1
  if (z1 .gt. ucob(1)) z1 = lcob(1)
  if (z2 .gt. ucob(2)) z2 = lcob(2)
  if (z3 .lt. lcob(3)) z3 = ucob(3)
  coarray(ubv(1),ubv(2),lbv(3),:) =         &
  coarray(lbr(1),lbr(2),ubr(3),:) [z1,z2,z3]
end if

! corner 5
if (imgpos(1) .eq. lcob(1) .or.             &
    imgpos(2) .eq. lcob(2) .or.             &
    imgpos(3) .eq. ucob(3))                 &
then
  z1 = imgpos1mns1
  z2 = imgpos2mns1
  z3 = imgpos3pls1
  if (z1 .lt. lcob(1)) z1 = ucob(1)
  if (z2 .lt. lcob(2)) z2 = ucob(2)
  if (z3 .gt. ucob(3)) z3 = lcob(3)
  coarray(lbv(1),lbv(2),ubv(3),:) =         &
  coarray(ubr(1),ubr(2),lbr(3),:) [z1,z2,z3]
end if

! corner 6
if (imgpos(1) .eq. ucob(1) .or.             &
    imgpos(2) .eq. lcob(2) .or.             &
    imgpos(3) .eq. ucob(3))                 &
then
  z1 = imgpos1pls1
  z2 = imgpos2mns1
  z3 = imgpos3pls1
```

```
  if (z1 .gt. ucob(1)) z1 = lcob(1)
  if (z2 .lt. lcob(2)) z2 = ucob(2)
  if (z3 .gt. ucob(3)) z3 = lcob(3)
  coarray(ubv(1),lbv(2),ubv(3),:) =         &
  coarray(lbr(1),ubr(2),lbr(3),:) [z1,z2,z3]
end if

! corner 7
if (imgpos(1) .eq. lcob(1) .or.             &
    imgpos(2) .eq. ucob(2) .or.             &
    imgpos(3) .eq. ucob(3))                 &
then
  z1 = imgpos1mns1
  z2 = imgpos2pls1
  z3 = imgpos3pls1
  if (z1 .lt. lcob(1)) z1 = ucob(1)
  if (z2 .gt. ucob(2)) z2 = lcob(2)
  if (z3 .gt. ucob(3)) z3 = lcob(3)
  coarray(lbv(1),ubv(2),ubv(3),:) =         &
  coarray(ubr(1),lbr(2),lbr(3),:) [z1,z2,z3]
end if

! corner 8
if (imgpos(1) .eq. ucob(1) .or.             &
    imgpos(2) .eq. ucob(2) .or.             &
    imgpos(3) .eq. ucob(3))                 &
then
  z1 = imgpos1pls1
  z2 = imgpos2pls1
  z3 = imgpos3pls1
  if (z1 .gt. ucob(1)) z1 = lcob(1)
  if (z2 .gt. ucob(2)) z2 = lcob(2)
  if (z3 .gt. ucob(3)) z3 = lcob(3)
  coarray(ubv(1),ubv(2),ubv(3),:) =         &
  coarray(lbr(1),lbr(2),lbr(3),:) [z1,z2,z3]
end if

end subroutine cgca_hxg
```

15.2 cgca_m2hx/cgca_hxi

[cgca_m2hx] [Subroutines]

NAME

cgca_hxi

SYNOPSIS

```
subroutine cgca_hxi( coarray )
```

INPUT

```
integer( kind=iarr ), allocatable, intent( inout ) ::                  &
 coarray(:,:,:,:)[:,:,:]
```

SIDE EFFECTS

coarray is changed

DESCRIPTION

This routine does internal halo exchange. The routine exchanges halos on *all* cell state types. This is an overkill, as it is likely that only one cell state type needs to be halo exchanged at a time. However, it makes for an easier code, and there is virtually no performance penalty, so we do it this way.

NOTES

All images must call this routine!

USES

parameters from cgca_m1co (9)

USED BY

module cgca_m3clvg (26): cgca_clgvp

SOURCE

```
integer ::                                                             &
  lbv(4)      , & ! lower bounds of the "virtual" coarray
  ubv(4)      , & ! upper bounds of the "virtual" coarray
  lbr(4)      , & ! lower bounds of the "real" coarray, lbv+1
  ubr(4)      , & ! upper bounds of the "real" coarray, ubv-1
  lcob(3)     , & ! lower cobounds of the coarray
  ucob(3)     , & ! upper cobounds of the coarray
  imgpos(3)       ! position of the image in a coarray grid

! check for allocated
if ( .not. allocated( coarray ) ) then
  write (*,'(a)') "ERROR: cgca_hxi/cgca_m2hx: coarray is not allocated"
  error stop
end if

   lbv = lbound( coarray )
   ubv = ubound( coarray )
```

```
   lbr = lbv + 1
   ubr = ubv - 1
  lcob = lcobound( coarray )
  ucob = ucobound( coarray )
imgpos = this_image( coarray )

! Make sure only the virtual (halo) arrays are assigned to.
! The real array values must never appear on the left
! hand side of the assignment expressions.
! The halo exchange process is copying real array values into halos.
! There must not ever be copying real values
! to real, or halo to halo, or halo to real.
! Also, only local array must appear on the left.
! We are assigning values to the local array's virtual
! (halo) cells using values from real cells from arrays
! in other images.

! exchange 2D halos in direction 1

! op 1
if ( imgpos(1) .ne. lcob(1) )                                          &
  coarray( lbv(1) , lbr(2) : ubr(2) , lbr(3) : ubr(3) , : ) =          &
  coarray( ubr(1) , lbr(2) : ubr(2) , lbr(3) : ubr(3) , : )            &
    [ imgpos(1)-1 , imgpos(2) , imgpos(3) ]

! op 2
if ( imgpos(1) .ne. ucob(1) )                                          &
  coarray( ubv(1) , lbr(2) : ubr(2) , lbr(3) : ubr(3) , : ) =          &
  coarray( lbr(1) , lbr(2) : ubr(2) , lbr(3) : ubr(3) , : )            &
    [ imgpos(1)+1 , imgpos(2) , imgpos(3) ]

! exchange 2D halos in direction 2

! op 3
if ( imgpos(2) .ne. lcob(2) )                                          &
  coarray( lbr(1) : ubr(1) , lbv(2) , lbr(3) : ubr(3) , : ) =          &
  coarray( lbr(1) : ubr(1) , ubr(2) , lbr(3) : ubr(3) , : )            &
    [ imgpos(1) , imgpos(2)-1 , imgpos(3) ]

! op 4
if ( imgpos(2) .ne. ucob(2) )                                          &
  coarray( lbr(1) : ubr(1) , ubv(2) , lbr(3) : ubr(3) , : ) =          &
  coarray( lbr(1) : ubr(1) , lbr(2) , lbr(3) : ubr(3) , : )            &
    [ imgpos(1) , imgpos(2)+1 , imgpos(3) ]

! exchange 2D halos in direction 3

! op 5
if ( imgpos(3) .ne. lcob(3) )                                          &
  coarray( lbr(1) : ubr(1) , lbr(2) : ubr(2) , lbv(3) , : ) =          &
  coarray( lbr(1) : ubr(1) , lbr(2) : ubr(2) , ubr(3) , : )            &
    [ imgpos(1) , imgpos(2) , imgpos(3)-1 ]
```

```
! op 6
if ( imgpos(3) .ne. ucob(3) )                                          &
  coarray( lbr(1) : ubr(1) , lbr(2) : ubr(2) , ubv(3) , : ) =          &
  coarray( lbr(1) : ubr(1) , lbr(2) : ubr(2) , lbr(3) , : )            &
    [ imgpos(1) , imgpos(2) , imgpos(3)+1 ]

! exchange 1D halos parallel to direction 3

! op 7
if ( imgpos(1) .ne. lcob(1) .and. imgpos(2) .ne. lcob(2) )             &
  coarray( lbv(1) , lbv(2) , lbr(3) : ubr(3) , : ) =                   &
  coarray( ubr(1) , ubr(2) , lbr(3) : ubr(3) , : )                     &
    [ imgpos(1)-1 , imgpos(2)-1 , imgpos(3) ]

! op 8
if ( imgpos(1) .ne. ucob(1) .and. imgpos(2) .ne. lcob(2) )             &
  coarray( ubv(1) , lbv(2) , lbr(3) : ubr(3) , :) =                    &
  coarray( lbr(1) , ubr(2) , lbr(3) : ubr(3) , :)                      &
    [ imgpos(1)+1 , imgpos(2)-1 , imgpos(3) ]

! op 9
if ( imgpos(1) .ne. ucob(1) .and. imgpos(2) .ne. ucob(2) )             &
  coarray( ubv(1) , ubv(2) , lbr(3) : ubr(3) , : ) =                   &
  coarray( lbr(1) , lbr(2) , lbr(3) : ubr(3) , : )                     &
    [ imgpos(1)+1 , imgpos(2)+1 , imgpos(3) ]

! op 10
if ( imgpos(1) .ne. lcob(1) .and. imgpos(2) .ne. ucob(2) )             &
  coarray( lbv(1) , ubv(2) , lbr(3) : ubr(3) , : ) =                   &
  coarray( ubr(1) , lbr(2) , lbr(3) : ubr(3) , : )                     &
    [ imgpos(1)-1 , imgpos(2)+1 , imgpos(3) ]

! exchange 1D halos parallel to direction 1

! op 11
if ( imgpos(2) .ne. lcob(2) .and. imgpos(3) .ne. lcob(3) )             &
  coarray( lbr(1) : ubr(1) , lbv(2) , lbv(3) , : ) =                   &
  coarray( lbr(1) : ubr(1) , ubr(2) , ubr(3) , : )                     &
    [ imgpos(1) , imgpos(2)-1 , imgpos(3)-1 ]

! op 12
if ( imgpos(2) .ne. lcob(2) .and. imgpos(3) .ne. ucob(3) )             &
  coarray( lbr(1) : ubr(1) , lbv(2) , ubv(3) , : ) =                   &
  coarray( lbr(1) : ubr(1) , ubr(2) , lbr(3) , : )                     &
    [ imgpos(1) , imgpos(2)-1 , imgpos(3)+1 ]

! op 13
if ( imgpos(2) .ne. ucob(2) .and. imgpos(3) .ne. ucob(3) )             &
  coarray( lbr(1) : ubr(1) , ubv(2) , ubv(3) , : ) =                   &
  coarray( lbr(1) : ubr(1) , lbr(2) , lbr(3) , : )                     &
    [ imgpos(1) , imgpos(2)+1 , imgpos(3)+1 ]
```

```
! op 14
if ( imgpos(2) .ne. ucob(2) .and. imgpos(3) .ne. lcob(3) )             &
  coarray( lbr(1) : ubr(1) , ubv(2) , lbv(3) , : ) =                   &
  coarray( lbr(1) : ubr(1) , lbr(2) , ubr(3) , : )                     &
    [ imgpos(1) , imgpos(2)+1 , imgpos(3)-1 ]

! exchange 1D halos parallel to direction 2

! op 15
if ( imgpos(1) .ne. lcob(1) .and. imgpos(3) .ne. lcob(3) )             &
  coarray( lbv(1) , lbr(2) : ubr(2) , lbv(3) , : ) =                   &
  coarray( ubr(1) , lbr(2) : ubr(2) , ubr(3) , : )                     &
    [ imgpos(1)-1 , imgpos(2) , imgpos(3)-1 ]

! op 16
if ( imgpos(1) .ne. ucob(1) .and. imgpos(3) .ne. lcob(3) )             &
  coarray( ubv(1) , lbr(2) : ubr(2) , lbv(3) , : ) =                   &
  coarray( lbr(1) , lbr(2) : ubr(2) , ubr(3) , : )                     &
    [ imgpos(1)+1 , imgpos(2) , imgpos(3)-1 ]

! op 17
if ( imgpos(1) .ne. ucob(1) .and. imgpos(3) .ne. ucob(3) )             &
  coarray( ubv(1) , lbr(2) : ubr(2) , ubv(3) , : ) =                   &
  coarray( lbr(1) , lbr(2) : ubr(2) , lbr(3) , : )                     &
    [ imgpos(1)+1 , imgpos(2) , imgpos(3)+1 ]

! op 18
if ( imgpos(1) .ne. lcob(1) .and. imgpos(3) .ne. ucob(3) )             &
  coarray( lbv(1) , lbr(2) : ubr(2) , ubv(3) , : ) =                   &
  coarray( ubr(1) , lbr(2) : ubr(2) , lbr(3) , : )                     &
    [ imgpos(1)-1 , imgpos(2) , imgpos(3)+1 ]

! Exchange 8 scalar halos
! See diagram cgca1 in the manual.

! op 19
if ( (imgpos(1) .ne. lcob(1)) .and. (imgpos(2) .ne. lcob(2)) .and.     &
     (imgpos(3) .ne. lcob(3)) )                                        &
  coarray( lbv(1) , lbv(2) , lbv(3) , : ) =                            &
  coarray( ubr(1) , ubr(2) , ubr(3) , : )                              &
                  [ imgpos(1)-1 , imgpos(2)-1 , imgpos(3)-1 ]

! op 20
if ( (imgpos(1) .ne. ucob(1)) .and. (imgpos(2) .ne. lcob(2)) .and.     &
     (imgpos(3) .ne. lcob(3)) )                                        &
  coarray( ubv(1) , lbv(2) , lbv(3) , : ) =                            &
  coarray( lbr(1) , ubr(2) , ubr(3) , : )                              &
                  [ imgpos(1)+1 , imgpos(2)-1 , imgpos(3)-1 ]

! op 21
if ( (imgpos(1) .ne. ucob(1)) .and. (imgpos(2) .ne. ucob(2)) .and.     &
```

```
     (imgpos(3) .ne. lcob(3)) )                                          &
  coarray( ubv(1) , ubv(2) , lbv(3) , : ) =                              &
  coarray( lbr(1) , lbr(2) , ubr(3) , : )                                &
                  [ imgpos(1)+1 , imgpos(2)+1 , imgpos(3)-1 ]

! op 22
if ( (imgpos(1) .ne. lcob(1)) .and. (imgpos(2) .ne. ucob(2)) .and.      &
     (imgpos(3) .ne. lcob(3)) )                                         &
  coarray( lbv(1) , ubv(2) , lbv(3) , : ) =                             &
  coarray( ubr(1) , lbr(2) , ubr(3) , : )                               &
                  [ imgpos(1)-1 , imgpos(2)+1 , imgpos(3)-1 ]

! op 23
if ( (imgpos(1) .ne. lcob(1)) .and. (imgpos(2) .ne. lcob(2)) .and.      &
     (imgpos(3) .ne. ucob(3)) )                                         &
  coarray( lbv(1) , lbv(2) , ubv(3) , : ) =                             &
  coarray( ubr(1) , ubr(2) , lbr(3) , : )                               &
                  [ imgpos(1)-1 , imgpos(2)-1 , imgpos(3)+1 ]

! op 24
if ( (imgpos(1) .ne. ucob(1)) .and. (imgpos(2) .ne. lcob(2)) .and.      &
     (imgpos(3) .ne. ucob(3)) )                                         &
  coarray( ubv(1) , lbv(2) , ubv(3) , : ) =                             &
  coarray( lbr(1) , ubr(2) , lbr(3) , : )                               &
                  [ imgpos(1)+1 , imgpos(2)-1 , imgpos(3)+1 ]

! op 25
if ( (imgpos(1) .ne. ucob(1)) .and. (imgpos(2) .ne. ucob(2)) .and.      &
     (imgpos(3) .ne. ucob(3)) )                                         &
  coarray( ubv(1) , ubv(2) , ubv(3) , : ) =                             &
  coarray( lbr(1) , lbr(2) , lbr(3) , : )                               &
                  [ imgpos(1)+1 , imgpos(2)+1 , imgpos(3)+1 ]

! op 26
if ( (imgpos(1) .ne. lcob(1)) .and. (imgpos(2) .ne. ucob(2)) .and.      &
     (imgpos(3) .ne. ucob(3)) )                                         &
  coarray( lbv(1) , ubv(2) , ubv(3) , : ) =                             &
  coarray( ubr(1) , lbr(2) , lbr(3) , : )                               &
                  [ imgpos(1)-1 , imgpos(2)+1 , imgpos(3)+1 ]

end subroutine cgca_hxi
```

15.3 cgca_m2hx/cgca_hxic

[cgca_m2hx] [Subroutines]

NAME

cgca_hxic

SYNOPSIS

```
module procedure cgca_hxic
```

INPUT

```
!    See the parent module
```

OUTPUT

```
!    See INPUT
```

SIDE EFFECTS

None

DESCRIPTION

This routine checks that a prior hx call was done flagly, i.e. that the halo cell states are consistent with the states of the corresponding real boundary cells. This routine can be called for any internal hx algorithm, e.g. cgca_hxi (15.2) or cgca_hxir (15.4).

NOTES

All images must call this routine! Lots of remote calls.

USES

All data objects from parent module cgca_m2hx (15) by host association.

USED BY

module cgca_m2hx (15)

SOURCE

```
integer ::                                                          &
  lbv(4)      , & ! lower bounds of the "virtual" coarray
  ubv(4)      , & ! upper bounds of the "virtual" coarray
  lbr(4)      , & ! lower bounds of the "real" coarray, lbv+1
  ubr(4)      , & ! upper bounds of the "real" coarray, ubv-1
  lcob(3)     , & ! lower cobounds of the coarray
  ucob(3)     , & ! upper cobounds of the coarray
  imgpos(3)       ! position of the image in a coarray grid

! Start with 0. Any error must resuult in a positive value.
flag = 0

! check for allocated
if ( .not. allocated( coarray ) )                                   &
```

```
  error stop "ERROR: cgca_hxic/m2hx_hxic: coarray is not allocated"

   lbv = lbound( coarray )
   ubv = ubound( coarray )
   lbr = lbv + 1
   ubr = ubv - 1
  lcob = lcobound( coarray )
  ucob = ucobound( coarray )
imgpos = this_image( coarray )

! Make sure only the virtual (halo) arrays are assigned to.
! The real array values must never appear on the left
! hand side of the assignment expressions.
! The halo exchange process is copying real array values into halos.
! There must not ever be copying real values
! to real, or halo to halo, or halo to real.
! Also, only local array must appear on the left.
! We are assigning values to the local array's virtual
! (halo) cells using values from real cells from arrays
! in other images.

! Check 2D halos in direction 1

! op 1
if ( imgpos(1) .ne. lcob(1) ) then
  if ( any(                                                          &
     coarray( lbv(1) , lbr(2) : ubr(2) , lbr(3) : ubr(3) , : ) .ne.  &
     coarray( ubr(1) , lbr(2) : ubr(2) , lbr(3) : ubr(3) , : )       &
       [ imgpos(1)-1 , imgpos(2) , imgpos(3) ] ) ) then
    flag = 1
    ! And return immediately
    return
  end if
end if

! op 2
if ( imgpos(1) .ne. ucob(1) ) then
  if ( any(                                                          &
    coarray( ubv(1) , lbr(2) : ubr(2) , lbr(3) : ubr(3) , : ) .ne.   &
    coarray( lbr(1) , lbr(2) : ubr(2) , lbr(3) : ubr(3) , : )        &
      [ imgpos(1)+1 , imgpos(2) , imgpos(3) ] ) ) then
    flag = 1
    ! And return immediately
    return
  end if
end if

! exchange 2D halos in direction 2

! op 3
if ( imgpos(2) .ne. lcob(2) ) then
  if ( any(                                                          &
```

```
    coarray( lbr(1) : ubr(1) , lbv(2) , lbr(3) : ubr(3) , : ) .ne.     &
    coarray( lbr(1) : ubr(1) , ubr(2) , lbr(3) : ubr(3) , : )          &
      [ imgpos(1) , imgpos(2)-1 , imgpos(3) ] ) ) then
    flag = 1
    ! And return immediately
    return
  end if
end if

! op 4
if ( imgpos(2) .ne. ucob(2) ) then
  if ( any(                                                            &
    coarray( lbr(1) : ubr(1) , ubv(2) , lbr(3) : ubr(3) , : ) .ne.     &
    coarray( lbr(1) : ubr(1) , lbr(2) , lbr(3) : ubr(3) , : )          &
      [ imgpos(1) , imgpos(2)+1 , imgpos(3) ] ) ) then
    flag = 1
    ! And return immediately
    return
  end if
end if

! exchange 2D halos in direction 3

! op 5
if ( imgpos(3) .ne. lcob(3) ) then
  if ( any(                                                            &
    coarray( lbr(1) : ubr(1) , lbr(2) : ubr(2) , lbv(3) , : ) .ne.     &
    coarray( lbr(1) : ubr(1) , lbr(2) : ubr(2) , ubr(3) , : )          &
      [ imgpos(1) , imgpos(2) , imgpos(3)-1 ] ) ) then
    flag = 1
    ! And return immediately
    return
  end if
end if

! op 6
if ( imgpos(3) .ne. ucob(3) ) then
  if ( any(                                                            &
  coarray( lbr(1) : ubr(1) , lbr(2) : ubr(2) , ubv(3) , : ) .ne.       &
  coarray( lbr(1) : ubr(1) , lbr(2) : ubr(2) , lbr(3) , : )            &
    [ imgpos(1) , imgpos(2) , imgpos(3)+1 ] ) ) then
    flag = 1
    ! And return immediately
    return
  end if
end if

! exchange 1D halos parallel to direction 3

! op 7
if ( imgpos(1) .ne. lcob(1) .and. imgpos(2) .ne. lcob(2) ) then
  if ( any(                                                            &
```

```
    coarray( lbv(1) , lbv(2) , lbr(3) : ubr(3) , : ) .ne.              &
    coarray( ubr(1) , ubr(2) , lbr(3) : ubr(3) , : )                   &
      [ imgpos(1)-1 , imgpos(2)-1 , imgpos(3) ] ) ) then
    flag = 1
    ! And return immediately
    return
  end if
end if

! op 8
if ( imgpos(1) .ne. ucob(1) .and. imgpos(2) .ne. lcob(2) ) then
  if ( any(                                                            &
    coarray( ubv(1) , lbv(2) , lbr(3) : ubr(3) , :) .ne.               &
    coarray( lbr(1) , ubr(2) , lbr(3) : ubr(3) , :)                    &
      [ imgpos(1)+1 , imgpos(2)-1 , imgpos(3) ] ) ) then
    flag = 1
    ! And return immediately
    return
  end if
end if

! op 9
if ( imgpos(1) .ne. ucob(1) .and. imgpos(2) .ne. ucob(2) ) then
  if ( any(                                                            &
    coarray( ubv(1) , ubv(2) , lbr(3) : ubr(3) , : ) .ne.              &
    coarray( lbr(1) , lbr(2) , lbr(3) : ubr(3) , : )                   &
      [ imgpos(1)+1 , imgpos(2)+1 , imgpos(3) ] ) ) then
    flag = 1
    ! And return immediately
    return
  end if
end if

! op 10
if ( imgpos(1) .ne. lcob(1) .and. imgpos(2) .ne. ucob(2) ) then
  if ( any(                                                            &
    coarray( lbv(1) , ubv(2) , lbr(3) : ubr(3) , : ) .ne.              &
    coarray( ubr(1) , lbr(2) , lbr(3) : ubr(3) , : )                   &
      [ imgpos(1)-1 , imgpos(2)+1 , imgpos(3) ] ) ) then
    flag = 1
    ! And return immediately
    return
  end if
end if

! exchange 1D halos parallel to direction 1

! op 11
if ( imgpos(2) .ne. lcob(2) .and. imgpos(3) .ne. lcob(3) ) then
  if ( any(                                                            &
    coarray( lbr(1) : ubr(1) , lbv(2) , lbv(3) , : ) .ne.              &
    coarray( lbr(1) : ubr(1) , ubr(2) , ubr(3) , : )                   &
```

```
      [ imgpos(1) , imgpos(2)-1 , imgpos(3)-1 ] ) ) then
    flag = 1
    ! And return immediately
    return
  end if
end if

! op 12
if ( imgpos(2) .ne. lcob(2) .and. imgpos(3) .ne. ucob(3) ) then
  if ( any(                                                          &
    coarray( lbr(1) : ubr(1) , lbv(2) , ubv(3) , : ) .ne.            &
    coarray( lbr(1) : ubr(1) , ubr(2) , lbr(3) , : )                 &
      [ imgpos(1) , imgpos(2)-1 , imgpos(3)+1 ] ) ) then
    flag = 1
    ! And return immediately
    return
  end if
end if

! op 13
if ( imgpos(2) .ne. ucob(2) .and. imgpos(3) .ne. ucob(3) ) then
  if ( any(                                                          &
    coarray( lbr(1) : ubr(1) , ubv(2) , ubv(3) , : ) .ne.            &
    coarray( lbr(1) : ubr(1) , lbr(2) , lbr(3) , : )                 &
      [ imgpos(1) , imgpos(2)+1 , imgpos(3)+1 ] ) ) then
    flag = 1
    ! And return immediately
    return
  end if
end if

! op 14
if ( imgpos(2) .ne. ucob(2) .and. imgpos(3) .ne. lcob(3) ) then
  if ( any(                                                          &
    coarray( lbr(1) : ubr(1) , ubv(2) , lbv(3) , : ) .ne.            &
    coarray( lbr(1) : ubr(1) , lbr(2) , ubr(3) , : )                 &
      [ imgpos(1) , imgpos(2)+1 , imgpos(3)-1 ] ) ) then
    flag = 1
    ! And return immediately
    return
  end if
end if

! exchange 1D halos parallel to direction 2

! op 15
if ( imgpos(1) .ne. lcob(1) .and. imgpos(3) .ne. lcob(3) ) then
  if ( any(                                                          &
    coarray( lbv(1) , lbr(2) : ubr(2) , lbv(3) , : ) .ne.            &
    coarray( ubr(1) , lbr(2) : ubr(2) , ubr(3) , : )                 &
      [ imgpos(1)-1 , imgpos(2) , imgpos(3)-1 ] ) ) then
    flag = 1
```

```
    ! And return immediately
    return
  end if
end if

! op 16
if ( imgpos(1) .ne. ucob(1) .and. imgpos(3) .ne. lcob(3) ) then
  if ( any(                                                          &
    coarray( ubv(1) , lbr(2) : ubr(2) , lbv(3) , : ) .ne.            &
    coarray( lbr(1) , lbr(2) : ubr(2) , ubr(3) , : )                 &
      [ imgpos(1)+1 , imgpos(2) , imgpos(3)-1 ] ) ) then
    flag = 1
    ! And return immediately
    return
  end if
end if

! op 17
if ( imgpos(1) .ne. ucob(1) .and. imgpos(3) .ne. ucob(3) ) then
  if ( any(                                                          &
    coarray( ubv(1) , lbr(2) : ubr(2) , ubv(3) , : ) .ne.            &
    coarray( lbr(1) , lbr(2) : ubr(2) , lbr(3) , : )                 &
      [ imgpos(1)+1 , imgpos(2) , imgpos(3)+1 ] ) ) then
    flag = 1
    ! And return immediately
    return
  end if
end if

! op 18
if ( imgpos(1) .ne. lcob(1) .and. imgpos(3) .ne. ucob(3) ) then
  if ( any(                                                          &
    coarray( lbv(1) , lbr(2) : ubr(2) , ubv(3) , : ) .ne.            &
    coarray( ubr(1) , lbr(2) : ubr(2) , lbr(3) , : )                 &
      [ imgpos(1)-1 , imgpos(2) , imgpos(3)+1 ] ) ) then
    flag = 1
    ! And return immediately
    return
  end if
end if

! Exchange 8 scalar halos
! See diagram cgca1 in the manual.

! op 19
if ( (imgpos(1) .ne. lcob(1)) .and. (imgpos(2) .ne. lcob(2)) .and.  &
     (imgpos(3) .ne. lcob(3)) ) then
  if ( any(                                                          &
    coarray( lbv(1) , lbv(2) , lbv(3) , : ) .ne.                     &
    coarray( ubr(1) , ubr(2) , ubr(3) , : )                          &
      [ imgpos(1)-1 , imgpos(2)-1 , imgpos(3)-1 ] ) ) then
    flag = 1
```

```
    ! And return immediately
    return
  end if
end if

! op 20
if ( (imgpos(1) .ne. ucob(1)) .and. (imgpos(2) .ne. lcob(2)) .and.     &
     (imgpos(3) .ne. lcob(3)) ) then
  if ( any(                                                            &
    coarray( ubv(1) , lbv(2) , lbv(3) , : ) .ne.                       &
    coarray( lbr(1) , ubr(2) , ubr(3) , : )                            &
      [ imgpos(1)+1 , imgpos(2)-1 , imgpos(3)-1 ] ) ) then
    flag = 1
    ! And return immediately
    return
  end if
end if

! op 21
if ( (imgpos(1) .ne. ucob(1)) .and. (imgpos(2) .ne. ucob(2)) .and.     &
     (imgpos(3) .ne. lcob(3)) ) then
  if ( any(                                                            &
    coarray( ubv(1) , ubv(2) , lbv(3) , : ) .ne.                       &
    coarray( lbr(1) , lbr(2) , ubr(3) , : )                            &
      [ imgpos(1)+1 , imgpos(2)+1 , imgpos(3)-1 ] ) ) then
    flag = 1
    ! And return immediately
    return
  end if
end if

! op 22
if ( (imgpos(1) .ne. lcob(1)) .and. (imgpos(2) .ne. ucob(2)) .and.     &
     (imgpos(3) .ne. lcob(3)) ) then
  if ( any(                                                            &
    coarray( lbv(1) , ubv(2) , lbv(3) , : ) .ne.                       &
    coarray( ubr(1) , lbr(2) , ubr(3) , : )                            &
      [ imgpos(1)-1 , imgpos(2)+1 , imgpos(3)-1 ] ) ) then
    flag = 1
    ! And return immediately
    return
  end if
end if

! op 23
if ( (imgpos(1) .ne. lcob(1)) .and. (imgpos(2) .ne. lcob(2)) .and.     &
     (imgpos(3) .ne. ucob(3)) ) then
  if ( any(                                                            &
    coarray( lbv(1) , lbv(2) , ubv(3) , : ) .ne.                       &
    coarray( ubr(1) , ubr(2) , lbr(3) , : )                            &
      [ imgpos(1)-1 , imgpos(2)-1 , imgpos(3)+1 ] ) ) then
    flag = 1
```

```
    ! And return immediately
    return
  end if
end if

! op 24
if ( (imgpos(1) .ne. ucob(1)) .and. (imgpos(2) .ne. lcob(2)) .and.     &
     (imgpos(3) .ne. ucob(3)) ) then
  if ( any(                                                            &
    coarray( ubv(1) , lbv(2) , ubv(3) , : ) .ne.                       &
    coarray( lbr(1) , ubr(2) , lbr(3) , : )                            &
      [ imgpos(1)+1 , imgpos(2)-1 , imgpos(3)+1 ] ) ) then
    flag = 1
    ! And return immediately
    return
  end if
end if

! op 25
if ( (imgpos(1) .ne. ucob(1)) .and. (imgpos(2) .ne. ucob(2)) .and.     &
     (imgpos(3) .ne. ucob(3)) ) then
  if ( any(                                                            &
    coarray( ubv(1) , ubv(2) , ubv(3) , : ) .ne.                       &
    coarray( lbr(1) , lbr(2) , lbr(3) , : )                            &
      [ imgpos(1)+1 , imgpos(2)+1 , imgpos(3)+1 ] ) ) then
    flag = 1
    ! And return immediately
    return
  end if
end if

! op 26
if ( (imgpos(1) .ne. lcob(1)) .and. (imgpos(2) .ne. ucob(2)) .and.     &
     (imgpos(3) .ne. ucob(3)) ) then
  if ( any(                                                            &
    coarray( lbv(1) , ubv(2) , ubv(3) , : ) .ne.                       &
    coarray( ubr(1) , lbr(2) , lbr(3) , : )                            &
      [ imgpos(1)-1 , imgpos(2)+1 , imgpos(3)+1 ] ) ) then
    flag = 1
    ! And return immediately
    return
  end if
end if

end procedure cgca_hxic
```

15.4 cgca_m2hx/cgca_hxir

[cgca_m2hx] [Subroutines]

NAME

cgca_hxir

SYNOPSIS

```
module procedure cgca_hxir
```

INPUT

```
!    See the parent module
```

OUTPUT

```
!    See INPUT
```

SIDE EFFECTS

coarray is changed

DESCRIPTION

This routine does internal halo exchange in random order. The routine exchanges halos on *all* cell state types. This is an overkill, as it is likely that only one cell state type needs to be halo exchanged at a time. However, it makes for an easier code, and there is virtually no performance penalty, so we do it this way.

NOTES

All images must call this routine!

USES

All data objects from parent module cgca_m2hx (15) by host association.

USED BY

module cgca_m2hx (15)

SOURCE

```
! Number of groups of remote calls
! This parameter is used to randomoise the order of remote calls
integer, parameter :: ngroups=3

integer ::                                                              &
  lbv(4)      , & ! lower bounds of the "virtual" coarray
  ubv(4)      , & ! upper bounds of the "virtual" coarray
  lbr(4)      , & ! lower bounds of the "real" coarray, lbv+1
  ubr(4)      , & ! upper bounds of the "real" coarray, ubv-1
  lcob(3)     , & ! lower cobounds of the coarray
  ucob(3)     , & ! upper cobounds of the coarray
  imgpos(3)   , & ! position of the image in a coarray grid
  imgpos1mns1 , & ! positions of the neighbouring images
```

```
  imgpos1pls1 , & ! along 3 directions
  imgpos2mns1 , & !
  imgpos2pls1 , & !
  imgpos3mns1 , & !
  imgpos3pls1 , & !
  istart      , & ! starting group of remote calls
  i, idx          ! loop indices

real :: rnd

! check for allocated
if ( .not. allocated( coarray ) )                                        &
  error stop "ERROR: m2hx_hxir/cgca_hxir: coarray is not allocated"

 lbv = lbound( coarray )
 ubv = ubound( coarray )
 lbr = lbv + 1
 ubr = ubv - 1
lcob = lcobound( coarray )
ucob = ucobound( coarray )

     imgpos = this_image( coarray )
imgpos1mns1 = imgpos(1) - 1
imgpos1pls1 = imgpos(1) + 1
imgpos2mns1 = imgpos(2) - 1
imgpos2pls1 = imgpos(2) + 1
imgpos3mns1 = imgpos(3) - 1
imgpos3pls1 = imgpos(3) + 1

! Make sure only the virtual (halo) arrays are assigned to.
! The real array values must never appear on the left
! hand side of the assignment expressions.
! The halo exchange process is copying real array values into halos.
! There must not ever be copying real values
! to real, or halo to halo, or halo to real.
! Also, only local array must appear on the left.
! We are assigning values to the local array's virtual
! (halo) cells using values from real cells from arrays
! in other images.

! I split all transfers into several groups.
! The starting group is chosen at random.
call random_number( rnd )         ! [ 0 .. 1 )
istart = int( rnd * ngroups ) + 1 ! [ 1 .. ngroups ]

!rcalls: do i = istart , istart + ngroups - 1
rcalls: do i = 1,1

  ! Remote call group index
  idx = i
  if ( idx .gt. ngroups ) idx = idx - ngroups
```

```
  groups: if ( idx .eq. 1 ) then
    ! 1st group of remote calls

    ! exchange 2D halos in direction 1

    if ( imgpos(1) .ne. lcob(1) )                                    &
      coarray( lbv(1) , lbr(2) : ubr(2) , lbr(3) : ubr(3) , : ) =    &
      coarray( ubr(1) , lbr(2) : ubr(2) , lbr(3) : ubr(3) , : )      &
        [ imgpos1mns1 , imgpos(2) , imgpos(3) ]

    if ( imgpos(1) .ne. ucob(1) )                                    &
      coarray( ubv(1) , lbr(2) : ubr(2) , lbr(3) : ubr(3) , : ) =    &
      coarray( lbr(1) , lbr(2) : ubr(2) , lbr(3) : ubr(3) , : )      &
        [ imgpos1pls1 , imgpos(2) , imgpos(3) ]

    ! exchange 1D halos parallel to direction 1

    if (imgpos(2) .ne. lcob(2) .and. imgpos(3) .ne. lcob(3))         &
      coarray(lbr(1):ubr(1),lbv(2),lbv(3),:) =                       &
      coarray(lbr(1):ubr(1),ubr(2),ubr(3),:)                         &
        [imgpos(1),imgpos2mns1,imgpos3mns1]

    if (imgpos(2) .ne. ucob(2) .and. imgpos(3) .ne. lcob(3))         &
      coarray(lbr(1):ubr(1),ubv(2),lbv(3),:) =                       &
      coarray(lbr(1):ubr(1),lbr(2),ubr(3),:)                         &
        [imgpos(1),imgpos2pls1,imgpos3mns1]

    if (imgpos(2) .ne. ucob(2) .and. imgpos(3) .ne. ucob(3))         &
      coarray(lbr(1):ubr(1),ubv(2),ubv(3),:) =                       &
      coarray(lbr(1):ubr(1),lbr(2),lbr(3),:)                         &
        [imgpos(1),imgpos2pls1,imgpos3pls1]

    if (imgpos(2) .ne. lcob(2) .and. imgpos(3) .ne. ucob(3))         &
      coarray(lbr(1):ubr(1),lbv(2),ubv(3),:) =                       &
      coarray(lbr(1):ubr(1),ubr(2),lbr(3),:)                         &
        [imgpos(1),imgpos2mns1,imgpos3pls1]

!   else if ( idx .eq. 2 ) then
    ! 2nd group of remote calls

    ! exchange 2D halos in direction 2

    if ( imgpos(2) .ne. lcob(2) )                                    &
      coarray( lbr(1) : ubr(1) , lbv(2) , lbr(3) : ubr(3) , : ) =    &
      coarray( lbr(1) : ubr(1) , ubr(2) , lbr(3) : ubr(3) , : )      &
        [ imgpos(1) , imgpos2mns1 , imgpos(3) ]

    if ( imgpos(2) .ne. ucob(2) )                                    &
      coarray( lbr(1) : ubr(1) , ubv(2) , lbr(3) : ubr(3) , : ) =    &
      coarray( lbr(1) : ubr(1) , lbr(2) , lbr(3) : ubr(3) , : )      &
        [ imgpos(1) , imgpos2pls1 , imgpos(3) ]
```

```
    ! exchange 1D halos parallel to direction 2

    if (imgpos(1) .ne. lcob(1) .and. imgpos(3) .ne. lcob(3))           &
      coarray(lbv(1),lbr(2):ubr(2),lbv(3),:) =                         &
      coarray(ubr(1),lbr(2):ubr(2),ubr(3),:)                           &
        [imgpos1mns1,imgpos(2),imgpos3mns1]

    if (imgpos(1) .ne. ucob(1) .and. imgpos(3) .ne. lcob(3))           &
      coarray(ubv(1),lbr(2):ubr(2),lbv(3),:) =                         &
      coarray(lbr(1),lbr(2):ubr(2),ubr(3),:)                           &
        [imgpos1pls1,imgpos(2),imgpos3mns1]

    if (imgpos(1) .ne. ucob(1) .and. imgpos(3) .ne. ucob(3))           &
      coarray(ubv(1),lbr(2):ubr(2),ubv(3),:) =                         &
      coarray(lbr(1),lbr(2):ubr(2),lbr(3),:)                           &
        [imgpos1pls1,imgpos(2),imgpos3pls1]

    if (imgpos(1) .ne. lcob(1) .and. imgpos(3) .ne. ucob(3))           &
      coarray(lbv(1),lbr(2):ubr(2),ubv(3),:) =                         &
      coarray(ubr(1),lbr(2):ubr(2),lbr(3),:)                           &
        [imgpos1mns1,imgpos(2),imgpos3pls1]

!   else if ( idx .eq. 3 ) then
    ! 3rd group of remote calls

    ! exchange 2D halos in direction 3

    if ( imgpos(3) .ne. lcob(3) )                                      &
      coarray( lbr(1) : ubr(1) , lbr(2) : ubr(2) , lbv(3) , : ) =      &
      coarray( lbr(1) : ubr(1) , lbr(2) : ubr(2) , ubr(3) , : )        &
        [ imgpos(1) , imgpos(2) , imgpos3mns1 ]

    if ( imgpos(3) .ne. ucob(3) )                                      &
      coarray( lbr(1) : ubr(1) , lbr(2) : ubr(2) , ubv(3) , : ) =      &
      coarray( lbr(1) : ubr(1) , lbr(2) : ubr(2) , lbr(3) , : )        &
        [ imgpos(1) , imgpos(2) , imgpos3pls1 ]

    ! exchange 1D halos parallel to direction 3

    if ( imgpos(1) .ne. lcob(1) .and. imgpos(2) .ne. lcob(2) )         &
      coarray( lbv(1) , lbv(2) , lbr(3) : ubr(3) , : ) =               &
      coarray( ubr(1) , ubr(2) , lbr(3) : ubr(3) , : )                 &
        [ imgpos1mns1 , imgpos2mns1 , imgpos(3) ]

    if (imgpos(1) .ne. ucob(1) .and. imgpos(2) .ne. lcob(2))           &
      coarray(ubv(1),lbv(2),lbr(3):ubr(3),:) =                         &
      coarray(lbr(1),ubr(2),lbr(3):ubr(3),:)                           &
        [imgpos1pls1,imgpos2mns1,imgpos(3)]

    if (imgpos(1) .ne. ucob(1) .and. imgpos(2) .ne. ucob(2))           &
      coarray(ubv(1),ubv(2),lbr(3):ubr(3),:) =                         &
      coarray(lbr(1),lbr(2),lbr(3):ubr(3),:)                           &
```

```
    [imgpos1pls1,imgpos2pls1,imgpos(3)]

if (imgpos(1) .ne. lcob(1) .and. imgpos(2) .ne. ucob(2))           &
  coarray(lbv(1),ubv(2),lbr(3):ubr(3),:) =                         &
  coarray(ubr(1),lbr(2),lbr(3):ubr(3),:)                           &
    [imgpos1mns1,imgpos2pls1,imgpos(3)]

! exchange 8 scalar halos
! first 4 statements are for the lower bound virtual
! corners of the coarray along dimension 3.

if ((imgpos(1) .ne. lcob(1)) .and. (imgpos(2) .ne. lcob(2)) .and.  &
    (imgpos(3) .ne. lcob(3)))                                      &
  coarray( lbv(1), lbv(2), lbv(3), : ) =                           &
  coarray( ubr(1), ubr(2), ubr(3), : )                             &
                    [ imgpos1mns1, imgpos2mns1, imgpos3mns1 ]

if ((imgpos(1) .ne. ucob(1)) .and. (imgpos(2) .ne. lcob(2)) .and.  &
    (imgpos(3) .ne. lcob(3)))                                      &
  coarray( ubv(1), lbv(2), lbv(3), : ) =                           &
  coarray( lbr(1), ubr(2), ubr(3), : )                             &
                    [ imgpos1pls1, imgpos2mns1, imgpos3mns1 ]

if ((imgpos(1) .ne. lcob(1)) .and. (imgpos(2) .ne. ucob(2)) .and.  &
    (imgpos(3) .ne. lcob(3)))                                      &
  coarray( lbv(1), ubv(2), lbv(3), : ) =                           &
  coarray( ubr(1), lbr(2), ubr(3), : )                             &
                    [ imgpos1mns1, imgpos2pls1, imgpos3mns1 ]

if ((imgpos(1) .ne. ucob(1)) .and. (imgpos(2) .ne. ucob(2)) .and.  &
    (imgpos(3) .ne. lcob(3)))                                      &
  coarray( ubv(1), ubv(2), lbv(3), : ) =                           &
  coarray( lbr(1), lbr(2), ubr(3), : )                             &
                    [ imgpos1pls1, imgpos2pls1, imgpos3mns1 ]

! these 4 statements are for the uppper bound virtual
! corners of the coarray along dimension 3.

if ((imgpos(1) .ne. lcob(1)) .and. (imgpos(2) .ne. lcob(2)) .and.  &
    (imgpos(3) .ne. ucob(3)))                                      &
  coarray (lbv(1), lbv(2), ubv(3), : ) =                           &
  coarray (ubr(1), ubr(2), lbr(3), : )                             &
                    [ imgpos1mns1, imgpos2mns1, imgpos3pls1 ]

if ((imgpos(1) .ne. ucob(1)) .and. (imgpos(2) .ne. lcob(2)) .and.  &
    (imgpos(3) .ne. ucob(3)))                                      &
  coarray (ubv(1), lbv(2), ubv(3), : ) =                           &
  coarray (lbr(1), ubr(2), lbr(3), : )                             &
                    [ imgpos1pls1, imgpos2mns1, imgpos3pls1 ]

if ((imgpos(1) .ne. lcob(1)) .and. (imgpos(2) .ne. ucob(2)) .and.  &
    (imgpos(3) .ne. ucob(3)))                                      &
```

```
      coarray (lbv(1), ubv(2), ubv(3), : ) =                          &
      coarray (ubr(1), lbr(2), lbr(3), : )                            &
                        [ imgpos1mns1, imgpos2pls1, imgpos3pls1 ]

    if ((imgpos(1) .ne. ucob(1)) .and. (imgpos(2) .ne. ucob(2)) .and.  &
        (imgpos(3) .ne. ucob(3)))                                     &
      coarray (ubv(1), ubv(2), ubv(3), : ) =                          &
      coarray (lbr(1), lbr(2), lbr(3), : )                            &
                        [ imgpos1pls1, imgpos2pls1, imgpos3pls1 ]

  end if groups

end do rcalls

end procedure cgca_hxir
```

15.5 cgca_m2hx/m2hx_hxic

[cgca_m2hx] [Submodules]

NAME

m2hx_hxic

SYNOPSIS

```
!$Id: m2hx_hxic.f90 430 2017-06-30 07:39:43Z mexas $

submodule ( cgca_m2hx ) m2hx_hxic
```

DESCRIPTION

Submodule of cgca_m2hx (15) with a hx subroutine.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENCE

CONTAINS

cgca_hxic (15.3)

USES

Variables and parameters from the parent module cgca_m2hx (15).

USED BY

The parent module cgca_m2hx (15).

SOURCE

```
implicit none

contains
```

15.6 cgca_m2hx/m2hx_hxir

[cgca_m2hx] [Submodules]

NAME

m2hx_hxir

SYNOPSIS

```
!$Id: m2hx_hxir.f90 423 2017-06-25 20:54:50Z mexas $

submodule ( cgca_m2hx ) m2hx_hxir
```

DESCRIPTION

Submodule of cgca_m2hx (15) with an internal halo exchange routine with random sequence of remote operations.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENCE

CONTAINS

cgca_hxir (15.4)

USES

Variables and parameters from the parent module cgca_m2hx (15).

USED BY

The parent module cgca_m2hx (15).

SOURCE

```
implicit none

contains
```

16 CGPACK/cgca_m2lnklst

[Modules]

NAME

cgca_m2lnklst

SYNOPSIS

```
!$Id: cgca_m2lnklst.f90 380 2017-03-22 11:03:09Z mexas $

module cgca_m2lnklst
```

DESCRIPTION

Module with link list types and routines. The module is mainly useful for linking CGPAK to FE. In case the CA box is sticking outside of the FE model. Routines of this module help effectively find all cells which are inside and outside of the FE model.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

CONTAINS

All public. Derived types cgca_lnklst_tpayld (16.5), cgca_lnklst_node (16.4). Routines cgca_inithead (16.3), cgca_addhead (16.1), cgca_addmiddle (16.2), cgca_rmhead (16.7), cgca_rmmiddle (16.8), cgca_lstdmp (16.6).

USES

cgca_m1co (9)

USED BY

cgca_m3pfem (30)

SOURCE

```
use cgca_m1co, only : idef
implicit none

private
public ::                                                          &
! derived types
          cgca_lnklst_tpayld, cgca_lnklst_node, &
! routines
          cgca_inithead, &
          cgca_addhead, &
          cgca_addmiddle, cgca_rmhead, cgca_rmmiddle, cgca_lstdmp
```

16.1 cgca_m2lnklst/cgca_addhead

[cgca_m2lnklst] [Subroutines]

NAME

cgca_addhead

SYNOPSIS

```
subroutine cgca_addhead( head, payload )
```

INPUTS

```
type( cgca_lnklst_node ), pointer, intent( inout ) :: head
type( cgca_lnklst_tpayld ), intent( in ) :: payload
```

SIDE EFFECTS

Memory for one entity of type "node" is allocated. The value of that memory is set to "payload". The pointer to this memory is returned as "head". The new head points to the old head.

DESCRIPTION

This routine adds another node on top of the head, i.e. puts another node higher than current head. The new node becomes the new head. It points to the old head.

SOURCE

```
  type( cgca_lnklst_node ), pointer :: tmp

  allocate( tmp )
  tmp%value = head%value
  tmp%next => head%next
  allocate( head )
  head%value = payload
  head%next => tmp
end subroutine cgca_addhead
```

16.2 cgca_m2lnklst/cgca_addmiddle

[cgca_m2lnklst] [Subroutines]

NAME

cgca_addmiddle

SYNOPSIS

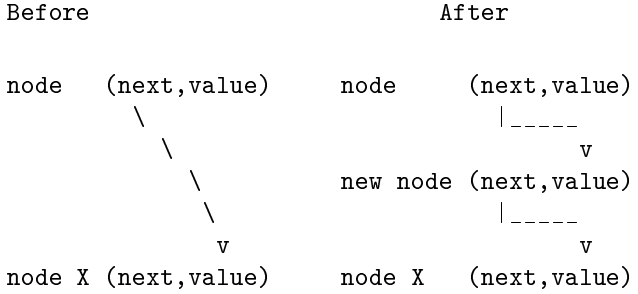
```
subroutine cgca_addmiddle( node, payload )
```

INPUTS

```
type( cgca_lnklst_node ), pointer, intent( in ) :: node
type( cgca_lnklst_tpayld ), intent( in ) :: payload
```

SIDE EFFECTS

Memory for one entity of type cgca_lnklst_node (16.4) is allocated. The value of that memory is set to "payload". The new entity points to where "node" pointed before. Node now points to the new entity. A schematic diagram:



DESCRIPTION

This routine adds another node *lower* than the given node. Lower here means further from the head and closer to NULL. The new node points to where the old node pointed. The old node points to the new node. So the list length is +1.

SOURCE

```
  type( cgca_lnklst_node ), pointer :: tmp

  allocate( tmp )
  tmp%value = payload
   tmp%next => node%next
  node%next => tmp
end subroutine cgca_addmiddle
```

16.3 cgca_m2lnklst/cgca_inithead

[cgca_m2lnklst] [Subroutines]

NAME

cgca_inithead

SYNOPSIS

```
subroutine cgca_inithead( head, payload )
```

INPUT

```
type( cgca_lnklst_tpayld ), intent( in ) :: payload
```

OUTPUT

```
type( cgca_lnklst_node ), pointer, intent( out ) :: head
```

SIDE EFFECTS

Memory for one entity of type "node" is allocated. The pointer to this memory is returned as "head". The value of that memory is set to "payload".

DESCRIPTION

This routine initialises the head node of the linked list. The head node is the node at the very top of the list. The head node has nothing higher. This is the only node that can be accessed directly. Access to all other nodes is via the head node and pointers therein.

SOURCE

```
  allocate( head )
  head%value = payload
  head%next => null()
end subroutine cgca_inithead
```

16.4 cgca_m2lnklst/cgca_lnklst_node

[cgca_m2lnklst] [Data structures]

NAME

cgca_lnklst_node

SYNOPSIS

```
type cgca_lnklst_node
  type( cgca_lnklst_tpayld ) :: value
  type( cgca_lnklst_node ), pointer :: next
end type cgca_lnklst_node
```

DESCRIPTION

A derived type for a node in the linked list. A very traditional type. The payload is of derived type cgca_lnklst_tpayld (16.5).

USED BY

All routines of module cgca_m2lnklst (16).

16.5 cgca_m2lnklst/cgca_lnklst_tpayld

[cgca_m2lnklst] [Data structures]

NAME

cgca_lnklst_tpayld

SYNOPSIS

```
type cgca_lnklst_tpayld
  integer :: lwr(3), upr(3)
end type cgca_lnklst_tpayld
```

DESCRIPTION

Payload type for all link list routines. The payload consists of two integer arrays of length 3. The arrays contain lower and upper corner coordinates of a CA box in local CA coord. system.

USED BY

All routines of module cgca_m2lnklst (16).

16.6 cgca_m2lnklst/cgca_lstdmp

[cgca_m2lnklst] [Subroutines]

NAME

cgca_lstdmp

SYNOPSIS

```
subroutine cgca_lstdmp( head )
```

INPUT

```
type( cgca_lnklst_node ), pointer, intent( in ) :: head
```

SIDE EFFECTS

Values of all nodes are dumped to stdout.

DESCRIPTION

This routine dumps all nodes, one per line, starting from HEAD, till it reaches NULL.

SOURCE

```
  type( cgca_lnklst_node ), pointer :: tmp

  if ( .not. associated( head ) ) return
  tmp => head
  do
    write (*,*) tmp%value
    tmp => tmp%next
    if ( .not. associated( tmp ) ) exit
  end do
end subroutine cgca_lstdmp
```

16.7 cgca_m2lnklst/cgca_rmhead

[cgca_m2lnklst] [Subroutines]

NAME

cgca_rmhead

SYNOPSIS

```
subroutine cgca_rmhead( head, stat )
```

INPUT

```
type( cgca_lnklst_node ), pointer, intent( inout ) :: head
```

OUTPUT

```
!    stat - integer, 0 if no problem, 1 if the head node is NULL.

integer( kind=idef ), intent( out ) :: stat
```

SIDE EFFECTS

Memory for one entity of type "node" is freed. The pointer to the old head now points to where the old head was pointing. This pointer is returned as "head".

DESCRIPTION

This routine removes the head node. The list length decreases by 1. The pointer to the old head is given on entry. On exit this pointer points to where the old head was pointing, i.e. one node closer to NULL. If there was only a single node on top of head, then the head will return null (unassociated) and stat will be 1. If there is no head node already, i.e. head is not associated already, head will not be changed and stat of 1 will be returned.

SOURCE

```
type( cgca_lnklst_node ), pointer :: tmp
stat = 0

if ( associated( head ) ) then
  tmp => head
  head => head%next
  deallocate( tmp )
end if

! This pointer is not associated only if NULL has been reached.
! Do nothing and set the output flag accordingly.
if ( .not. associated( head ) ) stat = 1

end subroutine cgca_rmhead
```

16.8 cgca_m2lnklst/cgca_rmmiddle

[cgca_m2lnklst] [Subroutines]

NAME

cgca_rmmiddle

SYNOPSIS

```
subroutine cgca_rmmiddle( node, stat )
```

INPUT

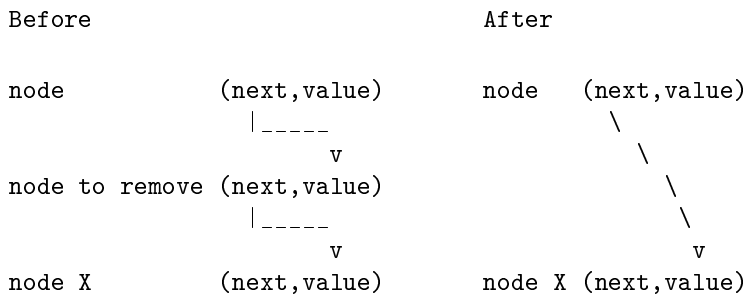
```
type( cgca_lnklst_node ), pointer, intent( in ) :: node
```

OUTPUT

```
integer( kind=idef), intent( out ) :: stat
```

SIDE EFFECTS

Memory for one entity of type cgca_lnklst_node (16.4) is freed.



DESCRIPTION

The node below the given node is removed. Below here means further from the head and closer to NULL. The node that pointed to the node to remove before, now points to where the node to remove was pointing.

NOTES

On output stat=0 means no problem. If stat=1, then an attempt has been made to remove NULL.

SOURCE

```
  type( cgca_lnklst_node ), pointer :: tmp

  stat = 0
  tmp => node%next
  if ( associated( tmp ) ) then
    node%next => tmp%next
    deallocate( tmp )
  else
    ! This pointer is not associated only if NULL has been reached.
```

```
    ! Do nothing but set the output flag accordingly.
    stat = 1
  end if
end subroutine cgca_rmmiddle
```

17 CGPACK/cgca_m2mpiio

[Modules]

NAME

cgca_m2mpiio

SYNOPSIS

```
!$Id: cgca_m2mpiio.f90 380 2017-03-22 11:03:09Z mexas $

module cgca_m2mpiio
```

DESCRIPTION

Module with MPI/IO related routines

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

CONTAINS

Subroutines: cgca_pswci2 (17.1)

USES

cgca_m1co (9)

USED BY

cgca

SOURCE

```
use cgca_m1co
use mpi
implicit none
private
public :: cgca_pswci2

contains
```

17.1 cgca_m2mpiio/cgca_pswci2

[cgca_m2mpiio] [Subroutines]

NAME

cgca_pswci2

SYNOPSIS

```
subroutine cgca_pswci2( coarray, stype, fname )
```

INPUTS

```
    integer( kind=iarr ), allocatable, intent( in ) ::                &
      coarray(:,:,:,:)[:,:,:]
    integer( kind=idef ),intent( in ) :: stype
    character( len=* ), intent( in ) :: fname
```

OUTPUTS

```
!     None
```

SIDE EFFECTS

A single binary file is created using MPI/IO with contents of coarray.

DESCRIPTION

Parallel Stream Write Coarray of Integers:

- coarray - what array to dump
- stype - what cell state type to dump
- fname - what file name to use

NOTES

All images must call this routine!

MPI must be initialised prior to calling this routine, most probably in the main program. Likewise MPI must be terminated only when no further MPI routines can be called. This will most likely be in the main program. There are some assumptions about the shape of the passed array.

The default integer is assumed for the array at present!

AUTHOR

Anton Shterenlikht, adapted from the code written by David Henty, EPCC

COPYRIGHT

Note that this routine has special Copyright conditions.

```
    !------------------------------------------------------------------------------!
    !                                                                              !
    !  MPI-IO routine for Fortran Coarrays                                         !
```

```
    !                                                                              !
    !  David Henty, EPCC; d.henty@epcc.ed.ac.uk                                    !
    !                                                                              !
    !  Copyright 2013 the University of Edinburgh                                  !
    !                                                                              !
    !  Licensed under the Apache LICENSE, Version 2.0 (the "LICENSE");             !
    !  you may not use this file except in compliance with the LICENSE.            !
    !  You may obtain a copy of the LICENSE at                                     !
    !                                                                              !
    !      http://www.apache.org/licenses/LICENSE-2.0                              !
    !                                                                              !
    !  Unless required by applicable law or agreed to in writing, software         !
    !  distributed under the LICENSE is distributed on an "AS IS" BASIS,           !
    !  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.    !
    !  See the LICENSE for the specific language governing permissions and         !
    !  limitations under the LICENSE.                                              !
    !                                                                              !
    !------------------------------------------------------------------------------!
```

USES

cgca_m1co (9), MPI library

USED BY

none, end user.

SOURCE

```
integer, parameter :: totdim = 4, arrdim = totdim-1, coardim = 3

integer :: img, nimgs, comm, ierr=0, rank=0, mpisize=0, filetype,      &
           mpi_subarray, fh, funit
integer, dimension(totdim) :: asizehal
integer, dimension(arrdim) :: arrsize, arstart, artsize
integer, dimension(coardim) :: coarsize, copos
integer( kind=MPI_OFFSET_KIND ) :: disp = 0
integer, dimension(MPI_STATUS_SIZE) :: mpistat

character( len=80 ) :: iomsg

  img = this_image()
nimgs = num_images()

asizehal(:) = shape( coarray )
   copos(:) = this_image( coarray )

! Subtract halos
 arrsize(:) = asizehal(1:arrdim) - 2
coarsize(:) = ucobound(coarray) - lcobound(coarray) + 1

! Does the array fit exactly?
if ( product( coarsize ) .ne. nimgs) then
 write(*,*) 'ERROR: cgca_m2mpiio/cgca_pswci2: non-conforming coarray'
```

```
 error stop
end if

comm = MPI_COMM_WORLD
call MPI_Comm_size( comm, mpisize, ierr )
call MPI_Comm_rank( comm, rank, ierr )

! Sanity check
if ( mpisize .ne. nimgs .or. rank .ne. img-1 ) then
  write(*,*) 'ERROR: cgca_m2mpiio/cgca_pswci2: MPI/coarray mismatch'
  error stop
end if

! Define filetype for this process, ie what portion of the global array
! this process owns. Starting positions use C-indexing
! (ie counting from 0).
artsize(:) = arrsize(:) * coarsize(:)
arstart(:) = arrsize(:) * (copos(:)-1)

! debug
!write (*,*) "image",img, "asizehal", asizehal, "copos", copos,         &
!  "arrsize", arrsize, "coarsize", coarsize,                            &
!  "artsize", artsize, "arstart", arstart, "stype", stype

call MPI_Type_create_subarray( arrdim, artsize, arrsize, arstart,      &
 MPI_ORDER_FORTRAN, MPI_INTEGER, filetype, ierr )
if ( ierr .ne. MPI_SUCCESS ) then
 write (*,'(a,i0)') 'ERROR: cgca_m2mpiio/cgca_pswci2:' //              &
   " MPI_type_create_subarray filetype: rank: ", rank
 error stop
end if

call MPI_Type_commit( filetype, ierr )

! Define subarray for this process, ie what portion of the local array
! is to be written (excludes halos); starting positions use C-indexing.

arstart(:) = 1

call MPI_Type_create_subarray( arrdim, asizehal, arrsize, arstart,     &
 MPI_ORDER_FORTRAN, MPI_INTEGER, mpi_subarray, ierr)
if ( ierr .ne. MPI_SUCCESS ) then
 write (*,'(a,i0)') 'ERROR: cgca_m2mpiio/cgca_pswci2:' //              &
   " MPI_type_create_subarray mpi_subarray: rank: ", rank
 error stop
end if

call MPI_Type_commit(mpi_subarray, ierr)

!  "Striping information cannot be changed on an existing
! file, so to set the stripe count (and stripe size) for the amount of
! parallelism you want to achieve, the file must be deleted if it exists."
```

```
! From: Cray Getting Started on MPI I/O manual, S-2490-40 - Dec 2009:
! http://docs.cray.com/books/S-2490-40/

!if ( rank .eq. 0 ) then
!  call MPI_File_delete( fname, MPI_INFO_NULL, ierr )
!  if ( ierr .ne. MPI_SUCCESS )                                          &
!    error stop "ERROR: cgca_pswci2: MPI_file_delete: rank 0"
!end if

! All ranks wait till rank 0 deletes the file
!call MPI_Barrier( comm, ierr )
!if ( ierr .ne. MPI_SUCCESS ) then
!  write (*,*) 'ERROR: cgca_pswci2: MPI_file_open: rank ', rank
!  error stop
!end if

! Overwriting MPI/IO files does not involve erasing the file first.
! So if the old file was bigger, the new smaller file will still
! be sized on disk as the old file, with only a part of it overwritten
! with new data. That would be bad. To avoid this
! image 1 removes all previous contents of this file, if it exists.
rm: if ( img .eq. 1 ) then

  ! this should not be necessary, but Cray issues a caution otherwise
  funit = 0

  open( newunit=funit, file=fname, status="replace", iostat=ierr,      &
        iomsg=iomsg )
  if ( ierr .ne. 0 ) then
    write (*,*) "ERROR: cgca_m2mpiio/cgca_pswci2: open( fname )," //    &
      " iostat=", ierr, "iomsg:", iomsg
    error stop
  end if

  write( funit, * , iostat=ierr) ""
  if ( ierr .ne. 0 ) then
    write (*,*) "ERROR: cgca_m2mpiio/cgca_pswci2: write( fname )," //   &
      " iostat=", ierr
    error stop
  end if

!  flush( funit, iostat=ierr )
!  if ( ierr .ne. 0 ) then
!    write (*,*) "ERROR: cgca_m2mpiio: flush( fname ), iostat=",ierr
!    error stop
!  end if

  close( funit, iostat=ierr )
  if ( ierr .ne. 0 ) then
    write (*,*) "ERROR: cgca_m2mpiio/cgca_pswci2: close( fname )," //   &
      " iostat=", ierr
    error stop
```

```
  end if

end if rm

! all images wait till image 1 erases the previous file
sync all

!  Open the file for writing only and attach to file handle fh
!  No IO hints are passed since MPI_INFO_NULL is specified
call MPI_File_open( comm, fname, ior(MPI_MODE_WRONLY, MPI_MODE_CREATE),&
                    MPI_INFO_NULL, fh, ierr )
if ( ierr .ne. MPI_SUCCESS ) then
  write (*,'(a,i0)') "ERROR: cgca_m2mpiio/cgca_pswci2:" //             &
    " MPI_file_open: rank: ", rank
  error stop
end if

!  Set view for this process using appropriate datatype
call MPI_File_set_view(                                               &
 fh, disp, MPI_INTEGER, filetype, 'native', MPI_INFO_NULL, ierr)
if ( ierr .ne. MPI_SUCCESS ) then
  write (*,'(a,i0)') 'ERROR: cgca_m2mpiio/cgca_pswci2:' //            &
    " MPI_file_set_view: rank: ", rank
  error stop
end if

! Write all the data for this process.
! Remove halo data by passing an explicit Fortran subarray
call MPI_File_write_all( fh, coarray(:,:,:,stype), 1, mpi_subarray,   &
                         mpistat, ierr )
if ( ierr .ne. MPI_SUCCESS ) then
  write (*,'(a,i0)') 'ERROR: cgca_m2mpiio/cgca_pswci2:' //            &
    " MPI_file_write_all: rank: ", rank
  error stop
end if

!  Close file
call MPI_File_close( fh, ierr )
if ( ierr .ne. MPI_SUCCESS ) then
 write (*,'(a,i0)') 'ERROR: cgca_m2mpiio/cgca_pswci2:' //             &
   " MPI_file_close: rank: ", rank
 error stop
end if

call MPI_Type_free( filetype, ierr )
call MPI_Type_free( mpi_subarray, ierr )

end subroutine cgca_pswci2
```

18 CGPACK/cgca_m2netcdf

[Modules]

NAME

cgca_m2netcdf

SYNOPSIS

```
!$Id: cgca_m2netcdf.f90 380 2017-03-22 11:03:09Z mexas $

module cgca_m2netcdf
```

DESCRIPTION

Module with netCDF related routines

AUTHOR

Luis Cebamanos

COPYRIGHT

See LICENSE (33)

CONTAINS

Subroutines: cgca_pswci3 (18.1)

USES

cgca_m1co (9)

USED BY

cgca

SOURCE

```
use cgca_m1co
use mpi
use netcdf
implicit none
private
public :: cgca_pswci3

contains
```

18.1 cgca_m2netcdf/cgca_pswci3

[cgca_m2netcdf] [Subroutines]

NAME

cgca_pswci3

SYNOPSIS

```
subroutine cgca_pswci3( coarray, stype, fname )
```

INPUTS

```
    integer( kind=iarr ), allocatable, intent( in ) ::                &
      coarray(:,:,:,:)[:,:,:]
    integer( kind=idef ),intent( in ) :: stype
    character( len=* ), intent( in ) :: fname
```

OUTPUTS

```
!     None
```

SIDE EFFECTS

A single binary file is created using netcdf with contents of coarray.

DESCRIPTION

Parallel Stream Write Coarray of Integers:

- coarray - what array to dump
- stype - what cell state type to dump
- fname - what file name to use

NOTES

All images must call this routine!

MPI must be initialised prior to calling this routine, most probably in the main program. Likewise MPI must be terminated only when no further MPI routines can be called. This will most likely be in the main program. There are some assumptions about the shape of the passed array.

The default integer is assumed for the array at present!

AUTHOR

Luis Cebamanos, adapted from the code written by David Henty, EPCC

COPYRIGHT

Note that this routine has special Copyright conditions.

```
    !-------------------------------------------------------------------------!
    !                                                                         !
    !  netCDF routine for Fortran Coarrays                                    !
```

```
    !                                                                              !
    !  David Henty, EPCC; d.henty@epcc.ed.ac.uk                                    !
    !                                                                              !
    !  Copyright 2013 the University of Edinburgh                                  !
    !                                                                              !
    !  Licensed under the Apache LICENSE, Version 2.0 (the "LICENSE");             !
    !  you may not use this file except in compliance with the LICENSE.            !
    !  You may obtain a copy of the LICENSE at                                     !
    !                                                                              !
    !      http://www.apache.org/licenses/LICENSE-2.0                              !
    !                                                                              !
    !  Unless required by applicable law or agreed to in writing, software         !
    !  distributed under the LICENSE is distributed on an "AS IS" BASIS,           !
    !  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.    !
    !  See the LICENSE for the specific language governing permissions and         !
    !  limitations under the LICENSE.                                              !
    !                                                                              !
    !------------------------------------------------------------------------------!
```

USES

cgca_m1co (9), MPI library, netCDF library

USED BY

none, end user.

SOURCE

```
integer, parameter :: totdim = 4, arrdim = totdim-1, coardim = 3

integer :: img, nimgs, comm, ierr=0, rank=0, mpisize=0

integer, dimension(totdim) :: asizehal
integer, dimension(arrdim) :: arrsize, arstart, artsize
integer, dimension(coardim) :: coarsize, copos

integer :: ncid, varid, dimids(arrdim)
integer :: x_dimid=0, y_dimid=0, z_dimid=0

  img = this_image()
nimgs = num_images()

asizehal(:) = shape( coarray )
   copos(:) = this_image( coarray )

! Subtract halos
 arrsize(:) = asizehal(1:arrdim) - 2
coarsize(:) = ucobound(coarray) - lcobound(coarray) + 1

! Does the array fit exactly?
if ( product( coarsize ) .ne. nimgs) then
 write(*,*) 'ERROR: cgca_m2netcdf/cgca_pswci3: non-conforming coarray'
 error stop
```

```
end if

comm = MPI_COMM_WORLD
call MPI_Comm_size( comm, mpisize, ierr )
call MPI_Comm_rank( comm, rank, ierr )

! Sanity check
if ( mpisize .ne. nimgs .or. rank .ne. img-1 ) then
  write(*,*) 'ERROR: cgca_m2netcdf/cgca_pswci3: MPI/coarray mismatch'
  error stop
end if


! This is the global arrayy
artsize(:) = arrsize(:) * coarsize(:)

! Correspondent portion of this global array
arstart(:) = arrsize(:) * (copos(:)-1) + 1 ! Use Fortran indexing

! ! debug
! write (*,*) "netCDF-image",img, "asizehal", asizehal, "copos", copos,         &
!  "arrsize", arrsize, "coarsize", coarsize,                                    &
!  "artsize", artsize, "arstart", arstart, "stype", stype

! Create (i.e. open) the netCDF file. The NF90_NETCDF4 flag causes a
! HDF5/netCDF-4 type file to be created. The comm and info parameters
! cause parallel I/O to be enabled.
call check( nf90_create(fname, ior(nf90_netcdf4,nf90_mpiio), ncid, &
       comm=comm, info=MPI_INFO_NULL))


! Define the dimensions. NetCDF returns an ID for each. Any
! metadata operations must take place on ALL processors
call check( nf90_def_dim(ncid, "x", artsize(1), x_dimid) )
call check( nf90_def_dim(ncid, "y", artsize(2), y_dimid) )
call check( nf90_def_dim(ncid, "z", artsize(3), z_dimid) )

! The dimids array is used to pass the ID's of the dimensions of
! the variables.
dimids = (/ x_dimid , y_dimid, z_dimid /)

! Define the variable. The type of the variable in this case is
! NF90_INT (4-byte int).
call check( nf90_def_var(ncid, "data", NF90_INT, dimids, varid) )

! Make sure file it not filled with default values which doubles wrote volume
call check ( nf90_def_var_fill(ncid, varid, 1, 1) )

! End define mode. This tells netCDF we are done defining
! metadata. This operation is collective and all processors will
! write their metadata to disk.
call check( nf90_enddef(ncid) )
```

```
! Parallel access
call check( nf90_var_par_access(ncid, varid, nf90_collective) )

! Write the data to file, start will equal the displacement from the
! start of the file and count is the number of points each proc writes.
call check1( nf90_put_var(ncid, varid, coarray(1:arrsize(1), 1:arrsize(2), 1:arrsize(3),stype), &
     start = arstart, count = arrsize) )

! Close the file. This frees up any internal netCDF resources
! associated with the file, and flushes any buffers.
call check( nf90_close(ncid) )

end subroutine cgca_pswci3

subroutine check(status )
  integer, intent ( in) :: status
  if(status /= nf90_noerr) then
     print *, trim(nf90_strerror(status))
     stop
  end if
end subroutine check

subroutine check1(status)
  integer, intent ( in) :: status
  if(status /= nf90_noerr) then
     print *, "put_var: ",trim(nf90_strerror(status))
     stop
  end if
end subroutine check1
```

19 CGPACK/cgca_m2out

[Modules]

NAME

cgca_m2out

SYNOPSIS

```
!$Id: cgca_m2out.f90 380 2017-03-22 11:03:09Z mexas $

module cgca_m2out
```

DESCRIPTION

Module dealing with output

AUTHOR

Anton Shterenlikht

COPYRIGHT

CONTAINS

Subroutines: cgca_swci (19.2), cgca_fwci (19.1). Submodules: m2out_sm1 (19.3), m2out_sm2_mpi (19.4).

USES

cgca_m1co (9)

USED BY

cgca

SOURCE

```
use cgca_m1co
implicit none

private
public :: cgca_swci, cgca_pc, cgca_pswci, cgca_fwci

interface
  module subroutine cgca_pc( coarray, stype, fname )
    ! In submodule m2out_sm1.f90
    ! coarray - what array to dump
    ! stype - what cell state type to dump
    ! fname - what file name to use
    integer( kind=iarr ), allocatable, intent( in ) ::                &
      coarray(:,:,:,:)[:,:,:]
    integer( kind=idef ),intent( in ) :: stype
    character( len=* ), intent( in ) :: fname
  end subroutine cgca_pc

  module subroutine cgca_pswci( coarray, stype, fname )
```

```
    ! In submodule m2out_sm2_mpi.f90
    ! Parallel Stream Write Coarray of Integers:
    ! - coarray - what array to dump
    ! - stype - what cell state type to dump
    ! - fname - what file name to use
    integer( kind=iarr ), allocatable, intent( in ) ::                &
      coarray(:,:,:,:)[:,:,:]
    integer( kind=idef ),intent( in ) :: stype
    character( len=* ), intent( in ) :: fname
  end subroutine cgca_pswci

end interface

contains
```

19.1 cgca_m2out/cgca_fwci

[cgca_m2out] [Subroutines]

NAME

cgca_fwci

SYNOPSIS

```
subroutine cgca_fwci( coarray, stype, fname )
```

INPUTS

```
integer( kind=iarr ),allocatable,intent( in ) :: coarray(:,:,:,:)[:,:,:]
integer( kind=idef ),intent( in ) :: stype
character( len=* ),intent( in ) :: fname
```

SIDE EFFECTS

A single formatted file is created on image 1 with contents of one layer of the coarray.

DESCRIPTION

Formatted Write Coarray of Integers:

- coarray - what array to dump
- stype - what cell state type to dump
- fname - what file name to use

NOTES

The main purpose of this routine is to provide an easy means for checking whether the results are reproducible. Arguably a formatted output gives a better clue than a binary file. This routine is *very* slow - writing a single cell value per line.

All images call this routine! However only image 1 does all the work. The other images are waiting.

USES

none

USED BY

none, end user.

SOURCE

```
integer :: errstat, coi1, coi2, coi3, i1, i2, i3, funit,              &
  lb(4),   & ! lower bounds   of the coarray
  ub(4),   & ! upper bounds   of the coarray
  lcob(3), & ! lower cobounds of the coarray
  ucob(3)    ! upper cobounds of the coarray

! Only image1 does this. All other images do nothing.
! So sync all probably should be used after a call to
```

```
! this routine in the program.

! Give funit any value, just to avoid compiler warnings
funit = 111

main: if ( this_image() .eq. 1 ) then
  errstat = 0

  ! Assume the coarray has halos. Don't write those.
    lb = lbound( coarray ) + 1
    ub = ubound( coarray ) - 1
  lcob = lcobound( coarray )
  ucob = ucobound( coarray )

  open( newunit=funit, file=fname, form="formatted",                    &
        access="sequential", status="replace", iostat=errstat )
  if ( errstat .ne. 0 ) then
    write (*,'(a,i0)') "ERROR: cgca_fwci/cgca_m2out: " //              &
      "open file for writing: err code: ", errstat
    error stop
  end if

  ! nested loops for writing out from all images
  do coi3 = lcob(3), ucob(3)
  do coi2 = lcob(2), ucob(2)
  do coi1 = lcob(1), ucob(1)
    do i3 = lb(3), ub(3)
    do i2 = lb(2), ub(2)
    do i1 = lb(1), ub(1)
      write( unit=funit, iostat=errstat, fmt="(7(i0,tr1))" )           &
        coi1, coi2, coi3, i1, i2, i3, coarray( i1, i2, i3, stype )     &
                                              [ coi1, coi2, coi3 ]
      if (errstat .ne. 0) then
         write (*,'(a,i0)') "ERROR: cgca_fwci/cgca_m2out: " //         &
           "write: err code: ", errstat
         error stop
      end if
    end do
    end do
    end do
  end do
  end do
  end do

  close( unit=funit, iostat=errstat )
  if ( errstat .ne. 0 ) then
    write (*,'(a,i0)') "ERROR: cgca_fwci/cgca_m2out: " //              &
      "close file: err code: ", errstat
    error stop
  end if

end if main
```

```
end subroutine cgca_fwci
```

19.2 cgca_m2out/cgca_swci

[cgca_m2out] [Subroutines]

NAME

cgca_swci

SYNOPSIS

```
subroutine cgca_swci( coarray, stype, iounit, fname )
```

INPUTS

```
integer( kind=iarr ),allocatable,intent( in ) :: coarray(:,:,:,:)[:,:,:]
integer( kind=idef ),intent( in ) :: stype, iounit
character( len=* ),intent( in ) :: fname
```

SIDE EFFECTS

A single binary file is created on image 1 with contents of coarray.

DESCRIPTION

Stream Write Coarray of Integers:

- coarray - what array to dump
- stype - what cell state type to dump
- iounit - which I/O unit to use
- fname - what file name to use

NOTES

All images call this routine! However only image 1 does all the work. The other images are waiting.

USES

none

USED BY

none, end user.

SOURCE

```
integer :: errstat, coi1, coi2, coi3, i2, i3, &
  lb(4),   & ! lower bounds   of the coarray
  ub(4),   & ! upper bounds   of the coarray
  lcob(3), & ! lower cobounds of the coarray
  ucob(3)    ! upper cobounds of the coarray

! Only image1 does this. All other images do nothing.
! So sync all probably should be used after a call to
! this routine in the program.
```

```
main: if ( this_image() .eq. 1 ) then
  errstat = 0

  ! Assume the coarray has halos. Don't write those.
    lb = lbound( coarray ) + 1
    ub = ubound( coarray ) - 1
  lcob = lcobound( coarray )
  ucob = ucobound( coarray )

!write (*,*) "DEBUG: cgca_swci: lb: " , lb
!write (*,*) "DEBUG: cgca_swci: ub: " , ub
!write (*,*) "DEBUG: cgca_swci: lcob: " , lcob
!write (*,*) "DEBUG: cgca_swci: ucob: " , ucob

  open( unit=iounit, file=fname, form="unformatted", access="stream",  &
        status="replace", iostat=errstat )
  if ( errstat .ne. 0 ) then
    write (*,'(a,i0)') "ERROR: cgca_swci/cgca_m2out: " //              &
      "open file for writing: err code: ", errstat
    error stop
  end if

!write (*,*) "DEBUG: cgca_swci: starting data output"

 ! nested loops for writing in correct order from all images
 do coi3 = lcob(3), ucob(3)
   do i3 = lb(3), ub(3)
     do coi2 = lcob(2), ucob(2)
       do i2 = lb(2), ub(2)
         do coi1 = lcob(1), ucob(1)

  write( unit=iounit, iostat=errstat )                                 &
    coarray( lb(1):ub(1), i2, i3, stype ) [ coi1, coi2, coi3 ]
  if (errstat .ne. 0) then
    write (*,'(a,i0)') "ERROR: cgca_swci/cgca_m2out: " //              &
      "write: err code: ", errstat
    error stop
  end if

!write (*,*) "DEBUG: cgca_swci: wrote cells with: i2, i3, coi1, coi2, coi3", i2, i3, coi1, coi2, coi3

         end do
       end do
     end do
   end do
 end do

!write (*,*) "DEBUG: cgca_swci: finished data output"

 close( unit=iounit, iostat=errstat )
 if ( errstat .ne. 0 ) then
   write (*,'(a,i0)') "ERROR: cgca_swci/cgca_m2out: " //              &
```

```
     "close file: err code: ", errstat
  error stop
 end if

end if main

end subroutine cgca_swci
```

19.3 cgca_m2out/m2out_sm1

[cgca_m2out] [Submodules]

NAME

m2out_sm1

SYNOPSIS

```
!$Id: m2out_sm1.f90 380 2017-03-22 11:03:09Z mexas $

submodule ( cgca_m2out) m2out_sm1
```

DESCRIPTION

Submodule with output routines using Cray parallel coarray IO extensions - basically writing to a shared direct access file.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

CONTAINS

cgca_pc (19.3.1)

USES

Parameters and variables of module cgca_m2out (19) by host association.

USED BY

Module cgca_m2out (19)

SOURCE

```
implicit none

contains
```

19.3.1 m2out_sm1/cgca_pc

[m2out_sm1] [Subroutines]

NAME

cgca_pc

SYNOPSIS

```
module procedure cgca_pc
```

INPUTS

```
!    See the interface block in the parent module cgca_m2out.
```

OUTPUTS

```
!    See the interface block in the parent module cgca_m2out.
```

SIDE EFFECTS

creates a binary file from *all images* and writes coarray to it

DESCRIPTION

Parallel Cray output routine. This routine works only on Cray systems using non-standandard proprietary Cray extension, beware! Refer to Cray "Enhanced I/O: Using the assign Environment", section 13 of Cray Fortran Reference Manual, S-3901-83, June 2014 or later version: http://docs.cray.com/cgi-bin/craydoc.cgi?mode=View;id=S-3901-83

First Cray assign enviroment is established, setting fname for shared IO access. Then image 1 writes the last record in the file. After that all images know the length of the file. Then all images can write their data in the correct place in this file in parallel. The expectation is that this is faster than a serial writer, cgca_swci (19.2).

All images must call this routine! Performance is not guaranteed, use with caution!

USES

none

USED BY

parent module cgca_m2out (19)

SOURCE

```
integer :: errstat=0, img, nimgs, i2, i3, fu, reclen, recnum,          &
  help1, help2, help3,                                                 &
  cosub(3),  & ! set of cosubscripts for this image
  lb(4),     & ! lower bounds   of the coarray
  ub(4),     & ! upper bounds   of the coarray
  spsz(4),   & ! size of the space array along each dimension
  lcob(3),   & ! lower cobounds of the coarray
  ucob(3),   & ! upper cobounds of the coarray
  cosz(3)      ! size of the space coarray along each dimension
```

```
! Assume the coarray has halos. Don't write those.
   lb = lbound( coarray ) + 1
   ub = ubound( coarray ) - 1
 spsz = ub - lb + 1
 lcob = lcobound( coarray )
 ucob = ucobound( coarray )
 cosz = ucob - lcob + 1
  img = this_image()
cosub = this_image( coarray )
nimgs = num_images()

! Initialise Cray assign environment
! -m on - "Special handling of direct access file that will be accessed
!          concurrently by several processes or tasks"
! -F system - no buffering
! fname - this assign enviroment will apply only to file name "fname".
call asnfile( trim(fname), '-m on -F system', errstat )

! Need to set up record length
inquire( iolength=reclen ) coarray( lb(1):ub(1), 1, 1, stype) [1,1,1]
if ( img .eq. 1 ) then
  write (*,*) "INFO: cgca_pc: asnfile errstat:", errstat
  write (*,*) "INFO: cgca_pc: record length:", reclen
  write (*,*) "INFO: cgca_pc: last record num:", spsz(2)*spsz(3)*nimgs
end if

!give fu any value
fu=-2
! open file on image 1, write the last record to it and close it
if ( img .eq. nimgs ) then
  open( newunit=fu, file=trim(fname), access="direct", recl=reclen,    &
        form="unformatted", status="replace" )
  recnum = spsz(2) * spsz(3) * nimgs
  write( fu, rec= recnum ) coarray( lb(1):ub(1), ub(2), ub(3), stype )
  close( fu )
end if

! all images wait until the file size is known
sync all

! open file on all images
open( unit=fu, file=fname, access="direct", recl=reclen,               &
      form="unformatted", status="old" )

! Calculate intermediate variables to reduce the FLOPs
! The exact expression for recnum
!     ( (cosub(3)-1) * spsz(3) + i3 - 1 ) * cosz(2) * spsz(2) * cosz(1) &
!   + ( (cosub(2)-1) * spsz(2) + i2 - 1 ) * cosz(1) + cosub(1)
help3 = (cosub(3)-1) * spsz(3) - 1
help2 = (cosub(2)-1) * spsz(2) - 1
help1 = cosz(2) * spsz(2) * cosz(1)
```

```
! write data
do i3 = lb(3), ub(3)
do i2 = lb(2), ub(2)
  recnum = (help3 + i3) * help1 + (help2 + i2) * cosz(1) + cosub(1)
  write( unit=fu, rec= recnum ) coarray( lb(1):ub(1), i2, i3, stype )
end do
end do

! flush data
flush( unit=fu )

! wait till all images wrote data and flushed
sync all

! close the file
close( fu )

end procedure cgca_pc
```

19.4 cgca_m2out/m2out_sm2_mpi

[cgca_m2out] [Submodules]

NAME

m2out_sm2_mpi

SYNOPSIS

```
!$Id: m2out_sm2_mpi.f90 380 2017-03-22 11:03:09Z mexas $

submodule ( cgca_m2out ) m2out_sm2_mpi
```

DESCRIPTION

Submodule of cgca_m2out (19) dealing with parallel IO using MPI/IO library

AUTHOR

David Henty, modified by Anton Shterenlikht

COPYRIGHT

Note that this routine has special Copyright conditions.

```
!-------------------------------------------------------------------------------!
!                                                                               !
!  MPI-IO routine for Fortran Coarrays                                          !
!                                                                               !
!  David Henty, EPCC; d.henty@epcc.ed.ac.uk                                     !
!                                                                               !
!  Copyright 2013 the University of Edinburgh                                   !
!                                                                               !
!  Licensed under the Apache LICENSE, Version 2.0 (the "LICENSE");              !
!  you may not use this file except in compliance with the LICENSE.             !
!  You may obtain a copy of the LICENSE at                                      !
!                                                                               !
!      http://www.apache.org/licenses/LICENSE-2.0                               !
!                                                                               !
!  Unless required by applicable law or agreed to in writing, software          !
!  distributed under the LICENSE is distributed on an "AS IS" BASIS,            !
!  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.     !
!  See the LICENSE for the specific language governing permissions and          !
!  limitations under the LICENSE.                                               !
!                                                                               !
!-------------------------------------------------------------------------------!
```

CONTAINS

cgca_pswci (19.4.1)

USES

Variables and parameters of the parent module cgca_m2out (19) via host association.

USED BY

The parent module cgca_m2out (19).

SOURCE

```
use mpi
implicit none

contains
```

19.4.1 m2out_sm2_mpi/cgca_pswci

[m2out_sm2_mpi] [Subroutines]

NAME

cgca_pswci

SYNOPSIS

```
!module procedure cgca_pswci
  module subroutine cgca_pswci( coarray, stype, fname )
    ! In submodule m2out_sm2_mpi.f90
    ! Parallel Stream Write Coarray of Integers:
    ! - coarray - what array to dump
    ! - stype - what cell state type to dump
    ! - fname - what file name to use
    integer( kind=iarr ), allocatable, intent( in ) ::                &
      coarray(:,:,:,:)[:,:,:]
    integer( kind=idef ),intent( in ) :: stype
    character( len=* ), intent( in ) :: fname
```

INPUTS

```
!    See the interface block in the parent module cgca_m2out.
```

OUTPUTS

```
!    None
```

SIDE EFFECTS

A single binary file is created using MPI/IO with contents of coarray.

DESCRIPTION

Parallel Stream Write Coarray of Integers:

- coarray - what array to dump
- stype - what cell state type to dump
- fname - what file name to use

NOTES

All images must call this routine!

MPI must be initialised prior to calling this routine, most probably in the main program. Likewise MPI must be terminated only when no further MPI routines can be called. This will most likely be in the main program. There are some assumptions about the shape of the passed array.

The default integer is assumed for the array at present!

AUTHOR

Anton Shterenlikht, adapted from the code written by David Henty, EPCC

USES

cgca_m1co (9), MPI library

USED BY

none, end user.

SOURCE

```
integer, parameter :: totdim = 4, arrdim = totdim-1, coardim = 3

integer :: img, nimgs, comm, ierr=0, rank=0, mpisize=0, filetype,      &
           mpi_subarray, fh, funit
integer, dimension(totdim) :: asizehal
integer, dimension(arrdim) :: arrsize, arstart, artsize
integer, dimension(coardim) :: coarsize, copos
integer( kind=MPI_OFFSET_KIND ) :: disp = 0
integer, dimension(MPI_STATUS_SIZE) :: mpistat

character( len=80 ) :: iomsg

  img = this_image()
nimgs = num_images()

asizehal(:) = shape( coarray )
   copos(:) = this_image( coarray )

! Subtract halos
 arrsize(:) = asizehal(1:arrdim) - 2
coarsize(:) = ucobound(coarray) - lcobound(coarray) + 1

! Does the array fit exactly?
if ( product( coarsize ) .ne. nimgs) then
 write(*,*) 'ERROR: m2out_sm2_mpi/cgca_pswci: non-conforming coarray'
 error stop
end if

comm = MPI_COMM_WORLD
call MPI_Comm_size( comm, mpisize, ierr )
call MPI_Comm_rank( comm, rank, ierr )

! Sanity check
if ( mpisize .ne. nimgs .or. rank .ne. img-1 ) then
  write(*,*) 'ERROR: m2out_sm2_mpi/cgca_pswci: MPI/coarray mismatch'
  error stop
end if

! Define filetype for this process, ie what portion of the global array
! this process owns. Starting positions use C-indexing
! (ie counting from 0).
artsize(:) = arrsize(:) * coarsize(:)
arstart(:) = arrsize(:) * (copos(:)-1)
```

```
! debug
!write (*,*) "image",img, "asizehal", asizehal, "copos", copos,         &
!  "arrsize", arrsize, "coarsize", coarsize,                            &
!  "artsize", artsize, "arstart", arstart, "stype", stype

call MPI_Type_create_subarray( arrdim, artsize, arrsize, arstart,      &
 MPI_ORDER_FORTRAN, MPI_INTEGER, filetype, ierr )
if ( ierr .ne. MPI_SUCCESS ) then
 write (*,*) 'ERROR: m2out_sm2_mpi/cgca_pswci:&
   & MPI_type_create_subarray filetype: rank ', rank
 error stop
end if

call MPI_Type_commit( filetype, ierr )

! Define subarray for this process, ie what portion of the local array
! is to be written (excludes halos); starting positions use C-indexing.

arstart(:) = 1

call MPI_Type_create_subarray( arrdim, asizehal, arrsize, arstart,     &
 MPI_ORDER_FORTRAN, MPI_INTEGER, mpi_subarray, ierr)
if ( ierr .ne. MPI_SUCCESS ) then
 write (*,*) 'ERROR: cgca_pswci: MPI_type_create_subarray&
   & mpi_subarray: rank ', rank
 error stop
end if

call MPI_Type_commit(mpi_subarray, ierr)

!  "Striping information cannot be changed on an existing
! file, so to set the stripe count (and stripe size) for the amount of
! parallelism you want to achieve, the file must be deleted if it exists."
! From: Cray Getting Started on MPI I/O manual, S-2490-40 - Dec 2009:
! http://docs.cray.com/books/S-2490-40/

!if ( rank .eq. 0 ) then
!  call MPI_File_delete( fname, MPI_INFO_NULL, ierr )
!  if ( ierr .ne. MPI_SUCCESS )                                         &
!    error stop "ERROR: cgca_pswci: MPI_file_delete: rank 0"
!end if

! All ranks wait till rank 0 deletes the file
!call MPI_Barrier( comm, ierr )
!if ( ierr .ne. MPI_SUCCESS ) then
!  write (*,*) 'ERROR: cgca_pswci: MPI_file_open: rank ', rank
!  error stop
!end if

! Overwriting MPI/IO files does not involve erasing the file first.
! So if the old file was bigger, the new smaller file will still
! be sized on disk as the old file, with only a part of it overwritten
```

```
! with new data. That would be bad. To avoid this
! image 1 removes all previous contents of this file, if it exists.
rm: if ( img .eq. 1 ) then

  ! this should not be necessary, but Cray issues a caution otherwise
  funit = 0

write (*,*) "DEBUG: fname:", fname

  open( newunit=funit, file=fname, status="replace", iostat=ierr,      &
        iomsg=iomsg )
  if ( ierr .ne. 0 ) then
    write (*,*) "ERROR: m2out_sm2_mpi/cgca_pswci: open( fname ),&
      & iostat=", ierr, "iomsg:", iomsg
    error stop
  end if

  write( funit, * , iostat=ierr) ""
  if ( ierr .ne. 0 ) then
    write (*,*) "ERROR: m2out_sm2_mpi/cgca_pswci: write( fname ),&
      & iostat=", ierr
    error stop
  end if

!  flush( funit, iostat=ierr )
!  if ( ierr .ne. 0 ) then
!    write (*,*) "ERROR: cgca_m2mpiio: flush( fname ), iostat=",ierr
!    error stop
!  end if

  close( funit, iostat=ierr )
  if ( ierr .ne. 0 ) then
    write (*,*) "ERROR: m2out_sm2_mpi/cgca_pswci: close( fname ),&
      & iostat=", ierr
    error stop
  end if

end if rm

! all images wait till image 1 erases the previous file
sync all

!  Open the file for writing only and attach to file handle fh
!  No IO hints are passed since MPI_INFO_NULL is specified
call MPI_File_open( comm, fname, ior(MPI_MODE_WRONLY, MPI_MODE_CREATE),&
                    MPI_INFO_NULL, fh, ierr )
if ( ierr .ne. MPI_SUCCESS ) then
  write (*,*) 'ERROR: m2out_sm2_mpi/cgca_pswci: MPI_file_open: rank ', &
    rank
  error stop
end if
```

```
!  Set view for this process using appropriate datatype
call MPI_File_set_view(                                                 &
 fh, disp, MPI_INTEGER, filetype, 'native', MPI_INFO_NULL, ierr)
if ( ierr .ne. MPI_SUCCESS ) then
  write (*,*) 'ERROR: m2out_sm2_mpi/cgca_pswci: MPI_file_set_view:&
    & rank ', rank
  error stop
end if

! Write all the data for this process.
! Remove halo data by passing an explicit Fortran subarray
call MPI_File_write_all( fh, coarray(:,:,:,stype), 1, mpi_subarray,    &
                         mpistat, ierr )
if ( ierr .ne. MPI_SUCCESS ) then
  write (*,*) 'ERROR: m2out_sm2_mpi/cgca_pswci: MPI_file_write_all:&
    & rank ', rank
  error stop
end if

!  Close file
call MPI_File_close( fh, ierr )
if ( ierr .ne. MPI_SUCCESS ) then
 write (*,*) 'ERROR: m2out_sm2_mpi/cgca_pswci: MPI_file_close: rank ', &
   rank
 error stop
end if

call MPI_Type_free( filetype, ierr )
call MPI_Type_free( mpi_subarray, ierr )

!end procedure cgca_pswci
end subroutine cgca_pswci
```

20 CGPACK/cgca_m2pck

[Modules]

NAME

cgca_m2pck

SYNOPSIS

```
!$Id: cgca_m2pck.f90 380 2017-03-22 11:03:09Z mexas $

module cgca_m2pck
```

DESCRIPTION

Module dealing with checking consistency of the many various global CGPACK parameters set in cgca_m1co (9).

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

CONTAINS

cgca_as (10.2), cgca_ds (10.5), cgca_av (10.3), cgca_dv (10.6), cgca_art (10.1), cgca_drt (10.4)

USES

cgca_m1co (9)

USED BY

cgca

SOURCE

```
use cgca_m1co
implicit none

private
public :: cgca_pdmp

contains
```

20.1 cgca_m2pck/cgca_pdmp

[cgca_m2pck] [Subroutines]

NAME

cgca_pdmp

SYNOPSIS

```
subroutine cgca_pdmp
```

DESCRIPTION

Dump global CGPACK parameters from cgca_m1co (9) to stdout. The user might want to see all values in one place.

USES

cgca_m1co (9)

USED BY

cgca_m2alloc (10)

SOURCE

```
! ifort 16 still does not support this
! Cray apparently has these 2 functions, but *not* in iso_fortran_env!
! So just comment out for now. If really needed, these can be
! added to the main program.
!write (* , "(a,a)" ) "CGPACK compiled with: ", compiler_version()
!write (* , "(a,a)" ) "CGPACK compiler options: ", compiler_options()
write (*,"(a)") "CGPACK cell state types:"
write (*,"(a40,i0)") "cgca_state_type_grain: ", cgca_state_type_grain
write (*,"(a40,i0)") "cgca_state_type_frac: ", cgca_state_type_frac
write (*,*)
write (*,"(a)") "CGPACK grain layer states:"
write (*,"(a40,i0)") "cgca_liquid_state: ", cgca_liquid_state
write (*,*)
write (*,"(a)") "CGPACK fracture layer states:"
write (*,"(a40,i0)") "cgca_state_null: ", cgca_state_null
write (*,"(a40,i0)") "cgca_gb_state_intact: ", cgca_gb_state_intact
write (*,"(a40,i0)") "cgca_gb_state_fractured: ",                    &
                      cgca_gb_state_fractured
write (*,"(a40,i0)") "cgca_intact_state: ", cgca_intact_state
write (*,"(a40,i0)") "cgca_clvg_state_100_flank: ",                  &
                      cgca_clvg_state_100_flank
write (*,"(a40,i0)") "cgca_clvg_state_100_edge: ",                   &
                      cgca_clvg_state_100_edge
write (*,"(a40,i0)") "cgca_clvg_state_110_flank: ",                  &
                      cgca_clvg_state_110_flank
write (*,"(a40,i0)") "cgca_clvg_state_110_edge: ",                   &
                      cgca_clvg_state_110_edge
write (*,"(a40,i0)") "cgca_clvg_state_111_flank: ",                  &
                      cgca_clvg_state_111_flank
write (*,"(a40,i0)") "cgca_clvg_state_111_edge: ",                   &
```

```
                     cgca_clvg_state_111_edge
write (*,"(a40,999(i0,tr1))") "cgca_clvg_states_flank: ",              &
                              cgca_clvg_states_flank
write (*,"(a40,999(i0,tr1))") "cgca_clvg_states_edge: ",               &
                              cgca_clvg_states_edge
write (*,"(a40,999(i0,tr1))") "cgca_clvg_states: ", cgca_clvg_states
write (*,"(a40,999(i0,tr1))") "cgca_frac_states: ", cgca_frac_states
write (*,"(a40,i0)") "cgca_clvg_lowest_state: ", cgca_clvg_lowest_state
write (*,*)
write (*,"(a)") "CGPACK lowest state for both types:"
write (*,"(a40,i0)") "cgca_lowest_state: ", cgca_lowest_state
write (*,*)
write (*,"(a)") "CGPACK kinds:"
write (*,"(a40,i0)") "iarr: ", iarr
write (*,"(a40,i0)") "idef: ", idef
write (*,"(a40,i0)") "ilrg: ", ilrg
write (*,"(a40,i0)") "ldef: ", ldef
write (*,"(a40,i0)") "rdef: ", rdef
write (*,"(a40,i0)") "rlrg: ", rlrg
write (*,*)
write (*,"(a)") "CGPACK other parameters:"
write (*,"(a40,f20.10)") "pi: ", cgca_pi

end subroutine cgca_pdmp
```

21 CGPACK/cgca_m2phys

[Modules]

NAME

cgca_m2phys

SYNOPSIS

```
!$Id: cgca_m2phys.f90 543 2018-04-05 15:08:41Z mexas $

module cgca_m2phys
```

DESCRIPTION

Module dealing with physical units - length and time.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

CONTAINS

cgca_cadim (21.1), cgca_gdim (21.2), cgca_imco (21.3)

USES

cgca_m1co (9)

USED BY

perhaps the user directly?

SOURCE

```
use cgca_m1co, only: idef, iarr, ilrg, rdef, rlrg
implicit none

private
public :: cgca_gdim, cgca_cadim, cgca_imco

contains
```

21.1 cgca_m2phys/cgca_cadim

[cgca_m2phys] [Subroutines]

NAME

cgca_cadim

SYNOPSIS

```
subroutine cgca_cadim( bsz, res, dm, ir, c, lres, ng )
```

INPUT

```
real( kind=rdef ), intent( inout ) :: bsz(3)
real( kind=rdef ), intent( in ) :: res, dm
integer, intent( inout ) :: ir(3)
```

OUTPUT

```
integer, intent( out ) :: c(3), ng
real( kind=rdef ), intent( out ) :: lres
```

SIDE EFFECTS

Arrays bsz and ir are input/output. On exit the values of ir are rearranged. The values of bsz are changed.

DESCRIPTION

Inputs:

- bsz - box size, the size of the CA space in physical units of length. The unit itself is not defined. It's use must be consistent across the whole of CGPACK. In particular, speeds will depend on the choice of the length unit.
- res - the model resolution, cells per grain. Note that this is *not* spatial resolution. The meaning is that RES cells are required to resolve the shape of a grain. This setting should not depend on the grain size.
- dm - the mean grain size, in physical units.
- ir - coarray grid dimensions. The intention is that ir is calculate by a call to cgca_gdim (21.2).

Outputs:

- ir - rearranged coarray grid dimensions
- bsz - new box dimension is calculated, see note 3.
- c - numbers of cells in the space coarray
- lres - linear resolution, cells per unit of length
- ng - number of grains in the whole model

NOTES

1. The space coarray should be declared with something like this

```
space( c(1), c(2), c(3), nlayers )[ir(1),ir(2),*]
```

on all images.

2. An important feature is that the coarray grid dimensions can be rearranged to better suit the physical dimensions of the "box". For example, on input ir(1) is the biggest. However, if the physical dimension of the box is smallest along 1, it makes sense to swap ir(1) with ir(3). This will help achive balanced linear resolution along each dimension.

3. It not generally possible to have the same linear resolution is all 3 directions and to keep the physical size of the box exactly as given. The decision is made here to give preference to the linear resolution. So the algorithm chooses the same linear resolution in all 3 directions. As a consequence, the physical sizes of the box along all 3 directions can be slightly smaller or bigger than given. The biggest deviations probably arise when the shape is very far from cubic.

AUTHOR

Anton Shterenlikht

COPYRIGHT

USES USED BY

via module cgca_m2phys (21)

SOURCE

```
real( kind=rdef ), parameter :: one = 1.0_rdef, third = one/3.0_rdef

real( kind=rdef ) :: dm3  ! dm**3

real( kind=rlrg ) :: bvol ! "box" volume in physical units

integer :: j

! Derived type for easy sorting
type bsz_order
  real( kind=rdef ) :: bsz
  integer :: i
end type bsz_order

type( bsz_order ) :: box(3), tmp

! Assume that ir is already sorted in decreasing order,
! as returned by cgca_gdim. Check this assertion.
if ( ir(1) .lt. ir(2) .or. ir(2) .lt. ir(3) .or. ir(1) .lt. ir(3) ) then
  write (*,*) "ERROR: cgca_m2phys/cgca_cadim: ir is not sorted on entry"
  error stop
end if

! Set the initial descending order, 1 to 3
```

```
do j=1,3
  box(j)%bsz = bsz(j)
  box(j)%i = j
end do

!write (*,*) "ir on entry:", ir
!write (*,*) "box:", box

! Now sort box%bsz in decreasing order and the resulting order
! or box%i is what I need.

if ( box(1)%bsz .lt. box(2)%bsz ) then
     tmp = box(1)
  box(1) = box(2)
  box(2) = tmp
end if

if ( box(2)%bsz .lt. box(3)%bsz ) then
     tmp = box(2)
  box(2) = box(3)
  box(3) = tmp
end if

if ( box(1)%bsz .lt. box(2)%bsz ) then
     tmp = box(1)
  box(1) = box(2)
  box(2) = tmp
end if

! Reassign elements in desired order
ir( box%i ) = ir

!write (*,*) "ir on exit:", ir

! box volume
bvol = product( real( bsz, kind=rlrg ) )

! grain volume
dm3 = dm**3

! number of grains in the whole model, integer
ng = int( bvol/dm3 )

! linear resolution
! res**third      - cells per mean grain size length, dm
! res**third / dm - cells per unit length
lres = res**third / dm

! numbers of cells
! res**third/dm * bsz(i)       - cells per box length along i
! res**third/dm * bsz(i)/ir(i) - cells per image along i
c = nint( lres * bsz/ir )
```

```
! cannot have zero as the array dimension
if ( c(1) .eq. 0 ) c(1) = 1
if ( c(2) .eq. 0 ) c(2) = 1
if ( c(3) .eq. 0 ) c(3) = 1

! warn the user, the box is likely to be very different
! from the input values.
if ( any( c .eq. 1 ) ) then
  write (*,"(a)")                                                    &
   "WARN: cgca_cadim: the new box sizes are probably wrong, check"
end if

! new box size
bsz = real( ir*c, kind=rdef ) / lres

end subroutine cgca_cadim
```

21.2 cgca_m2phys/cgca_gdim

[cgca_m2phys] [Subroutines]

NAME

cgca_gdim

SYNOPSIS

```
subroutine cgca_gdim( n, ir, qual )
```

INPUT

```
integer( kind=idef ), intent( in ) :: n
```

OUTPUT

```
integer( kind=idef ), intent( out ) :: ir(3)
real( kind=rdef ), intent( out ) :: qual
```

SIDE EFFECTS

None

DESCRIPTION

The purpose of this routine is to find 3 coarray grid dimensions, ir(1) >= ir(2) >= ir(3), such that for a given number of images the coarray grid is as "cubic" as possible. In mathematical terms the aim is to find F = min(max(ir(1) - ir(3))). The quality of this minimum is defined as QUAL=1-F/(N-1). Inputs:

- N is the total number of images, num_images().

Outputs:

- ir - the array of 3 coarray grid dimensions.
- qual - is the quality of the fitted grid. qual=1 means F=0, i.e. the coarray grid is a cube. qual=0 means F=N-1, i.e. the coarray grid is 1D, i.e. [N,1,1].

The outputs of this routine, ir, are used to choose the dimensions of space coarray. The QUAL output is for information only.

NOTES

If N<1 is given, then the routine returns immediately with QUAL=-1. So the caller can/should check for this condition.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES USED BY

Supposed to be called prior to calling cgca_cadim (21.1)

SOURCE

```
real( kind=rdef ), parameter :: one = 1.0_rdef, zero = 0.0_rdef, &
  third = one/3.0_rdef
integer( kind=idef) :: i, j, k, f, ftrial

! default output
  ir(1) = n
ir(2:3) = 1
   qual = zero

! return immediately if n=1
if ( n .eq. 1 ) return

! return with QUAL=-1 if N<1
if ( n .lt. 1 ) then
  qual = -one
  return
end if

! Set the initial value of the objective function
f = n-1

loopi1: do i = n/2, int( real(n)**third ), -1
          if ( mod( n,i ) .ne. 0 ) cycle loopi1
loopi2:   do j = n/i, 2, -1
            if ( j .gt. i ) cycle loopi2
            if ( mod( n,(i*j) ) .ne. 0 ) cycle loopi2
            k = n/(i*j)
            if ( k .gt. j ) cycle loopi2
            ftrial = i-k
            if ( ftrial .ge. f ) cycle loopi2
            f = ftrial
            ir(1) = i
            ir(2) = j
            ir(3) = k
            if ( f .eq. 0 ) exit loopi1
          end do loopi2
        end do loopi1

qual = one - real(f)/(n-one)

end subroutine cgca_gdim
```

21.3 cgca_m2phys/cgca_imco

[cgca_m2phys] [Subroutines]

NAME

cgca_imco

SYNOPSIS

```
subroutine cgca_imco( space, lres, bcol, bcou )
```

INPUT

```
integer( kind=iarr ), allocatable, intent( in ) :: &
 space(:,:,:,:)[:,:,:]
real( kind=rdef ), intent( in ) :: lres
```

INPUTS

```
!    - space - the model coarray. Error will result if it's
!      not allocated. The array is used only to calculate the
!      position of this image within the coarray grid, as
!       this_image( space )
!    - lres - linear resolution of the space cooarray,
!      cells per unit of length.
```

OUTPUT

```
real( kind=rdef ), intent( out ) :: bcol(3), bcou(3)
```

OUTPUTS

```
!    - bcol - physical coordinates of the lower bounds of the coarray
!      on this image in CA CS.
!    - bcou - physical coordinates of the upper bounds of the coarray
!      on this image in CA CS.
```

SIDE EFFECTS

none this image index Size of the space array. Remember that space array has 1 halo cell on each boundary along each dimension. Don't count those: vectors in CA CS Make sure there are no gaps between the upper and the next lower boundary. The gap equals to the physical size of a single cell. So add half a cell size to the upper boundary and subtract half a cell size from the lower boundary.

DESCRIPTION

IMCO stands for IMage COordinates. This routine calculates the lower and the upper physical coordinates of the coarray on this image in CA CS.

AUTHOR

Anton Shterenlikht

COPYRIGHT

22 CGPACK/cgca_m2red

[Modules]

NAME

cgca_m2red

SYNOPSIS

```
!$Id: cgca_m2red.f90 380 2017-03-22 11:03:09Z mexas $

module cgca_m2red
```

DESCRIPTION

Module dealing with collective reduction operations, including all required image syncronisation.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

CONTAINS

cgca_redand (22.1)

USES

cgca_m1co (9)

USED BY

cgca

SOURCE

```
use cgca_m1co
implicit none

private
public :: cgca_redand

contains
```

22.1 cgca_m2red/cgca_redand

[cgca_m2red] [Subroutines]

NAME

cgca_redand

SYNOPSIS

```
subroutine cgca_redand(coarray,p)
```

INPUTS

```
logical(kind=ldef),intent(inout) :: coarray[*]
integer(kind=idef),intent(in) :: p
```

SIDE EFFECTS

coarray values change

DESCRIPTION

This routine does collective AND operation over coarray values across all images. The result is returned in coarray on every image. The result is TRUE if coarray values on all images are TRUE, and FALSE otherwise. The algorithm implements a divide and conquer scheme that works only when the number of images, n, is a power of 2 - n=2**p. p is the input to this routine.

If the number of images is 2**p, then reduction takes p iterations. In this example I have 2**4=16, so it takes 4 iterations.

```
    img1 img2 img3 img4 img5 img6 img7 img8 img9 img10 img11 img12 img13 img14 img15 img16
 1. img1 _/   img3 _/   img5 _/   img7 _/   img9 _/    img11 _/    img13 _/    img15 _/
 2. img1 ______/        img5 ______/        img9 _______/          img13 _______/
 3. img1 _________________/                 img9 ___________________/
 4. img1 _____________________________________/
    img1
```

NOTE

For efficiency no check is made that n = 2**p. This check must be made in the calling routine or the main program.

USES

none

USED BY

cgca_m2red (22)

SOURCE

```
integer(kind=idef) :: i, img, step, stepold

img = this_image()

step    = 2
```

```
stepold = 1

! do the reduction

redu: do i = 1,p

 if (mod(img,step)-1 .eq. 0) then
  sync images (img+stepold)
  coarray = coarray .and. coarray[img+stepold]
 else if (mod(img+stepold,step)-1 .eq. 0) then
  sync images (img-stepold)
 end if

 stepold = step
 step = step * 2

end do redu

! now send the result, which is in z[1] to all images.
! all images wait for image 1, so can use sync images(*),
! but, as the standard suggests, sync images is probably faster.

sync all

coarray = coarray[1]

end subroutine cgca_redand
```

23 CGPACK/cgca_m2rnd

[Modules]

NAME

cgca_m2rnd

SYNOPSIS

```
!$Id: cgca_m2rnd.f90 380 2017-03-22 11:03:09Z mexas $

module cgca_m2rnd
```

DESCRIPTION

Module dealing with random number generation

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

CONTAINS

cgca_irs (23.2), cgca_ins (23.1)

USES

cgca_m1co (9)

USED BY

cgca

SOURCE

```
use cgca_m1co, only: ldef
implicit none

private
public :: cgca_irs, cgca_ins

contains
```

23.1 cgca_m2rnd/cgca_ins

[cgca_m2rnd] [Subroutines]

NAME

cgca_irs (23.2)

SYNOPSIS

```
subroutine cgca_ins(debug)
```

INPUT

```
logical( kind=ldef ), intent(in) :: debug
```

SIDE EFFECTS

Initialise the seed based only on the image number.

DESCRIPTION

This routine sets reproducible random seeds on each image. If the number of images is kept constant, then the results of the CGPACK simulation should be (need to prove this rigorously!) reproducible. If some changes to the code are not supposed to change the results, use this random seed routine.

USES

none

USED BY

none, end user

SOURCE

```
integer :: i, n, errstat=0
integer, allocatable :: seed(:)

call random_seed( size = n )

allocate( seed(n), stat=errstat )
if ( errstat .ne. 0 ) then
  write (*,*) "ERROR: cgca_ins/cgca_m2rnd: allocate( seed )," //       &
              " err. stat:", errstat
  error stop
end if

! Set seed to this_image() related values.
seed = this_image() * (/ (i, i=1, n) /)
call random_seed(put = seed)

! Don't need to do this, but just to be double sure, read
! the seed back from the subroutine
call random_seed(get = seed)

! Debug output
```

```
if (debug) write (*, "(3(a,i0),9999(tr1,i0))" )                          &
  "DEBUG: cgca_ins/cgca_m2rnd: this_image(): ", this_image(),            &
  " size: ", n, " seed: ", seed

deallocate( seed, stat=errstat )
if ( errstat .ne. 0 ) then
  write (*,*) "ERROR: cgca_ins/cgca_m2rnd: deallocate( seed )," //       &
              " err. stat:", errstat
  error stop
end if

end subroutine cgca_ins
```

23.2 cgca_m2rnd/cgca_irs

[cgca_m2rnd] [Subroutines]

NAME

cgca_irs

SYNOPSIS

```
subroutine cgca_irs(debug)
```

INPUT

```
logical(kind=ldef),intent(in) :: debug
```

SIDE EFFECTS

initialise random seed on all images

DESCRIPTION

Initialise random seed based on system_clock intrinsic, adapted from: http://gcc.gnu.org/onlinedocs/gfortran/RANDOM_005f
Note that the seed is based on THIS_IMAGE intrinsic, hence each image uses a different seed.

USES

none

USED BY

none, end user

SOURCE

```
integer :: i, n, clock, errstat=0
integer, allocatable :: seed(:)

call random_seed( size = n )

allocate( seed(n), stat=errstat )
if ( errstat .ne. 0 ) then
  write (*,*) "ERROR: cgca_irs/cgca_m2rnd: allocate( seed )," //       &
              " err. stat:", errstat
  error stop
end if

call system_clock(count=clock)

seed = int(real(clock)/real(this_image())) +                          &
  999999937*(/ (i - 1, i = 1, n) /)

call random_seed(put = seed)

! Debug output
if (debug) write (*,*) "DEBUG: cgca_irs/cgca_m2rnd: this_image():",    &
  this_image(), "; size:", n, "; seed:", seed
```

```
deallocate( seed, stat=errstat )
if ( errstat .ne. 0 ) then
  write (*,*) "ERROR: cgca_irs/cgca_m2rnd: deallocate( seed )," //     &
              " err. stat:", errstat
  error stop
end if

end subroutine cgca_irs
```

24 CGPACK/cgca_m2rot

[Modules]

NAME

cgca_m2rot

SYNOPSIS

```
!$Id: cgca_m2rot.f90 526 2018-03-25 23:44:51Z mexas $

module cgca_m2rot
```

DESCRIPTION

Module dealing with tensor rotations, orientations, mis-orientations

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

CONTAINS

cgca_rt (24.6), cgca_ckrt (24.1), rtprint (24.7), cgca_csym (24.2), cgca_csym_pure (24.3), cgca_mis (24.4), cgca_miscsym (24.5)

USES

cgca_m1co (9)

USED BY

cgca

SOURCE

```
use cgca_m1co, only : idef, ldef, rdef, cgca_pi
implicit none

private
public :: cgca_ckrt, cgca_csym, cgca_csym_pure, cgca_mis, &
  cgca_miscsym, cgca_rt

contains
```

24.1 cgca_m2rot/cgca_ckrt

[cgca_m2rot] [Subroutines]

NAME

cgca_ckrt

SYNOPSIS

```
subroutine cgca_ckrt(r,debug,flag)
```

INPUTS

```
real(kind=rdef),intent(in) :: r(3,3)
logical(kind=ldef),intent(in) :: debug
```

OUTPUT

```
integer(kind=idef),intent(out) :: flag
```

DESCRIPTION

Check that the given rotation tensor r(3,3) is orthogonal. If debug .eq. .true., then verbose diagnostics is printed on error. In this case a non-zero return flag shows the number of the failed test.

USES

rtprint (24.7)

USED BY

none, end user

SOURCE

```
real(kind=rdef),parameter :: one=1.0_rdef,factor=1.0e1_rdef
integer :: i
real :: maxerr, orthogon(3,3), tmp(3,3)

maxerr = factor * epsilon(one)

flag = 0
tmp = transpose(r)
 do i=1,2
   if ( i .eq. 1 ) orthogon = matmul( r, tmp )
   if ( i .eq. 2 ) orthogon = matmul( tmp, r )

   if (abs(orthogon(1,1)-one) .gt. maxerr) then
     if (debug) call rtprint( r, tmp, orthogon, maxerr )
     flag = 1 * i
     return
   end if

   if (abs(orthogon(1,2))     .gt. maxerr) then
     if (debug) call rtprint( r, tmp, orthogon, maxerr )
```

```
      flag = 2 * i
      return
    end if

    if (abs(orthogon(1,3))     .gt. maxerr) then
      if (debug) call rtprint( r, tmp, orthogon, maxerr )
      flag = 3 * i
      return
    end if

    if (abs(orthogon(2,1))     .gt. maxerr) then
      if (debug) call rtprint( r, tmp, orthogon, maxerr )
      flag = 4 * i
      return
    end if

    if (abs(orthogon(2,2)-one) .gt. maxerr) then
      if (debug) call rtprint( r, tmp, orthogon, maxerr )
      flag = 5 * i
      return
    end if

    if (abs(orthogon(2,3))     .gt. maxerr) then
      if (debug) call rtprint( r, tmp, orthogon, maxerr )
      flag = 6 * i
      return
    end if

    if (abs(orthogon(3,1))     .gt. maxerr) then
      if (debug) call rtprint( r, tmp, orthogon, maxerr )
      flag = 7 * i
      return
    end if

    if (abs(orthogon(3,2))     .gt. maxerr) then
      if (debug) call rtprint( r, tmp, orthogon, maxerr )
      flag = 8 * i
      return
    end if

    if (abs(orthogon(3,3)-one) .gt. maxerr) then
      if (debug) call rtprint( r, tmp, orthogon, maxerr )
      flag = 9 * i
      return
    end if

  end do

end subroutine cgca_ckrt
```

24.2 cgca_m2rot/cgca_csym

[cgca_m2rot] [Subroutines]

NAME

cgca_csym

SYNOPSIS

```
subroutine cgca_csym( num, rs )
```

INPUT

```
integer, intent( in ) :: num
```

OUTPUT

```
real( kind=rdef ), intent( out ) :: rs(3,3)
```

DESCRIPTION

This routine stores, and outputs on demand, symmetry rotation tensors, for cubic crystals, 24 in total. The first tensor is the unit tensor (trivial case). Then there are 23 non-trivial tensors. There are 24*3*3=216 elements in total:

- num - rotation symmetry tensor number
- rs - rotation symmetry tensor

NOTES

The symmetry tensors and the misorientation angle equations can be found in "Introduction to texture analysis : macrotexture, microtexture, and orientation mapping", Olaf Engler, Valerie Randle, 2nd ed, Boca Raton, CRC Press, 2010, 456 p.

There is a copy in QBL.

USES

none

USED BY

cgca_miscsym (24.5)

SOURCE

```
real( kind=rdef ), parameter :: mnsone = -1.0_rdef,      &
                                zero =  0.0_rdef,      &
                                 one =  1.0_rdef

! Keep the identity tensor as well, at number zero, if required in future.
! Data are filled columns first!

real( kind=rdef ), parameter :: r(3,3,24) = reshape(     &
(/ one,zero,zero,    zero,one,zero,    zero,zero,one,    &
```

```
   one,zero,zero,    zero,zero,mnsone, zero,one,zero,    &
   one,zero,zero,    zero,mnsone,zero, zero,zero,mnsone, &

   one,zero,zero,    zero,zero,one,    zero,mnsone,zero, &
   zero,zero,one,    zero,one,zero,    mnsone,zero,zero, &
   mnsone,zero,zero, zero,one,zero,    zero,zero,mnsone, &

   zero,zero,mnsone, zero,one,zero,    one,zero,zero,    &
   zero,mnsone,zero, one,zero,zero,    zero,zero,one,    &
   mnsone,zero,zero, zero,mnsone,zero, zero,zero,one,    &
   zero,one,zero,    mnsone,zero,zero, zero,zero,one,    &

   zero,zero,one,    one,zero,zero,    zero,one,zero,    &
   mnsone,zero,zero, zero,zero,one,    zero,one,zero,    &
   zero,zero,mnsone, mnsone,zero,zero, zero,one,zero,    &

   zero,mnsone,zero, zero,zero,mnsone, one,zero,zero,    &
   mnsone,zero,zero, zero,zero,mnsone, zero,mnsone,zero, &
   zero,one,zero,    zero,zero,mnsone, mnsone,zero,zero, &

   zero,zero,one,    zero,mnsone,zero, one,zero,zero,    &
   zero,zero,mnsone, zero,mnsone,zero, mnsone,zero,zero, &
   zero,one,zero,    one,zero,zero,    zero,zero,mnsone, &
   zero,mnsone,zero, mnsone,zero,zero, zero,zero,mnsone, &

   zero,zero,one,    mnsone,zero,zero, zero,mnsone,zero, &
   zero,zero,mnsone, one,zero,zero,    zero,mnsone,zero, &
   zero,mnsone,zero, zero,zero,one,    mnsone,zero,zero, &
   zero,one,zero,    zero,zero,one,    one,zero,zero /), &
   (/3,3,24/) )

! sanity check

if (num .lt. 1 .or. num .gt. 24) then
  write (*,'(a,i0)') "ERROR: cgca_csym: image: ", this_image()
  write (*,'(a)') "ERROR: cgca_csym: num is out of range [1..24]"
  error stop
end if

! simply return the required rotation tensor

rs = r(:,:,num)

end subroutine cgca_csym
```

24.3 cgca_m2rot/cgca_csym_pure

[cgca_m2rot] [Subroutines]

NAME

cgca_csym_pure

SYNOPSIS

```
pure subroutine cgca_csym_pure( num, rs, flag )
```

INPUT

```
!    - num - rotation symmetry tensor number

integer( kind=idef ), intent(in) :: num
```

OUTPUT

```
!    - rs - rotation symmetry tensor

real( kind=rdef ), intent(out) :: rs(3,3)
integer, intent(out) :: flag
```

DESCRIPTION

This routine stores, and outputs on demand, symmetry rotation tensors, for cubic crystals, 24 in total. The first tensor is the unit tensor (trivial case). Then there are 23 non-trivial tensors. There are 24*3*3=216 elements in total. The symmetry tensors and the misorientation angle equations can be found in "Introduction to texture analysis : macrotexture, microtexture, and orientation mapping", Olaf Engler, Valerie Randle, 2nd ed, Boca Raton, CRC Press, 2010, 456 p.

NOTES

This is a PURE subroutine. Hence no external IO is required. Hence I'll report back the errors via a flag:

```
      0 - sucessful completion.
      1 - number is out of range [1..24].
```

USES

none

USED BY

cgca_miscsym (24.5)

SOURCE

```
real( kind=rdef ), parameter :: mnsone = -1.0_rdef,      &
                                zero =  0.0_rdef,      &
                                 one =  1.0_rdef

! Keep the identity tensor as well, at number zero,
```

```
! if required in future. Data are filled columns first!

real( kind=rdef ), parameter :: r(3,3,24) = reshape(     &
(/ one,zero,zero,    zero,one,zero,    zero,zero,one,    &
   one,zero,zero,    zero,zero,mnsone, zero,one,zero,    &
   one,zero,zero,    zero,mnsone,zero, zero,zero,mnsone, &

   one,zero,zero,    zero,zero,one,    zero,mnsone,zero, &
   zero,zero,one,    zero,one,zero,    mnsone,zero,zero, &
   mnsone,zero,zero, zero,one,zero,    zero,zero,mnsone, &

   zero,zero,mnsone, zero,one,zero,    one,zero,zero,    &
   zero,mnsone,zero, one,zero,zero,    zero,zero,one,    &
   mnsone,zero,zero, zero,mnsone,zero, zero,zero,one,    &
   zero,one,zero,    mnsone,zero,zero, zero,zero,one,    &

   zero,zero,one,    one,zero,zero,    zero,one,zero,    &
   mnsone,zero,zero, zero,zero,one,    zero,one,zero,    &
   zero,zero,mnsone, mnsone,zero,zero, zero,one,zero,    &

   zero,mnsone,zero, zero,zero,mnsone, one,zero,zero,    &
   mnsone,zero,zero, zero,zero,mnsone, zero,mnsone,zero, &
   zero,one,zero,    zero,zero,mnsone, mnsone,zero,zero, &

   zero,zero,one,    zero,mnsone,zero, one,zero,zero,    &
   zero,zero,mnsone, zero,mnsone,zero, mnsone,zero,zero, &
   zero,one,zero,    one,zero,zero,    zero,zero,mnsone, &
   zero,mnsone,zero, mnsone,zero,zero, zero,zero,mnsone, &

   zero,zero,one,    mnsone,zero,zero, zero,mnsone,zero, &
   zero,zero,mnsone, one,zero,zero,    zero,mnsone,zero, &
   zero,mnsone,zero, zero,zero,one,    mnsone,zero,zero, &
   zero,one,zero,    zero,zero,one,    one,zero,zero /), &
   (/3,3,24/) )

! Set to zero initially
rs = zero

! sanity check
if (num .lt. 1 .or. num .gt. 24) then
  flag = 1
  return
end if

! If there are no error conditions, simply return the
! required rotation tensor and set the flag to 0.

flag = 0
  rs = r( :, :, num )

end subroutine cgca_csym_pure
```

24.4 cgca_m2rot/cgca_mis

[cgca_m2rot] [Subroutines]

NAME

cgca_mis

SYNOPSIS

```
subroutine cgca_mis(r1,r2,angle)
```

INPUTS

```
!real(kind=rdef),intent(in) :: r1(3,3),r2(3,3)
real(kind=rdef),intent(in) :: r1(:,:),r2(:,:)
```

OUTPUT

```
real(kind=rdef),intent(out) :: angle
```

DESCRIPTION

This routine calculates grain misoreintation. Given 2 orientation tensors, r1 and r2, the misorientation angle (in rad) is: acos((tr(r1*r2$\hat{t}$)-1)/2), where "tr" is tensor trace. The angle is [0,pi (9.30)].

USES

none

USED BY

cgca_miscsym (24.5)

SOURCE

```
real(kind=rdef),parameter :: one=1.0_rdef
real(kind=rdef) :: misor(3,3),trace,arg

misor=matmul(r1,transpose(r2))
trace=misor(1,1)+misor(2,2)+misor(3,3)
arg = (trace-one) / 2

if (arg .gt.  one) arg= one
if (arg .lt. -one) arg=-one

angle = acos(arg)

if (isnan(angle)) then
  write (*,'(a,i0)') "ERROR: cgca_mis: image: ", this_image()
  write (*,'(a,i0)') "ERROR: cgca_mis: arg: ", arg
  write (*,'(a,i0)') "ERROR: cgca_mis: angle=acos(arg) is NAN"
  error stop
end if

end subroutine cgca_mis
```

24.5 cgca_m2rot/cgca_miscsym

[cgca_m2rot] [Subroutines]

NAME

cgca_miscsym

SYNOPSIS

```
subroutine cgca_miscsym(r1,r2,minang)
```

INPUTS

```
!real(kind=rdef),intent(in) :: r1(3,3),r2(3,3)
real(kind=rdef),intent(in) :: r1(:,:),r2(:,:)
```

OUTPUT

```
real(kind=rdef),intent(out) :: minang
```

DESCRIPTION

This routine calculates the grain misorientation. angle, taking cubic symmetry into account. Given 2 orientation tensors, r1 and r2, the misorientation angle (in rad) is: acos((tr(r1*r2$\hat{t}$)-1)/2), where "tr" is tensor trace. the angle is [0,pi (9.30)].

USES

cgca_csym (24.2), cgca_mis (24.4)

USED BY SOURCE

```
real(kind=rdef) :: rot(3,3), angle, tmp(3,3)
integer :: i

minang = 1.0e1 ! any number .gt. pi will do
angle  = 0.0

do i=1,24
  call cgca_csym(i,rot)
  tmp = matmul(rot,r2)
  !call cgca_mis(r1,matmul(rot,r2),angle)
  call cgca_mis(r1,tmp,angle)
  if (angle .lt. minang) minang=angle
end do

end subroutine cgca_miscsym
```

24.6 cgca_m2rot/cgca_rt

[cgca_m2rot] [Subroutines]

NAME

cgca_rt

SYNOPSIS

```
subroutine cgca_rt(r)
```

INPUT

```
!    Array r to store output
```

OUTPUT

```
real(kind=rdef),allocatable,intent(inout) :: r(:,:,:)[:,:,:]
```

DESCRIPTION

Choose grain rotation tensors at random and return in a coarray r(:,:,:)[:,:,:]. dimension 1 of the coarray is the grain number. dimensions 2 and 3 are for the rotation tensor for the grain, e.g. r(387,2,1) is the rotation tensor component r21 for grain 387.

Image 1 assigns rotation tensors to all grains. then all other images copy the coarray from image 1.

The method: choose 3 random angles. Then interpret them as rotations about axes 1, 2 and 3 in turn. The resulting rotation will be quite random.

Grain axes are the crystallographic directions:

```
axis  direction
  1     [100]
  2     [010]
  3     [001]
```

Then the grain orientation tensor for some grain z is defined as:

```
r(z,1,1) r(z,1,2) r(z,1,3)
r(z,2,1) r(z,2,2) r(z,2,3)
r(z,3,1) r(z,3,2) r(z,3,3)
```

Index 2 is the grain axis, index 3 is the space axis, i.e. r(z,i,j)= cos(x_i_grain,x_j_space), e.g. r(z,3,2) = cos(3_grain,2_space) is the angle between grain axis 3 and the space axix 2 for grain z.

This leads to the important convention:

```
x_grain =           r  * x_space
x_space = transpose(r) * x_grain
```

All other routines must adhere to this convention.

NOTES

Call cgca_art (10.1) before calling this routine.

USES

none

USED BY

none, end user

SOURCE

```
real( kind=rdef ), parameter :: zero = 0.0_rdef, one = 1.0_rdef,       &
  eight=8.0_rdef, twopi = 2 * cgca_pi
real(kind=rdef) :: a(3),q(3,3)
real :: rndnum(3)

integer(kind=idef) :: l(3),u(3),lcob(3),ucob(3),i

! check for allocated
if (.not. allocated(r)) error stop "error in cgca_rt: r is not allocated"

! define cobounds on all images. these are used at the end
! to read the values from image 1.
lcob=lcobound(r)
ucob=ucobound(r)

! Image 1 does all work, otherwise lots of syncs will be needed
image1: if (this_image() .eq. 1) then

! coarray bounds
l=lbound(r)
u=ubound(r)

! check that bounds for dimensions 2 and 3 are 1:3
if (l(2) .ne. 1 .or. l(3) .ne. 1 .or. &
    u(2) .ne. 3 .or. u(3) .ne. 3) then
  write (*,'(a)') &
    "error in cgca_rt: coarray bounds along dimensions 2 or 3 are wrong."
  write (*,'(a)') &
    "error in cgca_rt: these must be [1:3]."
  error stop
end if

! loop over all grains
main: do i=l(1),u(1)

call random_number(rndnum)         ! in [0,1)
a=real(rndnum,kind=rdef)*twopi     ! a in [0,2*pi)

! rotation about 1
 r(i,:,:)=zero
 r(i,1,1)=one
 r(i,2,2)=cos(a(1))
 r(i,2,3)=sin(a(1))
```

```
 r(i,3,2)=-r(i,2,3)
 r(i,3,3)=r(i,2,2)

! rotation about 2
 q=zero
 q(1,1)=cos(a(2))
 q(1,3)=sin(a(2))
 q(2,2)=one
 q(3,1)=-q(1,3)
 q(3,3)=q(1,1)

! intermediate compound rotation
 r(i,:,:) = matmul(q,r(i,:,:))

! rotation about 3
 q=zero
 q(1,1)=cos(a(3))
 q(1,2)=sin(a(3))
 q(2,1)=-q(1,2)
 q(2,2)=q(1,1)
 q(3,3)=one

! final compound rotation
 r(i,:,:) = matmul(q,r(i,:,:))

end do main

end if image1

! global sync here
sync all

! all images read r from image 1
r(:,:,:) = r(:,:,:)[lcob(1),lcob(2),lcob(3)]

! exit only after all images have the rotation tensors assigned.
sync all

end subroutine cgca_rt
```

24.7 cgca_m2rot/rtprint

[cgca_m2rot] [Subroutines]

NAME

rtprint

SYNOPSIS

```
subroutine rtprint(a,b,c,err)
```

INPUT

```
real(kind=rdef),intent(in) :: a(3,3),b(3,3),c(3,3),err
```

SIDE EFFECTS

dumps some text on stdout

DESCRIPTION

This routine prints on stdout the details of the rotation tensor that failed one of the tests. It prints the tensor itself, as a matrix, the transposed tensor, and then a product.

NOTES

This routine is not accessible from outside of module cgca_m2rot (24).

USES

none

USED BY

cgca_ckrt (24.1)

SOURCE

```
integer :: i

write (*,'(a)') "troublesome tensor:"
do i=1,3
  write (*,*) a(i,:)
end do

write (*,*)

write (*,'(a)') "transposed troublesome tensor:"
do i=1,3
  write (*,*) b(i,:)
end do

write (*,*)
write (*,'(a)') "this should have been the unit tensor:"
do i=1,3
  write (*,*) c(i,:)
end do
```

```
write (*,*)"max allowed error was:", err

end subroutine rtprint
```

25 CGPACK/cgca_m2stat

[Modules]

NAME

cgca_m2stat

SYNOPSIS

```
!$Id: cgca_m2stat.f90 380 2017-03-22 11:03:09Z mexas $

module cgca_m2stat
```

DESCRIPTION

Module with various statistical routines: grain volumes, fracture volumes, etc.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

CONTAINS

cgca_gv (25.2), cgca_gvl (25.3), cgca_fv (25.1)

USES

cgca_m1co (9)

USED BY

cgca

SOURCE

```
use cgca_m1co
implicit none

private
public :: cgca_fv, cgca_gv, cgca_gvl

contains
```

25.1 cgca_m2stat/cgca_fv

[cgca_m2stat] [Subroutines]

NAME

cgca_fv

SYNOPSIS

```
subroutine cgca_fv( coarray, fv )
```

INPUTS

```
integer( kind=iarr ), intent( inout ), allocatable ::                  &
 coarray( : , : , : , : ) [ : , : , : ]
real( kind=rdef ) , intent( out ) :: fv
```

SIDE EFFECTS

None

DESCRIPTION

This routine analyses the fracture layer of the coarray, i.e. coarray(: , : , : , cgca_state_type_frac (9.23)). It calculates the number (volume) of failed (fractured) cells. Cells of states cgca_frac_states (9.12) are considered failed.

NOTES

This routine can be called by and and all images.

SOURCE

```
integer, parameter :: frsize = size( cgca_frac_states )
integer :: lb(4), ub(4), i
integer( kind=ilrg) :: icount
real( kind=rdef) :: counter( frsize ) = 0.0e0

! don't forget the halo cells!
lb = lbound( coarray ) + 1
ub = ubound( coarray ) - 1

do i = 1 , frsize
  icount = count( coarray( lb(1):ub(1) , lb(2):ub(2) , lb(3):ub(3) ,   &
    cgca_state_type_frac ) .eq. cgca_frac_states(i), kind=ilrg )
  counter( i ) = counter( i ) + real( icount, kind=rdef )
end do

fv = sum( counter )

end subroutine cgca_fv
```

25.2 cgca_m2stat/cgca_gv

[cgca_m2stat] [Subroutines]

NAME

cgca_gv

SYNOPSIS

```
subroutine cgca_gv(coarray,gv)
```

INPUTS

```
integer(kind=iarr),allocatable,intent(in) :: coarray(:,:,:,:)[:,:,:]
integer(kind=ilrg),allocatable,intent(inout) :: gv(:)[:,:,:]
```

SIDE EFFECTS

The state of gv array changes

DESCRIPTION

This routine does grain volume calculation. For each cell (i,j,k) in coarray, the routine increments gv(coarray(i,j,k)).

NOTES

All images must call this routine!

There are several SYNC ALL barriers, because all images must get the updated gv array. It is possible (probable?) that there's too much syncronisation, leading to poor performance. This should be investigated at depth.

SOURCE

```
integer(kind=ilrg),allocatable :: gvimg1(:)
integer(kind=ilrg) :: imagevol
integer :: errstat, i1, i2, i3, &
  lbr(4)      ,& ! lower bounds of the "real" coarray, lower virtual+1
  ubr(4)      ,& ! uppper bounds of the "real" coarray, lower virtual+1
  lcob_coar(3),& ! lower cobounds of the coarray
  ucob_coar(3),& ! upper cobounds of the coarray
  lcob_gv(3)  ,& ! lower cobounds of gv
  ucob_gv(3)  ,& ! upper cobounds of gv
  nimages        ! to store num_images() output
logical(kind=ldef) :: image1

!*********************************************************************73
! checks
!*********************************************************************73

if (.not. allocated(coarray)) then
  write (*,'(a,i0)') "ERROR: cgca_gv: image: ", this_image()
  write (*,'(a)') "ERROR: cgca_gv: coarray is not allocated"
  error stop
end if
```

```
if (.not. allocated(gv)) then
  write (*,'(a,i0)') "ERROR: cgca_gv: image: ", this_image()
  write (*,'(a)') "ERROR: cgca_gv: gv is not allocated"
  error stop
end if

! make sure coarray and gv have the same cobounds

lcob_coar=lcobound(coarray)
ucob_coar=ucobound(coarray)
lcob_gv=lcobound(gv)
ucob_gv=ucobound(gv)

if ( any (lcob_coar .ne. lcob_gv .or. ucob_coar .ne. ucob_gv)) then
  write (*,'(a,i0)') "ERROR: cgca_gv: image: ", this_image()
  write (*,'(a)') &
    "ERROR: cgca_gv: codimensions of coarray and gv do not match"
  error stop
end if

!*********************************************************************73
! end of checks
!*********************************************************************73

! initialise few variables
errstat = 0
nimages = num_images()

! set image1
image1 = .false.
if (this_image() .eq. 1) image1 = .true.

! Assume the coarray has halos. Ignore those
lbr=lbound(coarray)+1
ubr=ubound(coarray)-1

! zero gv on every image
gv = 0_ilrg

! each image calculates its gv
do i3=lbr(3),ubr(3)
do i2=lbr(2),ubr(2)
do i1=lbr(1),ubr(1)
 gv(coarray(i1,i2,i3,cgca_state_type_grain)) = &
  gv(coarray(i1,i2,i3,cgca_state_type_grain)) + 1_ilrg
end do
end do
end do

! image volume
imagevol = int( size( coarray(lbr(1):ubr(1), lbr(2):ubr(2), &
```

```
 lbr(3):ubr(3), cgca_state_type_grain)), kind=ilrg)

! local check on each image: sum(gv) must equal the coarray volume
if (sum(gv) .ne. imagevol) then
  write (*,'(a,i0)') "ERROR: cgca_gv: image: ", this_image()
  write (*,'(a)') "ERROR: cgca_gv: sum(gv) does not match coarray volume"
  error stop
end if

! cannot proceed further until all images
! finish calculating their volumes
sync all

! image1 adds to its own volume volumes from all other images

if (image1) then

 ! preserve gv from image 1
 allocate( gvimg1(size(gv)), stat=errstat)
 if (errstat .ne. 0) then
  write (*,'(a)') "ERROR: cgca_gv: cannot allocate gvimg1"
  write (*,'(a,i0)') "ERROR: cgca_gv: error code: ", errstat
  error stop
 end if

 gvimg1 = gv

 do i3=lcob_gv(3),ucob_gv(3)
 do i2=lcob_gv(2),ucob_gv(2)
 do i1=lcob_gv(1),ucob_gv(1)
  ! image 1 will be counted twice! So need to subtract its
  ! preserved value, gvimg1, from the total
  gv(:) = gv(:) + gv(:)[i1,i2,i3]
 end do
 end do
 end do

 gv = gv - gvimg1

end if
sync all

! get the global volume from image 1
gv(:) = gv(:)[lcob_gv(1),lcob_gv(2),lcob_gv(3)]

! global check: sum(gv) must equal the model volume
if (sum(gv) .ne. imagevol*nimages) then
  write (*,'(a,i0)') "ERROR: cgca_gv: image: ", this_image()
  write (*,'(2(a,i0))') "ERROR: cgca_gv: sum(gv): ", sum(gv), &
    " does not match model volume: ", imagevol*nimages
  error stop
end if
```

```
! sync before leaving
sync all

end subroutine cgca_gv
```

25.3 cgca_m2stat/cgca_gvl

[cgca_m2stat] [Subroutines]

NAME

cgca_gvl

SYNOPSIS

```
subroutine cgca_gvl(coarray,gv)
```

INPUTS

```
integer(kind=iarr),allocatable,intent(in) :: coarray(:,:,:,:)[:,:,:]
integer(kind=ilrg),allocatable,intent(inout) :: gv(:)[:,:,:]
```

SIDE EFFECTS

The state of gv array changes

DESCRIPTION

This routine does grain volume calculation on every image. For each cell (i,j,k) in coarray, the routine increments gv(coarray(i,j,k)). The intention is that after a call to this routine a collective routine is called, e.g. CO_SUM, to calculate grain volumes across all images.

NOTES

All images must call this routine!

SOURCE

```
integer(kind=ilrg) :: imagevol
integer :: i1, i2, i3, &
  lbr(4)      ,& ! lower bounds of the "real" coarray, lower virtual+1
  ubr(4)      ,& ! uppper bounds of the "real" coarray, lower virtual+1
  lcob_coar(3),& ! lower cobounds of the coarray
  ucob_coar(3),& ! upper cobounds of the coarray
  lcob_gv(3)  ,& ! lower cobounds of gv
  ucob_gv(3)     ! upper cobounds of gv

!*********************************************************************73
! checks
!*********************************************************************73

if (.not. allocated(coarray)) then
  write (*,'(a,i0)') "ERROR: cgca_gvl: image: ", this_image()
  write (*,'(a)') "ERROR: cgca_gvl: coarray is not allocated"
  error stop
end if

if (.not. allocated(gv)) then
  write (*,'(a,i0)') "ERROR: cgca_gvl: image: ", this_image()
  write (*,'(a)') "ERROR: cgca_gvl: gv is not allocated"
  error stop
```

```
end if

! make sure coarray and gv have the same cobounds

lcob_coar=lcobound(coarray)
ucob_coar=ucobound(coarray)
lcob_gv=lcobound(gv)
ucob_gv=ucobound(gv)

if ( any (lcob_coar .ne. lcob_gv .or. ucob_coar .ne. ucob_gv)) then
  write (*,'(a,i0)') "ERROR: cgca_gvl: image: ", this_image()
  write (*,'(a)') &
    "ERROR: cgca_gvl: codimensions of coarray and gv do not match"
  error stop
end if

!*********************************************************************73
! end of checks
!*********************************************************************73

! Assume the coarray has halos. Ignore those
lbr=lbound(coarray)+1
ubr=ubound(coarray)-1

! zero gv
gv = 0_ilrg

! each image calculates its gv
do i3=lbr(3),ubr(3)
do i2=lbr(2),ubr(2)
do i1=lbr(1),ubr(1)
 gv(coarray(i1,i2,i3,cgca_state_type_grain)) = &
  gv(coarray(i1,i2,i3,cgca_state_type_grain)) + 1_ilrg
end do
end do
end do

! image volume
imagevol = int( size( coarray(lbr(1):ubr(1), lbr(2):ubr(2), &
 lbr(3):ubr(3), cgca_state_type_grain)), kind=ilrg)

! local check on each image: sum(gv) must equal the coarray volume
if (sum(gv) .ne. imagevol) then
  write (*,'(a,i0)') "ERROR: cgca_gvl: image: ", this_image()
  write (*,'(a)') "ERROR: cgca_gvl: sum(gv) .ne. coarray volume"
  error stop
end if

end subroutine cgca_gvl
```

26 CGPACK/cgca_m3clvg

[Modules]

NAME

cgca_m3clvg

SYNOPSIS

```
!$Id: cgca_m3clvg.f90 526 2018-03-25 23:44:51Z mexas $

module cgca_m3clvg
```

DESCRIPTION

Module dealing with cleavage

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

CONTAINS

Abstract interfaces: cgca_clvgs_abstract. Public subroutines: cgca_clvgn (26.1), cgca_clvgsd (26.5), cgca_clvgsp (26.7), cgca_clvgp_nocosum (26.4), cgca_clvgp1 (26.3), cgca_dacf (26.9), cgca_gcupda (26.15.1) (in submodule m3clvg_sm1 (26.15)), cgca_gcupdn (26.15.2) (in submodule m3clvg_sm1 (26.15)), cgca_tchk (26.16.1) (in submodule m3clvg_sm2 (26.16)), cgca_clvgp (26.17.1) (in submodule m3clvg_sm3 (26.17)).

USES

cgca_m2gb (11), cgca_m2hx (15), cgca_m2rot (24)

USED BY

cgca

SOURCE

```
use cgca_m1co
use cgca_m2gb,  only: cgca_gcr, cgca_gcf
use cgca_m2glm, only: cgca_ico
use cgca_m2hx,  only: cgca_hxi, cgca_hxg
use cgca_m2rot, only: cgca_csym
use cgca_m2rnd, only: cgca_irs

implicit none

private
public :: cgca_clvgp_nocosum, cgca_clvgp1, cgca_clvgsd, cgca_clvgsp, &
          cgca_dacf, cgca_dacf1, cgca_tchk, cgca_clvgp, cgca_gcupda, &
          cgca_gcupdn , gcupd_alloc, cgca_clvgn

! This array is used to update local gc arrays.
! The components are as follows:
```

```
!
!  gcupd(:,1) - grain
!  gcupd(:,2) - neighbour
!  gcupd(:,3) - state, either cgca_gb_state_intact    or
!                             cgca_gb_state_fractured
!
! The idea is that this array is updated every time a grain
! boundary is crossed. Then all local arrays are updated using
! cgca_gcf and this coarray.

integer( kind=iarr ), allocatable :: gcupd(:,:) [:,:,:]

! Counter of a gcupd pair. Set to 1 initially.
integer( kind=idef ) :: gcucnt=1

real( kind=rdef ), parameter ::    &
     zero = 0.0_rdef,              &
      one = 1.0_rdef,              &
    sqrt2 = sqrt(2.0_rdef),        &
 onesqrt2 = one/sqrt2,             &
    sqrt3 = sqrt(3.0_rdef),        &
 onesqrt3 = one/sqrt3,             &

 ! 27 unit vectors connecting the central cell with all its
 ! 26 neighbours + itself, which is a zero vector.
 e(3, -1:1, -1:1, -1:1) =          &
   reshape( (/                     &
 -onesqrt3,-onesqrt3,-onesqrt3,    & ! -1 -1 -1
  zero,    -onesqrt2,-onesqrt2,    & !  0 -1 -1
  onesqrt3,-onesqrt3,-onesqrt3,    & !  1 -1 -1

 -onesqrt2, zero,    -onesqrt2,    & ! -1  0 -1
  zero,     zero,    -one,         & !  0  0 -1
  onesqrt2, zero,    -onesqrt2,    & !  1  0 -1

 -onesqrt3, onesqrt3,-onesqrt3,    & ! -1  1 -1
  zero,     onesqrt2,-onesqrt2,    & !  0  1 -1
  onesqrt3, onesqrt3,-onesqrt3,    & !  1  1 -1

 -onesqrt2,-onesqrt2, zero,        & ! -1 -1  0
  zero,    -one,      zero,        & !  0 -1  0
  onesqrt2,-onesqrt2, zero,        & !  1 -1  0

 -one,      zero,     zero,        & ! -1  0  0
  zero,     zero,     zero,        & !  0  0  0
  one,      zero,     zero,        & !  1  0  0

 -onesqrt2, onesqrt2, zero,        & ! -1  1  0
  zero,     one,      zero,        & !  0  1  0
  onesqrt2, onesqrt2, zero,        & !  1  1  0

 -onesqrt3,-onesqrt3, onesqrt3,    & ! -1 -1  1
```

```
  zero,    -onesqrt2, onesqrt2,    & !  0 -1  1
  onesqrt3,-onesqrt3, onesqrt3,    & !  1 -1  1

 -onesqrt2, zero,     onesqrt2,    & ! -1  0  1
  zero,     zero,     one,         & !  0  0  1
  onesqrt2, zero,     onesqrt2,    & !  1  0  1

 -onesqrt3, onesqrt3, onesqrt3,    & ! -1  1  1
  zero,     onesqrt2, onesqrt2,    & !  0  1  1
  onesqrt3, onesqrt3, onesqrt3     & !  1  1  1
  /), (/ 3,3,3,3 /) )


! Abstract interface is a fortran 2003 feature.
! This interface is for cleavage "change state" routines.
! The interface for such routines - cgca_clvgsd and cgca_clvgsp,
! must match it.
! Using this interface, the cleavage "change state"
! routines can be passed as actual arguments to the cleavage
! propagation routines, cgca_clvgp and cgca_clvgp1, where
! the dummy arguments for these routines are defined by
!  procedure(cgca_clvgs_abstract)
!
! The interface is the exact copy of cgca_clvgsd, cgca_clvgsp
! It is used by cgca_clvgp, cgca_clvgp1, etc.

abstract interface
 subroutine cgca_clvgs_abstract( farr, marr, n, cstate, debug,         &
                                 newstate )
  use cgca_m1co
  integer, parameter :: l=-1, centre=l+1, u=centre+1
  integer( kind=iarr ), intent(in) :: farr(l:u,l:u,l:u),               &
    marr(l:u,l:u,l:u), cstate
  real( kind=rdef ), intent(in) :: n(3)
  logical( kind=ldef ), intent(in) :: debug
  integer( kind=iarr ), intent(out) :: newstate
 end subroutine cgca_clvgs_abstract

 subroutine gcupd_abstract( periodicbc )
  import :: ldef
  logical( kind=ldef ), intent( in ) :: periodicbc
 end subroutine
end interface

! Interfaces for submodule procedures.

interface
  ! in submodule m3clvg_sm1
  module subroutine cgca_gcupda( periodicbc )
    logical( kind=ldef ), intent( in ) :: periodicbc
  end subroutine cgca_gcupda
```

```
  ! in submodule m3clvg_sm1
  module subroutine cgca_gcupdn( periodicbc )
    logical( kind=ldef ), intent( in ) :: periodicbc
  end subroutine cgca_gcupdn

  ! in submodule m3clvg_sm2
  module subroutine cgca_tchk( num, maxmin, minmax )
    integer( kind=ilrg ), intent(in) :: num
    real( kind=rlrg ), intent(out) :: maxmin, minmax
  end subroutine cgca_tchk

  ! in submodule m3clvg_sm3
  module subroutine cgca_clvgp( coarray, rt, t, scrit, sub, gcus,      &
                                periodicbc, iter, heartbeat, debug )
    ! Inputs:
    ! coarray - cellular array
    integer( kind=iarr ), allocatable, intent(inout) ::                &
      coarray(:,:,:,:)[:,:,:]
    ! rt - rotation tensor coarray
    real( kind=rdef ), allocatable, intent(inout) :: rt(:,:,:)[:,:,:]
    ! t - stress tensor in spatial CS
    ! - scrit - critical values of cleavage stress on 100,
    !   110 and 111 planes
    real( kind=rdef ), intent(in) :: t(3,3), scrit(3)
    ! sub - name of the cleavage state calculation routine,
    !   either cgca_clvgsd, or cgca_clvgsp.
    procedure( cgca_clvgs_abstract ) :: sub
    ! gcus - name of the grain connectivity update subroutine, either
    !        cgca_gcupda - all-to-all, or cgca_gcupdn - nearest
    ! neighbour.
    procedure( gcupd_abstract ) :: gcus
    ! periodicbc - if .true. periodic boundary conditions are used,
    !   i.e. global halo exchange is called before every iteration
    logical( kind=ldef ), intent(in) :: periodicbc
    !      iter - number of cleavage iterations, if <=0 then error
    ! heartbeat - if >0 then dump a simple message every
    !             heartbeat iterations
    integer( kind=idef ), intent(in) :: iter, heartbeat
    ! debug - if .true. then will call cgca_dacf with debug
    logical( kind=ldef ), intent(in) :: debug
  end subroutine cgca_clvgp
end interface

contains
```

26.1 cgca_m3clvg/cgca_clvgn

[cgca_m3clvg] [Subroutines]

NAME

cgca_clvgn

SYNOPSIS

```
subroutine cgca_clvgn( t, r, tcrit, flag, n, cstate )
```

INPUTS

```
real( kind=rdef ), intent(in) :: t(3,3), r(3,3), tcrit(3)
```

OUTPUTS

```
logical( kind=ldef ), intent(out) :: flag
real(    kind=rdef ), intent(out) :: n(3)
integer( kind=iarr ), intent(out) :: cstate
```

DESCRIPTION

Given the stress tensor in the spatial CS, (t) and the crystal rotation tensor (r), this routine first calculates whether cleavage happens or not (flag). If cleavage conditions are met (flag=.true.), then the routine calculates the acting cleavage plane normal in spatial coord. system (n) and the type of the cleavage plane (cstate).

Inputs:

- t - stress tensor in spatial CS
- r - crystal rotation tensor
- tcrit - critical values of cleavage stress on 100, 110 and 111 planes

Outputs:

- flag - .true. if cleavage conditions are met, .false. otherwise
- n - unit normal vector defining the acting cleavage plane
- cstate - cell state, one of the cleavage type.

On output, if flag=.false. then n=0, cstate=cgca_instact_state

NOTES

Not accessible from outside of the cgca_clvg module

USES

cgca_csym (24.2)

USED BY

cgca_clvgsd (26.5), cgca_clvgsp (26.7)

SOURCE

```
! unit vectors defining 3 cleavage plane families
real( kind=rdef ), parameter ::                &
 n100(3) = (/ one,      zero,     zero     /), &
 n110(3) = (/ onesqrt2, onesqrt2, zero     /), &
 n111(3) = (/ onesqrt3, onesqrt3, onesqrt3 /)

integer :: i

real( kind=rdef ) :: &
 tc(3,3),     & ! stress tensor in crystal CS
 rsym(3,3),   & ! rotation symmetry tensor
 n100rot(3),  & ! normal to a 100 plane
 n110rot(3),  & ! normal to a 110 plane
 n111rot(3),  & ! normal to a 111 plane
 ttmp(3),     & ! normal stress to a crystallographic plane
 tmax(3),     & ! max normal stress to a crystallographic plane
 n100max(3),  & ! normal to the 100 plane that has max normal stress
 n110max(3),  & ! normal to the 110 plane that has max normal stress
 n111max(3),  & ! normal to the 111 plane that has max normal stress
 p(3),        & ! cleavage stress ratios
 pmax           ! max cleavage stress ratio

! Set important values to zero, and initialise the default
! values for outputs
n100max = zero
n110max = zero
n111max = zero
   flag = .false.
      n = zero
 cstate = cgca_intact_state
   tmax = zero

! stress tensor in crystal CS
! By our convention, r rotates a vector from the cryst.
! coord. system to spatial. Here we rotate the other
! way round, from spatial to crystal system, hence the
! transpose: ! tc = r^T . t . r
tc = matmul( matmul( transpose( r ), t ), r )

! Find the max normal stresses to three cleavage plane families.
do i = 1, 24

  ! pick a rotation tensor
  call cgca_csym( i, rsym )

  ! normals to partucular {100}, {110}, {111} planes
  n100rot = matmul( rsym, n100 )
  n110rot = matmul( rsym, n110 )
  n111rot = matmul( rsym, n111 )

  ! Projections of the stress tensor to normals, i.e.
  ! the normal stresses to crystallographic planes.
```

```
  ! (tc . n) . n
  ttmp(1) = dot_product( n100rot, matmul( tc, n100rot) )
  ttmp(2) = dot_product( n110rot, matmul( tc, n110rot) )
  ttmp(3) = dot_product( n111rot, matmul( tc, n111rot) )

  ! Choose the plane of each type, that has the max normal stress
  ! Signs are taken into account. Since tmax is zero initially,
  ! negative stresses are less than tmax, and thus do not cause
  ! cleavage.

  if ( ttmp(1) .gt. tmax(1) ) then   ! 100 plane
    tmax(1) = ttmp(1)                ! stress
    n100max = n100rot                ! normal to a particular 100 plane
  end if
  if ( ttmp(2) .gt. tmax(2) ) then   ! 110 plane
    tmax(2) = ttmp(2)                ! stress
    n110max = n110rot                ! normal to a particular 110 plane
  end if
  if ( ttmp(3) .gt. tmax(3) ) then   ! 111 plane
    tmax(3) = ttmp(3)                ! stress
    n111max = n111rot                ! normal to a particular 111 plane
  end if

end do

! calculate the cleavage factors (ratios)
p = tmax / tcrit
pmax = maxval(p)

! check for cleavage
if ( pmax .ge. one ) then

  ! cleavage is happening, on 100 by default
    flag = .true.
       n = n100max
  cstate = cgca_clvg_state_100_edge

  if ( p(2) .gt. p(1) ) then
    ! cleavage on 110
         n = n110max
    cstate = cgca_clvg_state_110_edge
  end if

  if ( p(3) .gt. p(1) .and. p(3) .gt. p(2) ) then
    ! cleavage on 111
         n = n111max
    cstate = cgca_clvg_state_111_edge
  end if

  ! rotate n back into the spatial coord. system
  n = matmul( r, n )
```

```
end if

end subroutine cgca_clvgn
```

26.2 cgca_m3clvg/cgca_clvgn_pure

[cgca_m3clvg] [Subroutines]

NAME

cgca_clvgn_pure

SYNOPSIS

```
pure subroutine cgca_clvgn_pure( t, grain, r, tcrit, flag, n, cstate,  &
  ierr )
```

INPUTS

```
!    r - rotation array with extents (number of grains, 3, 3)

real( kind=rdef ), intent(in) :: t(3,3), r(:,:,:), tcrit(3)
integer( kind=iarr), intent(in) :: grain
```

OUTPUTS

```
logical( kind=ldef ), intent(out) :: flag
real( kind=rdef ), intent(out) :: n(3)
integer( kind=iarr ), intent(out) :: cstate
integer, intent(out) :: ierr
```

SIDE EFFECTS

None, this is a PURE subroutine.

DESCRIPTION

Given the stress tensor in the spatial CS, (t) and the crystal rotation tensor (r), this routine first calculates whether cleavage happens or not (flag). If cleavage conditions are met (flag=.true.), then the routine calculates the acting cleavage plane normal in spatial coord. system (n) and the type of the cleavage plane (cstate).

Inputs:

- t - stress tensor in spatial CS
- r - crystal rotation tensor
- tcrit - critical values of cleavage stress on 100, 110 and 111 planes

Outputs:

- flag - .true. if cleavage conditions are met, .false. otherwise
- n - unit normal vector defining the acting cleavage plane
- cstate - cell state, one of the cleavage type.
- ierr - error flag. ierr=0 means a successful execution. Any positive value means an error. The value is taken from the output flag of cgca_csym_pure (24.3).

On output, if flag=.false. then n=0, cstate=cgca_instact_state

NOTES

This is a PURE subroutine, no side effects. It should be used instead of cgca_clvgn (26.1) where a pure subroutine is required, e.g. from a do concurrent.

NOTES

Not accessible from outside of the cgca_clvg module

USES

cgca_csym (24.2)

USED BY

cgca_clvgsd (26.5), cgca_clvgsp (26.7)

SOURCE

```
! unit vectors defining 3 cleavage plane families
real( kind=rdef ), parameter ::                &
 n100(3) = (/ one,      zero,     zero     /), &
 n110(3) = (/ onesqrt2, onesqrt2, zero     /), &
 n111(3) = (/ onesqrt3, onesqrt3, onesqrt3 /)

integer( kind=iarr ) :: i
integer :: iflag

real( kind=rdef ) :: &
 rlocal(3,3), & ! array to store r(grain,:,:), just a convenience
 tc(3,3),     & ! stress tensor in crystal CS
 rsym(3,3),   & ! rotation symmetry tensor
 n100rot(3),  & ! normal to a 100 plane
 n110rot(3),  & ! normal to a 110 plane
 n111rot(3),  & ! normal to a 111 plane
 ttmp(3),     & ! normal stress to a crystallographic plane
 tmax(3),     & ! max normal stress to a crystallographic plane
 n100max(3),  & ! normal to the 100 plane that has max normal stress
 n110max(3),  & ! normal to the 110 plane that has max normal stress
 n111max(3),  & ! normal to the 111 plane that has max normal stress
 p(3),        & ! cleavage stress ratios
 pmax           ! max cleavage stress ratio

! Set rlocal
rlocal = r( grain, :, : )

! Set important values to zero, and initialise the default
! values for outputs
n100max = zero
n110max = zero
n111max = zero
   flag = .false.
      n = zero
 cstate = cgca_intact_state
   tmax = zero
```

```
   ierr = 0

! stress tensor in crystal CS
! By our convention, r rotates a vector from the cryst.
! coord. system to spatial. Here we rotate the other
! way round, from spatial to crystal system, hence the
! transpose: ! tc = r^T . t . r
tc = matmul( matmul( transpose( rlocal ), t ), rlocal )

! Find the max normal stresses to three cleavage plane families.
do i = 1, 24

  ! Pick a rotation tensor. If iflag .ne. 0, then an error
  ! condition has occurred. Then abandon the computation, set
  ! flag accordingly and return immediately.
  call cgca_csym_pure( i, rsym, iflag )
  if ( iflag .ne. 0 ) then
    ierr = iflag
    return
  end if

  ! normals to partucular {100}, {110}, {111} planes
  n100rot = matmul( rsym, n100 )
  n110rot = matmul( rsym, n110 )
  n111rot = matmul( rsym, n111 )

  ! Projections of the stress tensor to normals, i.e.
  ! the normal stresses to crystallographic planes.
  ! (tc . n) . n
  ttmp(1) = dot_product( n100rot, matmul( tc, n100rot) )
  ttmp(2) = dot_product( n110rot, matmul( tc, n110rot) )
  ttmp(3) = dot_product( n111rot, matmul( tc, n111rot) )

  ! Choose the plane of each type, that has the max normal stress
  ! Signs are taken into account. Since tmax is zero initially,
  ! negative stresses are less than tmax, and thus do not cause
  ! cleavage.

  if ( ttmp(1) .gt. tmax(1) ) then   ! 100 plane
    tmax(1) = ttmp(1)                ! stress
    n100max = n100rot                ! normal to a particular 100 plane
  end if
  if ( ttmp(2) .gt. tmax(2) ) then   ! 110 plane
    tmax(2) = ttmp(2)                ! stress
    n110max = n110rot                ! normal to a particular 110 plane
  end if
  if ( ttmp(3) .gt. tmax(3) ) then   ! 111 plane
    tmax(3) = ttmp(3)                ! stress
    n111max = n111rot                ! normal to a particular 111 plane
  end if

end do
```

```
! calculate the cleavage factors (ratios)
p = tmax / tcrit
pmax = maxval(p)

! check for cleavage
if ( pmax .ge. one ) then

  ! cleavage is happening, on 100 by default
    flag = .true.
       n = n100max
  cstate = cgca_clvg_state_100_edge

  if ( p(2) .gt. p(1) ) then
    ! cleavage on 110
         n = n110max
    cstate = cgca_clvg_state_110_edge
  end if

  if ( p(3) .gt. p(1) .and. p(3) .gt. p(2) ) then
    ! cleavage on 111
         n = n111max
    cstate = cgca_clvg_state_111_edge
  end if

  ! rotate n back into the spatial coord. system
  n = matmul( rlocal, n )

end if

end subroutine cgca_clvgn_pure
```

26.3 cgca_m3clvg/cgca_clvgp1

[cgca_m3clvg] [Subroutines]

NAME

cgca_clvgp1

SYNOPSIS

```
subroutine cgca_clvgp1( coarray, rt, t, scrit, sub, debug )
```

INPUTS

```
integer( kind=iarr ), allocatable, intent( inout ) ::                  &
 coarray(:,:,:,:)[:,:,:]
real( kind=rdef ), allocatable, intent( inout ) :: rt(:,:,:)[:,:,:]
real( kind=rdef ), intent( in ) :: t(3,3), scrit(3)
procedure(cgca_clvgs_abstract) :: sub
logical(kind=ldef),intent(in) :: debug
```

SIDE EFFECTS

Many:

- change state of coarray
- change state of gc (11.11) array

DESCRIPTION

This is a cleavage propagation routine.

Inputs:

- coarray - cellular array
- rt - rotation tensor coarray
- s1 - max. principal stress vector (3)
- scrit - critical values of cleavage stress on 100, 110 and 111 planes
- sub - name of the cleavage state calculation routine, either cgca_clvgsd (26.5), or cgca_clvgsp (26.7).
- debug - if .true. then will call cgca_dacf (26.9) with debug

We copy the model (coarray) into the local array. We then analyse the local array, but update the coarray.

NOTES

All images must call this routine

USES

cgca_clvgsd (26.5), cgca_clvgsp (26.7), cgca_clvgn (26.1), cgca_hxi (15.2), cgca_hxg (15.1), cgca_dacf (26.9)

USED BY

none, end user

SOURCE

```
real(kind=rdef) :: n(3)
integer(kind=iarr),allocatable,save :: array(:,:,:)
integer(kind=iarr) :: grold, grnew, cstate

integer(kind=idef) :: &
  lbv(4) ,& ! lower bounds of the complete (plus virtual) coarray
  ubv(4) ,& ! upper bounds of the complete (plus virtual) coarray
  lbr(4) ,& ! lower bounds of the "real" coarray, lower virtual+1
  ubr(4) ,& ! upper bounds of the "real" coarray, upper virtual-1
  x1     ,& ! local coordinates in an array, which are also
  x2     ,& ! do loop counters
  x3

integer :: thisimage, errstat=0, nimages

logical(kind=ldef) :: clvgflag

! use local vars to save time
thisimage = this_image()
nimages = num_images()

! determine the extents
lbv = lbound(coarray)
ubv = ubound(coarray)
lbr = lbv+1
ubr = ubv-1

! no sanity checks in this routine!

! allocate the temp array on first call
if (.not. allocated(array))                                        &
  allocate( array( lbv(1):ubv(1),                                  &
                   lbv(2):ubv(2),                                  &
                   lbv(3):ubv(3) ), stat=errstat )
if (errstat.ne.0) then
  write (*,"(2(a,i0))") "ERROR: cgca_clvgp: image ",thisimage,   &
        " : cannot allocate array, errcode: ", errstat
  error stop
end if

! initialise the old grain to liquid
grold = cgca_liquid_state

sync all

! copy coarray fracture state type into a local array
array = coarray(:,:,:,cgca_state_type_frac)

! propagate cleavage
do x3 = lbr(3),ubr(3)
do x2 = lbr(2),ubr(2)
do x1 = lbr(1),ubr(1)
```

```
  ! scan only through undamaged cells
  live: if ( array(x1,x2,x3) .eq. cgca_intact_state ) then

   ! what grain are we in?
   grnew = coarray(x1,x2,x3,cgca_state_type_grain)

   ! If the new grain differs from the old, then we have crossed the
   ! grain boundary, and need to calculate the cleavage plane again.

   ! not needed, but Crays issues caution otherwise
   clvgflag = .false.

   if ( grnew .ne. grold ) then
    call cgca_clvgn( t, rt(grnew,:,:), scrit, clvgflag, n, cstate )
    grold = grnew
   end if

   ! If cleavage conditions are met, propagate cleavage into this cell.
   ! Note that we pass the local array, but return the new state
   ! of the central cell into the coarray. The sub name is provided as
   ! an input to _clvgp. It can be either the deterministic routine
   ! _clvgsd, or the probabilistic routine _clvgsp.
   if ( clvgflag ) call sub( array(x1-1:x1+1, x2-1:x2+1, x3-1:x3+1),   &
    coarray(x1-1:x1+1, x2-1:x2+1, x3-1:x3+1, cgca_state_type_grain),   &
            n, cstate, debug, coarray(x1,x2,x3,cgca_state_type_frac) )
  end if live

end do
end do
end do

! no sync in this routine, leave this to the calling routine

end subroutine cgca_clvgp1
```

26.4 cgca_m3clvg/cgca_clvgp_nocosum

[cgca_m3clvg] [Subroutines]

NAME

cgca_clvgp_nocosum

SYNOPSIS

```
subroutine cgca_clvgp_nocosum( coarray, rt, t, scrit, sub, gcus,       &
  periodicbc, iter, heartbeat, debug )
```

INPUTS

```
integer( kind=iarr ), allocatable, intent(inout) ::                    &
 coarray(:,:,:,:)[:,:,:]
real( kind=rdef ), allocatable, intent(inout) :: rt(:,:,:)[:,:,:]
real( kind=rdef ), intent(in) :: t(3,3), scrit(3)
procedure( cgca_clvgs_abstract ) :: sub
procedure( gcupd_abstract) :: gcus
logical( kind=ldef ), intent(in) :: periodicbc
integer( kind=idef ), intent(in) :: iter, heartbeat
logical( kind=ldef ), intent(in) :: debug
```

SIDE EFFECTS

- change state of gc (11.11) array

DESCRIPTION

This is a cleavage propagation routine.

Inputs:

- coarray - cellular array
- rt - rotation tensor coarray
- t - stress tensor in spatial CS
- scrit - critical values of cleavage stress on 100, 110 and 111 planes
- sub - name of the cleavage state calculation routine, either cgca_clvgsd (26.5), or cgca_clvgsp (26.7).
- gcus - a subrotine to use to update the grain connectivity array, either gcupd_a (all-to-all) or gcupd_n (nearest neighbour). Both these subroutines have identical interface gcupd_abstract.
- periodicbc - if .true. periodic boundary conditions are used, i.e. global halo exchange is called before every iteration
- iter - number of cleavage iterations, if <=0 then error
- heartbeat - if >0 then dump a simple message every heartbeat iterations
- debug - if .true. then will call cgca_dacf (26.9) with debug

We copy the model (coarray) into the local array. We then analyse the local array, but update the coarray.

For each real cell (we do not analyse halo cells) we look only at undamaged cells in the fracture layer, i.e. cells of state cgca_intact_state (9.17) or cgca_gb_state_intact (9.14). At present two routines can be called, a deterministic cgca_clvgsd (26.5), or a probabilistic cgca_clvgsp (26.7).

All images must call this routine

NOTES

For use with ifort, no CO_SUM here

USES

cgca_clvgs_abstract, cgca_clvgsd (26.5), cgca_clvgsp (26.7), cgca_clvgn (26.1), cgca_hxi (15.2), cgca_hxg (15.1), cgca_dacf (26.9)

USED BY

none, end user

SOURCE

```
real( kind=rdef ) :: n(3),                                              &
  ! tmp array to avoid copy in/out warnings
     tmparr(3,3)
integer( kind=iarr ), allocatable :: array(:,:,:)
integer( kind=iarr ) :: grold, grnew, cstate,                           &
  ! tmp arrays to avoid copy in/out warnings
     arrtmp1(3,3,3), arrtmp2(3,3,3)

integer( kind=idef ) :: i,                                              &
  lbv(4) ,& ! lower bounds of the complete (plus virtual) coarray
  ubv(4) ,& ! upper bounds of the complete (plus virtual) coarray
  lbr(4) ,& ! lower bounds of the "real" coarray, lower virtual+1
  ubr(4) ,& ! upper bounds of the "real" coarray, upper virtual-1
  x1     ,& ! local coordinates in an array, which are also
  x2     ,& ! do loop counters
  x3     ,& !
  iteration ! iteration counter

integer :: thisimage, errstat=0, nimages
integer, save :: clvgglob[*]

logical(kind=ldef) :: clvgflag

! Make sure to allocate gcupd!
if ( .not. allocated( gcupd ) ) call gcupd_alloc

! Set the global cleavage flag initially to zero on all images,
! i.e. no cleavage
clvgglob = 0

! Set the local cleavage flag to .false.
clvgflag = .false.

! use local vars to save time
```

```
thisimage = this_image()
  nimages = num_images()

! determine the extents
lbv = lbound(coarray)
ubv = ubound(coarray)
lbr = lbv+1
ubr = ubv-1

!************************************************
! Sanity checks
!************************************************

! check for allocated
if ( .not. allocated( coarray ) ) then
  write (*,'(a,i0)') "ERROR: cgca_clvgp: image ",thisimage
  write (*,'(a)')    "ERROR: cgca_clvgp: coarray is not allocated"
  error stop
end if
if ( .not. allocated( rt ) ) then
  write (*,'(a,i0)') "ERROR: cgca_clvgp: image ",thisimage
  write (*,'(a)')    "ERROR: cgca_clvgp: rt is not allocated"
  error stop
end if

! check there are no liquid cells in the grain array
if ( any( coarray(lbr(1):ubr(1),lbr(2):ubr(2),lbr(3):ubr(3),           &
  cgca_state_type_grain) .eq. cgca_liquid_state)) then
  write (*,'(a,i0,a)') "ERROR: cgca_clvgp: image ",thisimage,          &
   ": liquid phase in the model"
  error stop
end if

if ( iter .le. 0 ) then
  write (*,'(a,i0,a)') "ERROR: cgca_clvgp: image ",thisimage,          &
   ": negative number of iterations given"
  error stop
end if

!************************************************
! End of sanity checks
!************************************************

! allocate the temp array
allocate( array(lbv(1):ubv(1),lbv(2):ubv(2),lbv(3):ubv(3) ),           &
          stat=errstat )
if ( errstat .ne. 0 ) then
  write (*,"(2(a,i0))") "ERROR: cgca_clvgp: image ", thisimage,        &
   " : cannot allocate array, errcode: ", errstat
  error stop
end if
```

```
! initialise the iteration counter
iteration = 1

! initialise the old grain to liquid
grold = cgca_liquid_state

! initial halo exchange, to make sure the coarray is in a correct state
call cgca_hxi( coarray )
if ( periodicbc ) call cgca_hxg( coarray )
sync all

! start the main loop for cleavage iterations
main: do

  ! copy coarray fracture state type into a local array
  array = coarray(:,:,:,cgca_state_type_frac)

  ! propagate cleavage
  do x3 = lbr(3),ubr(3)
  do x2 = lbr(2),ubr(2)
  do x1 = lbr(1),ubr(1)

    ! scan only through undamaged cells
    live: if ( array(x1,x2,x3) .eq. cgca_intact_state .or.             &
               array(x1,x2,x3) .eq. cgca_gb_state_intact) then

      ! what grain are we in?
      grnew = coarray( x1, x2, x3, cgca_state_type_grain )

      ! If the new grain differs from the old, then
      ! we have crossed the grain boundary, and need
      ! to calculate the cleavage plane again.
      !
      ! If clvgflag=.true., it stays .true. until another GB is crossed.
      if ( grnew .ne. grold ) then

        ! Use a tmp array to avoid compiler and runtime
        ! copy in/out warnings
        tmparr = rt( grnew, : , : )
        call cgca_clvgn( t, tmparr, scrit, clvgflag, n, cstate )
        grold = grnew
      end if

      ! debug
      !   if (debug) write (*,"(a,i0,a,l1)")      &
      !    "DEBUG: cgca_clvgp: img ", thisimage,  &
      !    " clvgflag=", clvgflag

      ! If cleavage conditions are met, propagate cleavage into
      ! this cell. Note that we pass the local array, but return
      ! the new state of the central cell into the coarray.
      ! The sub name is provided as an input to cgca_clvgp.
```

```
    ! It can be either the deterministic routine cgca_clvgsd,
    ! or the probabilistic routine cgca_clvgsp.
    if ( clvgflag ) then

      ! mark that cleavage has occurred. The value is not important,
      ! any non-zero integer will do, but the same on all images.
      clvgglob = 1

      ! Use tmp arrays explicitly to avoid compiler or runtime
      ! warnings.
      arrtmp1 = array(   x1-1:x1+1, x2-1:x2+1, x3-1:x3+1 )
      arrtmp2 = coarray( x1-1:x1+1, x2-1:x2+1, x3-1:x3+1,            &
                                    cgca_state_type_grain )
      call sub( arrtmp1, arrtmp2, n, cstate, debug,                  &
                  coarray(x1,x2,x3,cgca_state_type_frac) )
    end if
  end if live

end do
end do
end do

! Add together all cleavage identifiers from all images
! image 1 does all the work, other images wait
! need to make a new executing segment for this.

sync all

if ( thisimage .eq. 1 ) then
  do i=2, nimages
    clvgglob = clvgglob + clvgglob[i]
  end do
end if

! distibute clvgglob[1] to all images
! need to make a new executing segment for this.

sync all

clvgglob = clvgglob[1]

sync all

! Check if cleavage happened anywhere in the model.
if ( clvgglob .eq. 0 ) then
  if ( thisimage .eq. 1 ) write (*,*)                              &
    "INFO: cgca_clvgp_nocosum: no cleavage anywhere, leaving"
  exit main
end if

sync all
```

```
  ! Udate all local gc arrays using the given subroutine
  call gcus( periodicbc )

  ! halo exchange after a cleavage propagation step
  call cgca_hxi( coarray )
  if ( periodicbc ) call cgca_hxg( coarray )

  sync all

  ! Reset all gcupd
  gcupd = cgca_gb_state_intact

  ! Reset the gcupd counter
  gcucnt = 1

  ! deactivate crack flanks, ignore grain boundaries
  call cgca_dacf( coarray, debug=.false. )

  sync all

  ! halo exchange after deactivation step
  call cgca_hxi( coarray )
  if ( periodicbc ) call cgca_hxg( coarray )

  sync all

  ! send heartbeat signal to terminal
  if (thisimage .eq. 1 .and. heartbeat .gt. 0) then
    if ( mod( iteration, heartbeat ) .eq. 0) write (*,'(a,i0)')        &
        "INFO: cgca_clvgp_nocosum: iterations completed: ", iteration
  end if

  if ( iteration .ge. iter ) exit main

  ! increment the iteration counter
  iteration = iteration + 1

end do main

deallocate( array, stat=errstat )
if ( errstat .ne. 0 ) then
  write (*,"(2(a,i0))") "ERROR: cgca_clvgp_nocosum: image ",thisimage, &
   " : cannot deallocate array, errcode: ", errstat
  error stop
end if

! sync before leaving
sync all

end subroutine cgca_clvgp_nocosum
```

26.5 cgca_m3clvg/cgca_clvgsd

[cgca_m3clvg] [Subroutines]

NAME

cgca_clvgsd

SYNOPSIS

```
subroutine cgca_clvgsd( farr, marr, n, cstate, debug, newstate )
```

PARAMETERS

```
integer, parameter :: l=-1, centre=l+1, u=centre+1
real( kind=rdef ), parameter :: trshld = 0.17325932611400130485_rdef
```

INPUTS

```
integer( kind=iarr ), intent( in ) :: farr( l:u, l:u, l:u ),           &
  marr( l:u, l:u, l:u ), cstate
real(    kind=rdef ), intent(in) :: n(3)
logical( kind=ldef ), intent(in) :: debug
```

OUTPUT

```
integer( kind=iarr ), intent(out) :: newstate
```

DESCRIPTION

This routine determines the cleavage (CLVG) state (S) of the central cell. This is a deterministic (D) routine. Hence the name is CLVGSD. If there is a cleaved neighbour, such that the vector connecting it to the central cell is on or close to the cleavage plane, then the central cell state is changed to the given cleavage state.

If the cleaved neighbour belongs to another grain, the analysis takes into account the grain boundary state. If the grain boundary is marked as intact, then the crack can cross it, and the central cell takes the values of the cleaved neighbour. If the grain boundary is marked as fractured, then the crack cannot cross it. This means that even if there is a cleaved neighbour on the cleavage plane, the central cell will still not change state.

If the grain boudary is intact, and crack crosses it, mark this GB as fractured immediately in the local CG array and add this GB fracture to gcupd on this image.

Inputs:

- farr - (3,3,3) array of failed states, cell state type cgca_state_type_frac (9.23)
- marr - (3,3,3) array of material states, cell state type cgca_state_type_grain (9.24)
- n - vector defining the cleavage plane
- cstate - cleavage state
- debug - if .true. will dump some debug info

Outputs:

- newstate - updated central cell state

NOTES

The threshold analysis is explained in A. Shterenlikht, L. Margetts, Three-dimensional cellular automata modelling of cleavage propagation across crystal boundaries in polycrystalline microstructures, Proc. Roy. Soc. A, accepted for publication, 5-MAR-2015. There are always at least 2 neighbouring cells for which dot_product(e,n) is less than ∼0.1732. However, 2 cells is not enough! If threshold is left at that, 2 cells lead to linear 1D cracks, which are not reasonable.

USES

cgca_gcr (11.7), cgca_gcf (11.4)

USED BY

cgca_clvgp (26.17.1)

SOURCE

```
integer( kind=idef ) :: &
 x1, x2, x3,            & ! coordinates within (3,3,3) neighbourhood
 img,                   & ! this_image()
 nimgs                    ! num_images()

logical :: intact

  img = this_image()
nimgs = num_images()

! initially the new central cell state is the same as old
newstate = farr( centre, centre, centre )

! scan all neighbourhood cells
outer: do x3 = l, u
       do x2 = l, u
       do x1 = l, u

  ! Check whether the neighbour has cleaved. The central cell is never
  ! cleaved, so the check will always fail for the central cell.
  clv: if ( any( cgca_clvg_states .eq. farr(x1,x2,x3)) ) then

    ! If the neighbour is from the same grain, then just check the
    ! orientation of the cleavage plane
    same: if ( marr( centre, centre, centre ) .eq. marr(x1,x2,x3) ) then

      ! debug
      !  if (debug) then
      !   call cgca_gcr(1,2, intact)
      !   if (.not. intact) write (*,"(2(a,i0),a,3(tr1,f7.4),a,f7.4)") &
      !    "zzz: grain=",marr(centre,centre,centre), &
      !    ", cstate=", cstate, &
      !    ", n=(", n, ") ", &
```

```
  !    abs(dot_product(e(:,x1,x2,x3),n))
  !  end if

  ! If the dot product is smaller than the threshold, i.e. if
  ! the vector connecting the central cell with the neighbour
  ! is in the cleavage plane, then the central cell has cleaved
  if ( abs( dot_product( e(:,x1,x2,x3), n ) ) .lt. trshld ) then
    newstate = cstate  ! change its state
    exit outer         ! and exit the main loop
  end if

else

  ! The neighbour is from a different grain. In this case
  ! the boundary must be intact in addition to the cleavage
  ! plane orientation criterion.
  intact = .false.

  ! get the recorded state of the boundary between the two grains
  !  cgca_gcr returns .TRUE. if intact and .FALSE. if fractured
  call cgca_gcr( marr(centre,centre,centre), marr(x1,x2,x3), intact)

  ! same as in the previous case, but with the added constraint
  ! that the grain boundary must be intact
  crossgb: if ( intact .and.                                          &
       abs( dot_product( e(:,x1,x2,x3), n ) ) .lt. trshld ) then

    ! change its state
    newstate = cstate

    ! Mark GB as fractured on this image
    call cgca_gcf( marr(centre, centre, centre), marr(x1, x2, x3) )

    ! Add the (grain, neighbour) pair to gcupd coarray
    ! on this image
    gcupd( gcucnt, : ) = (/ marr( centre , centre , centre ) , &
                            marr(   x1   ,   x2   ,   x3   ) ,     &
                            cgca_gb_state_fractured /)

    ! debug output
    if (debug)                                                        &
      write (*,"(4(a,i0),2(tr1,i0),')',a,27(i0,tr1),').')")         &
        "DEBUG: cgca_clvgsd: img: ", img, ": newstate=", newstate, &
        ", gcucnt=", gcucnt,                                          &
        ", calling cgca_gcf, gcupd=(", gcupd( gcucnt , : ),&
        ", marr=(", marr

    ! increment the gcupd pair counter
    gcucnt = gcucnt + 1

    ! Issue fatal error if the length of the gcupd has been
    ! exceeded.
```

```
        if ( gcucnt .gt. cgca_gcupd_size1 ) then
          write( *, '(a,i0)' ) "ERROR: cgca_m3clvg/cgca_clvgsd:&
            & gcucnt .gt. cgca_gcupd_size1, image: ", img
          error stop
        end if

        ! now exit the main loop
        exit outer

      end if crossgb

    end if same

 end if clv

 ! now check another neighbouring cell, i.e. increment the loop counter

end do
end do
end do outer

end subroutine cgca_clvgsd
```

26.6 cgca_m3clvg/cgca_clvgsdt

[cgca_m3clvg] [Subroutines]

NAME

cgca_clvgsdt

SYNOPSIS

```
pure subroutine cgca_clvgsdt( farr, marr, n, cstate, debug, newstate )
```

PARAMETERS

```
integer, parameter :: l=-1, centre=l+1, u=centre+1
real( kind=rdef ), parameter :: trshld = 0.17325932611400130485_rdef
```

INPUTS

```
integer( kind=iarr ), intent(in) :: farr(l:u,l:u,l:u), &
 marr(l:u,l:u,l:u), cstate
real(    kind=rdef ), intent(in) :: n(3)
logical( kind=ldef ), intent(in) :: debug
```

OUTPUT

```
integer( kind=iarr ), intent(out) :: newstate
```

DESCRIPTION

This routine determines the cleavage (CLVG) state (S) of the central cell. This is a deterministic (D) routine. Hence the name is CLVGSD. If there is a cleaved neighbour, such that the vector connecting it to the central cell is on or close to the cleavage plane, then the central cell state is changed to the given cleavage state.

If the cleaved neighbour belongs to another grain, the analysis takes into account the grain boundary state. If the grain boundary is marked as intact, then the crack can cross it, and the central cell takes the values of the cleaved neighbour. If the grain boundary is marked as fractured, then the crack cannot cross it. This means that even if there is a cleaved neighbour on the cleavage plane, the central cell will still not change state.

If the grain boudary is intact, and crack crosses it, mark this GB as fractured immediately in the local CG array and add this GB fracture to gcupd on this image.

Inputs:

- farr - (3,3,3) array of failed states, cell state type cgca_state_type_frac (9.23)
- marr - (3,3,3) array of material states, cell state type cgca_state_type_grain (9.24)
- n - vector defining the cleavage plane
- cstate - cleavage state
- debug - if .true. will dump some debug info

Outputs:

- newstate - updated central cell state

NOTES

The threshold analysis is explained in A. Shterenlikht, L. Margetts, Three-dimensional cellular automata modelling of cleavage propagation across crystal boundaries in polycrystalline microstructures, Proc. Roy. Soc. A 471:20150039, http://dx.doi.org/10.1098/rspa.2015.0039 . There are always at least 2 neighbouring cells for which dot_product(e,n) is less than ∼0.1732. However, 2 cells is not enough! If threshold is left at that, 2 cells lead to linear 1D cracks, which are not reasonable.

USES

cgca_gcr (11.7), cgca_gcf (11.4)

USED BY

cgca_clvgp (26.17.1)

SOURCE

```
integer( kind=idef ) :: &
 x1, x2, x3,            & ! coordinates within (3,3,3) neighbourhood
 img,                   & ! this_image()
 nimgs                    ! num_images()

integer :: iflag

logical :: intact

  img = this_image()
nimgs = num_images()

! initially the new central cell state is the same as old
newstate = farr( centre, centre, centre )

! scan all neighbourhood cells
outer: do x3 = l, u
       do x2 = l, u
       do x1 = l, u

  ! Check whether the neighbour has cleaved. The central cell is never
  ! cleaved, so the check will always fail for the central cell.
  clv: if ( any( cgca_clvg_states .eq. farr(x1,x2,x3)) ) then

    ! If the neighbour is from the same grain, then just check the
    ! orientation of the cleavage plane
    same: if ( marr(centre,centre,centre) .eq. marr(x1,x2,x3) ) then

      ! debug
      !  if (debug) then
      !   call cgca_gcr(1,2, intact)
      !   if (.not. intact) write (*,"(2(a,i0),a,3(tr1,f7.4),a,f7.4)") &
      !    "zzz: grain=",marr(centre,centre,centre), &
```

```
      !    ", cstate=", cstate, &
      !    ", n=(", n, ") ", &
      !    abs(dot_product(e(:,x1,x2,x3),n))
      !  end if

      ! If the dot product is smaller than the threshold, i.e. if
      ! the vector connecting the central cell with the neighbour
      ! is in the cleavage plane, then the central cell has cleaved
      if ( abs( dot_product( e(:,x1,x2,x3), n ) ) .lt. trshld ) then
        newstate = cstate  ! change its state
        exit outer         ! and exit the main loop
      end if

    else

      ! The neighbour is from a different grain. In this case
      ! the boundary must be intact in addition to the cleavage
      ! plane orientation criterion.
      intact = .false.

      ! Get the recorded state of the boundary between the two grains.
      ! cgca_gcr_pure returns .TRUE. if the boundary is intact
      ! and .FALSE. if the boundary is fractured. If iflag .ne. 0,
      ! then some error condition occurred, potentially fatal.
      ! However, to keep this routine PURE, we cannot abort here,
      ! so not sure what I can do with iflag, perhaps combine it
      ! somehow with the success flag of this subroutine?
      call cgca_gcr_pure( marr(centre,centre,centre), marr(x1,x2,x3),  &
                          intact, iflag )

      ! same as in the previous case, but with the added constraint
      ! that the grain boundary must be intact
      crossgb: if ( intact .and.                                      &
           abs( dot_product( e(:,x1,x2,x3), n ) ) .lt. trshld ) then

        ! change its state
        newstate = cstate

        ! Mark GB as fractured on this image
        ! The return diagnostic iflag is ignored for now.
        call cgca_gcf_pure( marr(centre, centre, centre),             &
          marr(x1, x2, x3), iflag )

        ! Add the (grain, neighbour) pair to gcupd coarray
        ! on this image
        gcupd( gcucnt, : ) = (/ marr( centre , centre , centre ) , &
                                marr(   x1   ,   x2   ,   x3   ) ,     &
                                cgca_gb_state_fractured /)

        ! debug output, not allowed in a pure procedure
!        if (debug)                                                    &
!          write (*,"(4(a,i0),2(tr1,i0),')',a,27(i0,tr1),').')")       &
```

```
!            "DEBUG: cgca_clvgsd: img: ", img, ": newstate=", newstate, &
!            ", gcucnt=", gcucnt,                                        &
!            ", calling cgca_gcf, gcupd=(", gcupd( gcucnt , : ),&
!            ", marr=(", marr

        ! increment the gcupd pair counter
        gcucnt = gcucnt + 1

        ! IO not allowed in a pure procedure, so just terminate without
        ! issuing the error.
        ! Issue fatal error if the length of the gcupd has been
        ! exceeded.
        if ( gcucnt .gt. cgca_gcupd_size1 ) then
!          write( *, '(a,i0)' ) "ERROR: cgca_m3clvg/cgca_clvgsd:&
!            & gcucnt .gt. cgca_gcupd_size1, image: ", img
          error stop
        end if

        ! now exit the main loop
        exit outer

      end if crossgb

    end if same

 end if clv

 ! now check another neighbouring cell, i.e. increment the loop counter

end do
end do
end do outer

end subroutine cgca_clvgsdt
```

26.7 cgca_m3clvg/cgca_clvgsp

[cgca_m3clvg] [Subroutines]

NAME

cgca_clvgsp

SYNOPSIS

```
subroutine cgca_clvgsp( farr, marr, n, cstate, debug, newstate )
```

PARAMETERS

```
integer,parameter :: l=-1, centre=l+1, u=centre+1
real(kind=rdef),parameter ::            &
 ltrshld = 0.17325932611400130485_rdef, & ! see cgca_clvgpd
 utrshld = 0.27_rdef, interval = utrshld-ltrshld
```

INPUTS

```
integer(kind=iarr),intent(in) :: farr(l:u,l:u,l:u), &
 marr(l:u,l:u,l:u), cstate
real(kind=rdef),intent(in) :: n(3)
logical(kind=ldef),intent(in) :: debug
```

OUTPUT

```
integer(kind=iarr),intent(out) :: newstate
```

DESCRIPTION

This routine determines the cleavage (CLVG) state (S) of the central cell. This is a probabilistic (P) routine. Hence the name is CLVGSP. If there is a cleaved neighbour, such that the vector connecting it to the central cell is on or close to the cleavage plane, then the central cell has a probability of changing state to the given cleavage state.

If the cleaved neighbour belongs to another grain, the analysis takes into account the grain boundary state. If the grain boundary is marked as intact, then the crack can cross it, and the central cell takes the values of the cleaved neighbour. If the grain boundary is marked as fractured, then the crack cannot cross it. This means that even if there is a cleaved neighbour on the cleavage plane, the central cell will still not change state.

Inputs:

- farr - array of failed states, cell state type cgca_state_type_frac (9.23)
- marr - array of material states, cel state type cgca_state_type_grain (9.24)
- n - vector defining the cleavage plane
- cstate - cleavage state
- debug - if .true. will dump some debug info

Outputs:

- newstate - updated central cell state

NOTES

This routine has two thresholds, the upper and the lower.

USES

cgca_gcr (11.7), cgca_gcf (11.4)

USED BY SOURCE

```
integer(kind=idef) :: x1,x2,x3
real(kind=rdef) :: rnd, proj, prob

logical :: intact

! initially the new central cell state is the same as old
newstate = farr(centre,centre,centre)

! scan all neighbourhood cells
outer: do x3=l,u
do x2=l,u
do x1=l,u

 ! Check whether the neighbour has cleaved. The central cell is never
 ! cleaved, so the check will always fail for the central cell.
 clv: if ( any( cgca_clvg_states .eq. farr(x1,x2,x3)) ) then

  proj = abs(dot_product(e(:,x1,x2,x3),n))

  same: if ( marr(centre,centre,centre) .eq. marr(x1,x2,x3) ) then
   ! The neighbour is from the same grain.
   ! If the central cell is close to the cleavage plane,
   ! change its state. This is a deterministic check for the
   ! lower threshold. If the central cell is further from the
   ! cleavage plane, but not too far, it has some probability
   ! to cleave. This is a probabilistic check for upper threshold.
   z1: if ( proj .lt. ltrshld ) then
    newstate = cstate
    exit outer
   else if ( proj .lt. utrshld ) then
    ! The power must be .ge. 1. If the power is 1,
    ! then the probability is a linear function of proj.
    ! If the power is > 1 then the probability is a power function.
    ! The higher the power, the steeper the descent.
    ! In other words, the higher the power, the lower the chances
    ! of cleavage for proj values greater than the lower threshold.
    prob = ((utrshld-proj) / interval)**1
    call random_number(rnd)
    if ( prob .gt. rnd) newstate = cstate
    exit outer
   end if z1
```

```
  else

   ! The neighbour is from another grain. As above, but the
   ! additional condition is that the grain boundary must be intact.
   intact = .false.
   call cgca_gcr(marr(centre,centre,centre), marr(x1,x2,x3), intact)
   inta: if ( intact ) then
    z2: if ( proj .lt. ltrshld ) then
     newstate = cstate

     ! Mark GB as fractured straight away
     call cgca_gcf( marr(centre,centre,centre), marr(x1,x2,x3) )

     ! debug output
     if (debug) write (*,"(2(a,i0),2(tr1,i0),').')")                  &
       "DEBUG: cgca_clvgsd: image ", this_image(),                    &
       ": called _gcf, set gcupd=(", gcupd

     exit outer
    else if ( proj .lt. utrshld ) then
     prob = ((utrshld-proj) / interval)**1
     call random_number(rnd)
     if ( prob .gt. rnd) then
      newstate = cstate
      ! Mark GB as fractured straight away
      call cgca_gcf( marr(centre,centre,centre), marr(x1,x2,x3) )
      ! debug output
      if (debug) write (*,"(2(a,i0),2(tr1,i0),').')")                 &
        "DEBUG: cgca_clvgsd: image ", this_image(),                  &
        ": called _gcf, set gcupd=(", gcupd
      exit outer
     end if
    end if z2
   end if inta

  end if same
 end if clv
end do
end do
end do outer

end subroutine cgca_clvgsp
```

26.8 cgca_m3clvg/cgca_clvgspt

[cgca_m3clvg] [Subroutines]

NAME

cgca_clvgspt

SYNOPSIS

```
pure subroutine cgca_clvgspt( farr, marr, n, cstate, debug, newstate )
```

PARAMETERS

```
integer,parameter :: l=-1, centre=l+1, u=centre+1
real(kind=rdef),parameter ::            &
 ltrshld = 0.17325932611400130485_rdef, & ! see cgca_clvgpd
 utrshld = 0.27_rdef, interval = utrshld-ltrshld
```

INPUTS

```
integer(kind=iarr),intent(in) :: farr(l:u,l:u,l:u), &
 marr(l:u,l:u,l:u), cstate
real(kind=rdef),intent(in) :: n(3)
logical(kind=ldef),intent(in) :: debug
```

OUTPUT

```
integer(kind=iarr),intent(out) :: newstate
```

DESCRIPTION

This routine determines the cleavage (CLVG) state (S) of the central cell. This is a probabilistic (P) routine. Hence the name is CLVGSP. If there is a cleaved neighbour, such that the vector connecting it to the central cell is on or close to the cleavage plane, then the central cell has a probability of changing state to the given cleavage state.

If the cleaved neighbour belongs to another grain, the analysis takes into account the grain boundary state. If the grain boundary is marked as intact, then the crack can cross it, and the central cell takes the values of the cleaved neighbour. If the grain boundary is marked as fractured, then the crack cannot cross it. This means that even if there is a cleaved neighbour on the cleavage plane, the central cell will still not change state.

Inputs:

- farr - array of failed states, cell state type cgca_state_type_frac (9.23)
- marr - array of material states, cel state type cgca_state_type_grain (9.24)
- n - vector defining the cleavage plane
- cstate - cleavage state
- debug - if .true. will dump some debug info

Outputs:

- newstate - updated central cell state

NOTES

This routine has two thresholds, the upper and the lower.

USES

cgca_gcr (11.7), cgca_gcf (11.4)

USED BY SOURCE

```
integer(kind=idef) :: x1,x2,x3
real(kind=rdef) :: rnd, proj, prob

logical :: intact

! initially the new central cell state is the same as old
newstate = farr(centre,centre,centre)

! scan all neighbourhood cells
outer: do x3=l,u
do x2=l,u
do x1=l,u

 ! Check whether the neighbour has cleaved. The central cell is never
 ! cleaved, so the check will always fail for the central cell.
 clv: if ( any( cgca_clvg_states .eq. farr(x1,x2,x3)) ) then

  proj = abs(dot_product(e(:,x1,x2,x3),n))

  same: if ( marr(centre,centre,centre) .eq. marr(x1,x2,x3) ) then
   ! The neighbour is from the same grain.
   ! If the central cell is close to the cleavage plane,
   ! change its state. This is a deterministic check for the
   ! lower threshold. If the central cell is further from the
   ! cleavage plane, but not too far, it has some probability
   ! to cleave. This is a probabilistic check for upper threshold.
   z1: if ( proj .lt. ltrshld ) then
    newstate = cstate
    exit outer
   else if ( proj .lt. utrshld ) then
    ! The power must be .ge. 1. If the power is 1,
    ! then the probability is a linear function of proj.
    ! If the power is > 1 then the probability is a power function.
    ! The higher the power, the steeper the descent.
    ! In other words, the higher the power, the lower the chances
    ! of cleavage for proj values greater than the lower threshold.
    prob = ((utrshld-proj) / interval)**1
    call random_number(rnd)
    if ( prob .gt. rnd) newstate = cstate
    exit outer
   end if z1
```

```
  else

   ! The neighbour is from another grain. As above, but the
   ! additional condition is that the grain boundary must be intact.
   intact = .false.
   call cgca_gcr(marr(centre,centre,centre), marr(x1,x2,x3), intact)
   inta: if ( intact ) then
    z2: if ( proj .lt. ltrshld ) then
     newstate = cstate

     ! Mark GB as fractured straight away
     call cgca_gcf( marr(centre,centre,centre), marr(x1,x2,x3) )

     ! debug output
     if (debug) write (*,"(2(a,i0),2(tr1,i0),').')")                  &
       "DEBUG: cgca_clvgsd: image ", this_image(),                    &
       ": called _gcf, set gcupd=(", gcupd

     exit outer
    else if ( proj .lt. utrshld ) then
     prob = ((utrshld-proj) / interval)**1
     call random_number(rnd)
     if ( prob .gt. rnd) then
      newstate = cstate
      ! Mark GB as fractured straight away
      call cgca_gcf( marr(centre,centre,centre), marr(x1,x2,x3) )
      ! debug output
      if (debug) write (*,"(2(a,i0),2(tr1,i0),').')")                 &
        "DEBUG: cgca_clvgsd: image ", this_image(),                  &
        ": called _gcf, set gcupd=(", gcupd
      exit outer
     end if
    end if z2
   end if inta

  end if same
 end if clv
end do
end do
end do outer

end subroutine cgca_clvgspt
```

26.9 cgca_m3clvg/cgca_dacf

[cgca_m3clvg] [Subroutines]

NAME

cgca_dacf

SYNOPSIS

```
subroutine cgca_dacf( coarray , debug )
```

INPUTS

```
integer( kind=iarr ), allocatable,intent(inout) ::                    &
 coarray(:,:,:,:)[:,:,:]
logical( kind=ldef ), intent(in) :: debug
```

SIDE EFFECTS

Changed state of coarray

DESCRIPTION

This routine DeActivates Crack Flanks, hence the name DACF. The idea is that we must distinguish crack edge cells, which can attract new cracked (cleaved) cells, and crack flanks, which are inactive, i.e. cells representing crack flanks have very low SIF (stress intensity factor), and maybe low stresses too. Crack flank cells cannot attract new cleaved cells. The cell states are defined in module cgca_m1co (9).

The distinction is made based on the number of cleaved neighbours. If there are too many cleaved neighbours, then the central cell is a crack flank.

The intention is that this routine is called after every cleavage propagation increment, to prevent cracks becoming large 3D bodies.

This routines runs only once over the coarray. So we don't put the sync here. But a sync all probably should be used before and after a call to this routine. Also, the halo exchange probably should be done after running this routine.

If debug=.true. then will dump *lots* of debug output

USES USED BY

cgca_clvgp (26.17.1)

SOURCE

```
integer, parameter ::                                                  &
 lclvg_states = lbound( cgca_clvg_states , dim=1 ),                    &
 uclvg_states = ubound( cgca_clvg_states , dim=1 ),                    &
 l=-1, centre=l+1, u=centre+1

integer( kind=idef ) :: &
 lbv(4) ,& ! lower bounds of the complete (plus virtual) coarray
 ubv(4) ,& ! upper bounds of the complete (plus virtual) coarray
 lbr(4) ,& ! lower bounds of the "real" coarray, lower virtual+1
 ubr(4) ,& ! upper bounds of the "real" coarray, upper virtual-1
 x1     ,& ! local coordinates in an array, which are also
```

```
 x2     ,& ! do loop counters
 x3
integer( kind=iarr ), allocatable, save :: array(:,:,:)
integer( kind=iarr ) :: neiarr(l:u,l:u,l:u)
integer :: thisimage, errstat=0, ncount, i
logical( kind=ldef ) :: clvnei(l:u,l:u,l:u), &
 samegrain(l:u,l:u,l:u), csg(l:u,l:u,l:u)

! get image number
thisimage = this_image()

! no sanity checks for speed

! determine the extents
lbv = lbound(coarray)
ubv = ubound(coarray)
lbr = lbv+1
ubr = ubv-1

! allocate the temp array
if (.not. allocated(array)) allocate( &
  array(lbv(1):ubv(1), lbv(2):ubv(2), lbv(3):ubv(3)), stat=errstat)
if (errstat.ne.0) then
  write (*,'(a,i0)') "ERROR: cgca_dacf: image ",thisimage
  write (*,'(a)') "ERROR: cgca_dacf: cannot allocate array"
  error stop
end if

! copy coarray state type 2, fracture states, to temp array
array = coarray(:,:,:,cgca_state_type_frac)

! loop over all cells
do x3 = lbr(3), ubr(3)
do x2 = lbr(2), ubr(2)
do x1 = lbr(1), ubr(1)

 ! analyse only crack edge cells
 cleav: if ( any (coarray(x1,x2,x3,cgca_state_type_frac) .eq.        &
          cgca_clvg_states_edge )) then

   ! count the number of cleaved neighbours of the same grain
   neiarr = coarray( x1-1:x1+1, x2-1:x2+1, x3-1:x3+1,                &
                     cgca_state_type_frac )

   ! Do not count the central cell, so set its state to some
   ! value that is not in the cgca_clvg_states. I use the intact
   ! state here.
   neiarr(centre,centre,centre) = cgca_intact_state

   ! logical array: .true. if the values are identical,
   ! .false. otherwise. samegrain is a (3,3,3) neighbourhood
   ! array around the central cell in question.
```

```
   samegrain = coarray( x1, x2, x3, cgca_state_type_grain ) .eq.     &
               coarray( x1-1:x1+1, x2-1:x2+1, x3-1:x3+1,             &
                        cgca_state_type_grain )

   ! debug
   if (debug) write (*,'(a,i0,a,27(tr1,i0),a,27(tr1,l1))')           &
    "DEBUG: cgca_dacf: image ", thisimage, ": neiarr=", neiarr,      &
    ", samegrain=", samegrain

   ncount = 0

   ! for all cleavage states
   do i = lclvg_states, uclvg_states
     clvnei = cgca_clvg_states(i) .eq. neiarr
        csg = clvnei .and. samegrain
     ncount = ncount + count(csg)

     ! debug
     if (debug) write (*,'(4(a,i0),2(a,27(tr1,l1)),a,i0)')           &
       "DEBUG: cgca_dacf: image ", thisimage, ": i=", i,             &
       ", cgca_clvg_states(",i,")=", cgca_clvg_states(i),            &
       ", clvnei=", clvnei, ", csg=", csg, ", ncount=", ncount

   end do

   ! if a cell has 6 cleaved neighbours or more,
   ! then mark it as crack flank.
   nei: if ( ncount .ge. 6) then

     ! debug
     if (debug) write (*,'(3(a,i0),2(tr1,i0),a)')                    &
       "DEBUG: cgca_dacf: image ", thisimage,                        &
       ": grain=", coarray(x1,x2,x3,cgca_state_type_grain),          &
       ", crack front cell x1,x2,x3=", x1, x2, x3, " deactivated."

     ! change the state preserving the cleavage plane family
     ! Note: we are reading from "coarray" but writing into
     ! the temp "array"
     if ( coarray( x1, x2 ,x3, cgca_state_type_frac ) .eq.           &
                     cgca_clvg_state_100_edge ) then
       array(x1,x2,x3) = cgca_clvg_state_100_flank
     else if ( coarray( x1, x2, x3, cgca_state_type_frac ) .eq.      &
                     cgca_clvg_state_110_edge ) then
       array(x1,x2,x3) = cgca_clvg_state_110_flank
     else
       array(x1,x2,x3) = cgca_clvg_state_111_flank
     end if

   end if nei
 end if cleav
end do
end do
```

```
end do

! write array to coarray
coarray(:,:,:,cgca_state_type_frac) = array

! do not deallocate array. Let it exist until the program
! terminates.

end subroutine cgca_dacf
```

26.10 cgca_m3clvg/cgca_dacf1

[cgca_m3clvg] [Subroutines]

NAME

cgca_dacf1

SYNOPSIS

```
subroutine cgca_dacf1(coarray,debug)
```

INPUTS

```
integer( kind=iarr ), allocatable, intent(inout) ::                    &
 coarray(:,:,:,:)[:,:,:]
logical( kind=ldef ), intent(in) :: debug
```

SIDE EFFECTS

Changed state of coarray

DESCRIPTION

Same as cgca_dacf (26.9), but no attention is paid to which grain a cell belongs to. So when we count the number of fractured neighbours, these can be from any grain. What this means is that, when a crack propagates from one grain to another, and the crack planes IN THE MODEL coinside, this routine will deactivate the fractured cells on the interface. This helps the grain boundary fracture analysis. Since there are no crack fronts at the boundary, there is no bondary to fracture! In other words, if a crack really slices from one grain to another with no deviation, then the grain boundary fracture does not happen, the two grain system in question is already fully separated into two different bodies.

USES USED BY

cgca_clvgp (26.17.1)

SOURCE

```
integer,parameter :: lclvg_states = lbound(cgca_clvg_states,dim=1), &
 uclvg_states = ubound(cgca_clvg_states,dim=1),                     &
 l=-1, centre=l+1, u=centre+1

integer(kind=idef) :: &
  lbv(4) ,& ! lower bounds of the complete (plus virtual) coarray
  ubv(4) ,& ! upper bounds of the complete (plus virtual) coarray
  lbr(4) ,& ! lower bounds of the "real" coarray, lower virtual+1
  ubr(4) ,& ! upper bounds of the "real" coarray, upper virtual-1
  x1     ,& ! local coordinates in an array, which are also
  x2     ,& ! do loop counters
  x3        !
integer(kind=iarr),allocatable,save :: array(:,:,:)
integer(kind=iarr) :: neiarr(l:u,l:u,l:u)
integer :: thisimage, errstat=0, ncount, i
logical(kind=ldef) :: clvnei(l:u,l:u,l:u)

! get image number
```

```
thisimage=this_image()

! no sanity checks for speed

! determine the extents
lbv = lbound(coarray)
ubv = ubound(coarray)
lbr = lbv + 1
ubr = ubv - 1

! allocate the temp array

if ( .not. allocated( array ) )                                              &
 allocate( array( lbv(1):ubv(1), lbv(2):ubv(2), lbv(3):ubv(3) ), stat=errstat )
if (errstat.ne.0) then
  write (*,'(2(a,i0))') "****** ERROR: cgca_dacf1: image ",thisimage,        &
     ": cannot allocate array, error code ", errstat
  error stop
end if

! copy coarray state type 2, fracture states, to temp array
array = coarray(:,:,:,cgca_state_type_frac)

! loop over all cells
do x3 = lbr(3),ubr(3)
do x2 = lbr(2),ubr(2)
do x1 = lbr(1),ubr(1)

 ! analyse only crack edge cells
 cleav: if ( any (coarray(x1,x2,x3,cgca_state_type_frac) .eq.      &
          cgca_clvg_states_edge )) then

  ! Count the number of cleaved neighbours. Do not count the central
  ! cell, so set its state to some value that is not in the
  ! cgca_clvg_states. I use the intact state here.
  neiarr = coarray(x1-1:x1+1, x2-1:x2+1, x3-1:x3+1, cgca_state_type_frac)
  neiarr(centre,centre,centre) = cgca_intact_state

  ncount = 0
  do i=lclvg_states,uclvg_states
   clvnei = cgca_clvg_states(i) .eq. neiarr
   ncount = ncount + count(clvnei)

   ! debug
   if (debug) write (*,'(4(a,i0),a,27(tr1,l1),a,i0)')              &
    "DEBUG: cgca_dacf: image ", thisimage, ": i=", i,              &
    ", cgca_clvg_states(",i,")=", cgca_clvg_states(i),             &
    ", clvnei=", clvnei, ", ncount=", ncount

  end do

  ! if a cell has 5 cleaved neighbours or more,
```

```
  ! then mark it as crack flank.
  nei: if ( ncount .ge. 5) then

   ! debug
   if (debug) write (*,'(3(a,i0),2(tr1,i0),a)')                    &
    "DEBUG: cgca_dacf1: image ", thisimage,                        &
    ": grain=", coarray(x1,x2,x3,cgca_state_type_grain),           &
    ", crack front cell x1,x2,x3=", x1, x2, x3, " deactivated."

   ! change the state preserving the cleavage plane family
   ! Note: we are reading from "coarray" but writing into the temp
   ! "array"
   if ( coarray(x1,x2,x3,cgca_state_type_frac) .eq.                &
         cgca_clvg_state_100_edge ) then
     array(x1,x2,x3) = cgca_clvg_state_100_flank
   else if ( coarray(x1,x2,x3,cgca_state_type_frac) .eq.           &
          cgca_clvg_state_110_edge ) then
     array(x1,x2,x3) = cgca_clvg_state_110_flank
   else
     array(x1,x2,x3) = cgca_clvg_state_111_flank
   end if

  end if nei
 end if cleav
end do
end do
end do

! write array to coarray
coarray(:,:,:,cgca_state_type_frac) = array

! do not deallocate array. Let it exist until the program
! terminates.

end subroutine cgca_dacf1
```

26.11 cgca_m3clvg/cgca_dacf1t

[cgca_m3clvg] [Subroutines]

NAME

cgca_dacf1t

SYNOPSIS

```
subroutine cgca_dacf1t(coarray,debug)
```

INPUTS

```
integer( kind=iarr ), allocatable, intent(inout) ::                    &
 coarray(:,:,:,:)[:,:,:]
logical( kind=ldef ), intent(in) :: debug
```

SIDE EFFECTS

Changed state of coarray

DESCRIPTION

Same as cgca_dacf (26.9), but no attention is paid to which grain a cell belongs to. So when we count the number of fractured neighbours, these can be from any grain. What this means is that, when a crack propagates from one grain to another, and the crack planes IN THE MODEL coinside, this routine will deactivate the fractured cells on the interface. This helps the grain boundary fracture analysis. Since there are no crack fronts at the boundary, there is no bondary to fracture! In other words, if a crack really slices from one grain to another with no deviation, then the grain boundary fracture does not happen, the two grain system in question is already fully separated into two different bodies.

USES USED BY

cgca_clvgp (26.17.1)

SOURCE

```
integer,parameter :: lclvg_states = lbound(cgca_clvg_states,dim=1), &
 uclvg_states = ubound(cgca_clvg_states,dim=1),                     &
 l=-1, centre=l+1, u=centre+1

integer(kind=idef) :: &
  lbv(4) ,& ! lower bounds of the complete (plus virtual) coarray
  ubv(4) ,& ! upper bounds of the complete (plus virtual) coarray
  lbr(4) ,& ! lower bounds of the "real" coarray, lower virtual+1
  ubr(4) ,& ! upper bounds of the "real" coarray, upper virtual-1
  x1     ,& ! local coordinates in an array, which are also
  x2     ,& ! do loop counters
  x3        !
integer(kind=iarr),allocatable,save :: array(:,:,:)
integer(kind=iarr) :: neiarr(l:u,l:u,l:u)
integer :: thisimage, errstat=0, ncount, i
logical(kind=ldef) :: clvnei(l:u,l:u,l:u)

! get image number
```

```
thisimage=this_image()

! no sanity checks for speed

! determine the extents
lbv = lbound(coarray)
ubv = ubound(coarray)
lbr = lbv + 1
ubr = ubv - 1

! allocate the temp array

if ( .not. allocated( array ) )                                              &
 allocate( array( lbv(1):ubv(1), lbv(2):ubv(2), lbv(3):ubv(3) ), stat=errstat )
if (errstat.ne.0) then
  write (*,'(2(a,i0))') "****** ERROR: cgca_dacf1: image ",thisimage,        &
     ": cannot allocate array, error code ", errstat
  error stop
end if

! copy coarray state type 2, fracture states, to temp array
array = coarray(:,:,:,cgca_state_type_frac)

! loop over all cells
do x3 = lbr(3),ubr(3)
do x2 = lbr(2),ubr(2)
do x1 = lbr(1),ubr(1)

 ! analyse only crack edge cells
 cleav: if ( any (coarray(x1,x2,x3,cgca_state_type_frac) .eq.      &
          cgca_clvg_states_edge )) then

  ! Count the number of cleaved neighbours. Do not count the central
  ! cell, so set its state to some value that is not in the
  ! cgca_clvg_states. I use the intact state here.
  neiarr = coarray(x1-1:x1+1, x2-1:x2+1, x3-1:x3+1, cgca_state_type_frac)
  neiarr(centre,centre,centre) = cgca_intact_state

  ncount = 0
  do i=lclvg_states,uclvg_states
   clvnei = cgca_clvg_states(i) .eq. neiarr
   ncount = ncount + count(clvnei)

   ! debug
   if (debug) write (*,'(4(a,i0),a,27(tr1,l1),a,i0)')              &
    "DEBUG: cgca_dacf: image ", thisimage, ": i=", i,              &
    ", cgca_clvg_states(",i,")=", cgca_clvg_states(i),             &
    ", clvnei=", clvnei, ", ncount=", ncount

  end do

  ! if a cell has 5 cleaved neighbours or more,
```

```
  ! then mark it as crack flank.
  nei: if ( ncount .ge. 5) then

   ! debug
   if (debug) write (*,'(3(a,i0),2(tr1,i0),a)')                    &
    "DEBUG: cgca_dacf1: image ", thisimage,                        &
    ": grain=", coarray(x1,x2,x3,cgca_state_type_grain),           &
    ", crack front cell x1,x2,x3=", x1, x2, x3, " deactivated."

   ! change the state preserving the cleavage plane family
   ! Note: we are reading from "coarray" but writing into the temp
   ! "array"
   if ( coarray(x1,x2,x3,cgca_state_type_frac) .eq.                &
         cgca_clvg_state_100_edge ) then
     array(x1,x2,x3) = cgca_clvg_state_100_flank
   else if ( coarray(x1,x2,x3,cgca_state_type_frac) .eq.           &
          cgca_clvg_state_110_edge ) then
     array(x1,x2,x3) = cgca_clvg_state_110_flank
   else
     array(x1,x2,x3) = cgca_clvg_state_111_flank
   end if

  end if nei
 end if cleav
end do
end do
end do

! write array to coarray
coarray(:,:,:,cgca_state_type_frac) = array

! do not deallocate array. Let it exist until the program
! terminates.

end subroutine cgca_dacf1t
```

26.12 cgca_m3clvg/cgca_dacft

[cgca_m3clvg] [Subroutines]

NAME

cgca_dacft

SYNOPSIS

```
subroutine cgca_dacft( coarray , debug )
```

INPUTS

```
integer( kind=iarr ), allocatable,intent(inout) ::                    &
 coarray(:,:,:,:)[:,:,:]
logical( kind=ldef ), intent(in) :: debug
```

SIDE EFFECTS

Changed state of coarray

DESCRIPTION

This routine DeActivates Crack Flanks, hence the name DACF. The idea is that we must distinguish crack edge cells, which can attract new cracked (cleaved) cells, and crack flanks, which are inactive, i.e. cells representing crack flanks have very low SIF (stress intensity factor), and maybe low stresses too. Crack flank cells cannot attract new cleaved cells. The cell states are defined in module cgca_m1co (9).

The distinction is made based on the number of cleaved neighbours. If there are too many cleaved neighbours, then the central cell is a crack flank.

The intention is that this routine is called after every cleavage propagation increment, to prevent cracks becoming large 3D bodies.

This routines runs only once over the coarray. So we don't put the sync here. But a sync all probably should be used before and after a call to this routine. Also, the halo exchange probably should be done after running this routine.

If debug=.true. then will dump *lots* of debug output

USES USED BY

cgca_clvgp (26.17.1)

SOURCE

```
integer, parameter ::                                                  &
 lclvg_states = lbound( cgca_clvg_states , dim=1 ),                    &
 uclvg_states = ubound( cgca_clvg_states , dim=1 ),                    &
 l=-1, centre=l+1, u=centre+1

integer( kind=idef ) :: &
 lbv(4) ,& ! lower bounds of the complete (plus virtual) coarray
 ubv(4) ,& ! upper bounds of the complete (plus virtual) coarray
 lbr(4) ,& ! lower bounds of the "real" coarray, lower virtual+1
 ubr(4) ,& ! upper bounds of the "real" coarray, upper virtual-1
 x1     ,& ! local coordinates in an array, which are also
```

```
 x2     ,& ! do loop counters
 x3
integer( kind=iarr ), allocatable, save :: array(:,:,:)
integer( kind=iarr ) :: neiarr(l:u,l:u,l:u)
integer :: thisimage, errstat=0, ncount, i
logical( kind=ldef ) :: clvnei(l:u,l:u,l:u), &
 samegrain(l:u,l:u,l:u), csg(l:u,l:u,l:u)

! get image number
thisimage = this_image()

! no sanity checks for speed

! determine the extents
lbv = lbound(coarray)
ubv = ubound(coarray)
lbr = lbv+1
ubr = ubv-1

! allocate the temp array
if (.not. allocated(array)) allocate( &
  array(lbv(1):ubv(1), lbv(2):ubv(2), lbv(3):ubv(3)), stat=errstat)
if (errstat.ne.0) then
  write (*,'(a,i0)') "ERROR: cgca_dacf: image ",thisimage
  write (*,'(a)') "ERROR: cgca_dacf: cannot allocate array"
  error stop
end if

! copy coarray state type 2, fracture states, to temp array
array = coarray(:,:,:,cgca_state_type_frac)

! loop over all cells
do x3 = lbr(3), ubr(3)
do x2 = lbr(2), ubr(2)
do x1 = lbr(1), ubr(1)

 ! analyse only crack edge cells
 cleav: if ( any (coarray(x1,x2,x3,cgca_state_type_frac) .eq.        &
          cgca_clvg_states_edge )) then

   ! count the number of cleaved neighbours of the same grain
   neiarr = coarray( x1-1:x1+1, x2-1:x2+1, x3-1:x3+1,                &
                     cgca_state_type_frac )

   ! Do not count the central cell, so set its state to some
   ! value that is not in the cgca_clvg_states. I use the intact
   ! state here.
   neiarr(centre,centre,centre) = cgca_intact_state

   ! logical array: .true. if the values are identical,
   ! .false. otherwise. samegrain is a (3,3,3) neighbourhood
   ! array around the central cell in question.
```

```
   samegrain = coarray( x1, x2, x3, cgca_state_type_grain ) .eq.     &
               coarray( x1-1:x1+1, x2-1:x2+1, x3-1:x3+1,             &
                        cgca_state_type_grain )

   ! debug
   if (debug) write (*,'(a,i0,a,27(tr1,i0),a,27(tr1,l1))')           &
    "DEBUG: cgca_dacf: image ", thisimage, ": neiarr=", neiarr,      &
    ", samegrain=", samegrain

   ncount = 0

   ! for all cleavage states
   do i = lclvg_states, uclvg_states
     clvnei = cgca_clvg_states(i) .eq. neiarr
        csg = clvnei .and. samegrain
     ncount = ncount + count(csg)

     ! debug
     if (debug) write (*,'(4(a,i0),2(a,27(tr1,l1)),a,i0)')           &
       "DEBUG: cgca_dacf: image ", thisimage, ": i=", i,             &
       ", cgca_clvg_states(",i,")=", cgca_clvg_states(i),            &
       ", clvnei=", clvnei, ", csg=", csg, ", ncount=", ncount

   end do

   ! if a cell has 6 cleaved neighbours or more,
   ! then mark it as crack flank.
   nei: if ( ncount .ge. 6) then

     ! debug
     if (debug) write (*,'(3(a,i0),2(tr1,i0),a)')                    &
       "DEBUG: cgca_dacf: image ", thisimage,                        &
       ": grain=", coarray(x1,x2,x3,cgca_state_type_grain),          &
       ", crack front cell x1,x2,x3=", x1, x2, x3, " deactivated."

     ! change the state preserving the cleavage plane family
     ! Note: we are reading from "coarray" but writing into
     ! the temp "array"
     if ( coarray( x1, x2 ,x3, cgca_state_type_frac ) .eq.           &
                     cgca_clvg_state_100_edge ) then
       array(x1,x2,x3) = cgca_clvg_state_100_flank
     else if ( coarray( x1, x2, x3, cgca_state_type_frac ) .eq.      &
                     cgca_clvg_state_110_edge ) then
       array(x1,x2,x3) = cgca_clvg_state_110_flank
     else
       array(x1,x2,x3) = cgca_clvg_state_111_flank
     end if

   end if nei
 end if cleav
end do
end do
```

```
end do

! write array to coarray
coarray(:,:,:,cgca_state_type_frac) = array

! do not deallocate array. Let it exist until the program
! terminates.

end subroutine cgca_dacft
```

26.13 cgca_m3clvg/gcupd_alloc

[cgca_m3clvg] [Subroutines]

NAME

gcupd_alloc

SYNOPSIS

```
subroutine gcupd_alloc
```

SIDE EFFECTS

gcupd is (re)allocated.

DESCRIPTION

This is a private routine, hence the name does not start with cgca_. This routine allocates gcupd array coarray. Therefore it involves implicit sync all. If gcupd is allocated already, it is first deallocated and then reallocated with new codimensions.

USES USED BY SOURCE

```
integer :: errstat = 0

! Deallocate if already allocated
if ( allocated( gcupd ) ) then
  deallocate( gcupd, stat=errstat )
  if ( errstat .ne. 0 ) then
    write (*,'(a,i0)') "ERROR: cgca_m3clvg/gcupd_alloc:&
     & deallocate( gcupd ), err. code:", errstat
    error stop
  end if
end if

! Allocate and set all values to cgca_gb_state_intact.
allocate( gcupd( cgca_gcupd_size1, cgca_gcupd_size2 )               &
  [ cgca_slcob(1):cgca_sucob(1), cgca_slcob(2):cgca_sucob(2),          &
    cgca_slcob(3):* ], source = cgca_gb_state_intact, stat=errstat )
if ( errstat .ne. 0 ) then
  write (*,'(a,i0)') "ERROR: cgca_m3clvg/gcupd_alloc:&
     & allocate( gcupd ), err. code:", errstat
  error stop
end if

end subroutine gcupd_alloc
```

26.14 cgca_m3clvg/gcupd_alloct

[cgca_m3clvg] [Subroutines]

NAME

gcupd_alloct

SYNOPSIS

```
subroutine gcupd_alloct
```

SIDE EFFECTS

gcupd is (re)allocated.

DESCRIPTION

This is a private routine, hence the name does not start with cgca_. This routine allocates gcupd array coarray. Therefore it involves implicit sync all. If gcupd is allocated already, it is first deallocated and then reallocated with new codimensions.

USES USED BY SOURCE

```
integer :: errstat = 0

! Deallocate if already allocated
if ( allocated( gcupd ) ) then
  deallocate( gcupd, stat=errstat )
  if ( errstat .ne. 0 ) then
    write (*,'(a,i0)') "ERROR: cgca_m3clvg/gcupd_alloc:&
     & deallocate( gcupd ), err. code:", errstat
    error stop
  end if
end if

! Allocate and set all values to cgca_gb_state_intact.
allocate( gcupd( cgca_gcupd_size1, cgca_gcupd_size2 )               &
  [ cgca_slcob(1):cgca_sucob(1), cgca_slcob(2):cgca_sucob(2),          &
    cgca_slcob(3):* ], source = cgca_gb_state_intact, stat=errstat )
if ( errstat .ne. 0 ) then
  write (*,'(a,i0)') "ERROR: cgca_m3clvg/gcupd_alloc:&
     & allocate( gcupd ), err. code:", errstat
  error stop
end if

end subroutine gcupd_alloct
```

26.15 cgca_m3clvg/m3clvg_sm1

[cgca_m3clvg] [Submodules]

NAME

m3clvg_sm1

SYNOPSIS

```
!$Id: m3clvg_sm1.f90 491 2018-02-20 22:22:58Z mexas $

submodule ( cgca_m3clvg ) m3clvg_sm1
```

DESCRIPTION

Submodule of module cgca_m3clvg (26). It contains subroutines dealing with updating the grain connectivity array.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

CONTAINS

cgca_gcupda (26.15.1), cgca_gcupdn (26.15.2)

USES

All variables and parameters of module cgca_m3clvg (26) by host association

USED BY

The parent module cgca_m3clvg (26).

SOURCE

```
implicit none

contains
```

26.15.1 m3clvg_sm1/cgca_gcupda

[m3clvg_sm1] [Subroutines]

NAME

cgca_gcupda

SYNOPSIS

```
  module subroutine cgca_gcupda( periodicbc )
    logical( kind=ldef ), intent( in ) :: periodicbc
```

SIDE EFFECTS

State of GB array in module cgca_m2gb (11) is updated.

DESCRIPTION

This routine reads gcupd from *all* images and adds the pairs to the local GB array on this image. In other words it implements an all-to-all communication pattern. If you want to use just the nearest neighbouring images, use cgca_gcupdn (26.15.2) instead. Synchronisation must be used before and after calling this routine, to comply with the standard.

NOTES

logical var periodicbc that is passed as input is ignored. This input var is provided only to make the interface this routine identical to cgca_gcupdn (26.15.2). Identical interfaces allow creating an abstract interface for both gcupd and gcupdn and thus pass the name of the routine as input to cgca_clvgp (26.17.1)* cleavage propagation routines.

USES

cgca_gcf (11.4), cgca_ico (13.2)

SOURCE

```
integer( kind=idef ) :: i, j, img, nimgs, img_curr, rndint,            &
  cosub( cgca_scodim ), flag
integer( kind=kind(gcupd) ) ::                                         &
  gcupd_local( cgca_gcupd_size1, cgca_gcupd_size2 )
real :: rnd

logical :: l_tmp

! Just to suppress the unused compiler warning
l_tmp = periodicbc

  img = this_image()
nimgs = num_images()

! choose the first image at random
call random_number( rnd )   ! [ 0 .. 1 )
rndint = int( rnd*nimgs )+1 ! [ 1 .. nimgs ]

! Initialise flag with any value
flag = 0
```

```
! loop over all images, starting at a randomly chosen image
images: do j = rndint, rndint+nimgs-1

  ! Get the current image number.
  ! If it's > nimgs, subtract nimgs
  img_curr = j
  if ( img_curr .gt. nimgs ) img_curr = img_curr - nimgs

  ! Skip this image, because the gc array has already been updated
  ! in cgca_clvgsd.
  if ( img_curr .eq. img ) cycle images

  ! Calculate cosubscripts:
  call cgca_ico( img_curr, cosub, flag )
  if ( flag .ne. 0 ) then
    write (*,*) &
      "ERROR: m3clvg/cgca_gcupda: cgca_ico exit with error: flag:", flag
    error stop
  end if

  ! copy gcupd from image j into a local var
  gcupd_local( : , : ) = gcupd( : , : ) [ cosub(1), cosub(2), cosub(3) ]

  gcarray: do i = 1, cgca_gcupd_size1

    ! The gcupd array is filled with fractured pairs from the beginning
    ! so exit as soon as the GB state is intact.
    if ( gcupd_local( i , 3 ) .eq. cgca_gb_state_intact ) exit gcarray

!write (*,*) "DEBUG: cgca_gcupd: img:", img, &
!            "gcupd_local(i,:):", gcupd_local( i , : )

    ! add the pair to the gc array on this image
    call cgca_gcf( gcupd_local( i , 1 ), gcupd_local( i , 2 ) )

 end do gcarray
end do images

end subroutine cgca_gcupda
```

26.15.2 m3clvg_sm1/cgca_gcupdn

[m3clvg_sm1] [Subroutines]

NAME

cgca_gcupdn

SYNOPSIS

```
  module subroutine cgca_gcupdn( periodicbc )
    logical( kind=ldef ), intent( in ) :: periodicbc
```

INPUT

```
!    periodicbc - logical, .true. if the CA space has periodic BC,
!    and .false. otherwise.
```

SIDE EFFECTS

State of GB array in module cgca_m2gb (11) is updated

DESCRIPTION

This routine reads gcupd from the *nearest neighbouring* images only, and adds the pairs to the local GB array on this image. If you want to read from all images, use cgca_gcupd. Synchronisation must be used before and after calling this routine, to comply with the standard.

NOTES

This routine must be used only after gcupd has been allocated. A runtime error will result if gcupd has not been allocated yet.

USES

cgca_gcf (11.4)

SOURCE

```
integer( kind=idef ) :: i, j, k, s, mycod( cgca_scodim ),              &
 neicod( cgca_scodim )
integer( kind=kind(gcupd) ) ::                                         &
 gcupd_local( cgca_gcupd_size1, cgca_gcupd_size2 )

! Get my coindex set
if ( .not. allocated( gcupd ) ) then
  write (*,'(a)')                                                      &
     "ERROR: cgca_m3clvg/cgca_gcupdn: gcupd not allocated"
  error stop
end if
mycod = this_image( gcupd )

! Loop over all nearest neighbours, taking special attention of
! the images at the edges of the model
do i = -1 , 1
do j = -1 , 1
inner: do k = -1 , 1
```

```
  ! Get the coindex set of the neighbour
  neicod = mycod + (/ i, j, k /)

  ! Skip this image
  if ( all( neicod .eq. mycod ) ) cycle inner

  ! Dealing with edges
  ! Loop over all codimensions
  edges: do s = 1 , cgca_scodim

    ! If the neighbour is below the lower edge
    if ( neicod( s ) .lt. cgca_slcob( s ) ) then
      if ( periodicbc ) then
        ! If periodic BC are in use, take the data from the opposite
        ! edge.
        neicod( s ) = cgca_sucob( s )
      else
        ! Otherwise, do not pull data from this neighbour, move to
        ! the next one.
        cycle inner
      end if
    end if

    ! If the neighbour is above the upper edge
    if ( neicod( s ) .gt. cgca_sucob( s ) ) then
      if ( periodicbc ) then
        ! If periodic BC are in use, take the data from the opposite
        ! edge.
        neicod( s ) = cgca_slcob( s )
      else
        ! Otherwise, do not pull data from this neighbour, move to
        ! the next one.
        cycle inner
      end if
    end if

  end do edges

  ! Now the coindex set of the neighbour has been obtained.
  ! Pull its data

  ! Copy gcupd from the neighbouring image into a local var.
  ! Remote read.
  gcupd_local( : , : ) =                                                &
    gcupd( : , : ) [ neicod(1), neicod(2), neicod(3) ]

  ! Scan all values in gcupd. Can reuse loop index s.
  gcarray: do s = 1, cgca_gcupd_size1

    ! The gcupd array is filled with fractured pairs from the beginning
    ! so exit as soon as the GB state is intact.
```

```
    if ( gcupd_local( s , 3 ) .eq. cgca_gb_state_intact ) exit gcarray

    !write (*,*) "DEBUG: cgca_gcupd: img:", img, &
    !            "gcupd_local(i,:):", gcupd_local( i , : )

    ! add the pair to the gc array on this image
    call cgca_gcf( gcupd_local( s , 1 ), gcupd_local( s , 2 ) )

  end do gcarray

end do inner
end do
end do

end subroutine cgca_gcupdn
```

26.16 cgca_m3clvg/m3clvg_sm2

[cgca_m3clvg] [Submodules]

NAME

m3clvg_sm2

SYNOPSIS

```
!$Id: m3clvg_sm2.f90 380 2017-03-22 11:03:09Z mexas $

submodule ( cgca_m3clvg ) m3clvg_sm2
```

DESCRIPTION

Submodule with aux routine for checking the misorientation thereshold.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

CONTAINS

cgca_tchk (26.16.1)

USES

Variables and routines of module cgca_m3clvg (26) by host association.

USED BY

Module cgca_m3clvg (26)

SOURCE

```
implicit none

contains
```

26.16.1 m3clvg_sm2/cgca_tchk

[m3clvg_sm2] [Subroutines]

NAME

cgca_tchk

SYNOPSIS

```
module procedure cgca_tchk
```

DESCRIPTION

Generate num random unit normal vectors. Calculate the MAX of the MIN for all 26 cell neighbourhood unit vectors. Find the maximum value of all num normal vectors and return it as maxmin. And the opposite - calculate the MIN of the MAX for all 26 cell neighbourhood unit vectors. Find the min value of all num normal vectors and return as minmax.

NOTE

Any image can call this routine

SOURCE

```
real(    kind=rlrg ) :: n(3), mag, prod, prodmax, prodmin
integer( kind=ilrg ) :: i, x1, x2, x3

maxmin = 0.0_rlrg
minmax = 1.0_rlrg

do i=1,num
  prodmin = 1.0_rlrg
  prodmax = 0.0_rlrg
  call random_number(n)
  mag = sqrt( sum( n**2 ) )
  n = n / mag
  do x3 = -1, 1
  do x2 = -1, 1
  do x1 = -1, 1
    if ( x1 .eq. 0 .and. x2 .eq. 0 .and. x3 .eq. 0) cycle
    prod = abs( dot_product( e(:,x1,x2,x3), n ) )
    if ( prod .lt. prodmin ) prodmin = prod
    if ( prod .gt. prodmax ) prodmax = prod
  end do
  end do
  end do

  if ( prodmin .gt. maxmin) maxmin = prodmin
  if ( prodmax .lt. minmax) minmax = prodmax

end do

end procedure cgca_tchk
```

26.17 cgca_m3clvg/m3clvg_sm3

[cgca_m3clvg] [Submodules]

NAME

m3clvg_sm3

SYNOPSIS

```
!$Id: m3clvg_sm3.f90 393 2017-03-24 14:54:10Z mexas $

submodule ( cgca_m3clvg ) m3clvg_sm3
```

DESCRIPTION

Submodule of cgca_m3clvg (26) with collective routines. This module cannot be used (yet) with ifort 16, so don't build there.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENCE

CONTAINS

cgca_clvgp (26.17.1)

USES

Variables and parameters from the parent module cgca_m3clvg (26).

USED BY

The parent module cgca_m3clvg (26).

SOURCE

```
implicit none

contains
```

26.17.1 m3clvg_sm3/cgca_clvgp

[m3clvg_sm3] [Subroutines]

NAME

cgca_clvgp

SYNOPSIS

```
  module subroutine cgca_clvgp( coarray, rt, t, scrit, sub, gcus,      &
                                periodicbc, iter, heartbeat, debug )
    ! Inputs:
    ! coarray - cellular array
    integer( kind=iarr ), allocatable, intent(inout) ::                &
      coarray(:,:,:,:)[:,:,:]
    ! rt - rotation tensor coarray
    real( kind=rdef ), allocatable, intent(inout) :: rt(:,:,:)[:,:,:]
    ! t - stress tensor in spatial CS
    ! - scrit - critical values of cleavage stress on 100,
    !   110 and 111 planes
    real( kind=rdef ), intent(in) :: t(3,3), scrit(3)
    ! sub - name of the cleavage state calculation routine,
    !   either cgca_clvgsd, or cgca_clvgsp.
    procedure( cgca_clvgs_abstract ) :: sub
    ! gcus - name of the grain connectivity update subroutine, either
    !        cgca_gcupda - all-to-all, or cgca_gcupdn - nearest
    ! neighbour.
    procedure( gcupd_abstract ) :: gcus
    ! periodicbc - if .true. periodic boundary conditions are used,
    !   i.e. global halo exchange is called before every iteration
    logical( kind=ldef ), intent(in) :: periodicbc
    !      iter - number of cleavage iterations, if <=0 then error
    ! heartbeat - if >0 then dump a simple message every
    !             heartbeat iterations
    integer( kind=idef ), intent(in) :: iter, heartbeat
    ! debug - if .true. then will call cgca_dacf with debug
    logical( kind=ldef ), intent(in) :: debug
```

INPUTS

```
!    See the interface in the parent module cgca_m3clvg.
```

OUTPUTS

```
!    None
```

SIDE EFFECTS

Many:

- change state of coarray
- change state of gc (11.11) array

DESCRIPTION

This is a cleavage propagation routine. We copy the model (coarray) into the local array. We then analyse the local array, but update the coarray.

NOTES

All images must call this routine

USES

cgca_clvgs_abstract, cgca_clvgsd (26.5), cgca_clvgsp (26.7), cgca_clvgn (26.1), cgca_hxi (15.2), cgca_hxg (15.1), cgca_dacf (26.9)

USED BY

none, end user

SOURCE

```
real( kind=rdef ) :: n(3),                                          &
  ! tmp array to avoid copy in/out warnings
     tmparr(3,3)
integer( kind=iarr ), allocatable :: array(:,:,:)
integer( kind=iarr ) :: grold, grnew, cstate,                       &
  ! tmp arrays to avoid copy in/out warnings
     arrtmp1(3,3,3), arrtmp2(3,3,3)

integer(kind=idef) ::                                               &
  lbv(4) ,& ! lower bounds of the complete (plus virtual) coarray
  ubv(4) ,& ! upper bounds of the complete (plus virtual) coarray
  lbr(4) ,& ! lower bounds of the "real" coarray, lower virtual+1
  ubr(4) ,& ! upper bounds of the "real" coarray, upper virtual-1
  x1     ,& ! local coordinates in an array, which are also
  x2     ,& ! do loop counters
  x3     ,& !
  iteration ! iteration counter

integer :: thisimage, errstat=0, nimages
integer, save :: clvgglob[*]

logical(kind=ldef) :: clvgflag

! Make sure to allocate gcupd!
if ( .not. allocated( gcupd ) ) call gcupd_alloc

! Set the global cleavage flag initially to zero on all images,
! i.e. no cleavage
clvgglob = 0

! Set the local cleavage flag to .false.
clvgflag = .false.

! use local vars to save time
thisimage = this_image()
  nimages = num_images()
```

```
! determine the extents
lbv = lbound(coarray)
ubv = ubound(coarray)
lbr = lbv+1
ubr = ubv-1

!************************************************
! Sanity checks
!************************************************

! check for allocated
if ( .not. allocated( coarray ) ) then
  write (*,'(a,i0)') "ERROR: cgca_clvgp: image ",thisimage
  write (*,'(a)')    "ERROR: cgca_clvgp: coarray is not allocated"
  error stop
end if
if ( .not. allocated( rt ) ) then
  write (*,'(a,i0)') "ERROR: cgca_clvgp: image ",thisimage
  write (*,'(a)')    "ERROR: cgca_clvgp: rt is not allocated"
  error stop
end if

! check there are no liquid cells in the grain array
if ( any( coarray(lbr(1):ubr(1),lbr(2):ubr(2),lbr(3):ubr(3),           &
  cgca_state_type_grain) .eq. cgca_liquid_state)) then
  write (*,'(a,i0,a)') "ERROR: cgca_clvgp: image ",thisimage,          &
   ": liquid phase in the model"
  error stop
end if

if ( iter .le. 0 ) then
  write (*,'(a,i0,a)') "ERROR: cgca_clvgp: image ",thisimage,          &
   ": negative number of iterations given"
  error stop
end if

!************************************************
! End of sanity checks
!************************************************

! allocate the temp array
allocate( array(lbv(1):ubv(1),lbv(2):ubv(2),lbv(3):ubv(3) ),           &
          stat=errstat )
if ( errstat .ne. 0 ) then
  write (*,"(2(a,i0))") "ERROR: cgca_clvgp: image ", thisimage,        &
   " : cannot allocate array, errcode: ", errstat
  error stop
end if

! initialise the iteration counter
iteration = 1
```

```
! initialise the old grain to liquid
grold = cgca_liquid_state

! initial halo exchange, to make sure the coarray is in a correct state
call cgca_hxi( coarray )
if ( periodicbc ) call cgca_hxg( coarray )
sync all

! start the main loop for cleavage iterations
main: do

  ! copy coarray fracture state type into a local array
  array = coarray(:,:,:,cgca_state_type_frac)

  ! propagate cleavage
  do x3 = lbr(3),ubr(3)
  do x2 = lbr(2),ubr(2)
  do x1 = lbr(1),ubr(1)

    ! scan only through undamaged cells
    live: if ( array(x1,x2,x3) .eq. cgca_intact_state .or.             &
               array(x1,x2,x3) .eq. cgca_gb_state_intact) then

      ! what grain are we in?
      grnew = coarray( x1, x2, x3, cgca_state_type_grain )

      ! If the new grain differs from the old, then
      ! we have crossed the grain boundary, and need
      ! to calculate the cleavage plane again.
      if ( grnew .ne. grold ) then

        ! Use a tmp array to avoid compiler and runtime
        ! copy in/out warnings
        tmparr = rt( grnew, : , : )
        call cgca_clvgn( t, tmparr, scrit, clvgflag, n, cstate )

        grold = grnew
      end if

      ! debug
      !   if (debug) write (*,"(a,i0,a,l1)")      &
      !    "DEBUG: cgca_clvgp: img ", thisimage,  &
      !    " clvgflag=", clvgflag

      ! If cleavage conditions are met, propagate cleavage into
      ! this cell. Note that we pass the local array, but return
      ! the new state of the central cell into the coarray.
      ! The sub name is provided as an input to cgca_clvgp.
      ! It can be either the deterministic routine cgca_clvgsd,
      ! or the probabilistic routine cgca_clvgsp.
      if ( clvgflag ) then
```

```
      ! mark that cleavage has occurred. The value is not important,
      ! any non-zero integer will do, but the same on all images.
      clvgglob = 1

      ! Use tmp arrays explicitly to avoid compiler or runtime
      ! warnings.
      arrtmp1 = array(   x1-1:x1+1, x2-1:x2+1, x3-1:x3+1 )
      arrtmp2 = coarray( x1-1:x1+1, x2-1:x2+1, x3-1:x3+1,           &
                                    cgca_state_type_grain )

      call sub( arrtmp1, arrtmp2, n, cstate, debug,                 &
                  coarray(x1,x2,x3,cgca_state_type_frac) )
    end if
  end if live

end do
end do
end do

! Add together all cleavage identifiers from all images
! no sync is required!
call co_sum( clvgglob )

! Check if cleavage happened anywhere in the model.
if ( clvgglob .eq. 0 ) then
  if ( thisimage .eq. 1 )                                           &
    write (*,*) "INFO: cgca_clvgp: no cleavage anywhere, leaving"
  exit main
end if

sync all

! update all local gc arrays using the given subroutine
call gcus( periodicbc )

! halo exchange after a cleavage propagation step
call cgca_hxi( coarray )
if ( periodicbc ) call cgca_hxg( coarray )

! deactivate crack flanks, ignore grain boundaries
call cgca_dacf( coarray, debug=.false. )

sync all

! Reset all gcupd
gcupd = cgca_gb_state_intact

! Reset the gcupd counter
gcucnt = 1

! halo exchange after deactivation step
```

```
  call cgca_hxi( coarray )
  if ( periodicbc ) call cgca_hxg( coarray )

  sync all

  ! send heartbeat signal to terminal
  if (thisimage .eq. 1 .and. heartbeat .gt. 0) then
    if ( mod( iteration, heartbeat ) .eq. 0) write (*,'(a,i0)')        &
        "INFO: cgca_clvgp: iterations completed: ", iteration
  end if

  if ( iteration .ge. iter ) exit main

  ! increment the iteration counter
  iteration = iteration + 1

end do main

deallocate( array, stat=errstat )
if ( errstat .ne. 0 ) then
  write (*,"(2(a,i0))") "ERROR: cgca_clvgp: image ",thisimage,         &
   " : cannot deallocate array, errcode: ", errstat
  error stop
end if

! sync before leaving
sync all

end subroutine cgca_clvgp
```

26.17.2 m3clvg_sm3/cgca_clvgpt

[m3clvg_sm3] [Subroutines]

NAME

cgca_clvgpt

SYNOPSIS

```
module procedure cgca_clvgpt
```

INPUTS

```
!    See the interface in the parent module cgca_m3clvg.
```

OUTPUTS

```
!    None
```

SIDE EFFECTS

Many:

- change state of coarray
- change state of gc (11.11) array

DESCRIPTION

This is a cleavage propagation routine. We copy the model (coarray) into the local array. We then analyse the local array, but update the coarray.

NOTES

This is a *thread safe* cleavage propagation routine. This means that the order of iterations in the inner loop is not important.

All images must call this routine

USES

cgca_clvgs_abstract, cgca_clvgsd (26.5), cgca_clvgsp (26.7), cgca_clvgn (26.1), cgca_hxi (15.2), cgca_hxg (15.1), cgca_dacf (26.9)

USED BY

none, end user

SOURCE

```
real( kind=rdef ) :: n(3)
integer( kind=iarr ), allocatable :: array(:,:,:)
integer( kind=iarr ) :: grain, cstate

integer(kind=idef) ::                                                &
  lbv(4) ,& ! lower bounds of the complete (plus virtual) coarray
  ubv(4) ,& ! upper bounds of the complete (plus virtual) coarray
```

```
  lbr(4) ,& ! lower bounds of the "real" coarray, lower virtual+1
  ubr(4) ,& ! upper bounds of the "real" coarray, upper virtual-1
  x1     ,& ! local coordinates in an array, which are also
  x2     ,& ! do loop counters
  x3     ,& !
  iteration ! iteration counter

integer :: thisimage, errstat=0, nimages, ierr
integer, save :: clvgglob[*]

logical(kind=ldef) :: clvgflag

! Make sure to allocate gcupd!
if ( .not. allocated( gcupd ) ) call gcupd_alloc

! Set the global cleavage flag initially to zero on all images,
! i.e. no cleavage
clvgglob = 0

! Set the local cleavage flag to .false.
clvgflag = .false.

! use local vars to save time
thisimage = this_image()
  nimages = num_images()

! determine the extents
lbv = lbound(coarray)
ubv = ubound(coarray)
lbr = lbv+1
ubr = ubv-1

!************************************************
! Sanity checks
!************************************************

! check for allocated
if ( .not. allocated( coarray ) ) then
  write (*,'(a,i0)') "ERROR: cgca_clvgp: image ",thisimage
  write (*,'(a)')    "ERROR: cgca_clvgp: coarray is not allocated"
  error stop
end if
if ( .not. allocated( rt ) ) then
  write (*,'(a,i0)') "ERROR: cgca_clvgp: image ",thisimage
  write (*,'(a)')    "ERROR: cgca_clvgp: rt is not allocated"
  error stop
end if

! check there are no liquid cells in the grain array
if ( any( coarray(lbr(1):ubr(1),lbr(2):ubr(2),lbr(3):ubr(3),           &
  cgca_state_type_grain) .eq. cgca_liquid_state)) then
  write (*,'(a,i0,a)') "ERROR: cgca_clvgp: image ",thisimage,          &
```

```
   ": liquid phase in the model"
  error stop
end if

if ( iter .le. 0 ) then
  write (*,'(a,i0,a)') "ERROR: cgca_clvgp: image ",thisimage,          &
   ": negative number of iterations given"
  error stop
end if

!************************************************
! End of sanity checks
!************************************************

! allocate the temp array
allocate( array(lbv(1):ubv(1),lbv(2):ubv(2),lbv(3):ubv(3) ),           &
          stat=errstat )
if ( errstat .ne. 0 ) then
  write (*,"(2(a,i0))") "ERROR: cgca_clvgp: image ", thisimage,        &
   " : cannot allocate array, errcode: ", errstat
  error stop
end if

! initialise the iteration counter
iteration = 1

! initial halo exchange, to make sure the coarray is in a correct state
call cgca_hxi( coarray )
if ( periodicbc ) call cgca_hxg( coarray )
sync all

! start the main loop for cleavage iterations
main: do

  ! copy coarray fracture state type into a local array
  array = coarray(:,:,:,cgca_state_type_frac)

  ! propagate cleavage
  do concurrent( x1=lbr(1):ubr(1), x2=lbr(2):ubr(2), x3=lbr(3):ubr(3) )

    ! scan only through undamaged cells
    live: if ( array(x1,x2,x3) .eq. cgca_intact_state .or.             &
               array(x1,x2,x3) .eq. cgca_gb_state_intact) then

      grain = coarray( x1, x2, x3, cgca_state_type_grain )
      call cgca_clvgn_pure( t, grain, rt, scrit, clvgflag, n, cstate,  &
        ierr )

      ! debug
      !   if (debug) write (*,"(a,i0,a,l1)")      &
      !    "DEBUG: cgca_clvgp: img ", thisimage,  &
      !    " clvgflag=", clvgflag
```

```
    ! If cleavage conditions are met, propagate cleavage into
    ! this cell. Note that we pass the local array, but return
    ! the new state of the central cell into the coarray.
    ! The sub name is provided as an input to cgca_clvgp.
    ! It can be either the deterministic routine cgca_clvgsd,
    ! or the probabilistic routine cgca_clvgsp.
    if ( clvgflag ) then

      ! Mark that cleavage has occurred. The value is not important,
      ! any non-zero integer will do, but the same on all images.
      clvgglob = 1
      call sub(   array(x1-1:x1+1, x2-1:x2+1, x3-1:x3+1),            &
                coarray(x1-1:x1+1, x2-1:x2+1, x3-1:x3+1,             &
                                    cgca_state_type_grain),          &
               n, cstate, debug,                                     &
                  coarray(x1,x2,x3,cgca_state_type_frac) )
    end if
  end if live

end do

if ( ierr .ne. 0 ) then
  write (*,'(a,i0)') "ERROR: cgca_clvgp_t/m3clvg_sm3:" //            &
    " cgca_clvgn_pure error, ierr:", ierr
  error stop
end if

! Add together all cleavage identifiers from all images
! no sync is required!
call co_sum( clvgglob )

! Check if cleavage happened anywhere in the model.
if ( clvgglob .eq. 0 ) then
  if ( thisimage .eq. 1 )                                            &
    write (*,*) "INFO: cgca_clvgp: no cleavage anywhere, leaving"
  exit main
end if

sync all

! update all local gc arrays using the given subroutine
call gcus( periodicbc )

! halo exchange after a cleavage propagation step
call cgca_hxi( coarray )
if ( periodicbc ) call cgca_hxg( coarray )

! deactivate crack flanks, ignore grain boundaries
call cgca_dacf( coarray, debug=.false. )

sync all
```

```
  ! Reset all gcupd
  gcupd = cgca_gb_state_intact

  ! Reset the gcupd counter
  gcucnt = 1

  ! halo exchange after deactivation step
  call cgca_hxi( coarray )
  if ( periodicbc ) call cgca_hxg( coarray )

  sync all

  ! send heartbeat signal to terminal
  if (thisimage .eq. 1 .and. heartbeat .gt. 0) then
    if ( mod( iteration, heartbeat ) .eq. 0) write (*,'(a,i0)')        &
        "INFO: cgca_clvgp: iterations completed: ", iteration
  end if

  if ( iteration .ge. iter ) exit main

  ! increment the iteration counter
  iteration = iteration + 1

end do main

deallocate( array, stat=errstat )
if ( errstat .ne. 0 ) then
  write (*,"(2(a,i0))") "ERROR: cgca_clvgp: image ",thisimage,         &
   " : cannot deallocate array, errcode: ", errstat
  error stop
end if

! sync before leaving
sync all

end procedure cgca_clvgpt
```

26.18 cgca_m3clvg/m3clvgt_sm1

[cgca_m3clvg] [Submodules]

NAME

m3clvgt_sm1

SYNOPSIS

```
!$Id: m3clvgt_sm1.f90 380 2017-03-22 11:03:09Z mexas $

submodule ( cgca_m3clvgt ) m3clvgt_sm1
```

DESCRIPTION

Submodule of cgca_m3clvg (26) with collective routines. This module cannot be used (yet) with ifort 16, so don't build there.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENCE

CONTAINS

cgca_clvgp (26.17.1)

USES

Variables and parameters from the parent module cgca_m3clvg (26).

USED BY

The parent module cgca_m3clvg (26).

SOURCE

```
implicit none

contains
```

27 CGPACK/cgca_m3clvgt

[Modules]

NAME

cgca_m3clvgt

SYNOPSIS

```
!$Id: cgca_m3clvgt.f90 380 2017-03-22 11:03:09Z mexas $

module cgca_m3clvgt
```

DESCRIPTION

Module containing only *thread safe* cleavage routines. In particular, all routines are PURE, designed to be usable from DO CONCURRENT. The module contains variables, submodules and subroutines.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

CONTAINS

cgca_clvgpt (26.17.2) (in submodule m3clvgt_sm1 (26.18)).

USES

cgca_m1co (9), cgca_m2gb (11), cgca_m2hx (15), cgca_m2rot (24), cgca_m2rnd (23), cgca_m2glm (13)

USED BY

cgca

SOURCE

```
use cgca_m1co
use cgca_m2gb,  only: cgca_gcr_pure, cgca_gcf_pure
use cgca_m2glm, only: cgca_ico
use cgca_m2hx,  only: cgca_hxi, cgca_hxg
use cgca_m2rot, only: cgca_csym_pure
use cgca_m2rnd, only: cgca_irs

implicit none

private
public :: cgca_clvgpt

! This array is used to update local gc arrays.
! The components are as follows:
!
!  gcupd(:,1) - grain
!  gcupd(:,2) - neighbour
!  gcupd(:,3) - state, either cgca_gb_state_intact    or
```

```
!                             cgca_gb_state_fractured
!
! The idea is that this array is updated every time a grain
! boundary is crossed. Then all local arrays are updated using
! cgca_gcf and this coarray.

integer( kind=iarr ), allocatable :: gcupd(:,:) [:,:,:]

! Counter of a gcupd pair. Set to 1 initially.
integer( kind=idef ) :: gcucnt=1

real( kind=rdef ), parameter ::    &
     zero = 0.0_rdef,              &
      one = 1.0_rdef,              &
    sqrt2 = sqrt(2.0_rdef),        &
 onesqrt2 = one/sqrt2,             &
    sqrt3 = sqrt(3.0_rdef),        &
 onesqrt3 = one/sqrt3,             &

 ! 27 unit vectors connecting the central cell with all its
 ! 26 neighbours + itself, which is a zero vector.
 e(3, -1:1, -1:1, -1:1) =          &
   reshape( (/                     &
 -onesqrt3,-onesqrt3,-onesqrt3,    & ! -1 -1 -1
  zero,    -onesqrt2,-onesqrt2,    & !  0 -1 -1
  onesqrt3,-onesqrt3,-onesqrt3,    & !  1 -1 -1

 -onesqrt2, zero,    -onesqrt2,    & ! -1  0 -1
  zero,     zero,    -one,         & !  0  0 -1
  onesqrt2, zero,    -onesqrt2,    & !  1  0 -1

 -onesqrt3, onesqrt3,-onesqrt3,    & ! -1  1 -1
  zero,     onesqrt2,-onesqrt2,    & !  0  1 -1
  onesqrt3, onesqrt3,-onesqrt3,    & !  1  1 -1

 -onesqrt2,-onesqrt2, zero,        & ! -1 -1  0
  zero,    -one,      zero,        & !  0 -1  0
  onesqrt2,-onesqrt2, zero,        & !  1 -1  0

 -one,      zero,     zero,        & ! -1  0  0
  zero,     zero,     zero,        & !  0  0  0
  one,      zero,     zero,        & !  1  0  0

 -onesqrt2, onesqrt2, zero,        & ! -1  1  0
  zero,     one,      zero,        & !  0  1  0
  onesqrt2, onesqrt2, zero,        & !  1  1  0

 -onesqrt3,-onesqrt3, onesqrt3,    & ! -1 -1  1
  zero,    -onesqrt2, onesqrt2,    & !  0 -1  1
  onesqrt3,-onesqrt3, onesqrt3,    & !  1 -1  1

 -onesqrt2, zero,     onesqrt2,    & ! -1  0  1
```

```
  zero,     zero,     one,         & !  0  0  1
  onesqrt2, zero,     onesqrt2,    & !  1  0  1

 -onesqrt3, onesqrt3, onesqrt3,    & ! -1  1  1
  zero,     onesqrt2, onesqrt2,    & !  0  1  1
  onesqrt3, onesqrt3, onesqrt3     & !  1  1  1
  /), (/ 3,3,3,3 /) )


! Abstract interface is a fortran 2003 feature.
! This interface is for *thread safe* cleavage "change state" routines.
! The interface for such routines - cgca_clvgsdt and
! cgca_clvgspt, must match it.
! Using this interface, the cleavage "change state"
! routines can be passed as actual arguments to the cleavage
! propagation routines, e.g. cgca_clvgpt, where
! the dummy arguments for these routines are defined by
!  procedure(cgca_clvgst_abstract)

abstract interface
 subroutine cgca_clvgst_abstract( farr, marr, n, cstate, debug,        &
   newstate )
  use cgca_m1co
  integer, parameter :: l=-1, centre=l+1, u=centre+1
  integer( kind=iarr ), intent(in) :: farr(l:u,l:u,l:u),               &
    marr(l:u,l:u,l:u), cstate
  real( kind=rdef ), intent(in) :: n(3)
  logical( kind=ldef ), intent(in) :: debug
  integer( kind=iarr ), intent(out) :: newstate
 end subroutine cgca_clvgst_abstract

 subroutine gcupd_abstract( periodicbc )
  import :: ldef
  logical( kind=ldef ), intent( in ) :: periodicbc
 end subroutine
end interface

! Interfaces for submodule procedures.

interface

  ! In submodule m3clvgt_sm1
  module subroutine cgca_clvgpt( coarray, rt, t, scrit, sub, gcus,    &
      periodicbc, iter, heartbeat, debug )
    ! Inputs:
    ! coarray - cellular array
      integer( kind=iarr ), allocatable, intent(inout) ::              &
        coarray(:,:,:,:)[:,:,:]
    ! rt - rotation tensor coarray
      real( kind=rdef ), allocatable, intent(inout) :: rt(:,:,:)[:,:,:]
    ! t - stress tensor in spatial CS
    ! - scrit - critical values of cleavage stress on 100,
```

```
    !   110 and 111 planes
    ! Clearly t and scrit must be in the same units!
      real( kind=rdef ), intent(in) :: t(3,3), scrit(3)
    ! sub - name of the cleavage state calculation routine,
    !   either cgca_clvgsd, or cgca_clvgsp.
      procedure( cgca_clvgst_abstract ) :: sub
    ! gcus - name of the grain connectivity update subroutine, either
    !        cgca_gcupda - all-to-all, or cgca_gcupdn - nearest
    ! neighbour.
      procedure( gcupd_abstract ) :: gcus
    ! periodicbc - if .true. periodic boundary conditions are used,
    !   i.e. global halo exchange is called before every iteration
      logical( kind=ldef ), intent(in) :: periodicbc
    !      iter - number of cleavage iterations, if <=0 then error
    ! heartbeat - if >0 then dump a simple message every
    !             heartbeat iterations
      integer( kind=idef ), intent(in) :: iter, heartbeat
    ! debug - if .true. then will call cgca_dacf with debug
      logical( kind=ldef ), intent(in) :: debug
  end subroutine cgca_clvgpt
end interface

contains
```

28 CGPACK/cgca_m3gbf

[Modules]

NAME

cgca_m3gbf

SYNOPSIS

```
!$Id: cgca_m3gbf.f90 529 2018-03-26 11:25:45Z mexas $

module cgca_m3gbf
```

DESCRIPTION

Module dealing with grain boundary fractures.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

CONTAINS

cgca_gbf1p (28.2), cgca_gbf1f (28.1)

USES

cgca_m1co (9), cgca_m2glm (13)

USED BY SOURCE

```
use cgca_m1co
use cgca_m2glm
implicit none

private
public :: cgca_gbf1p, cgca_gbf1f

contains
```

28.1 cgca_m3gbf/cgca_gbf1f

[cgca_m3gbf] [Subroutines]

NAME

cgca_gbf1f

SYNOPSIS

```
subroutine cgca_gbf1f( coarray )
```

INPUTS

```
integer( kind=iarr ), allocatable, intent(inout) ::                    &
  coarray(:,:,:,:)[:,:,:]
```

OUTPUTS

```
!    coarray, as it's intent(INOUT)
```

SIDE EFFECTS

None

DESCRIPTION

This routine does a single iteration of grain boundary fracture propagation assuming fixed boundary conditions.

NOTE

Use only with fixed BC. For periodic BC use cgca_gbf1p (28.2).

USES

cgca_lg (13.4)

SOURCE

```
integer( kind=iarr ), allocatable, save :: array( :, :, : )
integer :: range1(3), range2(3), range3(3)

integer(kind=idef) :: &
  lbv(4),    & ! lower bounds of the complete (plus virtual) coarray
  ubv(4),    & ! upper bounds of the complete (plus virtual) coarray
  lbr(4),    & ! lower bounds of the "real" coarray, lower virtual+1
  ubr(4),    & ! upper bounds of the "real" coarray, upper virtual-1
  x1,x2,x3,  & ! local coordinates in an array
  super(3),  & ! global (super) coordinates of a cell
  imgpos(3), & ! image position in the image grid
  local(3),  & ! local coordinates of a cell
  ubsuper(3),& ! upper bounds of the super array
  frnei,     & ! number of fractured neighbours
  i            ! loop counter

integer :: thisimage, errstat=0, nimages
```

```
! Do not check coarray for allocated, as this wastes time.
! Instead let the code fail if coarray is not allocated.

! use local vars to save time
thisimage = this_image()
  nimages = num_images()
   imgpos = this_image( coarray )

! determine the extents
lbv = lbound( coarray )
ubv = ubound( coarray )
lbr = lbv + 1
ubr = ubv - 1
ubsuper = (ubr(1:3) - lbr(1:3) + 1) * nimages

! Allocate the temp array if not already allocated.
! The array has the SAVE attribute, as this routine
! is likely to be called many times.

if ( .not. allocated(array) ) then
  allocate( array( lbv(1):ubv(1) , lbv(2):ubv(2) , lbv(3):ubv(3) ),    &
            stat=errstat )
  if ( errstat .ne. 0 ) then
    write (*,'(2(a,i0))') "ERROR: cgca_gbf1f/cgca_m3gbf: image: ",     &
      thisimage, "allocate( array ), stat: ", errstat
    error stop
  end if
end if

! Copy coarray fracture state type into a local array
array = coarray(:,:,:,cgca_state_type_frac)

! scan across all cells
do x3=lbr(3),ubr(3)
do x2=lbr(2),ubr(2)
do x1=lbr(1),ubr(1)

 ! Analyse only live cells
 if ( array(x1,x2,x3) .ne. cgca_intact_state ) cycle

 ! Skip cells adjacent to halo cells
 local = (/ x1, x2, x3 /)
 call cgca_lg(imgpos,local,coarray,super)
 if ( any( super .eq. 1) .or. any( super .eq. ubsuper) ) cycle

 ! set up ranges to save compute time
 range1 = (/ x1-1, x1, x1+1 /)
 range2 = (/ x2-1, x2, x2+1 /)
 range3 = (/ x3-1, x3, x3+1 /)

 ! count fractured neighbours, only cleavage edges and fractured GB
```

```
 frnei = 0
 ! first all cleavage edge states
 do i=1,size(cgca_clvg_states_edge)
  frnei = frnei + count(                                              &
   array(range1,range2,range3) .eq. cgca_clvg_states_edge(i) )
 end do
 ! then all GB fractured states
 frnei = frnei + count(                                               &
  array(range1,range2,range3) .eq. cgca_gb_state_fractured )

 ! If the cell
 !  (1) is on the grain boundary
 !  (2) has 5 fractured neighbours
 ! then it becomes cgca_gb_state_fractured.
 if ( any( coarray(range1,range2,range3,cgca_state_type_grain) .ne. &
           coarray(x1,x2,x3,cgca_state_type_grain) ) .and.          &
      (frnei .ge. 5) ) then
  coarray(x1,x2,x3,cgca_state_type_frac) = cgca_gb_state_fractured
 end if

end do
end do
end do

end subroutine cgca_gbf1f
```

28.2 cgca_m3gbf/cgca_gbf1p

[cgca_m3gbf] [Subroutines]

NAME

cgca_gbf1p

SYNOPSIS

```
subroutine cgca_gbf1p(coarray)
```

INPUTS

```
integer(kind=iarr),allocatable,intent(inout) :: coarray(:,:,:,:)[:,:,:]
```

SIDE EFFECTS

state of coarray changes

DESCRIPTION

This routine does a single iteration of grain boundary fracture propagation assuming periodic boundary conditions.

NOTE

Use only with periodic BC. For fixed BC use cgca_gbf1f (28.1).

SOURCE

```
integer(kind=iarr),allocatable,save :: array(:,:,:)

integer(kind=idef) :: &
  range1(3), range2(3), range3(3), &
  lbv(4) ,& ! lower bounds of the complete (plus virtual) coarray
  ubv(4) ,& ! upper bounds of the complete (plus virtual) coarray
  lbr(4) ,& ! lower bounds of the "real" coarray, lower virtual+1
  ubr(4) ,& ! upper bounds of the "real" coarray, upper virtual-1
  x1     ,& ! local coordinates in an array, which are also
  x2     ,& ! do loop counters
  x3

integer :: thisimage, errstat=0, nimages

! Do not check coarray for allocated, as this wastes time.
! Instead let the code fail if coarray is not allocated.

! use local vars to save time
thisimage = this_image()
nimages = num_images()

! determine the extents
lbv = lbound( coarray )
ubv = ubound( coarray )
lbr = lbv + 1
```

```
ubr = ubv - 1

! Allocate the temp array if not already allocated.
! The array has the SAVE attribute, as this routine
! is likely to be called many times.

! Initialise errstat to -1
errstat = -1

if (.not. allocated(array)) &
 allocate(array(lbv(1):ubv(1),lbv(2):ubv(2),lbv(3):ubv(3)),stat=errstat)
if (errstat.ne.0) then
 write (*,'(a,i0)') "ERROR: cgca_gbf1p: image ",thisimage
 write (*,'(a)') "ERROR: cgca_gbf1p: cannot allocate array"
 error stop
end if

! Copy coarray fracture state type into a local array
array = coarray(:,:,:,cgca_state_type_frac)

! scan across all cells
do x3=lbr(3),ubr(3)
do x2=lbr(2),ubr(2)
do x1=lbr(1),ubr(1)

 ! Analyse only live cells
 if ( array(x1,x2,x3) .ne. cgca_intact_state ) cycle

 ! set up ranges to save compute time
 range1 = (/ x1-1, x1, x1+1 /)
 range2 = (/ x2-1, x2, x2+1 /)
 range3 = (/ x3-1, x3, x3+1 /)

 ! If the cell
 !  (1) is on the grain boundary
 !  (2) has a fractured neighbour
 ! then it becomes cgca_gb_state_fractured.
 if ( any( coarray(range1,range2,range3,cgca_state_type_grain) .ne. &
           coarray(x1,x2,x3,cgca_state_type_grain) ) .and.          &
      any( array(range1,range2,range3) .ne. cgca_intact_state ) )   &
 then
  coarray(x1,x2,x3,cgca_state_type_frac) = cgca_gb_state_fractured
 end if

end do
end do
end do

end subroutine cgca_gbf1p
```

29 CGPACK/cgca_m3nucl

[Modules]

NAME

cgca_m3nucl

SYNOPSIS

```
!$Id: cgca_m3nucl.f90 380 2017-03-22 11:03:09Z mexas $

module cgca_m3nucl
```

DESCRIPTION

Module dealing with nucleation

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

CONTAINS

cgca_nr (29.1)

USES

cgca_m2glm (13)

USED BY

cgca

SOURCE

```
use cgca_m1co
use cgca_m2glm

implicit none

private
public :: cgca_nr

contains
```

29.1 cgca_m3nucl/cgca_nr

[cgca_m3nucl] [Subroutines]

NAME

cgca_nr

SYNOPSIS

```
subroutine cgca_nr( coarray , number , debug )
```

INPUTS

```
integer( kind=iarr ), allocatable, intent(inout) ::                    &
 coarray(:,:,:,:)[:,:,:]
integer( kind=idef ), intent( in ) :: number
logical( kind=ldef ), intent( in ) :: debug
```

SIDE EFFECTS

State of coarray changed

DESCRIPTION

This routine randomly scatters the given number of grain nuclei over the model. The grain nuclei are assigned unique numbers starting from zero.

All elements of the coarray must be in cgca_liquid_state (9.18) state. If not, the program will stop with a error.

The number of nuclei must not greater than "critfract" of the model size. I arbitrarily set this to 0.1 for now. However, even this is too high. Although, in principle each cell can be a nuclei, such model would have no physical sense.

Inputs:

- coarray - the model
- number - number of nuclei to scatter
- debug - if .true. dump some debug output

NOTES

All images must call this routine! However, the work will be done only by image 1. There's "sync all" at the end of this routine.

USES

cgca_gl (13.1)

USED BY

none, end user

SOURCE

```
real( kind=rdef), parameter :: critfract=0.1
```

```
integer( kind=idef ) :: &
  nuc    ,& ! running total of nuclei
  lbr(4) ,& ! lower bounds of the "real" coarray, lower virtual+1
  ubr(4) ,& ! upper bounds of the "real" coarray, upper virtual-1
  szr(3) ,& ! size or the "real" coarray, ubr-lbr+1
  lcob(3),& ! lower cobounds of the coarray
  ucob(3),& ! upper cobounds of the coarray
  supermax(3) ,& ! upper bound of the super array, szr*(ucob-lcob+1)
  supermin(3) ,& ! lower bound of the super array, always 1.
  super(3)    ,& ! coordinates in a super array
  imgpos(3)   ,& ! image position in a grid
  local(3)    ,& ! coordinates within an image
  thisimage   ,& ! this_image()
  nimages        ! num_images()

integer( kind=ilrg ) :: coarsize

real( kind=rdef ) :: candidate(3), frac
logical( kind=ldef ) :: image1

  nimages = num_images()
thisimage = this_image()
   image1 = .false.
if ( thisimage .eq. 1 ) image1 = .true.

!*********************************************************************72
! checks
!*********************************************************************72

if ( .not. allocated( coarray ) ) then
 write( *, '(a,i0)' ) "ERROR: cgca_nr/cgca_m3nucl: coarray is not" //  &
   " allocated, img: ", thisimage
 error stop
end if

! check that there are only liquid cells in coarray.

lbr = lbound( coarray ) + 1
ubr = ubound( coarray ) - 1

if ( any( coarray(lbr(1):ubr(1),lbr(2):ubr(2),lbr(3):ubr(3),           &
              cgca_state_type_grain) .ne. cgca_liquid_state)) then
 write( *, '(a,i0)' ) "ERROR: cgca_nr/cgca_m3nucl: non-liquid" //      &
   " elements in coarray, img: ", thisimage
 error stop
end if

! check that the number of nuclei is positive

if ( number .lt. 1 ) then
 write (*,'(a,i0)') "ERROR: cgca_nr/cgca_m3nucl: number of nuclei" //  &
   " must be 1 or more, img:", thisimage
```

```
 error stop
end if

!*********************************************************************72
! end of checks
!*********************************************************************72

! image 1 must not change values in other images before
! all images pass checks
sync all

img1: if ( image1 ) then

 ! Warn the user if there are too many nuclei
 coarsize = size( coarray( lbr(1) : ubr(1) , lbr(2) : ubr(2) ,         &
                           lbr(3) : ubr(3), cgca_state_type_grain ) ,  &
                  kind=ilrg )

 ! number of grains as a fraction of the model
 frac = number / ( real( nimages ) * real( coarsize ) )

 if ( frac .gt. critfract ) then
   write (*,'(a,g10.3)') "WARN: cgca_nr/cgca_m3nucl: too many " //     &
    "nuclei - no physical sense! nuclei/model size: ", frac
 end if

 ! The 4th dimension is the number of cell state types.
 ! It is not relevant here, so don't use it.
      szr = ubr(1:3)-lbr(1:3)+1
     lcob = lcobound(coarray)
     ucob = ucobound(coarray)
 supermax = szr * (ucob-lcob+1)
 supermin = 1

 nuc=1
 do
  call random_number(candidate)    ! 0 .le. candidate .lt. 1
  super=int(candidate*supermax)+1  ! 1 .le. super .le. supermax

  ! now translate to the image and local coordinates
  call cgca_gl( super, coarray, imgpos, local )

  ! If a cell is liquid then assign the running "nuc" number to it.
  ncln: if ( coarray( local(1), local(2), local(3),                    &
             cgca_state_type_grain ) [imgpos(1), imgpos(2), imgpos(3)] &
                .eq. cgca_liquid_state ) then
          coarray( local(1), local(2), local(3),                       &
          cgca_state_type_grain ) [imgpos(1),imgpos(2),imgpos(3)] = nuc

    ! If requested, dump some debug output
    if ( debug ) then
      write( *, "(2(a,3(i0,tr1)),a,i0)" ) "DEBUG:" //                  &
```

```
        " cgca_nr/cgca_m3nucl: local: ", local, "imgpos: ", imgpos,    &
        " nucleus: ", nuc
    end if

    ! Increment the running total of the nuclei generated
    nuc = nuc + 1
  end if ncln

  ! If "number" of nuclei have been generated, exit
  if ( nuc .gt. number ) exit

 end do

end if img1

! Global sync is required here
sync all

end subroutine cgca_nr
```

30 CGPACK/cgca_m3pfem

[Modules]

NAME

cgca_m3pfem

SYNOPSIS

```
!$Id: cgca_m3pfem.f90 380 2017-03-22 11:03:09Z mexas $

module cgca_m3pfem
```

DESCRIPTION

Module dealing with interfacing CGPACK with ParaFEM.

AUTHOR

Anton Shterenlikht, Luis Cebamanos

COPYRIGHT

See LICENSE (33)

CONTAINS

Public coarray variables of derived types: cgca_pfem_centroid_tmp (30.5), cgca_pfem_integrity (30.14), cgca_pfem_stress (30.20).

Public *local*, non-coarray, variable: cgca_pfem_enew (30.10).

Private *local*, non-coarray, variables: lcentr (30.23).

Public routines: cgca_pfem_boxin (30.1), cgca_pfem_cellin (30.2), cgca_pfem_cenc (30.3), cgca_pfem_cendmp (30.4), cgca_pfem_ctalloc (30.6), cgca_pfem_ctdalloc (30.7), cgca_pfem_ealloc (30.8), cgca_pfem_edalloc (30.9), cgca_pfem_intcalc1 (30.11), cgca_pfem_integalloc (30.12), cgca_pfem_integdalloc (30.13), cgca_pfem_lcentr_dump (30.24.1) (in submodule m3pfem_sm1 (30.24)), cgca_pfem_map (30.24.2) (in submodule m3pfem_sm1 (30.24)), cgca_pfem_partin (30.15), cgca_pfem_salloc (30.16), cgca_pfem_sdalloc (30.17), cgca_pfem_sdmp (30.18), cgca_pfem_simg (30.19), cgca_pfem_uym (30.21), cgca_pfem_wholein (30.22)

USES

Modules cgca_m1co (9), cgca_m2lnklst (16), cgca_m2geom (12)

USED BY

end user?

SOURCE

```
use :: cgca_m1co
use :: cgca_m2lnklst, only : cgca_lnklst_tpayld, cgca_lnklst_node,     &
  cgca_inithead, cgca_addhead, cgca_lstdmp, cgca_rmhead
use :: cgca_m2geom, only : cgca_boxsplit

implicit none

private
public ::                                                              &
```

```
! routines
  cgca_pfem_boxin, cgca_pfem_cellin, &
  cgca_pfem_cenc, cgca_pfem_cendmp, cgca_pfem_ctalloc,                &
  cgca_pfem_ctdalloc, cgca_pfem_ealloc, cgca_pfem_edalloc,            &
  cgca_pfem_integalloc, cgca_pfem_integdalloc, cgca_pfem_intcalc1,    &
  cgca_pfem_lcentr_dump, & ! in submodule m3pfem_sm1
  cgca_pfem_map,         & ! in submodule m3pfem_sm1
  cgca_pfem_partin, &
  cgca_pfem_salloc,  cgca_pfem_sdalloc, cgca_pfem_sdmp, cgca_pfem_simg,&
  cgca_pfem_uym, cgca_pfem_wholein,                          &
! variables
  cgca_pfem_centroid_tmp, cgca_pfem_enew, cgca_pfem_integrity,        &
  cgca_pfem_stress

! corresponds to typical double precision real.
integer, parameter :: cgca_pfem_iwp = selected_real_kind(15,300)

interface

  module subroutine cgca_pfem_lcentr_dump
  end subroutine cgca_pfem_lcentr_dump

  module subroutine cgca_pfem_map( origin, rot, bcol, bcou )
    real( kind=rdef ), intent( in ) ::                                &
      origin(3),        & ! origin of the "box" cs, in FE cs
      rot(3,3),         & ! rotation tensor *from* FE cs *to* CA cs
      bcol(3),          & ! lower phys. coords of the coarray on image
      bcou(3)             ! upper phys. coords of the coarray on image
  end subroutine cgca_pfem_map

end interface
```

30.1 cgca_m3pfem/cgca_pfem_boxin

[cgca_m3pfem] [Subroutines]

NAME

cgca_pfem_boxin

SYNOPSIS

```
subroutine cgca_pfem_boxin( lowr, uppr, lres, bcol, charlen, debug,    &
  iflag )
```

INPUTS

```
!      lowr(3) - integer, local coordinates of the lower corner cell
!                in the space coarray on *this* image.
!      uppr(3) - integer, local coordinates of the upper corner cell
!                in the space coarray on *this* image.
!         lres - real, linear resolution of the model, how many cells
!                per linear physical unit of length.
!      bcol(3) - real, coordinates of the coarray box on this image
!                in physical units, in CA coord. system.
!      charlen - real, characteristic length of an FE in the model.
!                This parameter is used to determine whether a cell
!                is "close enough" to a centroid of an FE.
!        debug - logical. If .true. will dump some dubug output

integer( kind=idef ), intent( in ) :: lowr(3), uppr(3)
real( kind=rdef ), intent( in ) :: lres, bcol(3), charlen
logical( kind=ldef ), intent( in ) :: debug
```

OUTPUT

```
!    iflag - integer, 1 if all 8 corner cells of the box are inside
!            FE, 2 if all 8 corner cells of the box are outside of FE,
!            3 otherwise.

integer( kind=idef ), intent( out ) :: iflag
```

SIDE EFFECTS

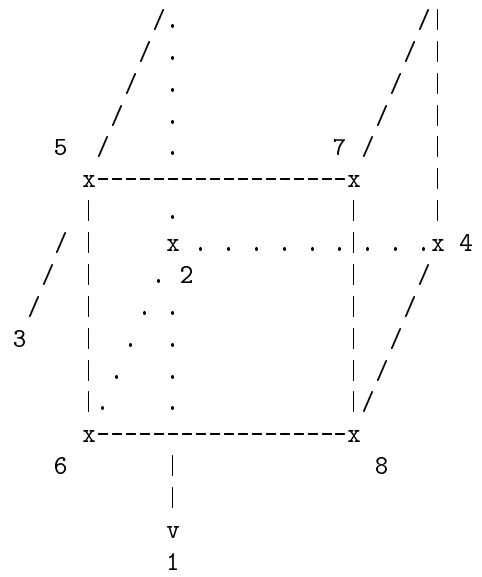
If debug is .true. dumps some output to stdout. Othewise none.

DESCRIPTION

This routine calculates whether 8 corner cells of the given box are inside FE or not. There are 3 possibilities: (1) all 8 cells are inside FE. In this case iflag is set to 1. (2) all 8 cells are outside FE. In this case iflag is set to 2. (3) some cells are inside and others are outside. In this case iflag is set to 3. iflag will be used by a calling routine to decide what to do next.

The cells in a box are numbered according the Fortran convention, the leftmost index changes first:

```
                1                 3
                x-----------------x   --> 2
```



This cell numbering convention is used for very small boxes, i.e. when the box size is 1 along some direction. So if the box size is 1 along 1, then following cells coincide: 2 as 1, 4 as 3, 6 as 5 and 8 as 7. If the box size is 1 along 2, then following cells coincide: 3 as 1, 4 as 2, 7 as 5 and 8 as 6. If the box size is 1 along 3, then following cells coincide: 5 as 1, 6 as 2, 7 as 3 and 8 as 4. This logic is used in the code. Logical array same(3) is used to show which box size is 1. same(i) is .true. if the box is of size 1 cell along dimension i. The box size is 1 when the lower cood. matches the upper coord. So same is calculated simly as: same = lowr .eq. uppr.

USES

cgca_pfem_cellin (30.2)

SOURCE

```
integer( kind=idef ) :: i, local(8,3), img
logical( kind=ldef ) :: same(3), flagarr(8)

img = this_image()

! From given corner cells, calculate 8 corner local coordinates
local( 1, : ) = (/ lowr(1), lowr(2), lowr(3) /)
local( 2, : ) = (/ uppr(1), lowr(2), lowr(3) /)
local( 3, : ) = (/ lowr(1), uppr(2), lowr(3) /)
local( 4, : ) = (/ uppr(1), uppr(2), lowr(3) /)
local( 5, : ) = (/ lowr(1), lowr(2), uppr(3) /)
local( 6, : ) = (/ uppr(1), lowr(2), uppr(3) /)
local( 7, : ) = (/ lowr(1), uppr(2), uppr(3) /)
local( 8, : ) = (/ uppr(1), uppr(2), uppr(3) /)

! Take care of repeated cells, i.e. when one or more box dimensions
! is 1. same(i) is .true. if the box lower and upper coord. are
! the same along dimension i.
same = lowr .eq. uppr

! Call cgca_pfem_cellin for each corner cell.
```

```
main: do i = 1, 8

  ! Cell 1 will always be evaluated, i.e. will call cgca_pfem_cellin.
  ! All other cells will be evaluated only if they are unique.

  ! Check direction 1
  dir1: if ( same(1) ) then
    ! Box is single cell long along 1. This means the following
    ! cells have the same flag: 2 as 1, 4 as 3, 6 as 5, 8 as 7.
    if ( ( i .eq. 2 ) .or. ( i .eq. 4 ) .or.                          &
         ( i .eq. 6 ) .or. ( i .eq. 8 ) ) then
      flagarr(i) = flagarr(i-1)
      cycle main
    end if
  end if dir1

  ! Check direction 2
  dir2: if ( same(2) ) then
    ! Box is single cell long along 2. This means the following
    ! cells have the same flag: 3 as 1, 4 as 2, 7 as 5, 8 as 6.
    if ( ( i .eq. 3 ) .or. ( i .eq. 4 ) .or.                          &
         ( i .eq. 7 ) .or. ( i .eq. 8 ) ) then
      flagarr(i) = flagarr(i-2)
      cycle main
    end if
  end if dir2

  ! Check direction 3
  dir3: if ( same(3) ) then
    ! Box is single cell long along 3. This means the following
    ! cells have the same flag: 5 as 1, 6 as 2, 7 as 3, 8 as 4.
    if ( ( i .eq. 5 ) .or. ( i .eq. 6 ) .or.                          &
         ( i .eq. 7 ) .or. ( i .eq. 8 ) ) then
      flagarr(i) = flagarr(i-4)
      cycle main
    end if
  end if dir3

  ! subroutine cgca_pfem_cellin( lc, lres, bcol, charlen, debug, flag )
  ! flag .eq. .true. if inside FE
  ! flag .eq. .false. if outside FE
  call cgca_pfem_cellin( i, local, lres, bcol, charlen, debug,    &
    flagarr(i) )

  if ( debug ) write (*,"(4(a,i0),a,l1)")                             &
    "DEBUG: cgca_pfem_boxin: img: ", img, " local cell coord (",      &
     local( i, 1 ), ",", local( i, 2 ), ",", local( i, 3 ),           &
     ") flag: ", flagarr(i)

end do main

! If all flags are .true. then the box is inside, set iflag to 1
```

```
! If all flags are .false. then the box is outside, set iflag to 2
! Otherwise, part of the box is in and part is out, set iflag to 3
if ( all( flagarr ) ) then
  iflag = 1
elseif ( all( .not. flagarr ) ) then
  iflag = 2
else
  iflag = 3
end if

end subroutine cgca_pfem_boxin
```

30.2 cgca_m3pfem/cgca_pfem_cellin

[cgca_m3pfem] [Subroutines]

NAME

cgca_pfem_cellin

SYNOPSIS

```
subroutine cgca_pfem_cellin( index, lc, lres, bcol, charlen, debug, flag )
```

INPUTS

```
!     index - represents each corner cell
!     lc(3) - integer, local coordinates of a cell in the space
!    coarray on *this* image.
!     lres - real, linear resolution of the model, how many cells per
!    linear physical unit of length.
!     bcol(3) - real, coordinates of the coarray box on this image
!    in physical units, in CA coord. system.
!     charlen - real, characteristic length of an FE in the model.
!    This parameter is used to determine whether a cell is "close
!    enough" to a centroid of an FE.
!     debug - logical. If .true. will dump some debug info

integer( kind=idef ), intent( in ) :: index
integer( kind=idef ), intent( in ) :: lc(index,3)
real( kind=rdef ), intent( in ) :: lres, bcol(3), charlen
logical( kind=ldef ), intent( in ) :: debug
```

OUTPUTS

```
!     flag - logical, .true. if the cell in "inside" the FE model,
!    .false. otherwise

logical( kind=ldef ), intent( out ) :: flag
```

SIDE EFFECTS

if debug is .true. dumps some output to stdout. Othewise none.

DESCRIPTION

Scan all FE in lcentr (30.23) array. If the coordinates of the cell in FE coord. system are close enough to the centroid of at least one element, then the cell is "inside" the FE model. Otherwise it is outside.

SOURCE

```
real( kind=rdef ) :: cacoord(3), cl2
integer( kind=idef ) :: i

! Assume the cell is outside by default. Only if it is proven to be in,
! change the flag to .true.
flag = .false.
```

```
! I need characteristic length squared for comparison
cl2 = charlen * charlen

! Calculate the global CA coord. of the given cell from the
! input local coord.
! lc / lres - distance in phys. units from the box lower corner
! lc / lres + bcol - coord. in phys units in CA CS
cacoord = lc(index,:) / lres + bcol
!cacoord = lc / lres + bcol

if ( debug ) write (*,"(a,i0,a,3(es9.2,a))")                          &
  "DEBUG: cgca_pfem_cellin: img: ", this_image(),                     &
  " cacoord (",  cacoord(1), ",", cacoord(2), ",", cacoord(3), ")"

! Loop over all elements in lcentr
elements: do i = 1, size( lcentr )

  ! If the square of the distance between the cell and the centroid
  ! is less than the square of the characteristic length, then it's in
  if ( sum( (cacoord - lcentr(i)%centr(:) )**2 ) .lt. cl2 ) then
    flag = .true.
    exit elements
  end if
end do elements

end subroutine cgca_pfem_cellin
```

30.3 cgca_m3pfem/cgca_pfem_cenc

[cgca_m3pfem] [Subroutines]

NAME

cgca_pfem_cenc

SYNOPSIS

```
subroutine cgca_pfem_cenc( origin, rot, bcol, bcou )
```

INPUTS

```
real( kind=rdef ), intent( in ) ::                                    &
 origin(3),        & ! origin of the "box" cs, in FE cs
 rot(3,3),         & ! rotation tensor *from* FE cs *to* CA cs
 bcol(3),          & ! lower phys. coords of the coarray on image
 bcou(3)             ! upper phys. coords of the coarray on image
```

SIDE EFFECTS

Array lcentr (30.23) is changed.

DESCRIPTION

CENC stands for CENtroids Collection. This routine reads centroids of all elements, in FE coord. system, from all MPI processes and adds those with centroids within its CA "box" to its lcentr (30.23) array.

NOTES

This routine must be called only after coarray cgca_pfem_centroid_tmp (30.5) has been established on all images. This routine *reads* coarrays on other images, hence sync must be used before calling this routine. However, the routine *does not* change coarrays, only reads. So no syncs are required inside this routine, as it constitutes a single execution segment. This routine uses all-to-all comm pattern. This might be inefficient on large numbers of PEs. In this case on can use cgca_pfem_map (30.24.2) (in submodule m3pfem_sm1 (30.24)) instead, which does the same calculation using collectives and large tmp arrays.

USES

lcentr (30.23) via host association.

SOURCE

```
! initial length of lcentr array. A good choice will reduce the number
! of deallocate/allocate and will use the memory better.
integer, parameter :: lclenini = 100

integer :: errstat, i, j, nimgs, nelements, img_curr, ndims, rndint, &
 lclen, & ! current length of the lcentr array
 lcel     ! number of elements in lcentr array

! centroid coords in CA cs
real( kind=cgca_pfem_iwp ) :: cen_ca(3) ! 3D case only
real( kind=cgca_pfem_iwp ), allocatable :: tmp(:,:)
real :: rnd
```

```
! temp array to expand/contract lcentr
type( mcen ), allocatable :: lctmp(:)

nimgs = num_images()

! Allocate lcentr to the initial guess size.
lclen = lclenini
allocate( lcentr( lclen ), stat=errstat )
if ( errstat .ne. 0 ) then
  write (*,'(a,i0)')                                                    &
   "ERROR: cgca_pfem_cenc: allocate( lcentr ), error code: ", errstat
  error stop
end if

! There are no elements yet in lcentr array
lcel = 0

! Choose the first image at random
! It is assumed that the RND has been initialised by a call
! to cgca_irs earlier on.
call random_number( rnd )   ! [ 0 .. 1 )
rndint = int( rnd*nimgs )+1 ! [ 1 .. nimgs ]

! loop over all images, starting at a randomly chosen image
images: do i=rndint, rndint+nimgs-1

  ! Get the current image number.
  ! If it's > nimgs, subtract nimgs
  img_curr = i
  if ( img_curr .gt. nimgs ) img_curr = img_curr - nimgs

  ! how many elements
      ndims = size( cgca_pfem_centroid_tmp[ img_curr ] % r, dim=1 )
  nelements = size( cgca_pfem_centroid_tmp[ img_curr ] % r, dim=2 )

  ! use a temp array to pull all centroids data in one call
  allocate( tmp( ndims, nelements ), source=0.0_cgca_pfem_iwp,         &
            stat=errstat )
  if ( errstat .ne. 0 )                                                &
    error stop "ERROR: cgca_m3pfem/cgca_pfem_cenc: allocate( tmp )"
  tmp = cgca_pfem_centroid_tmp[ img_curr ] % r

  ! loop over all elements on that image
  elements: do j = 1, nelements

    ! Convert centroid coordinates from FE cs to CA cs
    ! cgca_pfem_centroid_tmp[i]        - variable on image i
    ! cgca_pfem_centroid_tmp[i]%r      - component that is the centroids
    !                                    real array
    ! cgca_pfem_centroid_tmp[i]%r(:,j) - take finite element j, and all
    !                                    centroid coordinates for it.
```

```
    !
    ! old algorithm - lots of small remote calls:
    ! cen_ca = &
    !  matmul( rot, cgca_pfem_centroid_tmp[ img_curr ]%r(:,j) - origin )
    !
    cen_ca = matmul( rot, tmp(:,j) - origin )

    ! Check whether CA cs centroid is within the box.
    ! If all CA cs centroid coordinates are greater or equal to
    ! the lower bound of the box, and all of them are also
    ! less of equal to the upper bound of the box, then the centroid
    ! is inside. Then add the new entry.
    inside: if ( all( cen_ca .ge. bcol ) .and.                          &
                 all( cen_ca .le. bcou ) ) then

      ! Increment the number of elements
      lcel = lcel + 1

      ! Expand the array if there is no space left to add the new entry.
      expand: if ( lclen .lt. lcel ) then

        ! Double the length of the array
        lclen = 2 * lclen

        ! Allocate a temp array of this length
        allocate( lctmp( lclen ), stat=errstat )
        if ( errstat .ne. 0 ) error stop                                &
          "ERROR: cgca_pfem_cenc: allocate( lctmp ) 1"

        ! copy lcentr into the beginning of lctmp
        lctmp( 1:size( lcentr ) ) = lcentr

        ! move allocation from the temp array back to lcentr
        call move_alloc( lctmp, lcentr )

      end if expand

      ! Add new entry
      lcentr( lcel ) = mcen( img_curr, j, cen_ca )

    end if inside

  end do elements

  deallocate( tmp, stat=errstat )
  if ( errstat .ne. 0 )                                                 &
    error stop "ERROR: cgca_pfem_cenc: deallocate( tmp )"

end do images

! Trim lcentr if it is longer than the number of elements
if ( lclen .gt. lcel ) then
```

```
  ! Allocate temp array to the number of elements
  allocate( lctmp( lcel ), stat=errstat )
  if ( errstat .ne. 0 ) error stop                                    &
    "ERROR: cgca_pfem_cenc: allocate( lctmp ) 2"

  ! Copy lcentr elements to the temp array
  lctmp = lcentr( 1 : lcel )

  ! move allocation from lctmp back to lcentr
  call move_alloc( lctmp, lcentr )
end if

end subroutine cgca_pfem_cenc
```

30.4 cgca_m3pfem/cgca_pfem_cendmp

[cgca_m3pfem] [Subroutines]

NAME

cgca_pfem_cendmp

SYNOPSIS

```
subroutine cgca_pfem_cendmp
```

SIDE EFFECTS

Dumps some data to stdout

DESCRIPTION

CENDMP stands for CENtroids array dump. This routine dumps lcentr (30.23) to stdout.

NOTES

Must call from all images.

SOURCE

```
integer :: i, img

img = this_image()

do i = 1, size( lcentr )
  write (*,"(3(a,i0),a,3(es10.2,tr1))")                                &
              "CA on img "       , img             ,                   &
              " <-> FE "         , lcentr(i)%elnum ,                   &
              " on img "         , lcentr(i)%image ,                   &
              " centr. in CA cs" , lcentr(i)%centr
end do

end subroutine cgca_pfem_cendmp
```

30.5 cgca_m3pfem/cgca_pfem_centroid_tmp

[cgca_m3pfem] [Data structures]

NAME

cgca_pfem_centroid_tmp

SYNOPSIS

```
type rca
  real( kind=cgca_pfem_iwp ), allocatable :: r(:,:)
end type rca
type( rca ) :: cgca_pfem_centroid_tmp[*]
```

DESCRIPTION

RCA stands for Rugged CoArray. cgca_pfem_centroid_tmp is a temporary scalar *coarray* of derived type with allocatable array component, storing centroids of ParaFEM finite elements, in FE coord. system, on this image. The array might be of different length on different images, so have to use an allocatable component of a coarray variable of derived type.

USED BY

routines of this module + end user

30.6 cgca_m3pfem/cgca_pfem_ctalloc

[cgca_m3pfem] [Subroutines]

NAME

cgca_pfem_ctalloc

SYNOPSIS

```
subroutine cgca_pfem_ctalloc( ndim, nels_pp )
```

INPUTS

```
integer, intent( in ) :: ndim, nels_pp

!    ndim - integer, number of DOF per node.
!    nels_pp - elements per MPI process (per image).
```

SIDE EFFECTS

Allocatable array component cgca_pfem_centroid_tmp (30.5)%r becomes allocated.

DESCRIPTION

CTA stands for Centroids Temporary Allocate. This routine allocates an allocatable array component of scalar coarray cgca_pfem_centroid_tmp (30.5). The allocatable array stores FE centroid coordinates together with their numbers and MPI ranks where these are stored.

NOTES

The array component can of different length on different images.

USES USED BY SOURCE

```
integer :: errstat=0

allocate( cgca_pfem_centroid_tmp%r( ndim, nels_pp ),                  &
          source=0.0_cgca_pfem_iwp,                                   &
          stat=errstat )
if ( errstat .ne. 0 ) error stop                                      &
  "ERROR: cgca_pfem_ctalloc: allocate( cgca_pfem_centroid_tmp%r )"

end subroutine cgca_pfem_ctalloc
```

30.7 cgca_m3pfem/cgca_pfem_ctdalloc

[cgca_m3pfem] [Subroutines]

NAME

cgca_pfem_ctdalloc

SYNOPSIS

```
subroutine cgca_pfem_ctdalloc
```

SIDE EFFECTS

Allocatable array component of cgca_pfem_centroid_tmp (30.5) becomes deallocate

DESCRIPTION

CTD stands for Centroids Temporary Allocate. This routine deallocates an allocatable array component of coar. This must be done only after all images copied the contents of type(rca) :: cgca_pfem_centroid_tmp (30.5)[*] into their local, *not* coarray centroid arrays.

USES USED BY SOURCE

```
integer :: errstat=0

deallocate( cgca_pfem_centroid_tmp%r, stat=errstat )
if ( errstat .ne. 0 ) error stop                                        &
  "ERROR: cgca_pfem_ctd: deallocate( cgca_pfem_centroid_tmp%r )"

end subroutine cgca_pfem_ctdalloc
```

30.8 cgca_m3pfem/cgca_pfem_ealloc

[cgca_m3pfem] [Subroutines]

NAME

cgca_pfem_ealloc

SYNOPSIS

```
subroutine cgca_pfem_ealloc( nip, nels_pp )
```

INPUTS

```
!    nip - integer, number of integration points
!    nels_pp - elements per MPI process (per image).

integer, intent( in ) :: nip, nels_pp
```

SIDE EFFECTS

Allocatable *local* array enew becomes allocated

DESCRIPTION

This routine allocates an allocatable *local* array. The allocatable array stores the Young's modulus per FE element and integration point, for all FE that are stored on this image.

USED BY

end user?

SOURCE

```
integer :: errstat=0

allocate( cgca_pfem_enew( nip, nels_pp ), stat=errstat )
if ( errstat .ne. 0 ) then
  write (*,"(a,i0)")                                                  &
   "ERROR: cgca_pfem_ealloc: allocate( cgca_pfem_enew ), err. status", &
    errstat
  error stop
end if

end subroutine cgca_pfem_ealloc
```

30.9 cgca_m3pfem/cgca_pfem_edalloc

[cgca_m3pfem] [Subroutines]

NAME

cgca_pfem_edalloc

SYNOPSIS

```
subroutine cgca_pfem_edalloc
```

SIDE EFFECTS

Allocatable *local* array cgca_pfem_enew (30.10) becomes deallocated.

DESCRIPTION

This routine deallocates an allocatable *local* array used to store the Young's modulus per FE element and integration point

USES

cgca_pfem_enew (30.10) via host association.

USED BY

end user?

SOURCE

```
integer :: errstat=0

deallocate( cgca_pfem_enew, stat=errstat )
if ( errstat .ne. 0 ) error stop                                        &
  "ERROR: cgca_pfem_ealloc: deallocate( cgca_pfem_enew )"

end subroutine cgca_pfem_edalloc
```

30.10 cgca_m3pfem/cgca_pfem_enew

[cgca_m3pfem] [Data structures]

NAME

cgca_pfem_enew

SYNOPSIS

```
real( kind=cgca_pfem_iwp ), allocatable :: cgca_pfem_enew(:,:)
```

DESCRIPTION

Naming: E New as in new Young's modulus. This *local* array stores Young's moduli for each integration point of each FE on this image.

USED BY

cgca_pfem_uym (30.21) + end user

30.11 cgca_m3pfem/cgca_pfem_intcalc1

[cgca_m3pfem] [Subroutines]

NAME

cgca_pfem_intcalc1

SYNOPSIS

```
subroutine cgca_pfem_intcalc1( arrsize, fracvol )
```

INPUTS

```
!     arrsize - contains the 3 sizes of the space coarray.
!    Using the coarray sizes, the characteristic coarray
!    area is calculated.
!     fracvol - *real*, the number of failed (fractured) cells for each
!    image. It is calculated by cgca_fv, which *must* be called prior to
!    calling this routine.

integer( kind=iarr ), intent( in ) :: arrsize(3)
real( kind=rdef), intent( in ) :: fracvol
```

SIDE EFFECTS

cgca_pfem_integrity (30.14) array changes

DESCRIPTION

All FEs linked to this image get the same value of integrity. These are all FEs in lcentr (30.23) array. For entry i in this array this is FE cgca_pfem_integrity (30.14)(lcentr (30.23)(i)%elnum) on image lcentr (30.23)(i)%image.

The integrity is 1 minus the ratio of number of failed cells to the cracteristic coarray area. If integrity $<$ 0, set it to zero.

USES

lcentr (30.23) via host association

USED BY

end user?

SOURCE

```
real :: carea ! characteristic area
integer, parameter :: kind_integ = kind( cgca_pfem_integrity % i )
real( kind=kind_integ ), parameter :: one = 1_kind_integ
integer( kind=idef ) :: i

! Volume, in cells, is the product of 3 coarray sizes.
! Don't forget to remove the halo cells!
! Characteristic area is volume ** 2/3
carea = product( real( arrsize-2 ) ) ** 0.66666666666666666667

do i = 1, size( lcentr )
```

```
 ! integrity is calculate as: i = 1 - min(1,f),
 ! Integrity has the range [1..0], where i=1 for f=0, i=0 for f=1.
 ! f=fracvol / carea   - Fraction of failed cells, 0 if no fracture,
 !                       1 or above when I consider the CA to have no
 !                       load bearing capacity.
 ! min( 1, fraction) - To make sure fraction is [0..1].
 cgca_pfem_integrity[ lcentr(i)%image ] % i( lcentr(i)%elnum ) =       &
          one - min( one, fracvol / carea )
end do

end subroutine cgca_pfem_intcalc1
```

30.12 cgca_m3pfem/cgca_pfem_integalloc

[cgca_m3pfem] [Subroutines]

NAME

cgca_pfem_integalloc

SYNOPSIS

```
subroutine cgca_pfem_integalloc( nels_pp )
```

INPUT

```
!    nels_pp - elements per MPI process (per image).

integer, intent(in) :: nels_pp
```

SIDE EFFECTS

Allocatable array component cgca_pfem_integrity (30.14)%i becomes allocated

DESCRIPTION

This routine allocates cgca_pfem_integrity (30.14)%i on this image. This is a *local*, non-coarray, array. Hence this routine can be called by any or all images. It should be called by all images, of course.

The array is allocated with the length equal to the number FE stored on *that* image.

Must set i to 1, to take care of cases when some FE are not linked to CA. integrity for such FESuch FE which are

USES

cgca_pfem_integrity (30.14) via host association.

USED BY

end user?

SOURCE

```
integer :: errstat=0

allocate( cgca_pfem_integrity%i( nels_pp ),  source=1.0_rdef,          &
          stat=errstat )
if ( errstat .ne. 0 ) then
  write (*,"(a,i0)")                                                   &
    "ERROR: cgca_pfem_integalloc: allocate( cgca_pfem_integrity%i )",  &
    errstat
  error stop
end if

end subroutine cgca_pfem_integalloc
```

30.13 cgca_m3pfem/cgca_pfem_integdalloc

[cgca_m3pfem] [Subroutines]

NAME

cgca_pfem_integdalloc

SYNOPSIS

```
subroutine cgca_pfem_integdalloc
```

SIDE EFFECTS

Allocatable array component of cgca_pfem_integrity (30.14) coarray becomes deallocated.

DESCRIPTION

This routine deallocates allocatable array component of integrity: cgca_pfem_integrity (30.14)%i.

USES

cgca_pfem_integrity (30.14) via host association

USED BY

end user?

SOURCE

```
integer :: errstat=0

deallocate( cgca_pfem_integrity%i, stat=errstat )
if ( errstat .ne. 0 ) then
  write (*,"(a,i0)")                                                  &
   "ERROR: cgca_pfem_integalloc: deallocate( cgca_pfem_integrity%i )", &
   errstat
  error stop
end if

end subroutine cgca_pfem_integdalloc
```

30.14 cgca_m3pfem/cgca_pfem_integrity

[cgca_m3pfem] [Data structures]

NAME

cgca_pfem_integrity

SYNOPSIS

```
type cgca_pfem_integ_type
 real( kind=rdef ), allocatable :: i(:)
end type cgca_pfem_integ_type
type( cgca_pfem_integ_type ) :: cgca_pfem_integrity[*]
```

DESCRIPTION

A derived type is needed because the length of the integrity array will differ from image to image. So this is a scalar coarray of derived type with a single component: allocatable array of integrity, i. i=1 means to damage, i=0 means no remaining load bearing capacity. This data will be used to update the Young's modulus

NOTE

Set i to 1 on allocation to avoid problems later. The reason is that in cases when some FE are not connected to CA, the integrity of these FE will never be set of changed. So setting i to 1 on allocation is fool proof.

USED BY

cgca_pfem_uym (30.21) + end user?

30.15 cgca_m3pfem/cgca_pfem_partin

[cgca_m3pfem] [Subroutines]

NAME

cgca_pfem_partin

SYNOPSIS

```
subroutine cgca_pfem_partin( coarray, cadim, lres, bcol, charlen, debug)
```

INPUT

```
!      coarray - main model coarray
!        cadim - coarray dimensions. Can calculate from coarray, but
!                these will be known already anyway, so makes sense
!                to pass as input
!         lres - real, linear resolution of the model, how many cells
!                per linear physical unit of length.
!      bcol(3) - real, coordinates of the coarray box on this image
!                in physical units, in CA coord. system.
!      charlen - real, characteristic length of an FE in the model.
!                This parameter is used to determine whether a cell
!                is "close enough" to a centroid of an FE.
!        debug - logical. If .true. will dump some dubug output

integer( kind=iarr ), allocatable, intent( inout ) ::                  &
  coarray( : , : , : , : ) [ : , : , : ]
integer( kind=iarr ), intent( in ) :: cadim(3)
real( kind=rdef ), intent( in ) :: lres, bcol(3), charlen
logical( kind=ldef ), intent( in ) :: debug
```

SIDE EFFECTS

state of coarray changed

DESCRIPTION

This is the most thorough routine to decide if cells are "inside" the FE model or not. It starts by checking boxes the size of the whole coarray on this image. If a box in partially in and partially out, it is split into two smaller boxes and the process continues until each box is either fully in, or fully out. If it is fully out, all fracture level cells of this box are set to state cgca_state_null (9.22).

NOTES

This routine calls cgca_pfem_boxin (30.1), which in turn calls cgca_pfem_cellin (30.2), which accesses lcentr (30.23) array on its own image. This routine updates coarray *locally*, on its own image only. So there are no remote read or write operations in this routine. No sync is needed inside this routine. Sync is likely needed before and after. In particular, the coarray and lcentr (30.23) array must be defined on all images prior to calling this routine on any image.

USES

cgca_pfem_boxin (30.1), cgca_boxsplit (12.1), cgca_inithead (16.3), cgca_addhead (16.1), cgca_rmhead (16.7), cgca_lstdmp (16.6)

USED BY

end user?

SOURCE

```
integer( kind=idef ) :: lwr(3), upr(3), iflag, stat, lwr1(3),          &
  upr1(3), lwr2(3), upr2(3)
type( cgca_lnklst_tpayld ) :: payload
type( cgca_lnklst_node ), pointer :: head

integer :: iter, img

img = this_image()

! Start with a box the size of the whole coarray on this image
! Note a conversion from iarr to idef
lwr = 1
upr = int( cadim, kind=idef )

! Initialise the linked list with this box as the head node
! Returns the pointer to the head node
payload%lwr = lwr
payload%upr = upr
call cgca_inithead( head, payload )

! Start iteration counter
iter = 1

! Check all nodes on the list
list: do

  ! debug output
  if ( debug ) then
    write (*,'(2(a,i0),a)') "DEBUG: cgca_pfem_partin: img: ", img,     &
                         " iter: ", iter, " linked list dump:"
    call cgca_lstdmp( head )
  end if

  ! Get the payload from the head node on the list
  payload = head%value
  lwr = payload%lwr
  upr = payload%upr

  ! Initialise iflag before it's called for the first time
  iflag=-1

  ! Check if this box is in/out
  !subroutine cgca_pfem_boxin( lowr, uppr, lres, bcol, charlen, debug,
  !   iflag )
  call cgca_pfem_boxin( lwr, upr, lres, bcol, charlen, debug, iflag )

  ! Remove this box from the linked list in any case
  ! Check stat value later.
  call cgca_rmhead( head, stat )
```

```
  ! The whole box in, iflag=1
  chkiflag: if ( iflag .eq. 1 ) then

    ! Exit if the list has zero nodes
    if ( stat .eq. 1 ) exit list

  ! The whole box out, iflag=2
  else if ( iflag .eq. 2 ) then

    ! Mark all cells of this box, on *this* image, in the fracture
    ! layer as cgca_state_null.
    coarray( lwr(1):upr(1) , lwr(2):upr(2), lwr(3):upr(3) ,            &
             cgca_state_type_frac ) = cgca_state_null

    ! Exit if the list has zero nodes
    if ( stat .eq. 1 ) exit list

  ! Part in/part out
  else if ( iflag .eq. 3 ) then

    ! Split the box in two along the biggest dimension of the box
    call cgca_boxsplit( lwr, upr, lwr1, upr1, lwr2, upr2 )

    ! Add two new boxes on top of head.
    ! The first box.
    payload%lwr = lwr1
    payload%upr = upr1

    ! For the first node, if the head is not associated, i.e.
    ! stat is 1, then use cgca_inithead instead of cgca_addhead.
    ! The head will not be associated if the initial box, i.e.
    ! the whole CA on this image has to be split, which must always
    ! happen in the beginning of the process.
    if ( stat .eq. 0 ) then
      call cgca_addhead( head, payload )
    else
      call cgca_inithead( head, payload )
    end if

    ! The second box.
    payload%lwr = lwr2
    payload%upr = upr2
    call cgca_addhead( head, payload )

  end if chkiflag

  ! increase the iteration counter
  iter = iter + 1

end do list
```

```
end subroutine cgca_pfem_partin
```

30.16 cgca_m3pfem/cgca_pfem_salloc

[cgca_m3pfem] [Subroutines]

NAME

cgca_pfem_salloc

SYNOPSIS

```
subroutine cgca_pfem_salloc( nels_pp, intp, comp )
```

INPUTS

```
!    nels_pp - number of elements on this image
!    intp - number of integration points per element
!    comp - number of stress tensor components

integer, intent( in ) :: nels_pp, intp, comp
```

SIDE EFFECTS

Allocatable component array cgca_pfem_stress (30.20)%stress becomes allocated

DESCRIPTION

SALLOC stands for Allocate Stress tensor array. This routine allocates an allocatable array component of coar. The allocatable array stores all stress tensor components, for all integration points on all elements on an image.

USES

cgca_pfem_iwp, host association

USED BY

end user

SOURCE

```
integer :: errstat=0

allocate( cgca_pfem_stress%stress( nels_pp, intp, comp ),              &
          source=0.0_cgca_pfem_iwp, stat=errstat )
if ( errstat .ne. 0 ) then
  write (*,"(a,i0)") "ERROR: cgca_pfem_salloc: allocate( &
    &cgca_pfem_stress%stress ), err. status: ", errstat
  error stop
end if

end subroutine cgca_pfem_salloc
```

30.17 cgca_m3pfem/cgca_pfem_sdalloc

[cgca_m3pfem] [Subroutines]

NAME

cgca_pfem_sdalloc

SYNOPSIS

```
subroutine cgca_pfem_sdalloc
```

SIDE EFFECTS

allocatable array cgca_pfem_stress (30.20)%stress become deallocated

DESCRIPTION

SDALLOC stands for Deallocate Stress tensor array. This routine deallocates allocatable array component of coar. This routine should be called only when the analysis is complete. Any and every image can call this routine.

USES

cgca_pfem_stress (30.20)%stress, host association

USED BY SOURCE

```
integer :: errstat=0

deallocate( cgca_pfem_stress%stress, stat=errstat )
if ( errstat .ne. 0 ) then
  write (*,"(a,i0)") "ERROR: cgca_pfem_sdalloc: deallocate( &
    &cgca_pfem_stress%stress ), err. status: ", errstat
  error stop
end if

end subroutine cgca_pfem_sdalloc
```

30.18 cgca_m3pfem/cgca_pfem_sdmp

[cgca_m3pfem] [Subroutines]

NAME

cgca_pfem_sdmp

SYNOPSIS

```
subroutine cgca_pfem_sdmp
```

SIDE EFFECTS

Dumps some data to stdout

DESCRIPTION

SDMP stands for Stress tensor dump. This routine dumps stress tensors to stdout.

NOTES

Must call from all images.

SOURCE

```
integer :: img, nel, nintp, el, intp

  img = this_image()
  nel = size( cgca_pfem_stress%stress, dim=1 )
nintp = size( cgca_pfem_stress%stress, dim=2 )

do el = 1, nel
  do intp = 1, nintp
    write (*,*) "img", img, "FE", el, "int p.", intp, "stress",        &
                cgca_pfem_stress%stress( el, intp, : )
  end do
end do

end subroutine cgca_pfem_sdmp
```

30.19 cgca_m3pfem/cgca_pfem_simg

[cgca_m3pfem] [Subroutines]

NAME

cgca_pfem_simg

SYNOPSIS

```
subroutine cgca_pfem_simg( simg )
```

OUTPUT

```
!    simg - mean stress tensor over all integration points on all
!    finite elements linked to CA on this image.
!    Note that I use CGPACK kind, because this var will be input to
!    a CGPACK routine.

real( kind=rdef ), intent(out) :: simg(3,3)
```

DESCRIPTION

SIMG stands for mean Stress on an Image. The routine reads all stress tensors from all integration points for all elements which are linked to CA on this image, i.e. from lcentr (30.23) array, and calculates the mean value. This value is then used to pass to the cleavage routine.

NOTE

If size(lcentr (30.23)) .eq. 0, then there are no FE associated with coarray on this image. The set simg to 0.

SOURCE

```
integer, parameter :: comp=6 ! number of stress components

! Running total stress array
real( kind=rdef ) :: stot( comp )

! Temp stress array, to store remotely read values
real( kind=rdef ), allocatable :: str_tmp( : , : )

integer :: el, nel, rel, nintp, rimg, errstat

! Assertion check
! The number of stress components (6) is the same as dimension 3 of
! cgca_pfem_stress%stress
if ( size( cgca_pfem_stress%stress, dim=3 ) .ne. comp )                &
  error stop "ERROR: cgca_pfem_simg: &
   &size( cgca_pfem_stress%stress, dim=3 ) .ne. comp"

! Total number of elements linked to CA model on this image
! Total number of int. points per element
  nel = size( lcentr )
nintp = size( cgca_pfem_stress%stress, dim=2 )
```

```
! If there are no FE linked to this image, set simg to 0
! and return immediately.
if ( nel .eq. 0 ) then
  simg = 0.0_rdef
  return
end if

! Allocate tmp stress array
allocate( str_tmp( nintp, comp ), source=0.0_rdef, stat=errstat )
if ( errstat .ne. 0 )                                                  &
  error stop "ERROR: cgca_pfem_simg: allocate( str_tmp )"

! Add all stress tensors together. Loop over all elements linked
! to CA on this image and over all int. points.
stot = 0.0_rdef
do el=1, nel

  ! Calculate the image and the element numbers to read the stress
  ! data from.
  rimg = lcentr(el) % image
   rel = lcentr(el) % elnum

  ! Remote read of all stress values for this element
  str_tmp =                                                            &
    real( cgca_pfem_stress[ rimg ] % stress( rel, : , : ), kind=rdef )

  ! Sum over all int. points, i.e. 1st dimension
  stot = stot + sum( str_tmp( : , : ), dim=1 )

end do

! Construct a (3,3) matrix from (6) vector.
! Observe the component order of ParaFEM
!   sx=stress(1)
!   sy=stress(2)
!   sz=stress(3)
!   txy=stress(4)
!   tyz=stress(5)
!   tzx=stress(6)
!   sigm=(sx+sy+sz)/three
!https://code.google.com/p/parafem/source/browse/trunk/parafem/src/modules/shared/new_library.f90
simg(1,1) = stot(1)
simg(2,2) = stot(2)
simg(3,3) = stot(3)
simg(1,2) = stot(4)
simg(2,3) = stot(5)
simg(3,1) = stot(6)
simg(2,1) = simg(1,2)
simg(3,2) = simg(2,3)
simg(1,3) = simg(3,1)
```

```
! calculate the mean
simg = simg / real( nel*nintp, kind=rdef )

! deallocate temp stress array
deallocate( str_tmp, stat=errstat )
if ( errstat .ne. 0 ) then
  write (*,'(a,i0)')                                                    &
    "ERROR: cgca_pfem_simg: deallocate( str_tmp ), err. code: ", errstat
  error stop
end if

end subroutine cgca_pfem_simg
```

30.20 cgca_m3pfem/cgca_pfem_stress

[cgca_m3pfem] [Data structures]

NAME

cgca_pfem_stress

SYNOPSIS

```
type type_stress
  real( kind=cgca_pfem_iwp ), allocatable :: stress(:,:,:)
end type type_stress
type( type_stress ) :: cgca_pfem_stress[*]
```

DESCRIPTION

This is a coarray with a single allocatable array component, to store all stress components for all integration points for all elements on an image. Have to use a derived type because cgca_pfem_stress%stress can be allocated to different length on different images. This data will be read by all images.

30.21 cgca_m3pfem/cgca_pfem_uym

[cgca_m3pfem] [Subroutines]

NAME

cgca_pfem_uym

SYNOPSIS

```
subroutine cgca_pfem_uym( e_orig, nels_pp )
```

INPUTS

```
!     e_orig - *real* is the original Young's modulus.
!    For now assume a single value, i.e. all int points
!    have identical original value.
!     nels_pp - number of FEs for this image.

real( kind=cgca_pfem_iwp ), intent(in) :: e_orig
integer, intent(in) :: nels_pp
```

SIDE EFFECTS

The Young's modulus gets updated with integrity

DESCRIPTION

UYM stands for Update Young's Modulus This routine updates the value of the Young's modulus, e, e = e_original * integrity. Integrity - integer, cell integrity (from 0.0 to 1.0)

NOTES

Purely local routine, no coarray operations. It seems the Young's modulus of 0 causes instability. So don't let it get to 0, use a small factor instead, e.g. 1.0e-3.

USES

cgca_pfem_enew (30.10), cgca_pfem_integrity (30.14), all via host association.

USED BY

end user?

SOURCE

```
real( kind=rdef ), parameter :: factor = 1.0e-3_rdef
integer :: fe

do fe = 1, nels_pp

 cgca_pfem_enew( : , fe ) = max( factor*e_orig,                          &
                            e_orig * cgca_pfem_integrity % i( fe ) )
end do

end subroutine cgca_pfem_uym
```

30.22 cgca_m3pfem/cgca_pfem_wholein

[cgca_m3pfem] [Subroutines]

NAME

cgca_pfem_wholein

SYNOPSIS

```
subroutine cgca_pfem_wholein( coarray )
```

INPUT

```
!    coarray - main model coarray

integer( kind=iarr ), allocatable, intent( inout ) ::                  &
  coarray( : , : , : , : ) [ : , : , : ]
```

SIDE EFFECTS

state of coarray changed

DESCRIPTION

This is a very primitive routine to decide if cells are "inside" the FE model or not. It work on the whole coarray on an image. If there are no FE linked to coarray on this image, i.e. if lcentr (30.23) array is of zero length, then all cells in the fracture layer of the coarray on this image are turned to cgca_state_null (9.22).

SOURCE

```
if ( size( lcentr ) .eq. 0 )                                           &
  coarray( : , : , : , cgca_state_type_frac ) = cgca_state_null
end subroutine cgca_pfem_wholein
```

30.23 cgca_m3pfem/lcentr

[cgca_m3pfem] [Data structures]

NAME

lcentr

SYNOPSIS

```
type mcen
  integer( kind=idef ) :: image
  integer( kind=idef ) :: elnum
  real( kind=cgca_pfem_iwp ) :: centr(3)
end type mcen
type( mcen ), allocatable :: lcentr(:)
```

DESCRIPTION

A *private* *local* allocatable array of derived type with 3 components: (1) image number (2) the local element number on that image and (3) centroid coordinates in CA CS. Each entry in this array corresponds to an FE with centroid coordinates within the coarray "box" on this image.

Assumption!! This is a 3D problems, so the centroid is defined by 3 coordinates, hence centr(3).

MCEN stands for Mixed CENtroid data type. lcentr stands for *Local* array of CENTRoids.

NOTE

This is *private* array, hence the name does not start with "cgca_pfem".

USED BY

Many routines of this module.

30.24 cgca_m3pfem/m3pfem_sm1

[cgca_m3pfem] [Submodules]

NAME

m3pfem_sm1

SYNOPSIS

```
!$Id: m3pfem_sm1.f90 380 2017-03-22 11:03:09Z mexas $

submodule ( cgca_m3pfem ) m3pfem_sm1
```

DESCRIPTION

Submodule with routines using collectives.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

CONTAINS

cgca_pfem_map (30.24.2), cgca_pfem_lcentr_dump (30.24.1)

USES

All variables and parameters of module cgca_m3pfem (30) by host association

USED BY

The host module cgca_m3pfem (30)

SOURCE

30.24.1 m3pfem_sm1/cgca_pfem_lcentr_dump

[m3pfem_sm1] [Subroutines]

NAME

cgca_pfem_lcentr_dump

SYNOPSIS

```
module procedure cgca_pfem_lcentr_dump
```

SIDE EFFECTS

Dump lcentr (30.23) from this image to OUTPUT_UNIT.

DESCRIPTION

This routine is used for debugging, i.e. to check that cgca_pfem_cenc (30.3) and cgca_pfem_map (30.24.2) produce identical output, as they should.

SOURCE

```
integer :: i, img

img = this_image()

!write (*,*) "DEBUG: img:", this_image(), "size( lcentr ):", size(lcentr)

!if ( img .le. 50 ) then
  do i = lbound( lcentr, dim=1 ) , ubound( lcentr, dim=1 )
    write (*,"(2(a,i0),tr1,i0,3(tr1,es9.2))") "DEBUG: img: ", img,     &
      " lcentr: ", lcentr(i)
  end do
!end if

end procedure cgca_pfem_lcentr_dump
```

30.24.2 m3pfem_sm1/cgca_pfem_map

[m3pfem_sm1] [Subroutines]

NAME

cgca_pfem_map

SYNOPSIS

```
!module procedure cgca_pfem_map
  module subroutine cgca_pfem_map( origin, rot, bcol, bcou )
    real( kind=rdef ), intent( in ) ::                                  &
      origin(3),        & ! origin of the "box" cs, in FE cs
      rot(3,3),         & ! rotation tensor *from* FE cs *to* CA cs
      bcol(3),          & ! lower phys. coords of the coarray on image
      bcou(3)             ! upper phys. coords of the coarray on image
```

INPUTS

```
!    See interface in the host module cgca_m3pfem.
```

SIDE EFFECTS

Array lcentr (30.23) is changed.

DESCRIPTION

This routine reads centroids of all elements, in FE coord. system, from all MPI processes and adds those with centroids within its CA "box" to its lcentr (30.23) array.

NOTES

This routine is an alternative to cgca_pfem_cenc (30.3). While cgca_pfem_cenc (30.3) implements all-to-all algorithm, this routine uses collectives. This routine must be called only after coarray cgca_pfem_centroid_tmp (30.5) has been established on all images. This routine *reads* coarrays on other images, hence sync must be used before calling this routine. However, the routine *does not* change coarrays, only reads. So no syncs are required inside this routine, as it constitutes a single execution segment. This routine uses CO_MAX and CO_SUM collective. This routine allocates *large* tmp arrays on every image. The array size is equal or even bigger than the number of FE in the *whole* model, more precisely the array is allocated on every image as (5, <max no. of FE on any image>*num_images()). Hence this routine might give OOM for large models. In that case fall back to cgca_pfem_cenc (30.3).

USES

lcentr (30.23) via host association.

SOURCE

```
! Initial length of lcentr array. A good choice will reduce the number
! of deallocate/allocate and will use the memory better.
integer, parameter :: lclenini = 100

real( kind=cgca_pfem_iwp ), allocatable :: tmp(:,:)

! Centroid coords in CA cs
real( kind=cgca_pfem_iwp ) :: cen_ca(3) ! 3D case only
```

```
! Temp array to expand/contract lcentr
type( mcen ), allocatable :: lctmp(:)

integer( kind=idef ) :: img, nimgs, maxfe, pos_start, pos_end, lclen,  &
  lcel, j, ctmpsize

integer :: errstat

!*********************************************************************72
! First executable statement
  img = this_image()
nimgs = num_images()

! Calculate the max number of FE stored on this image, i.e. nels_pp.
maxfe = size( cgca_pfem_centroid_tmp%r, dim=2 )

! Save it in a separate var
ctmpsize = maxfe

! Calculate the max number of FE stored on any image.
! Use CO_MAX collective. RESULT_IMAGE is not used.
! Hence the result is assigned to maxfe on all images.
call co_max( maxfe )

!write (*,*) "DEBUG, after co_max, img:", img, " maxfe:", maxfe

! Allocate tmp array of length ( maxfe * nimgs )
! 5 real values are stored per each FE:
! 1 - image number, cast to real
! 2 - FE number, cast to real.
! 3-5 - coordinates of the centroid of this FE
! NOTE! Important to set to zero initially, either on allocatation,
! or later, but before use. The following algorithm relies on the
! fact that tmp is zero initially on all images.
allocate( tmp( maxfe * nimgs, 5 ), source = 0.0_cgca_pfem_iwp,         &
          stat=errstat )
if ( errstat .ne. 0 ) then
  write (*,'(2(a,i0))') "ERROR: m3pfem_sm1/cgca_pfem_map: img: ", img, &
    ", allocate( tmp ), stat: ", errstat
  error stop
end if

! Write values in correct places in array tmp on this image.
! Use this_image() as the offset.
pos_start = (img - 1) * maxfe + 1
pos_end = pos_start + ctmpsize - 1

! Write image number
tmp( pos_start : pos_end, 1 ) = real( img, kind=cgca_pfem_iwp )

! Write element number
```

```
tmp( pos_start : pos_end, 2 ) =                                          &
  real( (/ (j, j = 1, ctmpsize) /), kind=cgca_pfem_iwp )

! Write centroid coord
tmp( pos_start : pos_end, 3:5 ) = &
  transpose( cgca_pfem_centroid_tmp%r(:,:) )

! Calculate the sum of tmp arrays over all images.
! Because each image wrote its data in a unique location,
! the sum will just produce the tmp array with data from all images.
! Then this tmp array is delivered back to all images.
! Since RESULT_IMAGE is not used, the result is assigned to tmp
! on all images.
call co_sum( tmp )

! Now each image searches through the whole of tmp array and
! adds all elements with centroids inside its CA box, to its local
! (private) lcentr array.

! Allocate lcentr to the initial guess size.
lclen = lclenini
allocate( lcentr( lclen ), stat=errstat )
if ( errstat .ne. 0 ) then
  write (*,'(2(a,i0))') "ERROR: m3pfem_sm1/cgca_pfem_map: img: ", img, &
    ", allocate( lcentr ), stat: ", errstat
  error stop
end if

! There are no elements yet in lcentr array
lcel = 0

! Loop over all elements in tmp array
elements: do j = 1, size( tmp, dim=1 )

  ! Convert centroid coordinates from FE cs to CA cs.
  ! tmp( 3:5 , j ) - take finite element j, and all centroid
  ! coordinates for it.
  cen_ca = matmul( rot, tmp( j, 3:5 ) - origin )

  ! Check whether CA cs centroid is within the box.
  ! If all CA cs centroid coordinates are greater or equal to
  ! the lower bound of the box, and all of them are also
  ! less of equal to the upper bound of the box, then the centroid
  ! is inside. Then add the new entry.
  inside: if ( all( cen_ca .ge. bcol ) .and.                             &
               all( cen_ca .le. bcou ) ) then

    ! Skip zero elements
    if ( int( tmp(j,1), kind=idef ) .eq. 0_idef .or.                     &
         int( tmp(j,2), kind=idef ) .eq. 0_idef ) cycle elements

    ! Increment the number of elements
```

```
    lcel = lcel + 1

    ! Expand the array if there is no space left to add the new entry.
    expand: if ( lclen .lt. lcel ) then

      ! Double the length of the array
      lclen = 2 * lclen

      ! Allocate a temp array of this length
      allocate( lctmp( lclen ), stat=errstat )
      if ( errstat .ne. 0 ) then
        write (*,'(2(a,i0))') "ERROR: m3pfem_sm1/cgca_pfem_map: img: ",&
          img, ", allocate( lctmp ) 1, stat: ", errstat
        error stop
      end if

      ! copy lcentr into the beginning of lctmp
      lctmp( 1:size( lcentr ) ) = lcentr

      ! move allocation from the temp array back to lcentr
      call move_alloc( lctmp, lcentr )

    end if expand

    ! Add new entry
    lcentr( lcel ) = mcen( int( tmp(j,1), kind=idef ),                &
      int( tmp(j,2), kind=idef ), cen_ca )

  end if inside

end do elements

! Can now deallocate tmp
deallocate( tmp, stat=errstat )
if ( errstat .ne. 0 ) then
  write (*,'(2(a,i0))') "ERROR: m3pfem_sm1/cgca_pfem_map: img: ", img, &
    ", deallocate( tmp ), stat: ", errstat
  error stop
end if

! Trim lcentr if it is longer than the number of elements
if ( lclen .gt. lcel ) then

  ! Allocate temp array to the number of elements
  allocate( lctmp( lcel ), stat=errstat )
  if ( errstat .ne. 0 ) then
    write (*,'(2(a,i0))') "ERROR: m3pfem_sm1/cgca_pfem_map: img: ",    &
      img, ", allocate( lctmp ) 2, stat: ", errstat
    error stop
  end if

  ! Copy lcentr elements to the temp array
```

```
  lctmp = lcentr( 1 : lcel )

  ! move allocation from lctmp back to lcentr
  call move_alloc( lctmp, lcentr )
end if

!end procedure cgca_pfem_map
end subroutine cgca_pfem_map
```

31 CGPACK/cgca_m3sld

[Modules]

NAME

cgca_m3sld

SYNOPSIS

```
!$Id: cgca_m3sld.f90 431 2017-06-30 13:13:49Z mexas $

module cgca_m3sld
```

DESCRIPTION

Module dealing with solidification

AUTHOR

Anton Shterenlikht

COPYRIGHT

CONTAINS

Subroutines: cgca_sld (31.1), cgca_sld1 (31.2), cgca_sld2 (31.3), cgca_sld3 (31.5.1) (in submodule m3sld_sm1 (31.5)), cgca_sld_h (31.4.1) (in submodule m3sld_hc (31.4)).

USES

cgca_m1co (9), cgca_m2hx (15), cgca_m2red (22)

USED BY

cgca

SOURCE

```
use cgca_m1co
use cgca_m2hx
use cgca_m2red

implicit none

private
public :: cgca_sld, cgca_sld1, cgca_sld2, &
          cgca_sld3    , & ! in submodule m3sld_sm1
          cgca_sld_h       ! in submodule m3sld_hc

abstract interface
  subroutine halo_exchange( array )
    use cgca_m1co
    integer( kind=iarr ), allocatable, intent( inout ) ::              &
                                                array(:,:,:,:)[:,:,:]
  end subroutine halo_exchange
end interface
```

```
interface
  ! In submodule m3sld_sm1
  module subroutine cgca_sld3( coarray, iter, heartbeat, solid )
    integer( kind=iarr ), allocatable, intent( inout ) ::              &
      coarray(:,:,:,:)[:,:,:]
    integer( kind=idef ), intent( in ) :: iter,heartbeat
    logical( kind=ldef ), intent( out ) :: solid
  end subroutine cgca_sld3

  ! In submodule m3sld_hc
  module subroutine cgca_sld_h( coarray, hx, iter, heartbeat, solid )
    integer( kind=iarr ), allocatable, intent( inout ) ::              &
      coarray(:,:,:,:)[:,:,:]
    procedure( halo_exchange ) :: hx
    integer( kind=idef ), intent( in ) :: iter, heartbeat
    logical( kind=ldef ), intent( out ) :: solid
  end subroutine cgca_sld_h
end interface

contains
```

31.1 cgca_m3sld/cgca_sld

[cgca_m3sld] [Subroutines]

NAME

cgca_sld

SYNOPSIS

```
subroutine cgca_sld( coarray, periodicbc, iter, heartbeat, solid )
```

INPUTS

```
integer( kind=iarr ), allocatable, intent( inout ) ::                  &
  coarray(:,:,:,:)[:,:,:]
integer( kind=idef ), intent( in ) :: iter, heartbeat
logical( kind=ldef ), intent( in ) :: periodicbc
```

OUTPUT

```
logical( kind=ldef ), intent( out ) :: solid
```

SIDE EFFECTS

None

DESCRIPTION

This routine scans over all cgca_liquid_state (9.18) cells and gives them a chance to attach to a grain. Thus the grains grow and the liquid phase decreases. The routine can be run for a number of iterations, or until the whole model has solidified, i.e. no more liquid phase left.

Inputs:

- coarray - the model
- periodicbc - if .true. periodic boundary conditions are used, i.e. global halo exchange is called after every iteration
- iter - number of solidification iterations, if <=0 then do until the coarray has solidified; if >0 then proceed until solid or "iter" iterations have been completed, whichever is sooner
- heartbeat - if >0 then dump a simple message every heartbeat iterations

Outputs:

- solid - .true. if the coarray is fully solid, .false. otherwise

At least one value .gt. cgca_liquid_state (9.18) must exist on at least one image. At least one cgca_liquid_state (9.18) value must exist on at least one image.

NOTES

All images must call this routine! The grain numbers are *always* greater than the liquid state.

USES USED BY

none, end user

SOURCE

```
real :: candidate(3)

integer( kind=iarr ), allocatable :: array( : , : , : )
integer :: i, errstat
integer( kind=idef ) :: &
  lbv(4),      & ! lower bounds of the complete (plus virtual) coarray
  ubv(4),      & ! upper bounds of the complete (plus virtual) coarray
  lbr(4),      & ! lower bounds of the "real" coarray, lower virtual+1
  ubr(4),      & ! upper bounds of the "real" coarray, upper virtual-1
  thisimage   ,& ! to avoid repeated calls to this_image()
  numimages   ,& ! to avoid repeated calls to num_images()
  x1          ,& ! local coordinates in an array, which are also
  x2          ,& ! do loop counters
  x3          ,& !
  step(3)     ,& ! a random neighbouring cell, out of 26 possibilities
  iteration      ! solidification iteration

! Must be SAVEd to conform
logical( kind=ldef ), save ::                                          &
  yesnuclei [*],   & ! true if nuclei value exists on any image
  finished [*],    & ! true if solidified on this image
  allfinished [*]    ! true if solidified on all images

! .true. if a grain value in coarray exists at least on one image
logical( kind=ldef ) :: allyesnuc

character( len=100 ) :: e_message

! "finished" is calculated by each image and then the value
! is analysed by image 1 to calculate "allfinished".
! Then all other images read "allfinished" from image 1 for
! synchronisation.
!
! Important: when checking for finished, use only the real
! parts of coarray. Do not analyse the virtual (halo) elements!

! get image number and number of images
thisimage = this_image()
numimages = num_images()

!************************************************
! Sanity checks
!************************************************

! check for allocated
if ( .not. allocated( coarray ) ) &
  error stop "ERROR: cgca_sld: coarray is not allocated"

! determine the extents
```

```
lbv = lbound( coarray )
ubv = ubound( coarray )
lbr = lbv + 1
ubr = ubv - 1

! Check for at least one nuclei on this image.
yesnuclei = .false.
if ( any( coarray(lbr(1):ubr(1), lbr(2):ubr(2), lbr(3):ubr(3),         &
                  cgca_state_type_grain) .gt. cgca_liquid_state))      &
  yesnuclei=.true.

! Check if all cells on this image have solidified.
finished = .false.
if ( all( coarray(lbr(1):ubr(1), lbr(2):ubr(2), lbr(3):ubr(3),         &
                  cgca_state_type_grain) .gt. cgca_liquid_state))      &
  finished=.true.

         ! yesnuclei and finished defined
sync all ! new execution segment
         ! yesnuclei and finished used

! Image 1 calculates the global values.
if ( thisimage .eq. 1 ) then

  ! Need at least one nuclei on any image.
  allyesnuc = .false.
  do i = 1 , numimages
    if ( yesnuclei[i] ) then
      allyesnuc = .true.
      exit
    end if
  end do

  ! Need at least one non solidified cell on any image
  allfinished = .true.
  do i = 1 , numimages
    if ( .not. finished[i] ) then
      allfinished=.false.
      exit
    end if
  end do

  ! Report an error if no nuclei found
  if ( .not. allyesnuc ) then
    write (*,'(a)') "ERROR: cgca_sld/cgca_m3sld: no nuclei found"
    error stop
  end if

  ! Report an error if no liquid phase found
  if ( allfinished ) then
    write (*,'(a)') "ERROR: cgca_sld/cgca_m3sld: already solid"
    error stop
```

```
  end if

end if

! All images wait for image 1 here.
sync all

!************************************************
! End of sanity checks. All seems fine, proceed.
!************************************************

! Mark as not solid initially
solid = .false.

! allocate the temp array
allocate( array( lbv(1) : ubv(1) , lbv(2) : ubv(2) , lbv(3) : ubv(3) ),&
          stat = errstat,  errmsg = e_message )
if ( errstat .ne. 0 ) then
  write (*,'(a,i0,a)')                                                 &
    "ERROR: cgca_sld/cgca_m3sld: allocate( array ), err stat: ",       &
     errstat, " err message: " // e_message
  error stop
end if

! initialise the iteration counter
iteration = 1

! at this point allfinished=.false. on image 1 and not
! initialised on all other images!

! start the main loop
main: do

  ! halo exchange,        ===>>> remote comms <<<===
  call cgca_hxi( coarray )
  sync all

  ! global halo exchange, ===>>> remote comms <<<===
  if ( periodicbc ) call cgca_hxg( coarray )
  sync all

  ! do the iteration if not finished
  fini: if ( .not. finished ) then

    ! copy coarray, grain state type, into a local array
    array = coarray( : , : , : , cgca_state_type_grain )

    ! solidify array
    do x3 = lbr(3), ubr(3)
    do x2 = lbr(2), ubr(2)
    do x1 = lbr(1), ubr(1)
      if ( coarray( x1, x2, x3, cgca_state_type_grain ) .eq.           &
```

```
                                        cgca_liquid_state ) then
        call random_number( candidate )     ! 0 .le. candidate .lt. 1
        step = nint( candidate*2 - 1 )      ! step = [-1 0 1]
        array( x1, x2, x3 ) = coarray( x1 + step(1) ,                  &
                                       x2 + step(2) ,                  &
                                       x3 + step(3) ,                  &
                                         cgca_state_type_grain )
      end if
    end do
    end do
    end do

    ! update coarray on this image
    coarray( : , : , : , cgca_state_type_grain ) = array

    ! see if finished (all solid)
    finished = .true.
    if ( any( coarray( lbr(1):ubr(1) , lbr(2):ubr(2) , lbr(3):ubr(3),  &
          cgca_state_type_grain ) .eq. cgca_liquid_state ) ) then
      finished = .false.
    end if

  end if fini

  ! Global sync here. All images must calculate "finished"
  ! before image 1 reads these values from all images
  ! and calculates the global "allfinished".
  sync all

  ! image 1 checks if finished and sends heartbeat signal
  img1: if ( thisimage .eq. 1 ) then

    ! assume we are done
    allfinished = .true.
    do i = 1 , numimages

      ! but if any image is not done, then we are not done either
      if ( .not. finished[i] ) then
        allfinished = .false.
        exit
      end if
    end do

    ! send heartbeat signal to terminal
    if ( heartbeat .gt. 0 ) then
      if ( mod( iteration, heartbeat ) .eq. 0 )                        &
         write (*,'(a,i0)') "INFO: cgca_sld: iterations completed: ",  &
                            iteration
    end if

  end if img1
```

```
            ! allfinished calculated on image 1
  sync all  ! new segment
            ! allfinished used on all images

  ! get allfinished from image 1
  allfinished = allfinished [ 1 ]

  ! exit if done
  if ( allfinished ) then
    solid = .true.
    exit main
  end if

  ! Also exit is the max number of iterations has been reached
  if ( iter .gt. 0 .and. iteration .ge. iter ) exit main

  ! increment the iteration counter
  iteration = iteration + 1

end do main

deallocate( array, stat=errstat )
if ( errstat .ne. 0 ) then
  write( *, '(a,i0)' ) "ERROR: cgca_sld/cgca_m3sld:" //                &
    " deallocate( array ), img: ", thisimage
  error stop
end if

end subroutine cgca_sld
```

31.2 cgca_m3sld/cgca_sld1

[cgca_m3sld] [Subroutines]

NAME

cgca_sld1

SYNOPSIS

```
subroutine cgca_sld1(coarray,iter,heartbeat,solid)
```

INPUTS

```
integer(kind=iarr),allocatable,intent(inout) :: coarray(:,:,:,:)[:,:,:]
integer(kind=idef),intent(in) :: iter,heartbeat
```

OUTPUT

```
logical(kind=ldef),intent(out) :: solid
```

SIDE EFFECTS

State of coarray changed

DESCRIPTION

This is a simplified version of cgca_sld (31.1). Most checks have been removed and sync instances reduced. In addition, it does not support the periodic BC.

Inputs:

- coarray - the model
- iter - number of solidification iterations, if <=0 then do until the coarray has solidified; if >0 then proceed until solid or "iter" iterations have been completed, whichever is sooner
- heartbeat - if >0 then dump a simple message every heartbeat iterations

Outputs:

- solid - .true. if the coarray is fully solid, .false. otherwise

At least one value .ne. cgca_liquid_state (9.18) must exist on at least one image. At least one cgca_liquid_state (9.18) value must exist on at least one image.

NOTES

All images must call this routine!

USES USED BY

none, end user

SOURCE

```
real :: candidate(3)

integer(kind=iarr),allocatable :: array(:,:,:)
integer :: errstat
integer(kind=idef) :: &
  lbv(4)      ,& ! lower bounds of the complete (plus virtual) coarray
  ubv(4)      ,& ! upper bounds of the complete (plus virtual) coarray
  lbr(4)      ,& ! lower bounds of the "real" coarray, lower virtual+1
  ubr(4)      ,& ! upper bounds of the "real" coarray, upper virtual-1
  thisimage   ,& ! to avoid repeated calls to this_image() and num_images()
  nimages     ,& !
  x1,x2,x3    ,& ! local coordinates in an array, which are also
  step(3)     ,& ! a random neighbouring cell, out of 26 possibilities
  iteration      ! solidification iteration

! true if the local array has solidified
logical(kind=ldef) :: finished

! number of finished images
! NOTE: if *really* many images are used, then the kind will need to
! be increased!
integer(kind=idef),allocatable :: nfini[:]

! finished is set to .true. on all images at the beginning of each
! iteration. At the end of the iteration finished[1] is set to .false.
! by *all* images if their local finished is .false..
!
! Important: when checking for finished, use only the real
! parts of coarray. Do not analyse the virtual (halo) elements!

! get image number and number of images
thisimage = this_image()
nimages = num_images()

!************************************************
! Sanity checks
!************************************************

! check for allocated
if (.not. allocated(coarray)) &
  error stop "ERROR: cgca_sld1: coarray is not allocated"

!************************************************
! End of sanity checks. All seems fine, proceed.
!************************************************

! determine the extents
lbv=lbound(coarray)
ubv=ubound(coarray)
lbr=lbv+1
ubr=ubv-1
```

```
! allocate the integer array to store the number of finished images
allocate(nfini[*],stat=errstat)
if (errstat.ne.0) then
 write (*,'(a,i0)') "ERROR: cgca_sld1: image ",thisimage
 write (*,'(a)') "ERROR: cgca_sld1: cannot allocate nfini"
 error stop
end if

! allocate the temp array
allocate(array(lbv(1):ubv(1),lbv(2):ubv(2),lbv(3):ubv(3)),stat=errstat)
if (errstat.ne.0) then
  write (*,'(a,i0)') "ERROR: cgca_sld1: image ",thisimage
  write (*,'(a)') "ERROR: cgca_sld1: cannot allocate array"
  error stop
end if

! Mark as not solid initially.
solid = .false.

! set finished to .false.
finished = .false.

! initialise the iteration counter
iteration=1

! start the main loop
main: do

! set the number of finished images to zero.
nfini = 0

! do if not finished
fini: if (.not. finished) then

! copy coarray, grain state type, into a local array
array=coarray(:,:,:,cgca_state_type_grain)

! solidify array

do x3 = lbr(3),ubr(3)
do x2 = lbr(2),ubr(2)
do x1 = lbr(1),ubr(1)
 if (coarray(x1,x2,x3,cgca_state_type_grain) .eq. cgca_liquid_state) then
  call random_number(candidate)     ! 0 .le. candidate .lt. 1
  step = nint(candidate*2-1)        ! step = [-1 0 1]
  array (x1,x2,x3) = &
   coarray (x1+step(1),x2+step(2),x3+step(3),cgca_state_type_grain)
 end if
end do
end do
end do
```

```
! update coarray
coarray(:,:,:,cgca_state_type_grain) = array

! set finish to .true. if finished (all solid)
finished = all( coarray(lbr(1):ubr(1), lbr(2):ubr(2), lbr(3):ubr(3), &
                  cgca_state_type_grain) .ne. cgca_liquid_state)

end if fini

! Each image, but first, waits for the lower image. Then it sets
! finished[1] to .false. if not finished.
if (thisimage .eq. 1) then
 if ( finished ) nfini = nfini + 1
! write (*,*) "image", thisimage, finished, nfini
else
 sync images (thisimage-1)
 if ( finished ) nfini[1] = nfini[1] + 1
! write (*,*) "image", thisimage, finished, nfini[1]
end if

! each image, but last, waits for the next one
if (thisimage .lt. nimages) sync images (thisimage+1)

! halo exchange
call cgca_hxi(coarray)

! exit if done the required number of iterations
if (iter .gt. 0 .and. iteration .ge. iter) exit main

! increment the iteration counter
iteration = iteration+1

! image 1 sends heartbeat signal
if (thisimage .eq. 1) then
 if (heartbeat .gt. 0) then
   if (mod(iteration,heartbeat) .eq. 0) write (*,'(a,i0)') &
    "INFO: cgca_sld1: iterations completed: ", &
      iteration
 end if
end if

sync all

! exit if finished
if ( nfini[1] .eq. nimages ) then
 solid = .true.
 exit main
end if

end do main

! deallocate all local arrays
```

```
deallocate(array,stat=errstat)
if (errstat.ne.0) then
  write (*,'(a,i0)') "ERROR: cgca_sld1: image ",thisimage
  write (*,'(a)') "ERROR: cgca_sld1: cannot deallocate array"
  error stop
end if

deallocate(nfini,stat=errstat)
if (errstat.ne.0) then
  write (*,'(a,i0)') "ERROR: cgca_sld1: image ",thisimage
  write (*,'(a)') "ERROR: cgca_sld1: cannot deallocate nfini"
  error stop
end if

end subroutine cgca_sld1
```

31.3 cgca_m3sld/cgca_sld2

[cgca_m3sld] [Subroutines]

NAME

cgca_sld2

SYNOPSIS

```
subroutine cgca_sld2(coarray,p,iter,heartbeat,solid)
```

INPUTS

```
integer(kind=iarr),allocatable,intent(inout) :: coarray(:,:,:,:)[:,:,:]
integer(kind=idef),intent(in) :: p,iter,heartbeat
```

OUTPUT

```
logical(kind=ldef),intent(out) :: solid
```

SIDE EFFECTS

State of coarray changed

DESCRIPTION

This is a simplified version of cgca_sld (31.1). Most checks have been removed. cgca_redand (22.1) is called to check that all images have solidified. In addition, it does not support the periodic BC.

Inputs:

- coarray - the model
- p - this routine only works when the number of images is a power of 2. So p is the power: num_images = 2**p. Note that no check for this is made in this routine. This is left up to the calling routine, most probably the main program.
- iter - number of solidification iterations, if <=0 then do until the coarray has solidified; if >0 then proceed until solid or "iter" iterations have been completed, whichever is sooner
- heartbeat - if >0 then dump a simple message every heartbeat iterations

Outputs:

- solid - .true. if the coarray is fully solid, .false. otherwise

At least one value .ne. cgca_liquid_state (9.18) must exist on at least one image. At least one cgca_liquid_state (9.18) value must exist on at least one image.

NOTES

All images must call this routine!

USES

cgca_redand (22.1)

USED BY

none, end user

SOURCE

```
real :: candidate(3)

integer(kind=iarr),allocatable :: array(:,:,:)
integer :: errstat
integer(kind=idef) :: &
  lbv(4)      ,& ! lower bounds of the complete (plus virtual) coarray
  ubv(4)      ,& ! upper bounds of the complete (plus virtual) coarray
  lbr(4)      ,& ! lower bounds of the "real" coarray, lower virtual+1
  ubr(4)      ,& ! upper bounds of the "real" coarray, upper virtual-1
  img,nimages ,& ! to avoid repeated calls to this_image() and num_images()
  x1,x2,x3    ,& ! local coordinates in an array, which are also
  step(3)     ,& ! a random neighbouring cell, out of 26 possibilities
  iteration      ! solidification iteration

! true if the local array has solidified
! It must have SAVE attribute or be ALLOCATABLE. Right now SAVE seems
! faster. The memory freed after this routine, if ALLOCATABLE was used
! instead is insignificant.
logical(kind=ldef),save :: finished[*]

! "finished" is set to .false. on all images at the beginning of
! each iteration. At the end of an iteration, new "finished" value
! is calculated on each image. Then cgca_redall is called. It places
! the result in "finished" on every image. So after that every image
! needs to check just its local "finished". If it is .true., then exit
! the loop. If it is .false., then do another solidification iteration.

! get image number and number of images
img     = this_image()
nimages = num_images()

!************************************************
! Sanity checks
!************************************************

! check for allocated
if (.not. allocated(coarray)) &
  error stop "ERROR: cgca_sld2: coarray is not allocated"

!************************************************
! End of sanity checks. All seems fine, proceed.
!************************************************

! determine the extents
lbv=lbound(coarray)
ubv=ubound(coarray)
lbr=lbv+1
```

```
ubr=ubv-1

! Mark as not solid initially.
solid = .false.

! initialise the iteration counter
iteration=1

! allocate the temp array
! Implicit sync all here
allocate(array(lbv(1):ubv(1),lbv(2):ubv(2),lbv(3):ubv(3)),stat=errstat)
if (errstat.ne.0) then
  write (*,'(a,i0)') "ERROR: cgca_sld2: image ", img
  write (*,'(a)') "ERROR: cgca_sld2: cannot allocate array"
  error stop
end if

! start the main loop
main: do

 ! copy coarray, grain state type, into a local array
 array = coarray(:,:,:,cgca_state_type_grain)

 ! solidify array
 do x3 = lbr(3),ubr(3)
 do x2 = lbr(2),ubr(2)
 do x1 = lbr(1),ubr(1)
  if (coarray(x1,x2,x3,cgca_state_type_grain) .eq. cgca_liquid_state) then
   call random_number(candidate)     ! 0 .le. candidate .lt. 1
   step = nint(candidate*2-1)        ! step = [-1 0 1]
   array (x1,x2,x3) = &
    coarray (x1+step(1),x2+step(2),x3+step(3),cgca_state_type_grain)
  end if
 end do
 end do
 end do

 ! update coarray
 coarray(:,:,:,cgca_state_type_grain) = array

 ! image 1 sends heartbeat signal
 if ( img .eq. 1 ) then
  if ( heartbeat .gt. 0 ) then
   if ( mod(iteration,heartbeat) .eq. 0 ) write (*,'(a,i0)') &
    "INFO: cgca_sld2: iterations completed: ", iteration
  end if
 end if

 ! increment the iteration counter
 iteration = iteration+1

 ! exit if done the required number of iterations
```

```
 if (iter .gt. 0 .and. iteration .ge. iter) exit main

 ! set finish to .true. if finished (all solid)
 finished = all( coarray(lbr(1):ubr(1), lbr(2):ubr(2), lbr(3):ubr(3), &
                 cgca_state_type_grain) .ne. cgca_liquid_state)

 ! halo exchange in preparation for the next iteration
 call cgca_hxi(coarray)

! not sure if I need a global barrier here or not
 sync all

 ! do the collective AND on finished
 call cgca_redand(finished,p)

 ! Now all images will have the updated "finished".
 ! Exit if finished
 if ( finished ) then
  solid = .true.
  exit main
 end if

end do main

! deallocate all local arrays

deallocate(array,stat=errstat)
if (errstat.ne.0) then
  write (*,'(a,i0)') "ERROR: cgca_sld2: image ", img
  write (*,'(a)') "ERROR: cgca_sld2: cannot deallocate array"
  error stop
end if

end subroutine cgca_sld2
```

31.4 cgca_m3sld/m3sld_hc

[cgca_m3sld] [Submodules]

NAME

m3sld_hc

SYNOPSIS

```
!$Id: m3sld_hc.f90 431 2017-06-30 13:13:49Z mexas $

submodule ( cgca_m3sld ) m3sld_hc
```

DESCRIPTION

Submodule with solidification routines which offer a choice of halo exchange routines. Fortran 2015 collectives are used.

NOTES

Not supported by ifort 16.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

CONTAINS

cgca_sld3 (31.5.1)

USES

All date objects of the parent module cgca_m3sld (31) by host association.

USED BY

The parent module cgca_m3sld (31).

SOURCE

```
contains
```

31.4.1 m3sld_hc/cgca_sld_h

[m3sld_hc] [Subroutines]

NAME

cgca_sld_h

SYNOPSIS

```
!    module subroutine cgca_sld_h( coarray, hx, iter, heartbeat, solid )

module procedure cgca_sld_h
```

INPUTS

```
!    See the parent module cgca_m3sld.
```

OUTPUT

```
!    See the parent module cgca_m3sld.
```

SIDE EFFECTS

State of coarray changed

DESCRIPTION

This is a simplified version of cgca_sld (31.1). Most checks have been removed. We use co_sum reduction. The desired halo exchange routine is passed as input. In addition, it does not support the periodic BC.

Inputs:

- coarray - the model
- hx - the halo exchange procedure
- iter - number of solidification iterations, if <=0 then do until the coarray has solidified; if >0 then proceed until solid or "iter" iterations have been completed, whichever is sooner
- heartbeat - if >0 then dump a simple message every heartbeat iterations

Outputs:

- solid - .true. if the coarray is fully solid, .false. otherwise

At least one value .ne. cgca_liquid_state (9.18) must exist on at least one image. At least one cgca_liquid_state (9.18) value must exist on at least one image.

NOTES

All images must call this routine!

USES USED BY

none, end user

SOURCE

```
real :: candidate(3)

integer( kind=iarr ), allocatable :: array(:,:,:)
integer :: errstat
integer( kind=idef ) :: &
  lbv(4)      ,& ! lower bounds of the complete (plus virtual) coarray
  ubv(4)      ,& ! upper bounds of the complete (plus virtual) coarray
  lbr(4)      ,& ! lower bounds of the "real" coarray, lower virtual+1
  ubr(4)      ,& ! upper bounds of the "real" coarray, upper virtual-1
  img,nimages ,& ! to avoid repeated calls to this_image() and num_images()
  x1,x2,x3    ,& ! local coordinates in an array, which are also
  step(3)     ,& ! a random neighbouring cell, out of 26 possibilities
  iteration      ! solidification iteration

! true if the local array has solidified
logical( kind=ldef ) :: finished

character( len=100 ) :: e_message

! finished indicator (findicator) is an integer variable
! to use with CO_SUM.
! It is set to 1 on all images at the beginning of every loop.
! if (finished) findicator = 0.
! Then if co_sum(findicator) is zero, then all images have finished.
integer( kind=idef ), save :: findicator[*]

integer :: flag

! get image number and number of images
    img = this_image()
nimages = num_images()

!************************************************
! Sanity checks
!************************************************

! check for allocated
if ( .not. allocated(coarray) ) then
  write (*,*) "ERROR: cgca_sld_h/m3sld_hc: coarray is not allocated"
  error stop
end if

!************************************************
! End of sanity checks. All seems fine, proceed.
!************************************************

! determine the extents
lbv = lbound( coarray )
ubv = ubound( coarray )
lbr = lbv + 1
ubr = ubv - 1
```

```
! Mark as not solid initially.
solid = .false.

! initialise the iteration counter
iteration = 1

! allocate the temp array
allocate( array( lbv(1):ubv(1) , lbv(2):ubv(2) , lbv(3):ubv(3) ),      &
          source = 0_iarr,                                             &
          stat = errstat, errmsg = e_message )
if ( errstat .ne. 0 ) then
  write (*,'(a,i0,a)')                                                 &
    "ERROR: cgca_sld_h/m3sld_hc: allocate( array ), err stat: ",       &
     errstat, " err message: " // e_message
  error stop
end if

! start the main loop
main: do

 ! copy coarray, grain state type, into a local array
 array = coarray( : , : , : , cgca_state_type_grain )

 ! solidify array
 do x3 = lbr(3),ubr(3)
 do x2 = lbr(2),ubr(2)
 do x1 = lbr(1),ubr(1)
  if (coarray(x1,x2,x3,cgca_state_type_grain) .eq. cgca_liquid_state)  &
    then
      call random_number( candidate ) ! [0..1)
      step = nint( candidate*2-1 )    ! step = [-1 0 1]
      array( x1, x2, x3 ) =                                            &
                     coarray( x1+step(1), x2+step(2), x3+step(3),      &
                                              cgca_state_type_grain )
  end if
 end do
 end do
 end do

 ! update coarray
 coarray( :, :, :, cgca_state_type_grain ) = array

 ! image 1 sends heartbeat signal
 if ( img .eq. 1 ) then
  if ( heartbeat .gt. 0 ) then
   if ( mod(iteration,heartbeat) .eq. 0 ) write (*,'(a,i0)')           &
    "INFO: cgca_sld_h/m3sld_hc: iterations completed: ", iteration
  end if
 end if

 ! increment the iteration counter
 iteration = iteration + 1
```

```
 ! exit if done the required number of iterations
 if (iter .gt. 0 .and. iteration .gt. iter) exit main

 ! set finish to .true. if finished (all solid)
 finished = all( coarray(lbr(1):ubr(1), lbr(2):ubr(2), lbr(3):ubr(3),  &
                 cgca_state_type_grain) .ne. cgca_liquid_state)
 findicator = 1_idef
 if (finished) findicator = 0_idef

 sync all

 ! halo exchange in preparation for the next iteration
 ! no sync inside
 call hx( coarray )

 sync all

! Check hx, flag .eq. 0 means ok, flag .ne. 0 is an error.
call cgca_hxic( coarray, flag )
call co_sum( flag )
if ( img .eq. 1 ) then
  if ( flag .ne. 0 ) then
    write (*,'(a,i0)') "ERROR: cgca_sld_h/m3sld_hc: hx flag: ", flag
    error stop
  end if
end if

 ! do the collective AND on finished
 call co_sum( findicator )

 ! Now all images will have the updated findicator.
 ! Exit if finished
 if ( findicator .eq. 0 ) then
  solid = .true.
  exit main
 end if

end do main

! deallocate all local arrays

deallocate( array, stat=errstat, errmsg = e_message )
if ( errstat .ne. 0 ) then
  write (*,'(a,i0,a)')                                                 &
    "ERROR: cgca_sld_h/m3sld_hc: deallocate( array ), err stat: ",     &
     errstat, " err message: " // e_message
  error stop
end if

end procedure cgca_sld_h
```

31.5 cgca_m3sld/m3sld_sm1

[cgca_m3sld] [Submodules]

NAME

m3sld_sm1

SYNOPSIS

```
!$Id: m3sld_sm1.f90 400 2017-05-04 17:47:56Z mexas $

submodule ( cgca_m3sld ) m3sld_sm1
```

DESCRIPTION

Submodule with solidification routines using collectives. These are not supported by ifort 16 yet.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

CONTAINS

cgca_sld3 (31.5.1)

USES

All variables and parameters of the parent module cgca_m3sld (31).

USED BY

The parent module cgca_m3sld (31).

SOURCE

```
contains
```

31.5.1 m3sld_sm1/cgca_sld3

[m3sld_sm1] [Subroutines]

NAME

cgca_sld3

SYNOPSIS

```
!module procedure cgca_sld3
  module subroutine cgca_sld3( coarray, iter, heartbeat, solid )
    integer( kind=iarr ), allocatable, intent( inout ) ::              &
      coarray(:,:,:,:)[:,:,:]
    integer( kind=idef ), intent( in ) :: iter,heartbeat
    logical( kind=ldef ), intent( out ) :: solid
```

INPUTS

```
!    See the parent module cgca_m3sld.
```

OUTPUT

```
!    See the parent module cgca_m3sld.
```

SIDE EFFECTS

State of coarray changed

DESCRIPTION

This is a simplified version of cgca_sld (31.1). Most checks have been removed. We use co_sum reduction. In addition, it does not support the periodic BC.

Inputs:

- coarray - the model
- iter - number of solidification iterations, if <=0 then do until the coarray has solidified; if >0 then proceed until solid or "iter" iterations have been completed, whichever is sooner
- heartbeat - if >0 then dump a simple message every heartbeat iterations

Outputs:

- solid - .true. if the coarray is fully solid, .false. otherwise

At least one value .ne. cgca_liquid_state (9.18) must exist on at least one image. At least one cgca_liquid_state (9.18) value must exist on at least one image.

NOTES

All images must call this routine!

USES USED BY

none, end user

SOURCE

```
real :: candidate(3)

integer( kind=iarr ), allocatable :: array(:,:,:)
integer :: errstat
integer( kind=idef ) :: &
  lbv(4)      ,& ! lower bounds of the complete (plus virtual) coarray
  ubv(4)      ,& ! upper bounds of the complete (plus virtual) coarray
  lbr(4)      ,& ! lower bounds of the "real" coarray, lower virtual+1
  ubr(4)      ,& ! upper bounds of the "real" coarray, upper virtual-1
  img,nimages ,& ! to avoid repeated calls to this_image() and num_images()
  x1,x2,x3    ,& ! local coordinates in an array, which are also
  step(3)     ,& ! a random neighbouring cell, out of 26 possibilities
  iteration      ! solidification iteration

! true if the local array has solidified
logical( kind=ldef ) :: finished

character( len=100 ) :: e_message

! finished indicator (findicator) is an integer variable to use with CO_SUM.
! It is set to 1 on all images at the beginning of every loop.
! if (finished) findicator = 0.
! Then if co_sum(findicator) is zero, then all images have finished.
integer( kind=idef ),save :: findicator[*]

! get image number and number of images
    img = this_image()
nimages = num_images()

!************************************************
! Sanity checks
!************************************************

! check for allocated
if ( .not. allocated(coarray) ) &
  error stop "ERROR: cgca_sld3: coarray is not allocated"

!************************************************
! End of sanity checks. All seems fine, proceed.
!************************************************

! determine the extents
lbv = lbound( coarray )
ubv = ubound( coarray )
lbr = lbv + 1
ubr = ubv - 1

! Mark as not solid initially.
solid = .false.

! initialise the iteration counter
iteration = 1
```

```
! allocate the temp array
allocate( array( lbv(1):ubv(1) , lbv(2):ubv(2) , lbv(3):ubv(3) ),      &
          source = 0_iarr,                                              &
          stat = errstat, errmsg = e_message )
if ( errstat .ne. 0 ) then
  write (*,'(a,i0,a)')                                                  &
    "ERROR: cgca_sld3/m3sld_sm1: allocate( array ), err stat: ",        &
     errstat, " err message: " // e_message
  error stop
end if

! start the main loop
main: do

 ! copy coarray, grain state type, into a local array
 array = coarray( : , : , : , cgca_state_type_grain )

 ! solidify array
 do x3 = lbr(3),ubr(3)
 do x2 = lbr(2),ubr(2)
 do x1 = lbr(1),ubr(1)
  if (coarray(x1,x2,x3,cgca_state_type_grain) .eq. cgca_liquid_state) then
   call random_number(candidate)     ! 0 .le. candidate .lt. 1
   step = nint(candidate*2-1)        ! step = [-1 0 1]
   array (x1,x2,x3) = &
    coarray (x1+step(1),x2+step(2),x3+step(3),cgca_state_type_grain)
  end if
 end do
 end do
 end do

 ! update coarray
 coarray(:,:,:,cgca_state_type_grain) = array

 ! image 1 sends heartbeat signal
 if ( img .eq. 1 ) then
  if ( heartbeat .gt. 0 ) then
   if ( mod(iteration,heartbeat) .eq. 0 ) write (*,'(a,i0)') &
    "INFO: cgca_sld3: iterations completed: ", iteration
  end if
 end if

 ! increment the iteration counter
 iteration = iteration + 1

 ! exit if done the required number of iterations
 if (iter .gt. 0 .and. iteration .ge. iter) exit main

 ! set finish to .true. if finished (all solid)
 finished = all( coarray(lbr(1):ubr(1), lbr(2):ubr(2), lbr(3):ubr(3), &
                 cgca_state_type_grain) .ne. cgca_liquid_state)
```

```
 findicator = 1_idef
 if (finished) findicator = 0_idef


 ! halo exchange in preparation for the next iteration
 call cgca_hxi(coarray)

 ! do the collective AND on finished
 call co_sum(findicator)

 ! Now all images will have the updated findicator.
 ! Exit if finished
 if ( findicator .eq. 0 ) then
  solid = .true.
  exit main
 end if

end do main

! deallocate all local arrays

deallocate(array,stat=errstat)
if (errstat.ne.0) then
  write (*,'(a,i0)') "ERROR: cgca_sld3: image ", img
  write (*,'(a)') "ERROR: cgca_sld3: cannot deallocate array"
  error stop
end if

!end procedure cgca_sld3
end subroutine cgca_sld3
```

32 CGPACK/cgca_m4fr

[Modules]

NAME

cgca_m4fr

SYNOPSIS

```
!$Id: cgca_m4fr.f90 400 2017-05-04 17:47:56Z mexas $

module cgca_m4fr
```

DESCRIPTION

Module dealing with fracture

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

CONTAINS USES

cgca_m1co (9), cgca_m2gb (11), cgca_

USED BY SOURCE

```
use cgca_m1co
use cgca_m2gb
use cgca_m3clvg
use cgca_m3gbf
implicit none
private

contains
```

32.1 cgca_m4fr/cgca_fr

[cgca_m4fr] [Subroutines]

NAME

cgca_fr

SYNOPSIS

```
subroutine cgca_fr( coarray, rt, s1, scrit, periodicbc, iter, &
 heartbeat, debug )
```

INPUTS

```
integer( kind=iarr ), allocatable, intent( inout ) :: &
  coarray(:,:,:,:)[:,:,:]
real( kind=rdef ), allocatable, intent(inout) :: rt(:,:,:)[:,:,:]
real( kind=rdef ), intent(in) :: s1(3), scrit(3)
logical(kind=ldef),intent(in) :: periodicbc
integer(kind=idef),intent(in) :: iter,heartbeat
logical(kind=ldef),intent(in) :: debug
```

SIDE EFFECTS

state of coarray changes

DESCRIPTION

This routine does one iteration of cleavage propagation, followed by an iteration of grain boundary fracture. It does the halo exchange when required. Then it repeats this cycle for the given number of iterations, "iter". Do not check coarray for allocated, as this wastes time. Instead let the code fail if coarray is not allocated. determine the extents dummy code to suppress the warnings debug

33 CGPACK/LICENSE

[License]

NAME

LICENSE

SYNOPSIS

```
!     Unless explicitly specified otherwise, all files in the CGPACK
!     library are covered by 2-clause BSD LICENSE.
!     The full LICENSE text is below.
```

DESCRIPTION

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

34 CGPACK/Makefile-bc3-ifort-shared

[Make files]

NAME

Makefile-bc3-ifort-shared

SYNOPSIS

```
#$Id: Makefile-bc3-ifort-shared 382 2017-03-22 11:41:51Z mexas $

FC=             ifort
```

PURPOSE

Build/install CGPACK on the University of Bristol BlueCrystal computer with Intel Fortran compiler.

DESCRIPTION

This makefile is to build CGPACK with ifort for shared memory!

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

SOURCE

```
FFLAGS=         -c -assume realloc_lhs -coarray=shared \
                -traceback \
                -free -fPIC -warn all -O2 -qopt-report #-std08 #-warn stderrors
#               -coarray-config-file=xx14.conf -debug full \

CGPACK=         cgpack
MYLIB=          lib$(CGPACK).a
LIBDIR=         $(HOME)/lib
MODDIR=         $(HOME)/mod

# level 1

L1_SRC=         cgca_m1co.f90
L1_MOD=         $(L1_SRC:.f90=.mod)
L1_OBJ=         $(L1_SRC:.f90=.o)

# level 2 modules and submodules

L2_SRC_MOD=     cgca_m2alloc.f90 cgca_m2gb.f90 cgca_m2geom.f90 \
                cgca_m2glm.f90 cgca_m2hx.f90 cgca_m2lnklst.f90 \
                cgca_m2out.f90 cgca_m2pck.f90 \
                cgca_m2phys.f90 cgca_m2red.f90 cgca_m2rnd.f90 \
                cgca_m2rot.f90 cgca_m2stat.f90
L2_SRC_SUBMOD=  # m2out_sm1.f90 - Cray only
                # m2out_sm2_mpi.f90 - MPI only
```

```
L2_SRC=         $(L2_SRC_MOD) $(L2_SRC_SUBMOD)
L2_OBJ=         $(L2_SRC:.f90=.o)
L2_MOD=         $(L2_SRC_MOD:.f90=.mod)
L2_SUBMOD=      # cgca_m2out@m2out_sm1.smod - Cray only
                # cgca_m2out@m2out_sm2_mpi.smod - MPI only

# level 3 modules and submodules

L3_SRC_MOD=     cgca_m3clvg.f90 cgca_m3gbf.f90 cgca_m3nucl.f90 \
                cgca_m3pfem.f90 cgca_m3sld.f90
L3_SRC_SUBMOD=  m3clvg_sm1.f90 m3clvg_sm2.f90 m3sld_sm1.f90
                # m3clvg_sm3.f90 - uses collectives, not for ifort 16.0.0
                # m3pfem_sm1.f90 - uses collectives, not for ifort 16.0.0
L3_SRC=         $(L3_SRC_MOD) $(L3_SRC_SUBMOD)
L3_OBJ=         $(L3_SRC:.f90=.o)
L3_MOD=         $(L3_SRC_MOD:.f90=.mod)
L3_SUBMOD=      cgca_m3clvg@m3clvg_sm1.smod \
                cgca_m3clvg@m3clvg_sm2.smod \
                cgca_m3sld@m3sld_sm1.smod
                # cgca_m3clvg@m3clvg_sm3.smod

# level 4

L4_SRC=         cgca_m4fr.f90
L4_MOD=         $(L4_SRC:.f90=.mod)
L4_OBJ=         $(L4_SRC:.f90=.o)

# top level

LTOP_SRC=       cgca.f90
LTOP_MOD=       $(LTOP_SRC:.f90=.mod)
LTOP_OBJ=       $(LTOP_SRC:.f90=.o)

ALL_MOD=        $(L1_MOD) $(L2_MOD) $(L3_MOD) $(L4_MOD) $(LTOP_MOD)
ALL_SUBMOD=     $(L2_SUBMOD) $(L3_SUBMOD)
ALL_OBJ=        $(L1_OBJ) $(L2_OBJ) $(L3_OBJ) $(L4_OBJ) $(LTOP_OBJ)

ALL_CLEAN=      *.mod *.smod *.o *.optrpt $(MYLIB)

.SUFFIXES:
.SUFFIXES: .f90 .mod .o

all: $(MYLIB)

.f90.mod:
        $(FC) $(FFLAGS) $<

.f90.o:
        $(FC) $(FFLAGS) $<

# module dependencies
```

```
$(L2_MOD) $(L2_OBJ): $(L1_MOD)
$(L3_MOD) $(L3_OBJ): $(L2_MOD) $(L2_SUBMOD)
$(L4_MOD) $(L4_OBJ): $(L3_MOD) $(L3_SUBMOD)
$(LTOP_MOD) $(LTOP_OBJ): $(L4_MOD)
$(MYLIB): $(MOD_LTOP) $(OBJ_LTOP)

# Submodule dependencies
# level 2
m2out_sm1.o: cgca_m2out.mod cgca_m2out.o
# level 3
m3clvg_sm1.o m3clvg_sm2.o: cgca_m3clvg.mod cgca_m3clvg.o
m3sld_sm1.o: cgca_m3sld.mod cgca_m3sld.o

$(MYLIB): $(ALL_OBJ)
        @if [ -e $(MYLIB) ]; then \
                rm $(MYLIB); \
        fi
        ar -r $(MYLIB) $(ALL_OBJ)

install: $(MYLIB) $(ALL_MOD) $(ALL_SUBMOD)
        @if [ -e $(LIBDIR)/$(MYLIB) ]; then \
                echo $(LIBDIR)/$(MYLIB) already exists; \
                echo run \"make deinstall\" first; \
                exit 1; \
        fi
        cp $(MYLIB) $(LIBDIR)
        cp $(ALL_MOD) $(ALL_SUBMOD) $(MODDIR)

deinstall:
        cd $(LIBDIR) && rm $(MYLIB)
        cd $(MODDIR) && rm $(ALL_MOD) $(ALL_SUBMOD)

clean:
        rm $(ALL_CLEAN)
```

35 CGPACK/Makefile-bc3-mpiifort-tau

[Make files]

NAME

Makefile-bc3-mpiifort-tau

SYNOPSIS

```
#$Id: Makefile-bc3-mpiifort-tau 382 2017-03-22 11:41:51Z mexas $

FC=             tau_f90.sh
```

PURPOSE

Build/install CGPACK instrumented for profiling/tracing with TAU on the University of Bristol BlueCrystal computer with Intel Fortran compiler (mpiifort).

NOTES

This makefile is to build CGPACK for use with ParaFEM. According to Intel, when mixing MPI with coarrays, the easiest approach is to build the coarray library with "-coarray=single", and let the MPI parts of the program set up the MPI environment. See also: https://software.intel.com/en-us/forums/topic/559446

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

SOURCE

```
#TAU_MAKEFILE=  $(HOME)/tau-2.25.1.1/x86_64/lib/Makefile.tau-icpc-papi-mpi-pdt
TAU_MAKEFILE=   $(HOME)/tau-2.25.2-intel/x86_64/lib/Makefile.tau-icpc-papi-mpi-pdt
include $(TAU_MAKEFILE)

FFLAGS=         -c -qopt-report -assume realloc_lhs \
                -O2 -debug full -g -traceback -free -warn \
                -coarray=distributed -coarray-config-file=xx14.conf \
                $(TAU_INCLUDE) $(TAU_MPI_INCLUDE)
#-std08 -warn stderrors -fast

CGPACK=         cgpack
MYLIB=          lib$(CGPACK).a
LIBDIR=         $(HOME)/lib
MODDIR=         $(HOME)/mod

# level 1

L1_SRC=         cgca_m1co.f90
L1_MOD=         $(L1_SRC:.f90=.mod)
L1_OBJ=         $(L1_SRC:.f90=.o)
```

```
# level 2 modules and submodules

L2_SRC_MOD=     cgca_m2alloc.f90 cgca_m2gb.f90 cgca_m2geom.f90 \
                cgca_m2glm.f90 cgca_m2hx.f90 cgca_m2lnklst.f90 \
                cgca_m2out.f90 cgca_m2pck.f90 \
                cgca_m2phys.f90 cgca_m2red.f90 cgca_m2rnd.f90 \
                cgca_m2rot.f90 cgca_m2stat.f90 cgca_m2mpiio.f90
L2_SRC_SUBMOD=  m2out_sm2_mpi.f90 # m2out_sm1.f90 - only for Cray
L2_SRC=         $(L2_SRC_MOD) $(L2_SRC_SUBMOD)
L2_OBJ=         $(L2_SRC:.f90=.o)
L2_MOD=         $(L2_SRC_MOD:.f90=.mod)
L2_SUBMOD=      cgca_m2out@m2out_sm2_mpi.smod
                # cgca_m2out@m2out_sm1.smod - only for Cray

# level 3 modules and submodules

L3_SRC_MOD=     cgca_m3clvg.f90 cgca_m3gbf.f90 cgca_m3nucl.f90 \
                cgca_m3pfem.f90 cgca_m3sld.f90
L3_SRC_SUBMOD=  m3clvg_sm1.f90 m3clvg_sm2.f90 m3sld_sm1.f90
# don't build m3clvg_sm3.f90 with ifort 16.0.0 - uses collectives
# don't build m3pfem_sm1.f90 with ifort 16.0.0 - uses collectives
L3_SRC=         $(L3_SRC_MOD) $(L3_SRC_SUBMOD)
L3_OBJ=         $(L3_SRC:.f90=.o)
L3_MOD=         $(L3_SRC_MOD:.f90=.mod)
L3_SUBMOD=      cgca_m3clvg@m3clvg_sm1.smod \
                cgca_m3clvg@m3clvg_sm2.smod \
                cgca_m3sld@m3sld_sm1.smod

# level 4

L4_SRC=         cgca_m4fr.f90
L4_MOD=         $(L4_SRC:.f90=.mod)
L4_OBJ=         $(L4_SRC:.f90=.o)

# top level

LTOP_SRC=       cgca.f90
LTOP_MOD=       $(LTOP_SRC:.f90=.mod)
LTOP_OBJ=       $(LTOP_SRC:.f90=.o)

ALL_MOD=        $(L1_MOD) $(L2_MOD) $(L3_MOD) $(L4_MOD) $(LTOP_MOD)
ALL_SUBMOD=     $(L2_SUBMOD) $(L3_SUBMOD)
ALL_OBJ=        $(L1_OBJ) $(L2_OBJ) $(L3_OBJ) $(L4_OBJ) $(LTOP_OBJ)

ALL_CLEAN=      *.mod *.smod *.o *.optrpt $(MYLIB)

.SUFFIXES:
.SUFFIXES: .f90 .mod .o

all: $(MYLIB)

.f90.mod:
```

```
	$(FC) $(FFLAGS) $<

.f90.o:
	$(FC) $(FFLAGS) $<

# module dependencies

$(L2_MOD) $(L2_OBJ): $(L1_MOD)
$(L3_MOD) $(L3_OBJ): $(L2_MOD) $(L2_SUBMOD)
$(L4_MOD) $(L4_OBJ): $(L3_MOD) $(L3_SUBMOD)
$(LTOP_MOD) $(LTOP_OBJ): $(L4_MOD)
$(MYLIB): $(MOD_LTOP) $(OBJ_LTOP)

# Submodule dependencies
# level 2
m2out_sm1.o m2out_sm2_mpi.o: cgca_m2out.mod cgca_m2out.o

# level 3
m3clvg_sm1.o m3clvg_sm2.o: cgca_m3clvg.mod cgca_m3clvg.o
m3sld_sm1.o: cgca_m3sld.mod cgca_m3sld.o

$(MYLIB): $(ALL_OBJ)
	@if [ -e $(MYLIB) ]; then \
		rm $(MYLIB); \
	fi
	ar -r $(MYLIB) $(ALL_OBJ)

install: $(MYLIB) $(ALL_MOD) $(ALL_SUBMOD)
	@if [ -e $(LIBDIR)/$(MYLIB) ]; then \
		echo $(LIBDIR)/$(MYLIB) already exists; \
		echo run \"make deinstall\" first; \
		exit 1; \
	fi
	cp $(MYLIB) $(LIBDIR)
	cp $(ALL_MOD) $(ALL_SUBMOD) $(MODDIR)

deinstall:
	cd $(LIBDIR) && rm $(MYLIB)
	cd $(MODDIR) && rm $(ALL_MOD) $(ALL_SUBMOD)

clean:
	rm $(ALL_CLEAN)
```

36 CGPACK/Makefile-bc3-oca

[Make files]

NAME

Makefile-bc3-oca

SYNOPSIS

```
#$Id: Makefile-opencoarrays 446 2018-01-12 16:37:09Z mexas $

FC=             caf
```

PURPOSE

Build/install CGPACK with GCC/OpenCoarrays

NOTES

CGPACK uses parallel HDF5, NetCDF and NetCDF-Fortran libraries. Make sure these are installed. Either the include path must point to the location of their *mod files, or copy the *mod files into another convenient location. I do the latter. Some parts of CGPACK, in particular coarrays of derived type with pointer or allocatable components, require gcc7+. So need to build lang/opencoarrays with gcc7+. lang/opencoarrays can use either of 3 MPI ports:

```
net/mpich net/openmpi net/openmpi2
```

net/mpich is the default. The 2 openmpi ports have not been integrated fully with opencoarrays yet. Also need to rebuild a number of other ports:

```
science/hdf5 science/netcdf science/netcdf-fortran
```

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

SOURCE

```
FFLAGS=         -c -O2 -Wall -fPIC -g -fbacktrace -fall-intrinsics \
                -fcheck-array-temporaries \
                -I$(HOME)/mod

CGPACK=         cgpack
MYLIB=          lib$(CGPACK).a
LIBDIR=         $(HOME)/lib
MODDIR=         $(HOME)/mod

SRC=            cgca_m1clock.f90 cgca_m1co.f90 cgca_m2alloc.f90 cgca_m2gb.f90 \
                cgca_m2geom.f90 cgca_m2glm.f90 cgca_m2hdf5.f90 cgca_m2hx.f90 \
                cgca_m2lnklst.f90 cgca_m2mpiio.f90 cgca_m2netcdf.f90 cgca_m2out.f90 \
                cgca_m2pck.f90 cgca_m2phys.f90 cgca_m2red.f90 cgca_m2rnd.f90 \
```

```
                cgca_m2rot.f90 cgca_m2stat.f90 cgca_m3clvg.f90 cgca_m3gbf.f90 \
                cgca_m3nucl.f90 cgca_m3pfem.f90 cgca_m3sld.f90 cgca_m4fr.f90 \
                m2out_sm2_mpi.f90 m3clvg_sm1.f90 m3clvg_sm2.f90 m3clvg_sm3.f90 \
                m3sld_sm1.f90 \
                cgca.f90
                # cgca_m3clvgt.f90 # - broken, does not build
                # m2out_sm1.f90 # - Cray only
                # m3clvgt_sm1.f90 # - broken, does not build
                # m3pfem_sm1.f90 # - GCC7 ICE
OBJ=            $(SRC:.f90=.o)
MOD=            cgca*.mod
SMOD=           cgca*.smod

.SUFFIXES:
.SUFFIXES:      .f90 .o

all:            $(OBJ) $(MYLIB)

.f90.o:
                $(FC) $(FFLAGS) $<

$(MYLIB):       $(OBJ)
                ar -r $(MYLIB) $(OBJ)

install:        $(MYLIB)
                cp $(MYLIB) $(LIBDIR)
                cp cgca.mod $(MODDIR)

deinstall:
                cd $(LIBDIR) && rm $(MYLIB)
                cd $(MODDIR) && rm cgca.mod

clean:
                rm -f $(MOD) $(SMOD) $(OBJ) $(MYLIB)
```

37 CGPACK/Makefile-ifort

[Make files]

NAME

Makefile-ifort

SYNOPSIS

```
#$Id: Makefile-ifort 525 2018-03-19 21:54:26Z mexas $

FC=             ifort
```

PURPOSE

Build/install CGPACK on the University of Bristol BlueCrystal computer with Intel Fortran compiler.

NOTES

CGPACK uses parallel HDF5, NetCDF and NetCDF-Fortran libraries. Make sure these are installed. Either the include path must point to the location of their *mod files, or copy the *mod files into another convenient location. I do the first here.

According to Intel, when mixing MPI with coarrays, the easiest approach is to build the coarray library with "-coarray=single", and let the MPI parts of the program set up the MPI environment. See also: https://software.intel.com/en-us/forums/topic/559446

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

SOURCE

```
FFLAGS=         -c -assume realloc_lhs -free -fPIC \
                -debug full -traceback -warn all -O2 -qopt-report \
                -coarray=distributed
# -coarray-config-file=xx14.conf \
#               -I$(HOME)/soft/hdf5-1.10.1-ifort16u2-install/include \
#               -I$(HOME)/soft/netcdf-fortran-4.4.4-ifort-install/include
#-std08 #-warn stderrors

CGPACK=         cgpack
MYLIB=          lib$(CGPACK).a
LIBDIR=         $(HOME)/lib
MODDIR=         $(HOME)/include

SRC=            cgca_m1clock.f90 cgca_m1co.f90 cgca_m2alloc.f90 cgca_m2gb.f90 \
                cgca_m2geom.f90 cgca_m2glm.f90 cgca_m2hx.f90 \
                cgca_m2lnklst.f90 cgca_m2mpiio.f90 \
                cgca_m2out.f90 \
                cgca_m2pck.f90 cgca_m2phys.f90 cgca_m2red.f90 cgca_m2rnd.f90 \
                cgca_m2rot.f90 cgca_m2stat.f90 \
                cgca_m3clvg.f90 cgca_m3gbf.f90 \
```

```
                cgca_m3nucl.f90 cgca_m3pfem.f90 cgca_m3sld.f90 cgca_m4fr.f90 \
                m2hx_hxir.f90 \
 m3clvg_sm1.f90 m3clvg_sm2.f90 m3clvg_sm3.f90 \
                m3sld_sm1.f90 m3sld_hc.f90 m3pfem_sm1.f90 \
                cgca.f90
#               m2out_sm2_mpi.f90 - MPI routines
# cgca_m2netcdf.f90 \
#cgca_m2hdf5.f90
                # cgca_m3clvgt.f90 # - broken, does not build
                # m2out_sm1.f90 # - Cray only
                # m3clvgt_sm1.f90 # - broken, does not build
OBJ=            $(SRC:.f90=.o)
OPTRPT=         $(SRC:.f90=.optrpt)
MOD=            cgca*.mod
SMOD=           cgca*.smod

.SUFFIXES:
.SUFFIXES:      .f90 .o

all:            $(OBJ) $(MYLIB)

.f90.o:
        $(FC) $(FFLAGS) $<

$(MYLIB): $(OBJ)
        ar -r $(MYLIB) $(OBJ)

install: $(MYLIB)
        cp $(MYLIB) $(LIBDIR)
        cp $(MOD) $(SMOD) $(MODDIR)

deinstall:
        cd $(LIBDIR) && rm $(MYLIB)
        cd $(MODDIR) && rm $(MOD) $(SMOD)

clean:
        rm $(MOD) $(SMOD) $(OPTRPT) $(OBJ) $(MYLIB)
```

38 CGPACK/Makefile-mpiifort

[Make files]

NAME

Makefile-mpiifort

SYNOPSIS

```
#$Id: Makefile-mpiifort 520 2018-03-13 18:02:06Z mexas $

FC=             mpiifort
```

PURPOSE

Build/install CGPACK on the University of Bristol BlueCrystal computer with Intel Fortran compiler.

NOTES

CGPACK uses parallel HDF5, NetCDF and NetCDF-Fortran libraries. Make sure these are installed. Either the include path must point to the location of their *mod files, or copy the *mod files into another convenient location. I do the first here.

According to Intel, when mixing MPI with coarrays, the easiest approach is to build the coarray library with "-coarray=single", and let the MPI parts of the program set up the MPI environment. See also: https://software.intel.com/en-us/forums/topic/559446

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

SOURCE

```
# On BCp4 load these modules:
# languages/intel/2017.01 libs/netcdf/4.4.1.1.MPI libs/hdf5/1.10.1.MPI

FFLAGS=         -c -assume realloc_lhs -free -fPIC \
                -debug full -traceback -warn all -O2 -qopt-report \
                -coarray=distributed -coarray-config-file=xx14.conf \
                -I/mnt/storage/software/libraries/intel/hdf5-1.10.1-mpi/include \
                -I/mnt/storage/software/libraries/intel/netcdf-4.4.1.1-mpi/include
#               -I$(HOME)/soft/hdf5-1.10.1-ifort16u2-install/include \
#               -I$(HOME)/soft/netcdf-fortran-4.4.4-ifort-install/include
#-std08 #-warn stderrors

CGPACK=         cgpack
MYLIB=          lib$(CGPACK).a
LIBDIR=         $(HOME)/lib
MODDIR=         $(HOME)/include

SRC=            cgca_m1clock.f90 cgca_m1co.f90 \
                ca_hx.f90 ca_hx_mpi.f90 ca_hx_co.f90 \
                cgca_m2alloc.f90 cgca_m2gb.f90 \
```

```
                cgca_m2geom.f90 cgca_m2glm.f90 cgca_m2hx.f90 \
                cgca_m2lnklst.f90 cgca_m2mpiio.f90 \
                cgca_m2out.f90 \
                cgca_m2pck.f90 cgca_m2phys.f90 cgca_m2red.f90 cgca_m2rnd.f90 \
                cgca_m2rot.f90 cgca_m2stat.f90 \
                cgca_m3clvg.f90 cgca_m3gbf.f90 \
                cgca_m3nucl.f90 cgca_m3pfem.f90 cgca_m3sld.f90 cgca_m4fr.f90 \
                m2hx_hxir.f90 \
                cgca_m2netcdf.f90 cgca_m2hdf5.f90 \
                m2out_sm2_mpi.f90 m3clvg_sm1.f90 m3clvg_sm2.f90 m3clvg_sm3.f90 \
                m3sld_sm1.f90 m3sld_hc.f90 m3pfem_sm1.f90 \
                cgca.f90
OBJ=            $(SRC:.f90=.o)
OPTRPT=         $(SRC:.f90=.optrpt)
MOD=            cgca*.mod ca*.mod
SMOD=           cgca*.smod

                # cgca_m3clvgt.f90 # - broken, does not build
                # m2out_sm1.f90 # - Cray only
                # m3clvgt_sm1.f90 # - broken, does not build

.SUFFIXES:
.SUFFIXES:      .f90 .o

all:            $(OBJ) $(MYLIB)

.f90.o:
                $(FC) $(FFLAGS) $<

$(MYLIB):       $(OBJ)
                ar -r $(MYLIB) $(OBJ)

install:        $(MYLIB)
                cp $(MYLIB) $(LIBDIR)
                cp $(MOD) $(SMOD) $(MODDIR)

deinstall:
                cd $(LIBDIR) && rm $(MYLIB)
                cd $(MODDIR) && rm $(MOD) $(SMOD)

clean:
                rm -f $(MOD) $(SMOD) $(OPTRPT) $(OBJ) $(MYLIB)
```

39 CGPACK/Makefile-mpiifort-scorep

[Make files]

NAME

Makefile-mpiifort-scorep

SYNOPSIS

```
#$Id: Makefile-mpiifort-scorep 382 2017-03-22 11:41:51Z mexas $

FC=             scorep mpiifort
```

PURPOSE

Build/install CGPACK instrumented for profiling/tracing with TAU on the University of Bristol BlueCrystal computer with Intel Fortran compiler (mpiifort).

NOTES

This makefile is to build CGPACK for use with ParaFEM. According to Intel, when mixing MPI with coarrays, the easiest approach is to build the coarray library with "-coarray=single", and let the MPI parts of the program set up the MPI environment. See also: https://software.intel.com/en-us/forums/topic/559446

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

SOURCE

```
FFLAGS=         -c -qopt-report -assume realloc_lhs \
                -O2 -debug full -g -traceback -free -warn \
                -coarray=distributed -coarray-config-file=xx14.conf
#-std08 -warn stderrors -fast

CGPACK=         cgpack
MYLIB=          lib$(CGPACK).a
LIBDIR=         $(HOME)/lib
MODDIR=         $(HOME)/mod

# level 1

L1_SRC=         cgca_m1co.f90
L1_MOD=         $(L1_SRC:.f90=.mod)
L1_OBJ=         $(L1_SRC:.f90=.o)

# level 2 modules and submodules

L2_SRC_MOD=     cgca_m2alloc.f90 cgca_m2gb.f90 cgca_m2geom.f90 \
                cgca_m2glm.f90 cgca_m2hx.f90 cgca_m2lnklst.f90 \
                cgca_m2out.f90 cgca_m2pck.f90 \
```

```
               cgca_m2phys.f90 cgca_m2red.f90 cgca_m2rnd.f90 \
               cgca_m2rot.f90 cgca_m2stat.f90
L2_SRC_SUBMOD=  m2out_sm2_mpi.f90 # m2out_sm1.f90 - only for Cray
L2_SRC=         $(L2_SRC_MOD) $(L2_SRC_SUBMOD)
L2_OBJ=         $(L2_SRC:.f90=.o)
L2_MOD=         $(L2_SRC_MOD:.f90=.mod)
L2_SUBMOD=      cgca_m2out@m2out_sm2_mpi.smod
               # cgca_m2out@m2out_sm1.smod - only for Cray

# level 3 modules and submodules

L3_SRC_MOD=     cgca_m3clvg.f90 cgca_m3gbf.f90 cgca_m3nucl.f90 \
               cgca_m3pfem.f90 cgca_m3sld.f90
L3_SRC_SUBMOD=  m3clvg_sm1.f90 m3clvg_sm2.f90 m3sld_sm1.f90
# don't build m3clvg_sm3.f90 with ifort 16.0.0 - uses collectives
# don't build m3pfem_sm1.f90 with ifort 16.0.0 - uses collectives
L3_SRC=         $(L3_SRC_MOD) $(L3_SRC_SUBMOD)
L3_OBJ=         $(L3_SRC:.f90=.o)
L3_MOD=         $(L3_SRC_MOD:.f90=.mod)
L3_SUBMOD=      cgca_m3clvg@m3clvg_sm1.smod \
               cgca_m3clvg@m3clvg_sm2.smod \
               cgca_m3sld@m3sld_sm1.smod

# level 4

L4_SRC=         cgca_m4fr.f90
L4_MOD=         $(L4_SRC:.f90=.mod)
L4_OBJ=         $(L4_SRC:.f90=.o)

# top level

LTOP_SRC=       cgca.f90
LTOP_MOD=       $(LTOP_SRC:.f90=.mod)
LTOP_OBJ=       $(LTOP_SRC:.f90=.o)

ALL_MOD=        $(L1_MOD) $(L2_MOD) $(L3_MOD) $(L4_MOD) $(LTOP_MOD)
ALL_SUBMOD=     $(L2_SUBMOD) $(L3_SUBMOD)
ALL_OBJ=        $(L1_OBJ) $(L2_OBJ) $(L3_OBJ) $(L4_OBJ) $(LTOP_OBJ)

ALL_CLEAN=      *.mod *.smod *.o *.optrpt $(MYLIB)

.SUFFIXES:
.SUFFIXES: .f90 .mod .o

all: $(MYLIB)

.f90.mod:
        $(FC) $(FFLAGS) $<

.f90.o:
        $(FC) $(FFLAGS) $<
```

```
# module dependencies

$(L2_MOD) $(L2_OBJ): $(L1_MOD)
$(L3_MOD) $(L3_OBJ): $(L2_MOD) $(L2_SUBMOD)
$(L4_MOD) $(L4_OBJ): $(L3_MOD) $(L3_SUBMOD)
$(LTOP_MOD) $(LTOP_OBJ): $(L4_MOD)
$(MYLIB): $(MOD_LTOP) $(OBJ_LTOP)

# Submodule dependencies
# level 2
m2out_sm1.o m2out_sm2_mpi.o: cgca_m2out.mod cgca_m2out.o

# level 3
m3clvg_sm1.o m3clvg_sm2.o: cgca_m3clvg.mod cgca_m3clvg.o
m3sld_sm1.o: cgca_m3sld.mod cgca_m3sld.o

$(MYLIB): $(ALL_OBJ)
        @if [ -e $(MYLIB) ]; then \
                rm $(MYLIB); \
        fi
        ar -r $(MYLIB) $(ALL_OBJ)

install: $(MYLIB) $(ALL_MOD) $(ALL_SUBMOD)
        @if [ -e $(LIBDIR)/$(MYLIB) ]; then \
                echo $(LIBDIR)/$(MYLIB) already exists; \
                echo run \"make deinstall\" first; \
                exit 1; \
        fi
        cp $(MYLIB) $(LIBDIR)
        cp $(ALL_MOD) $(ALL_SUBMOD) $(MODDIR)

deinstall:
        cd $(LIBDIR) && rm $(MYLIB)
        cd $(MODDIR) && rm $(ALL_MOD) $(ALL_SUBMOD)

clean:
        rm $(ALL_CLEAN)
```

40 tests/future_ca_omp1

[Unit tests]

NAME

future_ca_omp1

SYNOPSIS

```
!$Id: future_ca_omp1.f90 550 2018-04-27 17:08:42Z mexas $

program future_ca_omp1
```

PURPOSE

The future* tests are not part of casup (3). These are to test emerging capabilities. This test checks coarrays inside an OpenMP parallel region.

DESCRIPTION

Run on 2 images only! This is just for demo purposes. A 1D integer array coarray is set to 0 on both images. The last element on image 2 is set to 1. The kernel copies the value to the right to itself. So gradually all values change to 1. The HX is implemented using sync images.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES USED BY SOURCE

```
use :: omp_lib
implicit none
integer, parameter :: n=20
!integer, external :: omp_get_num_threads, omp_get_thread_num
integer :: a(0:n+1)[*], b(0:n+1), i, img, iter, tmp, tid, nthr, nimgs

img = this_image()
nimgs = num_images()

if ( nimgs .ne. 2 ) then
  write (*,*) "ERROR: this demo program runs only on 2 images"
  error stop
end if

! Set b=0 on both images, except b(n+1)=1 on image 2
if (img .eq. 1 ) b = 0
if (img .eq. 2 ) then
  b = 0
  b(n+1) = 1
end if

! 2*n iterations are required to propagate 1 across both
```

```
! images.
main: do iter = 1, 2*n
  a = b
  !$omp parallel do default(none) private(i,tmp,tid) &
  !$omp shared(img,a,b,nthr)
  loop: do i=1, n
    nthr = omp_get_num_threads()
    if (img .eq. 1 .and. i .eq. n ) then
      tid = omp_get_thread_num()
      write (*,"(a,3(i0,tr1))") "img, nthr, tid: ", img, nthr, tid
      sync images   (2)
      a(n+1) = a(1) [2]
    end if
    if (img .eq. 2 .and. i .eq. 1 ) then
      tid = omp_get_thread_num()
      write (*,"(a,3(i0,tr1))") "img, nthr, tid: ", img, nthr, tid
      sync images   (1)
      a(0)   = a(n) [1]
    end if
    b(i) = max( a(i+1), a(i-1) )
  end do loop
  !$omp end parallel do
  write (*,"(a,i0,tr1,i0,tr1,999i1)") "iter, img, b: ", iter, img, b(1:n)
end do main

end program future_ca_omp1
```

41 tests/future_ca_omp2

[Unit tests]

NAME

future_ca_omp2

SYNOPSIS

```
!$Id: future_ca_omp2.f90 552 2018-08-31 11:30:39Z mexas $

program future_ca_omp2
```

PURPOSE

The future* tests are not part of casup (3). These are to test emerging capabilities. This test checks coarrays inside an OpenMP parallel region with 2D HX.

DESCRIPTION

Run on 4 images only! This is just for demo purposes. This is a simple relaxation problem. A 2D real array coarray is set real(this_image()) on all images. The halos are fixed boundary conditions. The 4-element kernel computes the average of the 4 neighbours, up, down, left, right. Run multiple iterations until convergence. The HX is implemented using sync images inside OMP.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES USED BY SOURCE

```
use :: omp_lib
implicit none
integer, parameter :: n=10000, niter=100
real :: a(0:n+1,0:n+1)[2,*], b(0:n+1,0:n+1), time1, time2
integer :: grid(2), i, j, img, iter, tmp, tid, nthr, nimgs

  img = this_image()
 grid = this_image( a )
nimgs = num_images()

if ( nimgs .ne. 4 ) then
  write (*,*) "ERROR: this demo program runs only on 4 images"
  error stop
end if

! Set to image number
a = real( img )
b = a

sync all
if (img .eq. 1 ) call dump( n, a, "start.raw" )
```

```
sync all

  ! Start counter
  if ( img .eq. 1 ) call cpu_time( time1 )

main: do iter = 1, niter
  a = b
  sync all
  !$omp parallel do default(none) private(i,tmp,tid) &
  !$omp shared(img,a,b,nthr,grid)
  loop_j: do j = 1, n
  loop_i: do i = 1, n
    nthr = omp_get_num_threads()
    if ( i .eq. n .and. grid(1) .ne. 2 ) a(n+1,j) = a(1,j) [ grid(1)+1, grid(2) ]
    if ( i .eq. 1 .and. grid(1) .ne. 1 ) a(0,  j) = a(n,j) [ grid(1)-1, grid(2) ]
    if ( j .eq. n .and. grid(2) .ne. 2 ) a(i,n+1) = a(i,1) [ grid(1), grid(2)+1 ]
    if ( j .eq. 1 .and. grid(2) .ne. 1 ) a(i,  0) = a(i,n) [ grid(1), grid(2)-1 ]
    b(i,j) = 0.25 * ( a(i-1,j) + a(i+1,j) + a(i,j-1) + a(i,j+1) )
  end do loop_i
  end do loop_j
  !$omp end parallel do
  if (img .eq. 1 ) write (*,*) "iter:", iter
!  write (*,"(a,i0,tr1,i0,tr1,999i1)") "iter, img, b: ", iter, img, b(1:n)
end do main

sync all

  ! Start counter
  if ( img .eq. 1 ) then
    call cpu_time( time2 )
    write (*,*) "Time, s:", time2-time1
  end if

a=b
if (img .eq. 1 ) call dump( n, a, "end.raw" )

contains

! Call this sub only from image 1!
subroutine dump( n, arr, fname )
integer, intent(in) :: n
real, intent(in) :: arr(0:n+1,0:n+1)[2,*]
character(len=*), intent(in) :: fname
integer :: fu, coi1, coi2, j
open( newunit=fu, file=fname, status="replace", form="unformatted", &
      access="stream")

! Assume 2x2 images grid
! nested loops for writing in correct order from all images
! Do not write halos
 do coi2 = 1,2
  do j = 1, n
```

```
   do coi1 = 1,2
!    write (*,'(es10.2)') arr(1:n, j) [ coi1, coi2 ]
    write (fu) arr(1:n, j) [ coi1, coi2 ]
   end do
  end do
 end do

close( fu )
end subroutine dump

end program future_ca_omp2
```

42 tests/hxvn

[Unit tests]

NAME

hxvn

SYNOPSIS

```
!$Id: hxvn.f90 561 2018-10-14 20:48:19Z mexas $

program hxvn
```

PURPOSE

Test ca_hx_all (1.4). The kernel is a simply copy. only halos are coarrays, the rest of the model is not.

DESCRIPTION

- ca_halloc (1.1) - user allocates halo coarrays once, obviously at the start of the simulation.
- ca_hx_all (1.4) - a high level routine to do all necessary hx operations, with necessary sync.

Must work on any number of images, except when a good decomposition cannot be made. The user needs know nothing about sync.

NOTE AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

casup (3)

USED BY

Part of casup (3) test suite

SOURCE

```
use casup

implicit none

integer( kind=iarr ), parameter :: huge_iarr = huge(0_iarr)

real( kind=rdef ) :: &
  qual,              & ! quality
  bsz0(3),           & ! the given "box" size
  bsz(3),            & ! updated "box" size
  dm,                & ! mean grain size, linear dim, phys units
  lres,              & ! linear resolution, cells per unit of length
  res                  ! resolutions, cells per grain
```

```
integer( kind=iarr ), allocatable :: space(:,:,:),                    &
  space1(:,:,:)

integer( kind=idef ) :: ir(3), nimgs, img, ng, c(3) ! coarray dimensions

integer( kind=ilrg ) :: icells, mcells

integer :: ierr, d, run

! logical :: flag

!*********************************************************************72
! first executable statement

! Read the box size from command line
call ca_cmd_real( n=3, data=bsz0 )

!bsz0 = (/ 50., 50., 50. /) ! numbers of cells in CA space
!bsz0 = (/ 4.0e1, 8.0e1, 6.0e1 /) ! for testing on FreeBSD laptop
  dm = 1.0 ! cell size
 res = 1.0 ! resolution

  img = this_image()
nimgs = num_images()

! do a check on image 1
if ( img .eq. 1 ) then
  write (*,*) "running on", nimgs, "images in a 3D grid"
  write (*,*) "iarr kind:", iarr, "huge(0_iarr):", huge_iarr

  ! In this test space is assigned image numbers - must be big enough
  ! integer kind to avoid integer overflow.
  if ( nimgs .gt. huge_iarr ) then
    write (*,*) "ERROR: num_images(): ", nimgs,              &
      " is greater than huge(0_iarr)"
    error stop
  end if
end if

! each image calculates the coarray grid dimensions
call cgca_gdim( nimgs, ir, qual )

! calculate the resolution and the actual phys dimensions of the box
! subroutine cgca_cadim( bsz, res, dm, ir, c, lres, ng )
! c - coarray sizes
! ir - coarray grid sizes
bsz = bsz0
call cgca_cadim( bsz, res, dm, ir, c, lres, ng )

! Check that the partition is sane
if ( img .eq. 1 ) then
  if ( any(int(bsz) .ne. int(bsz0) ) ) then
```

```
    write (*,*)                                                          &
      "ERROR: bad decomposition - use a 'nicer' number of images"
    write (*,*) "ERROR: wanted         :", int(bsz0)
    write (*,*) "ERROR: but got instead:", int(bsz)
    error stop
  end if
end if

! total number of cells in a coarray
icells = int( c(1), kind=ilrg ) * int( c(2), kind=ilrg ) *             &
         int( c(3), kind=ilrg )

! total number of cells in the model
mcells = icells * int( nimgs, kind=ilrg )

if ( img .eq. 1 ) then
  write ( *, "(8(a,i0),tr1,g10.3,tr1,g10.3,3(a,g10.3),a)" )            &
    "nimgs: ", nimgs, " (", c(1), "," , c(2), "," , c(3), ")[",        &
    ir(1), "," , ir(2), "," , ir(3), "] ", ng, qual, lres,             &
    " (", bsz(1), ",", bsz(2), ",", bsz(3), ")"
     write (*,'(a,i0,a)') "Each image has ",icells, " cells"
     write (*,'(a,i0,a)') "The model has ", mcells, " cells"
end if

! MPI
! init
! lines
! here
! in
! test_mpi_hxvn

! run=1 => ca_iter_tl
! run=2 => ca_iter_dc
! run=3 => ca_iter_omp
outer: do run=1,3

  if ( img .eq. 1 ) then
    select case( run )
    case(1)
      write (*,*) "Checking ca_iter_tl - triple loop"
    case(2)
      write (*,*) "Checking ca_iter_dc - do concurrent"
    case(3)
      write (*,*) "Checking ca_iter_omp - OpenMP"
    end select
  end if

  ! Loop over several halo depths
  ! The max halo depth is 1/4 of the min dimension
  ! of the space CA array
  main: do d=1, int( 0.25 * min( c(1), c(2), c(3) ) )
```

```
! allocate space array
!    space - CA array to allocate, with halos!
!        c - array with space dimensions
!        d - depth of the halo layer

call ca_spalloc( space,  c, d )
call ca_spalloc( space1, c, d )

! Set space to my image number
space  = int( img, kind=iarr )
space1 = space

! allocate hx arrays, implicit sync all
! ir(3) - codimensions
call ca_halloc( ir )

! MPI subarray types in
! test_mpi_hxvn

! do hx, remote ops
call ca_hx_all(     space )

! halo check, local ops
! space - space array, with halos
! flag - default integer
call ca_hx_check(    space=space, flag=ierr )
if ( ierr .ne. 0 ) then
  write (*,*) "ERROR: ca_hx_check    failed: img:", img,           &
              "flag:", ierr
  error stop
end if

! CA iterations
! subroutine ca_run(    space, hx_sub, iter_sub, kernel, niter )
select case( run )
case(1)
  call ca_run(    space = space, hx_sub = ca_hx_all,               &
    iter_sub = ca_iter_tl,  kernel = ca_kernel_copy, niter = 13 )
case(2)
  call ca_run(    space = space, hx_sub = ca_hx_all,               &
    iter_sub = ca_iter_dc,  kernel = ca_kernel_copy, niter = 13 )
case(3)
  call ca_run(    space = space, hx_sub = ca_hx_all,               &
    iter_sub = ca_iter_omp, kernel = ca_kernel_copy, niter = 13 )
end select

! Must be the same
if ( any( space(  1:c(1), 1:c(2), 1:c(3) ) .ne.                   &
          space1( 1:c(1), 1:c(2), 1:c(3) ) ) ) then
  write (*,*) "img:", img, "FAIL: space .ne. space1"
  error stop
end if
```

```
    ! deallocate halos, implicit sync all
    call ca_hdalloc

    ! Free MPI types in
    ! test_mpi_hxvn

    ! deallocate space
    deallocate( space )
    deallocate( space1 )

    if (img .eq. 1 ) write (*,*) "PASS, halo depth:", d

  end do main

end do outer

end program hxvn
```

43 tests/hxvn_1D

[Unit tests]

NAME

hxvn_1D

SYNOPSIS

```
!$Id: hxvn_1D.f90 561 2018-10-14 20:48:19Z mexas $

program hxvn_1D
```

PURPOSE

Test ca_1D_hx_sall (1.3.4). The kernel is a simply copy. only halos are coarrays, the rest of the model is not.

DESCRIPTION

- ca_1D_halloc (1.3.1) - user allocates halo coarrays once, obviously at the start of the simulation.
- ca_1D_hx_sall (1.3.4) - a high level routine to do all necessary hx operations, with a global barrier, sync all.

Must work on any number of images, except when a good decomposition cannot be made. The user needs know nothing about sync.

NOTE AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

casup (3)

USED BY

Part of casup (3) test suite

SOURCE

```
use casup

implicit none

integer( kind=iarr ), parameter :: huge_iarr = huge(0_iarr)

real( kind=rdef ) :: bsz(3)            ! "box" size
```

```
integer( kind=iarr ), allocatable :: space(:,:,:),                    &
  space1(:,:,:)

integer( kind=idef ) :: nimgs, img, c(3) ! coarray dimensions

integer( kind=ilrg ) :: icells, mcells

integer :: ierr, d, run

! logical :: flag

!*********************************************************************72
! first executable statement

! Read the box size from command line
call ca_cmd_real( n=3, data=bsz )

!bsz = (/ 50.0, 50.0, 24000.0 /) ! numbers of cells in CA space
!bsz0 = (/ 4.0e1, 8.0e1, 6.0e1 /) ! for testing on FreeBSD laptop



  img = this_image()
nimgs = num_images()

! do a check on image 1
if ( img .eq. 1 ) then
  write (*,*) "running on", nimgs, "images in a 1D grid"
  write (*,*) "iarr kind:", iarr, "huge(0_iarr):", huge_iarr

  ! In this test space is assigned image numbers - must be big enough
  ! integer kind to avoid integer overflow.
  if ( nimgs .gt. huge_iarr ) then
    write (*,*) "ERROR: num_images(): ", nimgs,              &
      " is greater than huge(0_iarr)"
    error stop
  end if
end if

! Only a single codimension, corank 1
c = int(bsz)
c(3) = int(bsz(3))/nimgs

! Check that the partition is sane
if ( img .eq. 1 ) then
  if ( c(3)*nimgs .ne. int(bsz(3)) ) then
    write (*,*)                                                    &
      "ERROR: bad decomposition: bsz(3) must be divisible by nimgs"
    write (*,*) "ERROR: bsz(3):", int(bsz(3))
    write (*,*) "ERROR: nimgs :", nimgs
```

```
    error stop
  end if
end if

! total number of cells in a coarray
icells = int( c(1), kind=ilrg ) * int( c(2), kind=ilrg ) *             &
         int( c(3), kind=ilrg )

! total number of cells in the model
mcells = icells * int( nimgs, kind=ilrg )

if ( img .eq. 1 ) then
  write ( *, "(5(a,i0),3(a,g10.3),a)" )                                &
    "nimgs: ", nimgs, " (", c(1), "," , c(2), "," , c(3),             &
    ")[", nimgs, "] (", bsz(1), ",", bsz(2), ",", bsz(3), ")"
     write (*,'(a,i0,a)') "Each image has ",icells, " cells"
     write (*,'(a,i0,a)') "The model has ", mcells, " cells"
end if

! MPI
! init
! lines
! here
! in
! test_mpi_hxvn

! run=1 => ca_iter_tl
! run=2 => ca_iter_dc
! run=3 => ca_iter_omp
outer: do run=1,3

  if ( img .eq. 1 ) then
    select case( run )
    case(1)
      write (*,*) "Checking ca_iter_tl - triple loop"
    case(2)
      write (*,*) "Checking ca_iter_dc - do concurrent"
    case(3)
      write (*,*) "Checking ca_iter_omp - OpenMP"
    end select
  end if

  ! Loop over several halo depths
  ! The max halo depth is 1/4 of the min dimension
  ! of the space CA array
  main: do d=1, int( 0.25 * min( c(1), c(2), c(3) ) )

    ! allocate space array
    !    space - CA array to allocate, with halos!
    !        c - array with space dimensions
    !        d - depth of the halo layer
```

```
call ca_spalloc( space,  c, d )
call ca_spalloc( space1, c, d )

! Set space to my image number
space  = int( img, kind=iarr )
space1 = space

! allocate hx arrays, implicit sync all

call ca_1D_halloc

! MPI subarray types in
! test_mpi_hxvn

! do hx, remote ops
call ca_1D_hx_sall(     space )

! halo check, local ops
! space - space array, with halos
! flag - default integer
call ca_1D_hx_check(    space=space, flag=ierr )
if ( ierr .ne. 0 ) then
  write (*,*) "ERROR: ca_hx_check    failed: img:", img,           &
              "flag:", ierr
  error stop
end if

! CA iterations
! subroutine ca_run(    space, hx_sub, iter_sub, kernel, niter )
select case( run )
case(1)
  call ca_run(    space = space, hx_sub = ca_1D_hx_sall,           &
    iter_sub = ca_iter_tl,  kernel = ca_kernel_copy, niter = 13 )
case(2)
  call ca_run(    space = space, hx_sub = ca_1D_hx_sall,           &
    iter_sub = ca_iter_dc,  kernel = ca_kernel_copy, niter = 13 )
case(3)
  call ca_run(    space = space, hx_sub = ca_1D_hx_sall,           &
    iter_sub = ca_iter_omp, kernel = ca_kernel_copy, niter = 13 )
end select

! Must be the same
if ( any( space(  1:c(1), 1:c(2), 1:c(3) ) .ne.                   &
          space1( 1:c(1), 1:c(2), 1:c(3) ) ) ) then
  write (*,*) "img:", img, "FAIL: space .ne. space1"
  error stop
end if

! deallocate halos, implicit sync all
call ca_1D_hdalloc

! Free MPI types in
```

```
    ! test_mpi_hxvn

    ! deallocate space
    deallocate( space )
    deallocate( space1 )

    if (img .eq. 1 ) write (*,*) "PASS, halo depth:", d

  end do main

end do outer

end program hxvn_1D
```

44 tests/hxvn_co

[Unit tests]

NAME

hxvn_co

SYNOPSIS

```
!$Id: hxvn_co.f90 561 2018-10-14 20:48:19Z mexas $

program hxvn_co
```

PURPOSE

Test HX routine ca_co_hx_all (1.6.1). Use coarrays for the whole model, not just halos.

DESCRIPTION

- ca_co_spalloc (1.6.7) - user allocates the space coarray, obviously at the start of the simulation.
- ca_co_hx_all (1.6.1) - a high level routine to do all necessary HX operations, with necessary sync.

Must work on any number of images, except when a good decomposition cannot be made. The user needs know nothing about sync.

NOTE AUTHOR

Anton Shterenlikht

COPYRIGHT

USES

casup (3)

USED BY

Part of casup (3) test suite

SOURCE

```
use casup

implicit none

integer( kind=iarr ), parameter :: huge_iarr = huge(0_iarr)

real( kind=rdef ) :: &
  qual,              & ! quality
  bsz0(3),           & ! the given "box" size
  bsz(3),            & ! updated "box" size
  dm,                & ! mean grain size, linear dim, phys units
  lres,              & ! linear resolution, cells per unit of length
  res                  ! resolutions, cells per grain
```

```
integer( kind=iarr ), allocatable :: space(:,:,:) [:,:,:],            &
  space1(:,:,:) [:,:,:]

integer( kind=idef ) :: ir(3), nimgs, img, ng, c(3) ! coarray dimensions

integer( kind=ilrg ) :: icells, mcells

integer :: ierr, d, run

! logical :: flag

!*********************************************************************72
! first executable statement

! Read the box size from command line
call ca_cmd_real( n=3, data=bsz0 )

!bsz0 = (/ 4.0e2, 8.0e2, 6.0e2 /) ! numbers of cells in CA space
!bsz0 = (/ 4.0e1, 8.0e1, 6.0e1 /) ! for testing on FreeBSD laptop
  dm = 1.0 ! cell size
 res = 1.0 ! resolution

  img = this_image()
nimgs = num_images()

! do a check on image 1
if ( img .eq. 1 ) then
  write (*,*) "running on", nimgs, "images in a 3D grid"
  write (*,*) "iarr kind:", iarr, "huge(0_iarr):", huge_iarr

  ! In this test space is assigned image numbers - must be big enough
  ! integer kind to avoid integer overflow.
  if ( nimgs .gt. huge_iarr ) then
    write (*,*) "ERROR: num_images(): ", nimgs,             &
      " is greater than huge(0_iarr)"
    error stop
  end if
end if

! each image calculates the coarray grid dimensions
call cgca_gdim( nimgs, ir, qual )

! calculate the resolution and the actual phys dimensions of the box
! subroutine cgca_cadim( bsz, res, dm, ir, c, lres, ng )
! c - coarray sizes
! ir - coarray grid sizes
bsz = bsz0
call cgca_cadim( bsz, res, dm, ir, c, lres, ng )

! Check that the partition is sane
if ( img .eq. 1 ) then
  if ( any(int(bsz) .ne. int(bsz0) ) ) then
```

```
    write (*,*)                                                          &
      "ERROR: bad decomposition - use a 'nicer' number of images"
    write (*,*) "ERROR: wanted         :", int(bsz0)
    write (*,*) "ERROR: but got instead:", int(bsz)
    error stop
  end if
end if

! total number of cells in a coarray
icells = int( c(1), kind=ilrg ) * int( c(2), kind=ilrg ) *             &
         int( c(3), kind=ilrg )

! total number of cells in the model
mcells = icells * int( nimgs, kind=ilrg )

if ( img .eq. 1 ) then
  write ( *, "(8(a,i0),tr1,g10.3,tr1,g10.3,3(a,g10.3),a)" )            &
    "nimgs: ", nimgs, " (", c(1), "," , c(2), "," , c(3), ")[",        &
    ir(1), "," , ir(2), "," , ir(3), "] ", ng, qual, lres,             &
    " (", bsz(1), ",", bsz(2), ",", bsz(3), ")"
     write (*,'(a,i0,a)') "Each image has ",icells, " cells"
     write (*,'(a,i0,a)') "The model has ", mcells, " cells"
end if

! MPI
! init
! lines
! here
! in
! test_mpi_hxvn

! run=1 => ca_iter_tl
! run=2 => ca_iter_dc
! run=3 => ca_iter_omp
outer: do run=1,3

  if ( img .eq. 1 ) then
    select case( run )
    case(1)
      write (*,*) "Checking ca_iter_tl - triple loop"
    case(2)
      write (*,*) "Checking ca_iter_dc - do concurrent"
    case(3)
      write (*,*) "Checking ca_iter_omp - OpenMP"
    end select
  end if

  ! Loop over several halo depths
  ! The max halo depth is 1/4 of the min dimension
  ! of the space CA array
  main: do d=1, int( 0.25 * min( c(1), c(2), c(3) ) )
```

```
! allocate space array coarrays
!    space - CA array to allocate, with halos!
!        c - array with space dimensions
!        d - depth of the halo layer
!       ir - codimensions
call ca_co_spalloc( space,  c, d, ir )
call ca_co_spalloc( space1, c, d, ir )

! Set space to my image number
space  = int( img, kind=iarr )
space1 = space

! No need
! for separate
! HX coarrays!

! MPI subarray types in
! test_mpi_hxvn

! do hx, remote ops
call ca_co_hx_all( space )

! halo check, local ops
! space - space array, with halos
! flag - default integer
call ca_co_hx_check( space=space, flag=ierr )
if ( ierr .ne. 0 ) then
  write (*,*) "ERROR: ca_co_hx_check failed: img:", img,           &
              "flag:", ierr
  error stop
end if

! CA iterations
! subroutine ca_co_run( space, hx_sub, iter_sub, kernel, niter )
select case( run )
case(1)
  call ca_co_run( space = space, hx_sub = ca_co_hx_all,            &
    iter_sub = ca_iter_tl,  kernel = ca_kernel_copy, niter = 13 )
case(2)
  call ca_co_run( space = space, hx_sub = ca_co_hx_all,            &
    iter_sub = ca_iter_dc,  kernel = ca_kernel_copy, niter = 13 )
case(3)
  call ca_co_run( space = space, hx_sub = ca_co_hx_all,            &
    iter_sub = ca_iter_omp, kernel = ca_kernel_copy, niter = 13 )
end select

! Must be the same
if ( any( space(  1:c(1), 1:c(2), 1:c(3) ) .ne.                   &
          space1( 1:c(1), 1:c(2), 1:c(3) ) ) ) then
  write (*,*) "img:", img, "FAIL: space .ne. space1"
  error stop
end if
```

```
    ! No separate halos
    ! to deallocate

    ! Free MPI types in
    ! test_mpi_hxvn

    ! deallocate space
    deallocate( space )
    deallocate( space1 )

    if (img .eq. 1 ) write (*,*) "PASS, halo depth:", d

  end do main

end do outer

end program hxvn_co
```

45 tests/hxvn_glbar

[Unit tests]

NAME

hxvn_glbar

SYNOPSIS

```
!$Id: hxvn_glbar.f90 561 2018-10-14 20:48:19Z mexas $

program hxvn_glbar
```

PURPOSE

Test ca_hx_glbar (1.7). The kernel is a simply copy. only halos are coarrays, the rest of the model is not.

DESCRIPTION

- ca_halloc (1.1) - user allocates halo coarrays once, obviously at the start of the simulation.
- ca_hx_glbar (1.7) - a high level routine to do all necessary hx operations, with a global barrier - sync all.

Must work on any number of images, except when a good decomposition cannot be made. The user needs know nothing about sync.

NOTE AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

casup (3)

USED BY

Part of casup (3) test suite

SOURCE

```
use casup

implicit none

integer( kind=iarr ), parameter :: huge_iarr = huge(0_iarr)

real( kind=rdef ) :: &
  qual,              & ! quality
  bsz0(3),           & ! the given "box" size
  bsz(3),            & ! updated "box" size
  dm,                & ! mean grain size, linear dim, phys units
  lres,              & ! linear resolution, cells per unit of length
  res                  ! resolutions, cells per grain
```

```
integer( kind=iarr ), allocatable :: space(:,:,:),                    &
  space1(:,:,:)

integer( kind=idef ) :: ir(3), nimgs, img, ng, c(3) ! coarray dimensions

integer( kind=ilrg ) :: icells, mcells

integer :: ierr, d, run

! logical :: flag

!*********************************************************************72
! first executable statement

! Read the box size from command line
call ca_cmd_real( n=3, data=bsz0 )

!bsz0 = (/ 50., 50., 50. /) ! numbers of cells in CA space
!bsz0 = (/ 4.0e1, 8.0e1, 6.0e1 /) ! for testing on FreeBSD laptop
  dm = 1.0 ! cell size
 res = 1.0 ! resolution

  img = this_image()
nimgs = num_images()

! do a check on image 1
if ( img .eq. 1 ) then
  write (*,*) "running on", nimgs, "images in a 3D grid"
  write (*,*) "iarr kind:", iarr, "huge(0_iarr):", huge_iarr

  ! In this test space is assigned image numbers - must be big enough
  ! integer kind to avoid integer overflow.
  if ( nimgs .gt. huge_iarr ) then
    write (*,*) "ERROR: num_images(): ", nimgs,              &
      " is greater than huge(0_iarr)"
    error stop
  end if
end if

! each image calculates the coarray grid dimensions
call cgca_gdim( nimgs, ir, qual )

! calculate the resolution and the actual phys dimensions of the box
! subroutine cgca_cadim( bsz, res, dm, ir, c, lres, ng )
! c - coarray sizes
! ir - coarray grid sizes
bsz = bsz0
call cgca_cadim( bsz, res, dm, ir, c, lres, ng )

! Check that the partition is sane
if ( img .eq. 1 ) then
```

```
  if ( any(int(bsz) .ne. int(bsz0) ) ) then
    write (*,*)                                                          &
      "ERROR: bad decomposition - use a 'nicer' number of images"
    write (*,*) "ERROR: wanted         :", int(bsz0)
    write (*,*) "ERROR: but got instead:", int(bsz)
    error stop
  end if
end if

! total number of cells in a coarray
icells = int( c(1), kind=ilrg ) * int( c(2), kind=ilrg ) *             &
         int( c(3), kind=ilrg )

! total number of cells in the model
mcells = icells * int( nimgs, kind=ilrg )

if ( img .eq. 1 ) then
  write ( *, "(8(a,i0),tr1,g10.3,tr1,g10.3,3(a,g10.3),a)" )            &
    "nimgs: ", nimgs, " (", c(1), "," , c(2), "," , c(3), ")[",        &
    ir(1), "," , ir(2), "," , ir(3), "] ", ng, qual, lres,             &
    " (", bsz(1), ",", bsz(2), ",", bsz(3), ")"
     write (*,'(a,i0,a)') "Each image has ",icells, " cells"
     write (*,'(a,i0,a)') "The model has ", mcells, " cells"
end if

! MPI
! init
! lines
! here
! in
! test_mpi_hxvn

! run=1 => ca_iter_tl
! run=2 => ca_iter_dc
! run=3 => ca_iter_omp
outer: do run=1,3

  if ( img .eq. 1 ) then
    select case( run )
    case(1)
      write (*,*) "Checking ca_iter_tl - triple loop"
    case(2)
      write (*,*) "Checking ca_iter_dc - do concurrent"
    case(3)
      write (*,*) "Checking ca_iter_omp - OpenMP"
    end select
  end if

  ! Loop over several halo depths
  ! The max halo depth is 1/4 of the min dimension
  ! of the space CA array
  main: do d=1, int( 0.25 * min( c(1), c(2), c(3) ) )
```

```
! allocate space array
!    space - CA array to allocate, with halos!
!        c - array with space dimensions
!        d - depth of the halo layer

call ca_spalloc( space,  c, d )
call ca_spalloc( space1, c, d )

! Set space to my image number
space  = int( img, kind=iarr )
space1 = space

! allocate hx arrays, implicit sync all
! ir(3) - codimensions
call ca_halloc( ir )

! MPI subarray types in
! test_mpi_hxvn

! do hx, remote ops
call ca_hx_glbar(     space )

! halo check, local ops
! space - space array, with halos
! flag - default integer
call ca_hx_check(    space=space, flag=ierr )
if ( ierr .ne. 0 ) then
  write (*,*) "ERROR: ca_hx_check    failed: img:", img,           &
              "flag:", ierr
  error stop
end if

! CA iterations
! subroutine ca_run(    space, hx_sub, iter_sub, kernel, niter )
select case( run )
case(1)
  call ca_run(    space = space, hx_sub = ca_hx_glbar,             &
    iter_sub = ca_iter_tl,  kernel = ca_kernel_copy, niter = 13 )
case(2)
  call ca_run(    space = space, hx_sub = ca_hx_glbar,             &
    iter_sub = ca_iter_dc,  kernel = ca_kernel_copy, niter = 13 )
case(3)
  call ca_run(    space = space, hx_sub = ca_hx_glbar,             &
    iter_sub = ca_iter_omp, kernel = ca_kernel_copy, niter = 13 )
end select

! Must be the same
if ( any( space(  1:c(1), 1:c(2), 1:c(3) ) .ne.                   &
          space1( 1:c(1), 1:c(2), 1:c(3) ) ) ) then
  write (*,*) "img:", img, "FAIL: space .ne. space1"
  error stop
```

```
    end if

    ! deallocate halos, implicit sync all
    call ca_hdalloc

    ! Free MPI types in
    ! test_mpi_hxvn

    ! deallocate space
    deallocate( space )
    deallocate( space1 )

    if (img .eq. 1 ) write (*,*) "PASS, halo depth:", d

  end do main

end do outer

end program hxvn_glbar
```

46 tests/hxvn_timing

[Unit tests]

NAME

hxvn_timing

SYNOPSIS

```
!$Id: hxvn_timing.f90 548 2018-04-27 14:49:39Z mexas $

program hxvn_timing
```

PURPOSE

Time HX for HCA, WCA and MPI. No computation or any other work is done. Several HX sizes are used.

DESCRIPTION

- ca_halloc (1.1) - user allocates halo coarrays once, obviously at the start of the simulation.
- ca_hx_all (1.4) - a high level routine to do all necessary hx operations, with necessary sync.

Must work on any number of images, except when a good decomposition cannot be made. The user needs know nothing about sync.

NOTE AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

casup (3)

USED BY

Part of casup (3) test suite

SOURCE

```
use casup

implicit none

integer( kind=iarr ), parameter :: huge_iarr = huge(0_iarr)

real( kind=rdef ) :: &
  qual,              & ! quality
  bsz0(3),           & ! the given "box" size
  bsz(3),            & ! updated "box" size
  dm,                & ! mean grain size, linear dim, phys units
  lres,              & ! linear resolution, cells per unit of length
  res                  ! resolutions, cells per grain
```

```
integer( kind=iarr ), allocatable :: space(:,:,:)

integer( kind=idef ) :: ir(3), nimgs, img, ng, c(3), hsize, i, iter

integer( kind=ilrg ) :: icells, mcells

real, allocatable :: rnd(:,:,:)
real( kind=kind(1.0d0) ) :: time1, time2

! integer :: ierr
! logical :: flag

!*****************************************************************72
! first executable statement

  img = this_image()
nimgs = num_images()

! do a check on image 1
if ( img .eq. 1 ) then
  write (*,*) "running on", nimgs, "images in a 3D grid"
  write (*,*) "iarr kind:", iarr, "huge(0_iarr):", huge_iarr

  ! In this test space is assigned only 0 or 1, and no collectives,
  ! so even 1 byte integers will do.
end if

! each image calculates the coarray grid dimensions
call cgca_gdim( nimgs, ir, qual )

  dm = 1.0 ! cell size
 res = 1.0 ! resolution

! Loop over several box sizes, i.e. halo sizes
main: do i=1,35

  ! 50, 100, 150
  bsz0 = (/ 100*i, 100*i, 100*i /) ! numbers of cells in CA space

  ! calculate the resolution and the actual phys dimensions of the box
  ! subroutine cgca_cadim( bsz, res, dm, ir, c, lres, ng )
  ! c - coarray sizes
  ! ir - coarray grid sizes
  bsz = bsz0
  call cgca_cadim( bsz, res, dm, ir, c, lres, ng )

  ! Check that the partition is sane
  if ( img .eq. 1 ) then
    if ( any(int(bsz) .ne. int(bsz0) ) ) then
      write (*,*)                                                    &
        "ERROR: bad decomposition - use a 'nicer' number of images"
```

```
      write (*,*) "ERROR: wanted         :", int(bsz0)
      write (*,*) "ERROR: but got instead:", int(bsz)
      error stop
    end if

    ! total number of cells in a coarray
    icells = int( c(1), kind=ilrg ) * int( c(2), kind=ilrg ) *         &
           int( c(3), kind=ilrg )

    ! total number of cells in the model
    mcells = icells * int( nimgs, kind=ilrg )

    ! Halo size
    hsize = c(1)*c(2)

    write ( *, "(8(a,i0),tr1,g10.3,tr1,g10.3,3(a,g10.3),a)" )          &
      "nimgs: ", nimgs, " (", c(1), "," , c(2), "," , c(3), ")[",      &
      ir(1), "," , ir(2), "," , ir(3), "] ", ng, qual, lres,           &
      " (", bsz(1), ",", bsz(2), ",", bsz(3), ")"
    write (*,'(a,i0,a)') "Each image has ",icells, " cells"
    write (*,'(a,i0,a)') "The model has ", mcells, " cells"
    write (*,'(a,i0,a)') "Halo size is ", hsize, " cells"
  end if

  ! MPI
  ! init
  ! lines
  ! here
  ! in
  ! test_mpi_hxvn

  ! allocate space array
  !    space - CA array to allocate, with halos!
  !        c - array with space dimensions
  !        d - depth of the halo layer

  call ca_spalloc(    space, c, 1 )

  ! Very crude RND
  allocate( rnd( size(space, dim=1), size(space, dim=2),               &
                 size(space, dim=3) ) )
  call random_number( rnd )
  space = nint( rnd, kind=iarr ) ! Just 0 or 1

  ! allocate hx array coarrays, implicit sync all
  ! ir(3) - codimensions
  call ca_halloc( ir )

  ! MPI subarray types in
  ! hxvn_timing_mpi

  ! do several HX, remote ops, sync inside
```

```
  ! Make sure the timer covers the slowest image
  sync all
  time1 = cgca_benchtime()
  do iter=1,10
    call ca_hx_all(     space )
  end do
  ! Make sure the timer covers the slowest image
  sync all
  time2 = cgca_benchtime()
  if (img .eq. 1) write (*,*) "Halo, cells:", hsize, ". Time, s:",     &
                              time2-time1

  ! deallocate arrays
  call ca_hdalloc
  deallocate( space )
  deallocate( rnd )

  ! Free MPI types in
  ! hxvn_timing_mpi

end do main

if (img .eq. 1 ) write (*,*) "PASS"

end program hxvn_timing
```

47 tests/hxvn_timing_co

[Unit tests]

NAME

hxvn_timing_co

SYNOPSIS

```
!$Id: hxvn_timing_co.f90 548 2018-04-27 14:49:39Z mexas $

program hxvn_timing_co
```

PURPOSE

Time HX for HCA, WCA and MPI. No computation or any other work is done. Several HX sizes are used.

DESCRIPTION

- ca_halloc (1.1) - user allocates halo coarrays once, obviously at the start of the simulation.
- ca_hx_all (1.4) - a high level routine to do all necessary hx operations, with necessary sync.

Must work on any number of images, except when a good decomposition cannot be made. The user needs know nothing about sync.

NOTE AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

casup (3)

USED BY

Part of casup (3) test suite

SOURCE

```
use casup

implicit none

integer( kind=iarr ), parameter :: huge_iarr = huge(0_iarr)

real( kind=rdef ) :: &
  qual,              & ! quality
  bsz0(3),           & ! the given "box" size
  bsz(3),            & ! updated "box" size
  dm,                & ! mean grain size, linear dim, phys units
  lres,              & ! linear resolution, cells per unit of length
  res                  ! resolutions, cells per grain
```

```
integer( kind=iarr ), allocatable :: space(:,:,:) [:,:,:]

integer( kind=idef ) :: ir(3), nimgs, img, ng, c(3), hsize, i, iter

integer( kind=ilrg ) :: icells, mcells

real, allocatable :: rnd(:,:,:)
real( kind=kind(1.0d0) ) :: time1, time2

! integer :: ierr
! logical :: flag

!*********************************************************************72
! first executable statement

  img = this_image()
nimgs = num_images()

! do a check on image 1
if ( img .eq. 1 ) then
  write (*,*) "running on", nimgs, "images in a 3D grid"
  write (*,*) "iarr kind:", iarr, "huge(0_iarr):", huge_iarr

  ! In this test space is assigned only 0 or 1, and no collectives,
  ! so even 1 byte integers will do.
end if

! each image calculates the coarray grid dimensions
call cgca_gdim( nimgs, ir, qual )

  dm = 1.0 ! cell size
 res = 1.0 ! resolution

! Loop over several box sizes, i.e. halo sizes
main: do i=1,35

  ! 50, 100, 150
  bsz0 = (/ 100*i, 100*i, 100*i /) ! numbers of cells in CA space

  ! calculate the resolution and the actual phys dimensions of the box
  ! subroutine cgca_cadim( bsz, res, dm, ir, c, lres, ng )
  ! c - coarray sizes
  ! ir - coarray grid sizes
  bsz = bsz0
  call cgca_cadim( bsz, res, dm, ir, c, lres, ng )

  ! Check that the partition is sane
  if ( img .eq. 1 ) then
    if ( any(int(bsz) .ne. int(bsz0) ) ) then
      write (*,*)                                                    &
        "ERROR: bad decomposition - use a 'nicer' number of images"
```

```
      write (*,*) "ERROR: wanted         :", int(bsz0)
      write (*,*) "ERROR: but got instead:", int(bsz)
      error stop
    end if

    ! total number of cells in a coarray
    icells = int( c(1), kind=ilrg ) * int( c(2), kind=ilrg ) *         &
           int( c(3), kind=ilrg )

    ! total number of cells in the model
    mcells = icells * int( nimgs, kind=ilrg )

    ! Halo size
    hsize = c(1)*c(2)

    write ( *, "(8(a,i0),tr1,g10.3,tr1,g10.3,3(a,g10.3),a)" )          &
      "nimgs: ", nimgs, " (", c(1), "," , c(2), "," , c(3), ")[",      &
      ir(1), "," , ir(2), "," , ir(3), "] ", ng, qual, lres,           &
      " (", bsz(1), ",", bsz(2), ",", bsz(3), ")"
    write (*,'(a,i0,a)') "Each image has ",icells, " cells"
    write (*,'(a,i0,a)') "The model has ", mcells, " cells"
    write (*,'(a,i0,a)') "Halo size is ", hsize, " cells"
  end if

  ! MPI
  ! init
  ! lines
  ! here
  ! in
  ! test_mpi_hxvn

  ! allocate space array
  !    space - CA array to allocate, with halos!
  !        c - array with space dimensions
  !        d - depth of the halo layer
  !       ir - codimensions
  call ca_co_spalloc( space, c, 1, ir )

  ! Very crude RND
  allocate( rnd( size(space, dim=1), size(space, dim=2),               &
                 size(space, dim=3) ) )
  call random_number( rnd )
  space = nint( rnd, kind=iarr ) ! Just 0 or 1

  ! allocate hx array coarrays, implicit sync all
  ! ir(3) - codimensions
  !call ca_halloc( ir )

  ! MPI subarray types in
  ! test_mpi_hxvn

  ! do several HX, remote ops, sync inside
```

```
  ! Make sure the timer covers the slowest image
  sync all
  time1 = cgca_benchtime()
  do iter=1,10
    call ca_co_hx_all(  space )
  end do
  ! Make sure the timer covers the slowest image
  sync all
  time2 = cgca_benchtime()
  if (img .eq. 1) write (*,*) "Halo, cells:", hsize, ". Time, s:",     &
                              time2-time1

  ! deallocate arrays
  !call ca_hdalloc
  deallocate( space )
  deallocate( rnd )

  ! Free MPI types in
  ! test_mpi_hxvn

end do main

if (img .eq. 1 ) write (*,*) "PASS"

end program hxvn_timing_co
```

48 tests/hxvn_timing_mpi

[Unit tests]

NAME

hxvn_timing_mpi

SYNOPSIS

```
!$Id: hxvn_timing_mpi.f90 548 2018-04-27 14:49:39Z mexas $

program hxvn_timing_mpi
```

PURPOSE

Time HX for HCA, WCA and MPI. No computation or any other work is done. Several HX sizes are used.

DESCRIPTION

- ca_halloc (1.1) - user allocates halo coarrays once, obviously at the start of the simulation.
- ca_hx_all (1.4) - a high level routine to do all necessary hx operations, with necessary sync.

Must work on any number of images, except when a good decomposition cannot be made. The user needs know nothing about sync.

NOTE AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

casup (3)

USED BY

Part of casup (3) test suite

SOURCE

```
use casup

implicit none

integer( kind=iarr ), parameter :: huge_iarr = huge(0_iarr)

real( kind=rdef ) :: &
  qual,              & ! quality
  bsz0(3),           & ! the given "box" size
  bsz(3),            & ! updated "box" size
  dm,                & ! mean grain size, linear dim, phys units
  lres,              & ! linear resolution, cells per unit of length
  res                  ! resolutions, cells per grain
```

```
integer( kind=iarr ), allocatable :: space(:,:,:)

integer( kind=idef ) :: ir(3), nimgs, img, ng, c(3), hsize, i, iter

integer( kind=ilrg ) :: icells, mcells

real, allocatable :: rnd(:,:,:)
real( kind=kind(1.0d0) ) :: time1, time2

logical :: flag
integer :: ierr

!*********************************************************************72
! first executable statement

  img = this_image()
nimgs = num_images()

! do a check on image 1
if ( img .eq. 1 ) then
  write (*,*) "running on", nimgs, "images in a 3D grid"
  write (*,*) "iarr kind:", iarr, "huge(0_iarr):", huge_iarr

  ! In this test space is assigned only 0 or 1, and no collectives,
  ! so even 1 byte integers will do.
end if

! each image calculates the coarray grid dimensions
call cgca_gdim( nimgs, ir, qual )

  dm = 1.0 ! cell size
 res = 1.0 ! resolution

! Loop over several box sizes, i.e. halo sizes
main: do i=1,35

  ! 50, 100, 150
  bsz0 = (/ 100*i, 100*i, 100*i /) ! numbers of cells in CA space

  ! calculate the resolution and the actual phys dimensions of the box
  ! subroutine cgca_cadim( bsz, res, dm, ir, c, lres, ng )
  ! c - coarray sizes
  ! ir - coarray grid sizes
  bsz = bsz0
  call cgca_cadim( bsz, res, dm, ir, c, lres, ng )

  ! Check that the partition is sane
  if ( img .eq. 1 ) then
    if ( any(int(bsz) .ne. int(bsz0) ) ) then
      write (*,*)                                                    &
        "ERROR: bad decomposition - use a 'nicer' number of images"
```

```
      write (*,*) "ERROR: wanted         :", int(bsz0)
      write (*,*) "ERROR: but got instead:", int(bsz)
      error stop
    end if

    ! total number of cells in a coarray
    icells = int( c(1), kind=ilrg ) * int( c(2), kind=ilrg ) *         &
           int( c(3), kind=ilrg )

    ! total number of cells in the model
    mcells = icells * int( nimgs, kind=ilrg )

    ! Halo size
    hsize = c(1)*c(2)

    write ( *, "(8(a,i0),tr1,g10.3,tr1,g10.3,3(a,g10.3),a)" )          &
      "nimgs: ", nimgs, " (", c(1), "," , c(2), "," , c(3), ")[",      &
      ir(1), "," , ir(2), "," , ir(3), "] ", ng, qual, lres,           &
      " (", bsz(1), ",", bsz(2), ",", bsz(3), ")"
    write (*,'(a,i0,a)') "Each image has ",icells, " cells"
    write (*,'(a,i0,a)') "The model has ", mcells, " cells"
    write (*,'(a,i0,a)') "Halo size is ", hsize, " cells"
  end if

! Initialise MPI if not done already
call MPI_INITIALIZED( flag, ierr)
if ( .not. flag ) then
  call MPI_INIT( ierr )
  if ( img .eq. 1 ) write (*,*) "MPI not initialised, doing now!"
end if

  ! allocate space array
  !    space - CA array to allocate, with halos!
  !        c - array with space dimensions
  !        d - depth of the halo layer

  call ca_spalloc(    space, c, 1 )

  ! Very crude RND
  allocate( rnd( size(space, dim=1), size(space, dim=2),               &
                 size(space, dim=3) ) )
  call random_number( rnd )
  space = nint( rnd, kind=iarr ) ! Just 0 or 1

  ! allocate hx array coarrays, implicit sync all
  ! ir(3) - codimensions
  call ca_halloc( ir )

  ! Create MPI subarray types
  call ca_mpi_halo_type_create( space )

  ! do several HX, remote ops, sync inside
```

```
  ! Make sure the timer covers the slowest image
  sync all
  time1 = cgca_benchtime()
  do iter=1,10
    call ca_mpi_hx_all( space )
  end do
  ! Make sure the timer covers the slowest image
  sync all
  time2 = cgca_benchtime()
  if (img .eq. 1) write (*,*) "Halo, cells:", hsize, ". Time, s:",     &
                              time2-time1

  ! deallocate arrays
  call ca_hdalloc
  deallocate( space )
  deallocate( rnd )

  ! free halo types
  call ca_mpi_halo_type_free

end do main

if (img .eq. 1 ) write (*,*) "PASS"

end program hxvn_timing_mpi
```

49 tests/ising

[Unit tests]

NAME

ising

SYNOPSIS

```
!$Id: ising.f90 561 2018-10-14 20:48:19Z mexas $

program ising
```

PURPOSE

Test ising magnetisation Halo coarrays only

DESCRIPTION

See ca_kernel_ising (1.15) and related routines for details. Note that I use a reproducible RND seed and generate a single sequence of RND values for the whole CA model. Thus the results must be exactly reproducible on any number of images. I include the reference value for the final magnetisation (unscaled, integer). If the test does not produce the same value, it fails. However... the ref magnetisation value is obtained here with gfortran7. It is possible (likely?) that other compliers will produce a different sequence of RND from the same seed. In such cases users need to replace the ref value accordingly.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

casup (3)

USED BY

Part of casup (3) test suite

SOURCE

```
use casup

implicit none

integer( kind=iarr ), parameter :: huge_iarr = huge(0_iarr)

! Reference values for different compilers for magnet_ref,
! magnetisation at the end of simulation
!integer( kind=iarr ), parameter :: magnet_ref = 863379 ! gfortran7
integer( kind=iarr ), parameter :: magnet_ref = 864070 ! Cray

real( kind=rdef ) :: &
  qual,              & ! quality
  bsz0(3),           & ! the given "box" size
```

```
  bsz(3),            & ! updated "box" size
  dm,                & ! mean grain size, linear dim, phys units
  lres,              & ! linear resolution, cells per unit of length
  res                  ! resolutions, cells per grain

integer( kind=iarr ), allocatable :: space(:,:,:),                    &
  space0(:,:,:)

integer( kind=idef ) :: ir(3), nimgs, img, ng, c(3) ! coarray dimensions

integer( kind=ilrg ) :: icells, mcells

!real, allocatable :: space_ini(:,:,:), rnd_array(:)

integer :: i, iter, seed_size, run
integer( kind=ilrg) :: energy0, energy1, energy2, magnet0, magnet1,    &
  magnet2
integer, allocatable :: seed_array(:)

! real :: time1, time2

!*********************************************************************72
! first executable statement

 dm = 1.0 ! Linear "size" of one spin cell
res = 1.0 ! resolution, CA cells per spin

! Read the box size from command line
call ca_cmd_real( n=3, data=bsz0 )

! When dm=res=1, then bsz0 is simply CA dimensions in cells!
!bsz0 = (/ 1.2e2, 1.2e2, 1.2e2 /) ! dimensions of the CA model

  img = this_image()
nimgs = num_images()

! each image calculates the coarray grid dimensions
call cgca_gdim( nimgs, ir, qual )

! calculate the resolution and the actual phys dimensions of the box
! subroutine cgca_cadim( bsz, res, dm, ir, c, lres, ng )
! c - coarray sizes
! ir - coarray grid sizes
bsz = bsz0
call cgca_cadim( bsz, res, dm, ir, c, lres, ng )

! do a check on image 1
if ( img .eq. 1 ) then
  write (*,*) "running on", nimgs, "images in a 3D grid"
  write (*,*) "iarr kind:", iarr, "huge(0_iarr):", huge_iarr

  ! Check that the partition is sane
```

```
  if ( any(int(bsz) .ne. int(bsz0) ) ) then
    write (*,*)                                                          &
      "ERROR: bad decomposition - use a 'nicer' number of images"
    write (*,*) "ERROR: wanted         :", int(bsz0)
    write (*,*) "ERROR: but got instead:", int(bsz)
    error stop
  end if

  ! total number of cells in a coarray
  icells = product( int( c, kind=ilrg ) )

  ! total number of cells in the model
  mcells = icells * int( nimgs, kind=ilrg )

  write ( *, "(8(a,i0),tr1,g10.3,tr1,g10.3,3(a,g10.3),a)" )              &
    "nimgs: ", nimgs, " (", c(1), "," , c(2), "," , c(3), ")[",          &
    ir(1), "," , ir(2), "," , ir(3), "] ", ng, qual, lres,               &
    " (", bsz(1), ",", bsz(2), ",", bsz(3), ")"
     write (*,'(a,i0,a)') "Each image has ",icells, " cells"
     write (*,'(a,i0,a)') "The model has ", mcells, " cells"

  ! In this test sum over all cells on an image is done, so the kind
  ! must be big enough to contain the total number of cells
  ! on an image.
  if ( icells .gt. huge_iarr ) then
    write (*,*) "ERROR: number of cells on an image:", icells,           &
      "is greater than huge(0_iarr)"
    error stop
  end if

end if

! allocate space arrays
!    space - CA array to allocate, with halos!
!        c - array with space dimensions
!        d - depth of the halo layer

call ca_spalloc( space,  c, 1 )
call ca_spalloc( space0, c, 1 )

! allocate hx arrays, implicit sync all
! mask_array is set inside too.
! ir(3) - codimensions
call ca_halloc( ir )

! Init RND
!call cgca_irs( debug = .false. )
! Use a reproducible RND here for verification
call random_seed( size = seed_size )
allocate( seed_array( seed_size ) )
seed_array = (/ (i, i=1,seed_size) /)
```

```
! Set space arrays
if (img .eq. 1) write (*,*) "RND, serial IO, etc. - wait..."
! Passing allocatable coarray into assumed-shape array, which is ok!
call ca_set_space_rnd( seed = seed_array, frac1=0.5, space = space )

! Calculate initial energy and magnetisation

! run=1 => ca_iter_tl
! run=2 => ca_iter_dc
! run=3 => ca_iter_omp
do run=1,3
  select case(run)
  case(1)
    call ca_ising_energy(     space = space, hx_sub = ca_hx_all,       &
      iter_sub = ca_iter_tl,  kernel = ca_kernel_ising_ener,           &
      energy = energy0, magnet = magnet0 )
  case(2)
    call ca_ising_energy(     space = space, hx_sub = ca_hx_all,       &
      iter_sub = ca_iter_dc,  kernel = ca_kernel_ising_ener,           &
      energy = energy1, magnet = magnet1 )
  case(3)
    call ca_ising_energy(     space = space, hx_sub = ca_hx_all,       &
      iter_sub = ca_iter_omp, kernel = ca_kernel_ising_ener,           &
      energy = energy2, magnet = magnet2 )
  end select
end do

if (img .eq. 1 ) then
  write (*,*) "Initial energy and magnetisation"
  write (*,*) "ca_iter_tl :", energy0, magnet0
  write (*,*) "ca_iter_dc :", energy1, magnet1
  write (*,*) "ca_iter_omp:", energy2, magnet2
  if ( energy0 .ne. energy1 .or. magnet0 .ne. magnet1 .or.             &
       energy0 .ne. energy2 .or. magnet0 .ne. magnet2 ) then
    write (*,*) "FAIL: ca_iter_tl, ca_iter_dc, ca_iter_omp differ"
    error stop
  else
    write (*,*) "PASS: ca_iter_tl, ca_iter_dc, ca_iter_omp agree"
  end if
end if

! save old space as space0
space0 = space

! run=1 => ca_iter_tl
! run=2 => ca_iter_dc
! run=3 => ca_iter_omp
main: do run=1,3

  ! Reset space to space0
  space = space0
```

```
! No IO here

! No timing
! here

  ! CA iterations
  loop: do iter = 1,100

    ! Check energy after every iter
    ! subroutine ca_run(    space, hx_sub, iter_sub, kernel, niter )
    select case(run)
    case(1)
      call ca_run(    space = space, hx_sub = ca_hx_all,               &
        iter_sub = ca_iter_tl,  kernel = ca_kernel_ising, niter = 1 )
      call ca_ising_energy(     space = space, hx_sub = ca_hx_all,     &
        iter_sub = ca_iter_tl,  kernel = ca_kernel_ising_ener,         &
        energy = energy1, magnet = magnet1 )
    case(2)
      call ca_run(    space = space, hx_sub = ca_hx_all,               &
        iter_sub = ca_iter_dc,  kernel = ca_kernel_ising, niter = 1 )
      call ca_ising_energy(     space = space, hx_sub = ca_hx_all,     &
        iter_sub = ca_iter_dc,  kernel = ca_kernel_ising_ener,         &
        energy = energy1, magnet = magnet1 )
    case(3)
      call ca_run(    space = space, hx_sub = ca_hx_all,               &
        iter_sub = ca_iter_omp, kernel = ca_kernel_ising, niter = 1 )
      call ca_ising_energy(     space = space, hx_sub = ca_hx_all,     &
        iter_sub = ca_iter_omp, kernel = ca_kernel_ising_ener,         &
        energy = energy1, magnet = magnet1 )
    end select

    if ( img .eq. 1 ) then
      if ( energy1 .ne. energy0 ) then
        write (*,*) "FAIL: energy0:", energy0, "energy1:", energy1
        error stop
      else
        if ( mod((iter-1), 100) .eq. 0 ) then
  !      write (*,"(a,i0,a,es18.6)") "Magnetisation_after_iter_",      &
  !        iter, ":", real(magnet1) / real(mcells)
          write (*,*) iter, real(magnet1) / real(mcells)
        end if
      end if
    end if

  end do loop

! no sync needed here
!
! no IO here
!
! no counter
! here
```

```
  if ( img .eq. 1 ) then
    select case(run)
      case(1)
        if ( magnet1 .eq. magnet_ref ) then
          write (*,*) "PASS: ca_iter_tl : final mag:", magnet1
! no timing here
        else
          write (*,"(2(a,i0))")                                          &
            "FAIL: ca_iter_tl : magnetisation ref value: ",              &
             magnet_ref, " my value: ", magnet1
        end if
      case(2)
        if ( magnet1 .eq. magnet_ref ) then
          write (*,*) "PASS: ca_iter_dc : final mag:", magnet1
! no timing here
        else
          write (*,"(2(a,i0))")                                          &
            "FAIL: ca_iter_dc : magnetisation ref value: ",              &
             magnet_ref, " my value: ", magnet1
        end if
      case(3)
        if ( magnet1 .eq. magnet_ref ) then
          write (*,*) "PASS: ca_iter_omp: final mag:", magnet1
! no timing here
        else
          write (*,"(2(a,i0))")                                          &
            "FAIL: ca_iter_omp: magnetisation ref value: ",              &
             magnet_ref, " my value: ", magnet1
        end if
    end select

  end if

end do main

! deallocate halos, implicit sync all
call ca_hdalloc

! deallocate space
deallocate( space )
deallocate( space0 )

end program ising
```

50 tests/ising_1D

[Unit tests]

NAME

ising_1D

SYNOPSIS

```
!$Id: ising_1D.f90 561 2018-10-14 20:48:19Z mexas $

program ising_1D
```

PURPOSE

Test ising (49) magnetisation Halo coarrays only, 1D co-rank version + sync all

DESCRIPTION

See ca_kernel_ising (1.15) and related routines for details. Note that I use a reproducible RND seed and generate a single sequence of RND values for the whole CA model. Thus the results must be exactly reproducible on any number of images. I include the reference value for the final magnetisation (unscaled, integer). If the test does not produce the same value, it fails. However... the ref magnetisation value is obtained here with gfortran7. It is possible (likely?) that other compliers will produce a different sequence of RND from the same seed. In such cases users need to replace the ref value accordingly.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

casup (3)

USED BY

Part of casup (3) test suite

SOURCE

```
use casup

implicit none

integer( kind=iarr ), parameter :: huge_iarr = huge(0_iarr)

! Reference values for different compilers for magnet_ref,
! magnetisation at the end of simulation
!integer( kind=iarr ), parameter :: magnet_ref = 863379 ! gfortran7
integer( kind=iarr ), parameter :: magnet_ref = 864070 ! Cray

real( kind=rdef ) :: bsz(3)            ! "box" size

integer( kind=iarr ), allocatable :: space(:,:,:),                    &
```

```
  space0(:,:,:)

integer( kind=idef ) :: nimgs, img, c(3) ! coarray dimensions

integer( kind=ilrg ) :: icells, mcells

!real, allocatable :: space_ini(:,:,:), rnd_array(:)

integer :: i, iter, seed_size, run
integer( kind=ilrg) :: energy0, energy1, energy2, magnet0, magnet1,    &
  magnet2
integer, allocatable :: seed_array(:)

! real :: time1, time2

!*********************************************************************72
! first executable statement

! Read the box size from command line
call ca_cmd_real( n=3, data=bsz )

! bsz is simply CA dimensions in cells! but in real, not integer!
!bsz = (/ 1.2e2, 1.2e2, 1.2e2 /) ! dimensions of the CA model

  img = this_image()
nimgs = num_images()

! do a check on image 1
if ( img .eq. 1 ) then
  write (*,*) "running on", nimgs, "images in a 1D grid"
  write (*,*) "iarr kind:", iarr, "huge(0_iarr):", huge_iarr

  ! In this test space is assigned image numbers - must be big enough
  ! integer kind to avoid integer overflow.
  if ( nimgs .gt. huge_iarr ) then
    write (*,*) "ERROR: num_images(): ", nimgs,              &
      " is greater than huge(0_iarr)"
    error stop
  end if
end if

! Only a single codimension, corank 1
c = int(bsz)
c(3) = int(bsz(3))/nimgs

! Check that the partition is sane
if ( img .eq. 1 ) then
  if ( c(3)*nimgs .ne. int(bsz(3)) ) then
    write (*,*)                                                  &
      "ERROR: bad decomposition: bsz(3) must be divisible by nimgs"
    write (*,*) "ERROR: bsz(3):", int(bsz(3))
    write (*,*) "ERROR: nimgs :", nimgs
```

```
    error stop
  end if

  ! total number of cells in a coarray
  icells = product( int( c, kind=ilrg ) )

  ! total number of cells in the model
  mcells = icells * int( nimgs, kind=ilrg )

  write ( *, "(5(a,i0),3(a,g10.3),a)" )                                &
    "nimgs: ", nimgs, " (", c(1), "," , c(2), "," , c(3),              &
    ")[", nimgs, "] (", bsz(1), ",", bsz(2), ",", bsz(3), ")"
     write (*,'(a,i0,a)') "Each image has ",icells, " cells"
     write (*,'(a,i0,a)') "The model has ", mcells, " cells"

  ! In this test sum over all cells on an image is done, so the kind
  ! must be big enough to contain the total number of cells
  ! on an image.
  if ( icells .gt. huge_iarr ) then
    write (*,*) "ERROR: number of cells on an image:", icells,         &
      "is greater than huge(0_iarr)"
    error stop
  end if

end if

! allocate space arrays
!    space - CA array to allocate, with halos!
!        c - array with space dimensions
!        d - depth of the halo layer

call ca_spalloc( space,  c, 1 )
call ca_spalloc( space0, c, 1 )

! allocate hx arrays, implicit sync all
! mask_array is set inside too.
call ca_1D_halloc

! Init RND
!call cgca_irs( debug = .false. )
! Use a reproducible RND here for verification
call random_seed( size = seed_size )
allocate( seed_array( seed_size ) )
seed_array = (/ (i, i=1,seed_size) /)

! Set space arrays
if (img .eq. 1) write (*,*) "RND, serial IO, etc. - wait..."
! Passing allocatable coarray into assumed-shape array, which is ok!
call ca_set_space_rnd( seed = seed_array, frac1=0.5, space = space )

! Calculate initial energy and magnetisation
```

```
! run=1 => ca_iter_tl
! run=2 => ca_iter_dc
! run=3 => ca_iter_omp
do run=1,3
  select case(run)
  case(1)
    call ca_ising_energy_col( space = space, hx_sub = ca_1D_hx_sall,   &
      iter_sub = ca_iter_tl,  kernel = ca_kernel_ising_ener,           &
      energy = energy0, magnet = magnet0 )
  case(2)
    call ca_ising_energy_col( space = space, hx_sub = ca_1D_hx_sall,   &
      iter_sub = ca_iter_dc,  kernel = ca_kernel_ising_ener,           &
      energy = energy1, magnet = magnet1 )
  case(3)
    call ca_ising_energy_col( space = space, hx_sub = ca_1D_hx_sall,   &
      iter_sub = ca_iter_omp, kernel = ca_kernel_ising_ener,           &
      energy = energy2, magnet = magnet2 )
  end select
end do

if (img .eq. 1 ) then
  write (*,*) "Initial energy and magnetisation"
  write (*,*) "ca_iter_tl :", energy0, magnet0
  write (*,*) "ca_iter_dc :", energy1, magnet1
  write (*,*) "ca_iter_omp:", energy2, magnet2
  if ( energy0 .ne. energy1 .or. magnet0 .ne. magnet1 .or.             &
       energy0 .ne. energy2 .or. magnet0 .ne. magnet2 ) then
    write (*,*) "FAIL: ca_iter_tl, ca_iter_dc, ca_iter_omp differ"
    error stop
  else
    write (*,*) "PASS: ca_iter_tl, ca_iter_dc, ca_iter_omp agree"
  end if
end if

! save old space as space0
space0 = space

! run=1 => ca_iter_tl
! run=2 => ca_iter_dc
! run=3 => ca_iter_omp
main: do run=1,3

  ! Reset space to space0
  space = space0

! No IO here

! No timing
! here

  ! CA iterations
  loop: do iter = 1,100
```

```
    ! Check energy after every iter
    ! subroutine ca_run(    space, hx_sub, iter_sub, kernel, niter )
    select case(run)
    case(1)
      call ca_run(    space = space, hx_sub = ca_1D_hx_sall,           &
        iter_sub = ca_iter_tl,  kernel = ca_kernel_ising, niter = 1 )
      call ca_ising_energy_col( space = space, hx_sub = ca_1D_hx_sall, &
        iter_sub = ca_iter_tl,  kernel = ca_kernel_ising_ener,         &
        energy = energy1, magnet = magnet1 )
    case(2)
      call ca_run(    space = space, hx_sub = ca_1D_hx_sall,           &
        iter_sub = ca_iter_dc,  kernel = ca_kernel_ising, niter = 1 )
      call ca_ising_energy_col( space = space, hx_sub = ca_1D_hx_sall, &
        iter_sub = ca_iter_dc,  kernel = ca_kernel_ising_ener,         &
        energy = energy1, magnet = magnet1 )
    case(3)
      call ca_run(    space = space, hx_sub = ca_1D_hx_sall,           &
        iter_sub = ca_iter_omp, kernel = ca_kernel_ising, niter = 1 )
      call ca_ising_energy_col( space = space, hx_sub = ca_1D_hx_sall, &
        iter_sub = ca_iter_omp, kernel = ca_kernel_ising_ener,         &
        energy = energy1, magnet = magnet1 )
    end select

    if ( img .eq. 1 ) then
      if ( energy1 .ne. energy0 ) then
        write (*,*) "FAIL: energy0:", energy0, "energy1:", energy1
        error stop
      else
        if ( mod((iter-1), 100) .eq. 0 ) then
  !      write (*,"(a,i0,a,es18.6)") "Magnetisation_after_iter_",      &
  !        iter, ":", real(magnet1) / real(mcells)
         write (*,*) iter, real(magnet1) / real(mcells)
        end if
      end if
    end if

  end do loop

! no sync needed here
!
! no IO here
!
! no counter
! here

  if ( img .eq. 1 ) then
    select case(run)
      case(1)
        if ( magnet1 .eq. magnet_ref ) then
          write (*,*) "PASS: ca_iter_tl : final mag:", magnet1
! no timing here
```

```
        else
          write (*,"(2(a,i0))")                                          &
            "FAIL: ca_iter_tl : magnetisation ref value: ",              &
             magnet_ref, " my value: ", magnet1
        end if
      case(2)
        if ( magnet1 .eq. magnet_ref ) then
          write (*,*) "PASS: ca_iter_dc : final mag:", magnet1
! no timing here
        else
          write (*,"(2(a,i0))")                                          &
            "FAIL: ca_iter_dc : magnetisation ref value: ",              &
             magnet_ref, " my value: ", magnet1
        end if
      case(3)
        if ( magnet1 .eq. magnet_ref ) then
          write (*,*) "PASS: ca_iter_omp: final mag:", magnet1
! no timing here
        else
          write (*,"(2(a,i0))")                                          &
            "FAIL: ca_iter_omp: magnetisation ref value: ",              &
             magnet_ref, " my value: ", magnet1
        end if
    end select

  end if

end do main

! deallocate halos, implicit sync all
call ca_1D_hdalloc

! deallocate space
deallocate( space )
deallocate( space0 )

end program ising_1D
```

51 tests/ising_co

[Unit tests]

NAME

ising_co

SYNOPSIS

```
!$Id: ising_co.f90 561 2018-10-14 20:48:19Z mexas $

program ising_co
```

PURPOSE

Test ising (49) magnetisation + timing. Use whole model coarrays, not just coarray halos.

DESCRIPTION

See ca_kernel_ising (1.15) and related routines for details. Note that I use a reproducible RND seed and generate a single sequence of RND values for the whole CA model. Thus the results must be exactly reproducible on any number of images. I include the reference value for the final magnetisation (unscaled, integer). If the test does not produce the same value, it fails. However... the ref magnetisation value is obtained here with gfortran7. It is possible (likely?) that other compliers will produce a different sequence of RND from the same seed. In such cases users need to replace the ref value accordingly.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

casup (3)

USED BY

Part of casup (3) test suite

SOURCE

```
use casup

implicit none

integer( kind=iarr ), parameter :: huge_iarr = huge(0_iarr)

! Reference values for different compilers for magnet_ref,
! magnetisation at the end of simulation
!integer( kind=iarr ), parameter :: magnet_ref = 863379 ! gfortran7
integer( kind=iarr ), parameter :: magnet_ref = 864070 ! Cray

real( kind=rdef ) :: &
  qual,              & ! quality
  bsz0(3),           & ! the given "box" size
```

```
  bsz(3),            & ! updated "box" size
  dm,                & ! mean grain size, linear dim, phys units
  lres,              & ! linear resolution, cells per unit of length
  res                  ! resolutions, cells per grain

integer( kind=iarr ), allocatable :: space(:,:,:) [:,:,:],             &
  space0(:,:,:) [:,:,:]

integer( kind=idef ) :: ir(3), nimgs, img, ng, c(3) ! coarray dimensions

integer( kind=ilrg ) :: icells, mcells

!real, allocatable :: space_ini(:,:,:), rnd_array(:)

integer :: i, iter, seed_size, run
integer( kind=ilrg) :: energy0, energy1, energy2, magnet0, magnet1,    &
  magnet2
integer, allocatable :: seed_array(:)

real :: time1, time2

!*********************************************************************72
! first executable statement

 dm = 1.0 ! Linear "size" of one spin cell
res = 1.0 ! resolution, CA cells per spin

! Read the box size from command line
call ca_cmd_real( n=3, data=bsz0 )

! When dm=res=1, then bsz0 is simply CA dimensions in cells!
!bsz0 = (/ 1.2e2, 1.2e2, 1.2e2 /) ! dimensions of the CA model

  img = this_image()
nimgs = num_images()

! each image calculates the coarray grid dimensions
call cgca_gdim( nimgs, ir, qual )

! calculate the resolution and the actual phys dimensions of the box
! subroutine cgca_cadim( bsz, res, dm, ir, c, lres, ng )
! c - coarray sizes
! ir - coarray grid sizes
bsz = bsz0
call cgca_cadim( bsz, res, dm, ir, c, lres, ng )

! do a check on image 1
if ( img .eq. 1 ) then
  write (*,*) "running on", nimgs, "images in a 3D grid"
  write (*,*) "iarr kind:", iarr, "huge(0_iarr):", huge_iarr

  ! Check that the partition is sane
```

```
  if ( any(int(bsz) .ne. int(bsz0) ) ) then
    write (*,*)                                                          &
      "ERROR: bad decomposition - use a 'nicer' number of images"
    write (*,*) "ERROR: wanted         :", int(bsz0)
    write (*,*) "ERROR: but got instead:", int(bsz)
    error stop
  end if

  ! total number of cells in a coarray
  icells = product( int( c, kind=ilrg ) )

  ! total number of cells in the model
  mcells = icells * int( nimgs, kind=ilrg )

  write ( *, "(8(a,i0),tr1,g10.3,tr1,g10.3,3(a,g10.3),a)" )              &
    "nimgs: ", nimgs, " (", c(1), "," , c(2), "," , c(3), ")[",          &
    ir(1), "," , ir(2), "," , ir(3), "] ", ng, qual, lres,               &
    " (", bsz(1), ",", bsz(2), ",", bsz(3), ")"
     write (*,'(a,i0,a)') "Each image has ",icells, " cells"
     write (*,'(a,i0,a)') "The model has ", mcells, " cells"

  ! In this test sum over all cells on an image is done, so the kind
  ! must be big enough to contain the total number of cells
  ! on an image.
  if ( icells .gt. huge_iarr ) then
    write (*,*) "ERROR: number of cells on an image:", icells,           &
      "is greater than huge(0_iarr)"
    error stop
  end if

end if

! allocate space arrays
!    space - CA array to allocate, with halos!
!        c - array with space dimensions
!        d - depth of the halo layer
!       ir - codimensions
call ca_co_spalloc( space,  c, 1, ir )
call ca_co_spalloc( space0, c, 1, ir )

! No need
! for
! HX
! arrays

! Init RND
!call cgca_irs( debug = .false. )
! Use a reproducible RND here for verification
call random_seed( size = seed_size )
allocate( seed_array( seed_size ) )
seed_array = (/ (i, i=1,seed_size) /)
```

```
! Set space arrays
if (img .eq. 1) write (*,*) "RND, serial IO, etc. - wait..."
! Passing allocatable coarray into assumed-shape array, which is ok!
call ca_set_space_rnd( seed = seed_array, frac1=0.5, space = space )

! Calculate initial energy and magnetisation

! run=1 => ca_iter_tl
! run=2 => ca_iter_dc
! run=3 => ca_iter_omp
do run=1,3
  select case(run)
  case(1)
    call ca_co_ising_energy( space = space, hx_sub = ca_co_hx_all,     &
      iter_sub = ca_iter_tl,  kernel = ca_kernel_ising_ener,           &
      energy = energy0, magnet = magnet0 )
  case(2)
    call ca_co_ising_energy( space = space, hx_sub = ca_co_hx_all,     &
      iter_sub = ca_iter_dc,  kernel = ca_kernel_ising_ener,           &
      energy = energy1, magnet = magnet1 )
  case(3)
    call ca_co_ising_energy( space = space, hx_sub = ca_co_hx_all,     &
      iter_sub = ca_iter_omp, kernel = ca_kernel_ising_ener,           &
      energy = energy2, magnet = magnet2 )
  end select
end do

if (img .eq. 1 ) then
  write (*,*) "Initial energy and magnetisation"
  write (*,*) "ca_iter_tl :", energy0, magnet0
  write (*,*) "ca_iter_dc :", energy1, magnet1
  write (*,*) "ca_iter_omp:", energy2, magnet2
  if ( energy0 .ne. energy1 .or. magnet0 .ne. magnet1 .or.             &
       energy0 .ne. energy2 .or. magnet0 .ne. magnet2 ) then
    write (*,*) "FAIL: ca_iter_tl, ca_iter_dc, ca_iter_omp differ"
    error stop
  else
    write (*,*) "PASS: ca_iter_tl, ca_iter_dc, ca_iter_omp agree"
  end if
end if

! save old space as space0
space0 = space

! run=1 => ca_iter_tl
! run=2 => ca_iter_dc
! run=3 => ca_iter_omp
main: do run=1,3

  ! Reset space to space0
  space = space0
```

```
call ca_co_naive_io( space, 'start.raw' )

  ! Start counter
  if ( img .eq. 1 ) call cpu_time( time1 )

  ! CA iterations
  loop: do iter = 1,100

    ! Check energy after every iter
    ! subroutine ca_co_run( space, hx_sub, iter_sub, kernel, niter )
    select case(run)
    case(1)
      call ca_co_run( space = space, hx_sub = ca_co_hx_all,            &
        iter_sub = ca_iter_tl,  kernel = ca_kernel_ising, niter = 1 )
      call ca_co_ising_energy( space = space, hx_sub = ca_co_hx_all,   &
        iter_sub = ca_iter_tl,  kernel = ca_kernel_ising_ener,         &
        energy = energy1, magnet = magnet1 )
    case(2)
      call ca_co_run( space = space, hx_sub = ca_co_hx_all,            &
        iter_sub = ca_iter_dc,  kernel = ca_kernel_ising, niter = 1 )
      call ca_co_ising_energy( space = space, hx_sub = ca_co_hx_all,   &
        iter_sub = ca_iter_dc,  kernel = ca_kernel_ising_ener,         &
        energy = energy1, magnet = magnet1 )
    case(3)
      call ca_co_run( space = space, hx_sub = ca_co_hx_all,            &
        iter_sub = ca_iter_omp, kernel = ca_kernel_ising, niter = 1 )
      call ca_co_ising_energy( space = space, hx_sub = ca_co_hx_all,   &
        iter_sub = ca_iter_omp, kernel = ca_kernel_ising_ener,         &
        energy = energy1, magnet = magnet1 )
    end select

    if ( img .eq. 1 ) then
      if ( energy1 .ne. energy0 ) then
        write (*,*) "FAIL: energy0:", energy0, "energy1:", energy1
        error stop
      else
        if ( mod((iter-1), 100) .eq. 0 ) then
!       write (*,"(a,i0,a,es18.6)") "Magnetisation_after_iter_",      &
!         iter, ":", real(magnet1) / real(mcells)
          write (*,*) iter, real(magnet1) / real(mcells)
        end if
      end if
    end if

  end do loop

  sync all

  call ca_co_naive_io( space, 'end.raw' )

  ! Stop counter
  if ( img .eq. 1 ) call cpu_time( time2 )
```

```
  if ( img .eq. 1 ) then
    select case(run)
      case(1)
        if ( magnet1 .eq. magnet_ref ) then
          write (*,*) "PASS: ca_iter_tl : final mag:", magnet1
          write (*,*) "TIME ca_iter_tl, s :", time2-time1
        else
          write (*,"(2(a,i0))")                                          &
            "FAIL: ca_iter_tl : magnetisation ref value: ",              &
             magnet_ref, " my value: ", magnet1
        end if
      case(2)
        if ( magnet1 .eq. magnet_ref ) then
          write (*,*) "PASS: ca_iter_dc : final mag:", magnet1
          write (*,*) "TIME ca_iter_dc, s :", time2-time1
        else
          write (*,"(2(a,i0))")                                          &
            "FAIL: ca_iter_dc : magnetisation ref value: ",              &
             magnet_ref, " my value: ", magnet1
        end if
      case(3)
        if ( magnet1 .eq. magnet_ref ) then
          write (*,*) "PASS: ca_iter_omp: final mag:", magnet1
          write (*,*) "TIME ca_iter_omp, s:", time2-time1
        else
          write (*,"(2(a,i0))")                                          &
            "FAIL: ca_iter_omp: magnetisation ref value: ",              &
             magnet_ref, " my value: ", magnet1
        end if
    end select

  end if

end do main

! No halos
! to deallocate

! deallocate space
deallocate( space )
deallocate( space0 )

end program ising_co
```

52 tests/ising_col

[Unit tests]

NAME

ising_col

SYNOPSIS

```
!$Id: ising_col.f90 561 2018-10-14 20:48:19Z mexas $

program ising_col
```

PURPOSE

Test ising (49) magnetisation + timing This test uses coarray collectives.

DESCRIPTION

See ca_kernel_ising (1.15) and related routines for details. Note that I use a reproducible RND seed and generate a single sequence of RND values for the whole CA model. Thus the results must be exactly reproducible on any number of images. I include the reference value for the final magnetisation (unscaled, integer). If the test does not produce the same value, it fails. However... the ref magnetisation value is obtained here with gfortran7. It is possible (likely?) that other compliers will produce a different sequence of RND from the same seed. In such cases users need to replace the ref value accordingly.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

casup (3)

USED BY

Part of casup (3) test suite

SOURCE

```
use casup

implicit none

integer( kind=iarr ), parameter :: huge_iarr = huge(0_iarr)

! Reference values for different compilers for magnet_ref,
! magnetisation at the end of simulation
!integer( kind=iarr ), parameter :: magnet_ref = 863379 ! gfortran7
integer( kind=iarr ), parameter :: magnet_ref = 864070 ! Cray

real( kind=rdef ) :: &
  qual,              & ! quality
  bsz0(3),           & ! the given "box" size
```

```
  bsz(3),            & ! updated "box" size
  dm,                & ! mean grain size, linear dim, phys units
  lres,              & ! linear resolution, cells per unit of length
  res                  ! resolutions, cells per grain

integer( kind=iarr ), allocatable :: space(:,:,:),                    &
  space0(:,:,:)

integer( kind=idef ) :: ir(3), nimgs, img, ng, c(3) ! coarray dimensions

integer( kind=ilrg ) :: icells, mcells

!real, allocatable :: space_ini(:,:,:), rnd_array(:)

integer :: i, iter, seed_size, run
integer( kind=ilrg) :: energy0, energy1, energy2, magnet0, magnet1,    &
  magnet2
integer, allocatable :: seed_array(:)

real :: time1, time2

!*********************************************************************72
! first executable statement

 dm = 1.0 ! Linear "size" of one spin cell
res = 1.0 ! resolution, CA cells per spin

! Read the box size from command line
call ca_cmd_real( n=3, data=bsz0 )

! When dm=res=1, then bsz0 is simply CA dimensions in cells!
!bsz0 = (/ 1.2e2, 1.2e2, 1.2e2 /) ! dimensions of the CA model

  img = this_image()
nimgs = num_images()

! each image calculates the coarray grid dimensions
call cgca_gdim( nimgs, ir, qual )

! calculate the resolution and the actual phys dimensions of the box
! subroutine cgca_cadim( bsz, res, dm, ir, c, lres, ng )
! c - coarray sizes
! ir - coarray grid sizes
bsz = bsz0
call cgca_cadim( bsz, res, dm, ir, c, lres, ng )

! do a check on image 1
if ( img .eq. 1 ) then
  write (*,*) "running on", nimgs, "images in a 3D grid"
  write (*,*) "iarr kind:", iarr, "huge(0_iarr):", huge_iarr

  ! Check that the partition is sane
```

```
  if ( any(int(bsz) .ne. int(bsz0) ) ) then
    write (*,*)                                                          &
      "ERROR: bad decomposition - use a 'nicer' number of images"
    write (*,*) "ERROR: wanted         :", int(bsz0)
    write (*,*) "ERROR: but got instead:", int(bsz)
    error stop
  end if

  ! total number of cells in a coarray
  icells = product( int( c, kind=ilrg ) )

  ! total number of cells in the model
  mcells = icells * int( nimgs, kind=ilrg )

  write ( *, "(8(a,i0),tr1,g10.3,tr1,g10.3,3(a,g10.3),a)" )              &
    "nimgs: ", nimgs, " (", c(1), "," , c(2), "," , c(3), ")[",          &
    ir(1), "," , ir(2), "," , ir(3), "] ", ng, qual, lres,               &
    " (", bsz(1), ",", bsz(2), ",", bsz(3), ")"
     write (*,'(a,i0,a)') "Each image has ",icells, " cells"
     write (*,'(a,i0,a)') "The model has ", mcells, " cells"

  ! In this test sum over all cells on an image is done, so the kind
  ! must be big enough to contain the total number of cells
  ! on an image.
  if ( icells .gt. huge_iarr ) then
    write (*,*) "ERROR: number of cells on an image:", icells,           &
      "is greater than huge(0_iarr)"
    error stop
  end if

end if

! allocate space arrays
!    space - CA array to allocate, with halos!
!        c - array with space dimensions
!        d - depth of the halo layer

call ca_spalloc( space,  c, 1 )
call ca_spalloc( space0, c, 1 )

! allocate hx arrays, implicit sync all
! mask_array is set inside too.
! ir(3) - codimensions
call ca_halloc( ir )

! Init RND
!call cgca_irs( debug = .false. )
! Use a reproducible RND here for verification
call random_seed( size = seed_size )
allocate( seed_array( seed_size ) )
seed_array = (/ (i, i=1,seed_size) /)
```

```
! Set space arrays
if (img .eq. 1) write (*,*) "RND, serial IO, etc. - wait..."
! Passing allocatable coarray into assumed-shape array, which is ok!
call ca_set_space_rnd( seed = seed_array, frac1=0.5, space = space )

! Calculate initial energy and magnetisation

! run=1 => ca_iter_tl
! run=2 => ca_iter_dc
! run=3 => ca_iter_omp
do run=1,3
  select case(run)
  case(1)
    call ca_ising_energy_col( space = space, hx_sub = ca_hx_all,       &
      iter_sub = ca_iter_tl,  kernel = ca_kernel_ising_ener,           &
      energy = energy0, magnet = magnet0 )
  case(2)
    call ca_ising_energy_col( space = space, hx_sub = ca_hx_all,       &
      iter_sub = ca_iter_dc,  kernel = ca_kernel_ising_ener,           &
      energy = energy1, magnet = magnet1 )
  case(3)
    call ca_ising_energy_col( space = space, hx_sub = ca_hx_all,       &
      iter_sub = ca_iter_omp, kernel = ca_kernel_ising_ener,           &
      energy = energy2, magnet = magnet2 )
  end select
end do

if (img .eq. 1 ) then
  write (*,*) "Initial energy and magnetisation"
  write (*,*) "ca_iter_tl :", energy0, magnet0
  write (*,*) "ca_iter_dc :", energy1, magnet1
  write (*,*) "ca_iter_omp:", energy2, magnet2
  if ( energy0 .ne. energy1 .or. magnet0 .ne. magnet1 .or.             &
       energy0 .ne. energy2 .or. magnet0 .ne. magnet2 ) then
    write (*,*) "FAIL: ca_iter_tl, ca_iter_dc, ca_iter_omp differ"
    error stop
  else
    write (*,*) "PASS: ca_iter_tl, ca_iter_dc, ca_iter_omp agree"
  end if
end if

! save old space as space0
space0 = space

! run=1 => ca_iter_tl
! run=2 => ca_iter_dc
! run=3 => ca_iter_omp
main: do run=1,3

  ! Reset space to space0
  space = space0
```

```
! No IO here

  ! Start counter
  if ( img .eq. 1 ) call cpu_time( time1 )

  ! CA iterations
  loop: do iter = 1,100

    ! Check energy after every iter
    ! subroutine ca_run(    space, hx_sub, iter_sub, kernel, niter )
    select case(run)
    case(1)
      call ca_run(    space = space, hx_sub = ca_hx_all,               &
        iter_sub = ca_iter_tl,  kernel = ca_kernel_ising, niter = 1 )
      call ca_ising_energy_col( space = space, hx_sub = ca_hx_all,     &
        iter_sub = ca_iter_tl,  kernel = ca_kernel_ising_ener,         &
        energy = energy1, magnet = magnet1 )
    case(2)
      call ca_run(    space = space, hx_sub = ca_hx_all,               &
        iter_sub = ca_iter_dc,  kernel = ca_kernel_ising, niter = 1 )
      call ca_ising_energy_col( space = space, hx_sub = ca_hx_all,     &
        iter_sub = ca_iter_dc,  kernel = ca_kernel_ising_ener,         &
        energy = energy1, magnet = magnet1 )
    case(3)
      call ca_run(    space = space, hx_sub = ca_hx_all,               &
        iter_sub = ca_iter_omp, kernel = ca_kernel_ising, niter = 1 )
      call ca_ising_energy_col( space = space, hx_sub = ca_hx_all,     &
        iter_sub = ca_iter_omp, kernel = ca_kernel_ising_ener,         &
        energy = energy1, magnet = magnet1 )
    end select

    if ( img .eq. 1 ) then
      if ( energy1 .ne. energy0 ) then
        write (*,*) "FAIL: energy0:", energy0, "energy1:", energy1
        error stop
      else

  !      write (*,"(a,i0,a,es18.6)") "Magnetisation_after_iter_",      &
  !        iter, ":", real(magnet1) / real(mcells)


      end if
    end if

  end do loop

! no sync needed here
!
! no IO here
!
  ! Stop counter
  if ( img .eq. 1 ) call cpu_time( time2 )
```

```
  if ( img .eq. 1 ) then
    select case(run)
      case(1)
        if ( magnet1 .eq. magnet_ref ) then
          write (*,*) "PASS: ca_iter_tl : final mag:", magnet1
          write (*,*) "TIME ca_iter_tl, s :", time2-time1
        else
          write (*,"(2(a,i0))")                                          &
            "FAIL: ca_iter_tl : magnetisation ref value: ",              &
             magnet_ref, " my value: ", magnet1
        end if
      case(2)
        if ( magnet1 .eq. magnet_ref ) then
          write (*,*) "PASS: ca_iter_dc : final mag:", magnet1
          write (*,*) "TIME ca_iter_dc, s :", time2-time1
        else
          write (*,"(2(a,i0))")                                          &
            "FAIL: ca_iter_dc : magnetisation ref value: ",              &
             magnet_ref, " my value: ", magnet1
        end if
      case(3)
        if ( magnet1 .eq. magnet_ref ) then
          write (*,*) "PASS: ca_iter_omp: final mag:", magnet1
          write (*,*) "TIME ca_iter_omp, s:", time2-time1
        else
          write (*,"(2(a,i0))")                                          &
            "FAIL: ca_iter_omp: magnetisation ref value: ",              &
             magnet_ref, " my value: ", magnet1
        end if
    end select

  end if

end do main

! deallocate halos, implicit sync all
call ca_hdalloc

! deallocate space
deallocate( space )
deallocate( space0 )

end program ising_col
```

53 tests/ising_glbar

[Unit tests]

NAME

ising_glbar

SYNOPSIS

```
!$Id: ising_glbar.f90 561 2018-10-14 20:48:19Z mexas $

program ising_glbar
```

PURPOSE

Test ising (49) magnetisation Halo coarrays only, global barrier version - sync all

DESCRIPTION

See ca_kernel_ising (1.15) and related routines for details. Note that I use a reproducible RND seed and generate a single sequence of RND values for the whole CA model. Thus the results must be exactly reproducible on any number of images. I include the reference value for the final magnetisation (unscaled, integer). If the test does not produce the same value, it fails. However... the ref magnetisation value is obtained here with gfortran7. It is possible (likely?) that other compliers will produce a different sequence of RND from the same seed. In such cases users need to replace the ref value accordingly.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

casup (3)

USED BY

Part of casup (3) test suite

SOURCE

```
use casup

implicit none

integer( kind=iarr ), parameter :: huge_iarr = huge(0_iarr)

! Reference values for different compilers for magnet_ref,
! magnetisation at the end of simulation
!integer( kind=iarr ), parameter :: magnet_ref = 863379 ! gfortran7
integer( kind=iarr ), parameter :: magnet_ref = 864070 ! Cray

real( kind=rdef ) :: &
  qual,              & ! quality
  bsz0(3),           & ! the given "box" size
```

```
  bsz(3),            & ! updated "box" size
  dm,                & ! mean grain size, linear dim, phys units
  lres,              & ! linear resolution, cells per unit of length
  res                  ! resolutions, cells per grain

integer( kind=iarr ), allocatable :: space(:,:,:),                    &
  space0(:,:,:)

integer( kind=idef ) :: ir(3), nimgs, img, ng, c(3) ! coarray dimensions

integer( kind=ilrg ) :: icells, mcells

!real, allocatable :: space_ini(:,:,:), rnd_array(:)

integer :: i, iter, seed_size, run
integer( kind=ilrg) :: energy0, energy1, energy2, magnet0, magnet1,    &
  magnet2
integer, allocatable :: seed_array(:)

! real :: time1, time2

!*********************************************************************72
! first executable statement

 dm = 1.0 ! Linear "size" of one spin cell
res = 1.0 ! resolution, CA cells per spin

! Read the box size from command line
call ca_cmd_real( n=3, data=bsz0 )

! When dm=res=1, then bsz0 is simply CA dimensions in cells!
!bsz0 = (/ 1.2e2, 1.2e2, 1.2e2 /) ! dimensions of the CA model

  img = this_image()
nimgs = num_images()

! each image calculates the coarray grid dimensions
call cgca_gdim( nimgs, ir, qual )

! calculate the resolution and the actual phys dimensions of the box
! subroutine cgca_cadim( bsz, res, dm, ir, c, lres, ng )
! c - coarray sizes
! ir - coarray grid sizes
bsz = bsz0
call cgca_cadim( bsz, res, dm, ir, c, lres, ng )

! do a check on image 1
if ( img .eq. 1 ) then
  write (*,*) "running on", nimgs, "images in a 3D grid"
  write (*,*) "iarr kind:", iarr, "huge(0_iarr):", huge_iarr

  ! Check that the partition is sane
```

```
  if ( any(int(bsz) .ne. int(bsz0) ) ) then
    write (*,*)                                                          &
      "ERROR: bad decomposition - use a 'nicer' number of images"
    write (*,*) "ERROR: wanted         :", int(bsz0)
    write (*,*) "ERROR: but got instead:", int(bsz)
    error stop
  end if

  ! total number of cells in a coarray
  icells = product( int( c, kind=ilrg ) )

  ! total number of cells in the model
  mcells = icells * int( nimgs, kind=ilrg )

  write ( *, "(8(a,i0),tr1,g10.3,tr1,g10.3,3(a,g10.3),a)" )              &
    "nimgs: ", nimgs, " (", c(1), "," , c(2), "," , c(3), ")[",          &
    ir(1), "," , ir(2), "," , ir(3), "] ", ng, qual, lres,               &
    " (", bsz(1), ",", bsz(2), ",", bsz(3), ")"
     write (*,'(a,i0,a)') "Each image has ",icells, " cells"
     write (*,'(a,i0,a)') "The model has ", mcells, " cells"

  ! In this test sum over all cells on an image is done, so the kind
  ! must be big enough to contain the total number of cells
  ! on an image.
  if ( icells .gt. huge_iarr ) then
    write (*,*) "ERROR: number of cells on an image:", icells,           &
      "is greater than huge(0_iarr)"
    error stop
  end if

end if

! allocate space arrays
!    space - CA array to allocate, with halos!
!        c - array with space dimensions
!        d - depth of the halo layer

call ca_spalloc( space,  c, 1 )
call ca_spalloc( space0, c, 1 )

! allocate hx arrays, implicit sync all
! mask_array is set inside too.
! ir(3) - codimensions
call ca_halloc( ir )

! Init RND
!call cgca_irs( debug = .false. )
! Use a reproducible RND here for verification
call random_seed( size = seed_size )
allocate( seed_array( seed_size ) )
seed_array = (/ (i, i=1,seed_size) /)
```

```
! Set space arrays
if (img .eq. 1) write (*,*) "RND, serial IO, etc. - wait..."
! Passing allocatable coarray into assumed-shape array, which is ok!
call ca_set_space_rnd( seed = seed_array, frac1=0.5, space = space )

! Calculate initial energy and magnetisation

! run=1 => ca_iter_tl
! run=2 => ca_iter_dc
! run=3 => ca_iter_omp
do run=1,3
  select case(run)
  case(1)
    call ca_ising_energy(     space = space, hx_sub = ca_hx_glbar,     &
      iter_sub = ca_iter_tl,  kernel = ca_kernel_ising_ener,           &
      energy = energy0, magnet = magnet0 )
  case(2)
    call ca_ising_energy(     space = space, hx_sub = ca_hx_glbar,     &
      iter_sub = ca_iter_dc,  kernel = ca_kernel_ising_ener,           &
      energy = energy1, magnet = magnet1 )
  case(3)
    call ca_ising_energy(     space = space, hx_sub = ca_hx_glbar,     &
      iter_sub = ca_iter_omp, kernel = ca_kernel_ising_ener,           &
      energy = energy2, magnet = magnet2 )
  end select
end do

if (img .eq. 1 ) then
  write (*,*) "Initial energy and magnetisation"
  write (*,*) "ca_iter_tl :", energy0, magnet0
  write (*,*) "ca_iter_dc :", energy1, magnet1
  write (*,*) "ca_iter_omp:", energy2, magnet2
  if ( energy0 .ne. energy1 .or. magnet0 .ne. magnet1 .or.             &
       energy0 .ne. energy2 .or. magnet0 .ne. magnet2 ) then
    write (*,*) "FAIL: ca_iter_tl, ca_iter_dc, ca_iter_omp differ"
    error stop
  else
    write (*,*) "PASS: ca_iter_tl, ca_iter_dc, ca_iter_omp agree"
  end if
end if

! save old space as space0
space0 = space

! run=1 => ca_iter_tl
! run=2 => ca_iter_dc
! run=3 => ca_iter_omp
main: do run=1,3

  ! Reset space to space0
  space = space0
```

```
! No IO here

! No timing
! here

  ! CA iterations
  loop: do iter = 1,100

    ! Check energy after every iter
    ! subroutine ca_run(    space, hx_sub, iter_sub, kernel, niter )
    select case(run)
    case(1)
      call ca_run(    space = space, hx_sub = ca_hx_glbar,             &
        iter_sub = ca_iter_tl,  kernel = ca_kernel_ising, niter = 1 )
      call ca_ising_energy(     space = space, hx_sub = ca_hx_glbar,   &
        iter_sub = ca_iter_tl,  kernel = ca_kernel_ising_ener,         &
        energy = energy1, magnet = magnet1 )
    case(2)
      call ca_run(    space = space, hx_sub = ca_hx_glbar,             &
        iter_sub = ca_iter_dc,  kernel = ca_kernel_ising, niter = 1 )
      call ca_ising_energy(     space = space, hx_sub = ca_hx_glbar,   &
        iter_sub = ca_iter_dc,  kernel = ca_kernel_ising_ener,         &
        energy = energy1, magnet = magnet1 )
    case(3)
      call ca_run(    space = space, hx_sub = ca_hx_glbar,             &
        iter_sub = ca_iter_omp, kernel = ca_kernel_ising, niter = 1 )
      call ca_ising_energy(     space = space, hx_sub = ca_hx_glbar,   &
        iter_sub = ca_iter_omp, kernel = ca_kernel_ising_ener,         &
        energy = energy1, magnet = magnet1 )
    end select

    if ( img .eq. 1 ) then
      if ( energy1 .ne. energy0 ) then
        write (*,*) "FAIL: energy0:", energy0, "energy1:", energy1
        error stop
      else
        if ( mod((iter-1), 100) .eq. 0 ) then
  !      write (*,"(a,i0,a,es18.6)") "Magnetisation_after_iter_",      &
  !        iter, ":", real(magnet1) / real(mcells)
          write (*,*) iter, real(magnet1) / real(mcells)
        end if
      end if
    end if

  end do loop

! no sync needed here
!
! no IO here
!
! no counter
! here
```

```
  if ( img .eq. 1 ) then
    select case(run)
      case(1)
        if ( magnet1 .eq. magnet_ref ) then
          write (*,*) "PASS: ca_iter_tl : final mag:", magnet1
! no timing here
        else
          write (*,"(2(a,i0))")                                          &
            "FAIL: ca_iter_tl : magnetisation ref value: ",              &
             magnet_ref, " my value: ", magnet1
        end if
      case(2)
        if ( magnet1 .eq. magnet_ref ) then
          write (*,*) "PASS: ca_iter_dc : final mag:", magnet1
! no timing here
        else
          write (*,"(2(a,i0))")                                          &
            "FAIL: ca_iter_dc : magnetisation ref value: ",              &
             magnet_ref, " my value: ", magnet1
        end if
      case(3)
        if ( magnet1 .eq. magnet_ref ) then
          write (*,*) "PASS: ca_iter_omp: final mag:", magnet1
! no timing here
        else
          write (*,"(2(a,i0))")                                          &
            "FAIL: ca_iter_omp: magnetisation ref value: ",              &
             magnet_ref, " my value: ", magnet1
        end if
    end select

  end if

end do main

! deallocate halos, implicit sync all
call ca_hdalloc

! deallocate space
deallocate( space )
deallocate( space0 )

end program ising_glbar
```

54 tests/ising_perf

[Unit tests]

NAME

ising_perf

SYNOPSIS

```
!$Id: ising_perf.f90 561 2018-10-14 20:48:19Z mexas $

program ising_perf
```

PURPOSE

Test performance of ising (49) magnetisation. This test uses coarray collectives. Reproducibility on any number of images requires a serial RND for the whole model. This is too slow. Here each image uses its own RND, meaning the results are *not* reproducible when the number of images change. This test is purely for performance analysis.

DESCRIPTION

See ca_kernel_ising (1.15) and related routines for details.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

cgca

USED BY

Part of CGPACK test suite

SOURCE

```
use casup

implicit none

integer( kind=iarr ), parameter :: huge_iarr = huge(0_iarr)

real( kind=rdef ) :: &
  qual,              & ! quality
  bsz0(3),           & ! the given "box" size
  bsz(3),            & ! updated "box" size
  dm,                & ! mean grain size, linear dim, phys units
  lres,              & ! linear resolution, cells per unit of length
  res                  ! resolutions, cells per grain

integer( kind=iarr ), allocatable :: space(:,:,:),                    &
  space0(:,:,:)
```

```
integer( kind=idef ) :: ir(3), nimgs, img, ng, c(3) ! coarray dimensions

integer( kind=ilrg ) :: icells, mcells

!real, allocatable :: space_ini(:,:,:), rnd_array(:)

integer :: i, iter, seed_size, run
integer( kind=ilrg ) :: energy0, energy1, energy2, magnet0, magnet1,   &
  magnet2
integer, allocatable :: seed_array(:)

! logical :: flag

real :: time1, time2
real, allocatable :: rnd_arr(:,:,:)

!*********************************************************************72
! first executable statement

 dm = 1.0 ! Linear "size" of one spin cell
res = 1.0 ! resolution, CA cells per spin

! Read the box size from command line
call ca_cmd_real( n=3, data=bsz0 )

! When dm=res=1, then bsz0 is simply CA dimensions in cells!
!bsz0 = (/ 2.0e3, 2.0e3, 2.0e3 /) ! dimensions of the CA model
!bsz0 = (/ 3.0e3, 3.0e3, 3.0e3 /) ! dimensions of the CA model
!bsz0 = (/ 1.2e2, 1.2e2, 1.2e2 /) ! for testing on FreeBSD laptop

  img = this_image()
nimgs = num_images()

! each image calculates the coarray grid dimensions
call cgca_gdim( nimgs, ir, qual )

! calculate the resolution and the actual phys dimensions of the box
! subroutine cgca_cadim( bsz, res, dm, ir, c, lres, ng )
! c - coarray sizes
! ir - coarray grid sizes
bsz = bsz0
call cgca_cadim( bsz, res, dm, ir, c, lres, ng )

! do a check on image 1
if ( img .eq. 1 ) then
  write (*,*) "running on", nimgs, "images in a 3D grid"
  write (*,*) "iarr kind:", iarr, "huge(0_iarr):", huge_iarr

! No
! check
! for
```

```
! bad
! partition
! in
! this
! test

  ! total number of cells in a coarray
  icells = product( int( c, kind=ilrg ) )

  ! total number of cells in the model
  mcells = icells * int( nimgs, kind=ilrg )

  write ( *, "(8(a,i0),tr1,g10.3,tr1,g10.3,3(a,g10.3),a)" )            &
    "nimgs: ", nimgs, " (", c(1), "," , c(2), "," , c(3), ")[",        &
    ir(1), "," , ir(2), "," , ir(3), "] ", ng, qual, lres,             &
    " (", bsz(1), ",", bsz(2), ",", bsz(3), ")"
     write (*,'(a,i0,a)') "Each image has ",icells, " cells"
     write (*,'(a,i0,a)') "The model has ", mcells, " cells"

  ! In this test sum over all cells on an image is done, so the kind
  ! must be big enough to contain the total number of cells
  ! on an image.
  if ( icells .gt. huge_iarr ) then
    write (*,*) "ERROR: number of cells on an image:", icells,         &
      "is greater than huge(0_iarr)"
    error stop
  end if

end if

! allocate space arrays and set all values to zero
!    space - CA array to allocate, with halos!
!        c - array with space dimensions
!        d - depth of the halo layer

call ca_spalloc( space,  c, 1 )
call ca_spalloc( space0, c, 1 )

! allocate hx arrays, implicit sync all
! mask_array is set inside too.
! ir(3) - codimensions
call ca_halloc( ir )

! Init RND
!call cgca_irs( debug = .false. )
! Use a reproducible RND here for verification
call random_seed( size = seed_size )
allocate( seed_array( seed_size ) )
seed_array = (/ (i, i=1,seed_size) /)
call random_seed( put = seed_array )

! Set space arrays
```

```
allocate( rnd_arr( lbound(space, dim=1) : ubound(space, dim=1),        &
                   lbound(space, dim=2) : ubound(space, dim=2),        &
                   lbound(space, dim=3) : ubound(space, dim=3) ) )
call random_number( rnd_arr )
space = nint( rnd_arr, kind=iarr )

! MPI
! init
! in
! test_mpi_ising_perf
!
!

! MPI types
! creation

! Calculate initial energy and magnetisation

! run=1 => ca_iter_tl
! run=2 => ca_iter_dc
! run=3 => ca_iter_omp
do run=1,3
  select case(run)
  case(1)
    call ca_ising_energy_col( space = space, hx_sub = ca_hx_all,       &
      iter_sub = ca_iter_tl,  kernel = ca_kernel_ising_ener,           &
      energy = energy0, magnet = magnet0 )
  case(2)
    call ca_ising_energy_col( space = space, hx_sub = ca_hx_all,       &
      iter_sub = ca_iter_dc,  kernel = ca_kernel_ising_ener,           &
      energy = energy1, magnet = magnet1 )
  case(3)
    call ca_ising_energy_col( space = space, hx_sub = ca_hx_all,       &
      iter_sub = ca_iter_omp, kernel = ca_kernel_ising_ener,           &
      energy = energy2, magnet = magnet2 )
  end select
end do

if (img .eq. 1 ) then
  write (*,*) "Initial energy and magnetisation"
  write (*,*) "ca_iter_tl :", energy0, magnet0
  write (*,*) "ca_iter_dc :", energy1, magnet1
  write (*,*) "ca_iter_omp:", energy2, magnet2
  if ( energy0 .ne. energy1 .or. magnet0 .ne. magnet1 .or.             &
       energy0 .ne. energy2 .or. magnet0 .ne. magnet2 ) then
    write (*,*) "FAIL: ca_iter_tl, ca_iter_dc, ca_iter_omp differ"
    error stop
  else
    write (*,*) "PASS: ca_iter_tl, ca_iter_dc, ca_iter_omp agree"
  end if
end if
```

```
! save old space as space0
space0 = space

! run=1 => ca_iter_tl
! run=2 => ca_iter_dc
! run=3 => ca_iter_omp
main: do run=1,1

  ! Reset space to space0
  space = space0

  ! Start counter
  if ( img .eq. 1 ) call cpu_time( time1 )

  ! CA iterations
  loop: do iter = 1,100

    ! Check energy after every iter
    ! subroutine ca_run(    space, hx_sub, iter_sub, kernel, niter )
    select case(run)
    case(1)
      call ca_run(    space = space, hx_sub = ca_hx_all,               &
        iter_sub = ca_iter_tl,  kernel = ca_kernel_ising, niter = 1 )
      call ca_ising_energy_col( space = space, hx_sub = ca_hx_all,     &
        iter_sub = ca_iter_tl,  kernel = ca_kernel_ising_ener,         &
        energy = energy1, magnet = magnet1 )
    case(2)
      call ca_run(    space = space, hx_sub = ca_hx_all,               &
        iter_sub = ca_iter_dc,  kernel = ca_kernel_ising, niter = 1 )
      call ca_ising_energy_col( space = space, hx_sub = ca_hx_all,     &
        iter_sub = ca_iter_dc,  kernel = ca_kernel_ising_ener,         &
        energy = energy1, magnet = magnet1 )
    case(3)
      call ca_run(    space = space, hx_sub = ca_hx_all,               &
        iter_sub = ca_iter_omp, kernel = ca_kernel_ising, niter = 1 )
      call ca_ising_energy_col( space = space, hx_sub = ca_hx_all,     &
        iter_sub = ca_iter_omp, kernel = ca_kernel_ising_ener,         &
        energy = energy1, magnet = magnet1 )
    end select

    if ( img .eq. 1 ) then
      if ( energy1 .ne. energy0 ) then
        write (*,*) "FAIL: energy0:", energy0, "energy1:", energy1
        error stop
      else
!       write (*,"(a,i0,a,es18.6)") "Magnetisation_after_iter_",      &
!         iter, ":", real(magnet1) / real(mcells)
      end if
    end if

  end do loop
```

```
  if ( img .eq. 1 ) then
    ! Stop counter
    call cpu_time( time2 )

    select case(run)
      case(1)
          write (*,*) "TIME ca_iter_tl, s:", time2-time1
      case(2)
          write (*,*) "TIME ca_iter_dc, s:", time2-time1
      case(3)
          write (*,*) "TIME ca_iter_omp,s:", time2-time1
    end select

  end if

end do main

! deallocate halos, implicit sync all
call ca_hdalloc

! No need to free
! MPI types

! deallocate space
deallocate( space )
deallocate( space0 )

end program ising_perf
```

55 tests/ising_perf_1D

[Unit tests]

NAME

ising_perf_1D

SYNOPSIS

```
!$Id: ising_perf_1D.f90 561 2018-10-14 20:48:19Z mexas $

program ising_perf_1D
```

PURPOSE

Test performance of ising (49) magnetisation with 1D (corank 1). This test uses coarray collectives. Reproducibility on any number of images requires a serial RND for the whole model. This is too slow. Here each image uses its own RND, meaning the results are *not* reproducible when the number of images change. This test is purely for performance analysis.

DESCRIPTION

See ca_kernel_ising (1.15) and related routines for details.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

cgca

USED BY

Part of CGPACK test suite

SOURCE

```
use casup

implicit none

integer( kind=iarr ), parameter :: huge_iarr = huge(0_iarr)

real( kind=rdef ) :: bsz(3)            ! updated "box" size




integer( kind=iarr ), allocatable :: space(:,:,:),                    &
  space0(:,:,:)
```

```
integer( kind=idef ) :: ir(3), nimgs, img, ng, c(3) ! coarray dimensions

integer( kind=ilrg ) :: icells, mcells

!real, allocatable :: space_ini(:,:,:), rnd_array(:)

integer :: i, iter, seed_size, run
integer( kind=ilrg ) :: energy0, energy1, energy2, magnet0, magnet1,   &
  magnet2
integer, allocatable :: seed_array(:)

! logical :: flag

real :: time1, time2
real, allocatable :: rnd_arr(:,:,:)

!*********************************************************************72
! first executable statement

! Read the box size from command line
call ca_cmd_real( n=3, data=bsz )

! When dm=res=1, then bsz is simply CA dimensions in cells!
!bsz = (/ 2.0e3, 2.0e3, 2.0e3 /) ! dimensions of the CA model
!bsz = (/ 3.0e3, 3.0e3, 3.0e3 /) ! dimensions of the CA model
!bsz = (/ 1.2e2, 1.2e2, 1.2e2 /) ! for testing on FreeBSD laptop

  img = this_image()
nimgs = num_images()

! do a check on image 1
if ( img .eq. 1 ) then
  write (*,*) "running on", nimgs, "images in a 1D grid"
  write (*,*) "iarr kind:", iarr, "huge(0_iarr):", huge_iarr

  ! In this test space is assigned image numbers - must be big enough
  ! integer kind to avoid integer overflow.
  if ( nimgs .gt. huge_iarr ) then
    write (*,*) "ERROR: num_images(): ", nimgs,              &
      " is greater than huge(0_iarr)"
    error stop
  end if
end if

! Only a single codimension, corank 1
c = int(bsz)
c(3) = int(bsz(3))/nimgs

! Check that the partition is sane
if ( img .eq. 1 ) then
  if ( c(3)*nimgs .ne. int(bsz(3)) ) then
```

```
    write (*,*)                                                        &
      "ERROR: bad decomposition: bsz(3) must be divisible by nimgs"
    write (*,*) "ERROR: bsz(3):", int(bsz(3))
    write (*,*) "ERROR: nimgs :", nimgs
    error stop
  end if

  ! total number of cells in a coarray
  icells = product( int( c, kind=ilrg ) )

  ! total number of cells in the model
  mcells = icells * int( nimgs, kind=ilrg )

  write ( *, "(5(a,i0),3(a,g10.3),a)" )                                &
    "nimgs: ", nimgs, " (", c(1), "," , c(2), "," , c(3),              &
    ")[", nimgs, "] (", bsz(1), ",", bsz(2), ",", bsz(3), ")"
     write (*,'(a,i0,a)') "Each image has ",icells, " cells"
     write (*,'(a,i0,a)') "The model has ", mcells, " cells"

  ! In this test sum over all cells on an image is done, so the kind
  ! must be big enough to contain the total number of cells
  ! on an image.
  if ( icells .gt. huge_iarr ) then
    write (*,*) "ERROR: number of cells on an image:", icells,         &
      "is greater than huge(0_iarr)"
    error stop
  end if

end if

! allocate space arrays and set all values to zero
!    space - CA array to allocate, with halos!
!        c - array with space dimensions
!        d - depth of the halo layer

call ca_spalloc( space,  c, 1 )
call ca_spalloc( space0, c, 1 )

! allocate hx arrays, implicit sync all
! mask_array is set inside too.
!
call ca_1D_halloc

! Init RND
!call cgca_irs( debug = .false. )
! Use a reproducible RND here for verification
call random_seed( size = seed_size )
allocate( seed_array( seed_size ) )
seed_array = (/ (i, i=1,seed_size) /)
call random_seed( put = seed_array )

! Set space arrays
```

```
allocate( rnd_arr( lbound(space, dim=1) : ubound(space, dim=1),        &
                   lbound(space, dim=2) : ubound(space, dim=2),        &
                   lbound(space, dim=3) : ubound(space, dim=3) ) )
call random_number( rnd_arr )
space = nint( rnd_arr, kind=iarr )

! MPI
! init
! in
! test_mpi_ising_perf
!
!

! MPI types
! creation

! Calculate initial energy and magnetisation

! run=1 => ca_iter_tl
! run=2 => ca_iter_dc
! run=3 => ca_iter_omp
do run=1,3
  select case(run)
  case(1)
    call ca_ising_energy_col( space = space, hx_sub = ca_1D_hx_sall,     &
      iter_sub = ca_iter_tl,  kernel = ca_kernel_ising_ener,           &
      energy = energy0, magnet = magnet0 )
  case(2)
    call ca_ising_energy_col( space = space, hx_sub = ca_1D_hx_sall,     &
      iter_sub = ca_iter_dc,  kernel = ca_kernel_ising_ener,           &
      energy = energy1, magnet = magnet1 )
  case(3)
    call ca_ising_energy_col( space = space, hx_sub = ca_1D_hx_sall,     &
      iter_sub = ca_iter_omp, kernel = ca_kernel_ising_ener,           &
      energy = energy2, magnet = magnet2 )
  end select
end do

if (img .eq. 1 ) then
  write (*,*) "Initial energy and magnetisation"
  write (*,*) "ca_iter_tl :", energy0, magnet0
  write (*,*) "ca_iter_dc :", energy1, magnet1
  write (*,*) "ca_iter_omp:", energy2, magnet2
  if ( energy0 .ne. energy1 .or. magnet0 .ne. magnet1 .or.             &
       energy0 .ne. energy2 .or. magnet0 .ne. magnet2 ) then
    write (*,*) "FAIL: ca_iter_tl, ca_iter_dc, ca_iter_omp differ"
    error stop
  else
    write (*,*) "PASS: ca_iter_tl, ca_iter_dc, ca_iter_omp agree"
  end if
end if
```

```
! save old space as space0
space0 = space

! run=1 => ca_iter_tl
! run=2 => ca_iter_dc
! run=3 => ca_iter_omp
main: do run=1,1

  ! Reset space to space0
  space = space0

  ! Start counter
  if ( img .eq. 1 ) call cpu_time( time1 )

  ! CA iterations
  loop: do iter = 1,100

    ! Check energy after every iter
    ! subroutine ca_run(    space, hx_sub, iter_sub, kernel, niter )
    select case(run)
    case(1)
      call ca_run(    space = space, hx_sub = ca_1D_hx_sall,           &
        iter_sub = ca_iter_tl,  kernel = ca_kernel_ising, niter = 1 )
      call ca_ising_energy_col( space = space, hx_sub = ca_1D_hx_sall, &
        iter_sub = ca_iter_tl,  kernel = ca_kernel_ising_ener,         &
        energy = energy1, magnet = magnet1 )
    case(2)
      call ca_run(    space = space, hx_sub = ca_1D_hx_sall,           &
        iter_sub = ca_iter_dc,  kernel = ca_kernel_ising, niter = 1 )
      call ca_ising_energy_col( space = space, hx_sub = ca_1D_hx_sall, &
        iter_sub = ca_iter_dc,  kernel = ca_kernel_ising_ener,         &
        energy = energy1, magnet = magnet1 )
    case(3)
      call ca_run(    space = space, hx_sub = ca_1D_hx_sall,           &
        iter_sub = ca_iter_omp, kernel = ca_kernel_ising, niter = 1 )
      call ca_ising_energy_col( space = space, hx_sub = ca_1D_hx_sall, &
        iter_sub = ca_iter_omp, kernel = ca_kernel_ising_ener,         &
        energy = energy1, magnet = magnet1 )
    end select

    if ( img .eq. 1 ) then
      if ( energy1 .ne. energy0 ) then
        write (*,*) "FAIL: energy0:", energy0, "energy1:", energy1
        error stop
      else
!       write (*,"(a,i0,a,es18.6)") "Magnetisation_after_iter_",      &
!         iter, ":", real(magnet1) / real(mcells)
      end if
    end if

  end do loop
```

```
  if ( img .eq. 1 ) then
    ! Stop counter
    call cpu_time( time2 )

    select case(run)
      case(1)
          write (*,*) "TIME ca_iter_tl, s:", time2-time1
      case(2)
          write (*,*) "TIME ca_iter_dc, s:", time2-time1
      case(3)
          write (*,*) "TIME ca_iter_omp,s:", time2-time1
    end select

  end if

end do main

! deallocate halos, implicit sync all
call ca_1D_hdalloc

! No need to free
! MPI types

! deallocate space
deallocate( space )
deallocate( space0 )

end program ising_perf_1D
```

56 tests/ising_perf_co

[Unit tests]

NAME

ising_perf_co

SYNOPSIS

```
!$Id: ising_perf_co.f90 561 2018-10-14 20:48:19Z mexas $

program ising_perf_co
```

PURPOSE

Test performance of ising (49) magnetisation. This test uses coarrays for the whole model, not just halos! Reproducibility on any number of images requires a serial RND for the whole model. This is too slow. Here each image uses its own RND, meaning the results are *not* reproducible when the number of images change. This test is purely for performance analysis.

DESCRIPTION

See ca_kernel_ising (1.15) and related routines for details.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

cgca

USED BY

Part of CGPACK test suite

SOURCE

```
use casup

implicit none

integer( kind=iarr ), parameter :: huge_iarr = huge(0_iarr)

real( kind=rdef ) :: &
  qual,              & ! quality
  bsz0(3),           & ! the given "box" size
  bsz(3),            & ! updated "box" size
  dm,                & ! mean grain size, linear dim, phys units
  lres,              & ! linear resolution, cells per unit of length
  res                  ! resolutions, cells per grain

integer( kind=iarr ), allocatable :: space(:,:,:) [:,:,:],            &
  space0(:,:,:) [:,:,:]
```

```
integer( kind=idef ) :: ir(3), nimgs, img, ng, c(3) ! coarray dimensions

integer( kind=ilrg ) :: icells, mcells

!real, allocatable :: space_ini(:,:,:), rnd_array(:)

integer :: i, iter, seed_size, run
integer( kind=ilrg ) :: energy0, energy1, energy2, magnet0, magnet1,   &
  magnet2
integer, allocatable :: seed_array(:)

! logical :: flag

real :: time1, time2
real, allocatable :: rnd_arr(:,:,:)

!*********************************************************************72
! first executable statement

 dm = 1.0 ! Linear "size" of one spin cell
res = 1.0 ! resolution, CA cells per spin

! Read the box size from command line
call ca_cmd_real( n=3, data=bsz0 )

! When dm=res=1, then bsz0 is simply CA dimensions in cells!
!bsz0 = (/ 2.0e3, 2.0e3, 2.0e3 /) ! dimensions of the CA model
!bsz0 = (/ 3.0e3, 3.0e3, 3.0e3 /) ! dimensions of the CA model
!bsz0 = (/ 1.2e2, 1.2e2, 1.2e2 /) ! for testing on FreeBSD laptop

  img = this_image()
nimgs = num_images()

! each image calculates the coarray grid dimensions
call cgca_gdim( nimgs, ir, qual )

! calculate the resolution and the actual phys dimensions of the box
! subroutine cgca_cadim( bsz, res, dm, ir, c, lres, ng )
! c - coarray sizes
! ir - coarray grid sizes
bsz = bsz0
call cgca_cadim( bsz, res, dm, ir, c, lres, ng )

! do a check on image 1
if ( img .eq. 1 ) then
  write (*,*) "running on", nimgs, "images in a 3D grid"
  write (*,*) "iarr kind:", iarr, "huge(0_iarr):", huge_iarr

! No
! check
! for
```

```
! bad
! partition
! in
! this
! test

  ! total number of cells in a coarray
  icells = product( int( c, kind=ilrg ) )

  ! total number of cells in the model
  mcells = icells * int( nimgs, kind=ilrg )

  write ( *, "(8(a,i0),tr1,g10.3,tr1,g10.3,3(a,g10.3),a)" )            &
    "nimgs: ", nimgs, " (", c(1), "," , c(2), "," , c(3), ")[",        &
    ir(1), "," , ir(2), "," , ir(3), "] ", ng, qual, lres,             &
    " (", bsz(1), ",", bsz(2), ",", bsz(3), ")"
     write (*,'(a,i0,a)') "Each image has ",icells, " cells"
     write (*,'(a,i0,a)') "The model has ", mcells, " cells"

  ! In this test sum over all cells on an image is done, so the kind
  ! must be big enough to contain the total number of cells
  ! on an image.
  if ( icells .gt. huge_iarr ) then
    write (*,*) "ERROR: number of cells on an image:", icells,         &
      "is greater than huge(0_iarr)"
    error stop
  end if

end if

! allocate space arrays and set all values to zero
!    space - CA array to allocate, with halos!
!        c - array with space dimensions
!        d - depth of the halo layer
!       ir - codimensions
call ca_co_spalloc( space,  c, 1, ir )
call ca_co_spalloc( space0, c, 1, ir )

! No
! need
! for HX
! arrays

! Init RND
!call cgca_irs( debug = .false. )
! Use a reproducible RND here for verification
call random_seed( size = seed_size )
allocate( seed_array( seed_size ) )
seed_array = (/ (i, i=1,seed_size) /)
call random_seed( put = seed_array )

! Set space arrays
```

```
allocate( rnd_arr( lbound(space, dim=1) : ubound(space, dim=1),        &
                   lbound(space, dim=2) : ubound(space, dim=2),        &
                   lbound(space, dim=3) : ubound(space, dim=3) ) )
call random_number( rnd_arr )
space = nint( rnd_arr, kind=iarr )

! MPI
! init
! in
! test_mpi_ising_perf
!
!

! MPI types
! creation

! Calculate initial energy and magnetisation

! run=1 => ca_iter_tl
! run=2 => ca_iter_dc
! run=3 => ca_iter_omp
do run=1,3
  select case(run)
  case(1)
    call ca_co_ising_energy(  space = space, hx_sub = ca_co_hx_all,    &
      iter_sub = ca_iter_tl,  kernel = ca_kernel_ising_ener,           &
      energy = energy0, magnet = magnet0 )
  case(2)
    call ca_co_ising_energy(  space = space, hx_sub = ca_co_hx_all,    &
      iter_sub = ca_iter_dc,  kernel = ca_kernel_ising_ener,           &
      energy = energy1, magnet = magnet1 )
  case(3)
    call ca_co_ising_energy(  space = space, hx_sub = ca_co_hx_all,    &
      iter_sub = ca_iter_omp, kernel = ca_kernel_ising_ener,           &
      energy = energy2, magnet = magnet2 )
  end select
end do

if (img .eq. 1 ) then
  write (*,*) "Initial energy and magnetisation"
  write (*,*) "ca_iter_tl :", energy0, magnet0
  write (*,*) "ca_iter_dc :", energy1, magnet1
  write (*,*) "ca_iter_omp:", energy2, magnet2
  if ( energy0 .ne. energy1 .or. magnet0 .ne. magnet1 .or.             &
       energy0 .ne. energy2 .or. magnet0 .ne. magnet2 ) then
    write (*,*) "FAIL: ca_iter_tl, ca_iter_dc, ca_iter_omp differ"
    error stop
  else
    write (*,*) "PASS: ca_iter_tl, ca_iter_dc, ca_iter_omp agree"
  end if
end if
```

```
! save old space as space0
space0 = space

! run=1 => ca_iter_tl
! run=2 => ca_iter_dc
! run=3 => ca_iter_omp
main: do run=1,1

  ! Reset space to space0
  space = space0

  ! Start counter
  if ( img .eq. 1 ) call cpu_time( time1 )

  ! CA iterations
  loop: do iter = 1,100

    ! Check energy after every iter
    ! subroutine ca_co_run( space, hx_sub, iter_sub, kernel, niter )
    select case(run)
    case(1)
      call ca_co_run( space = space, hx_sub = ca_co_hx_all,            &
        iter_sub = ca_iter_tl,  kernel = ca_kernel_ising, niter = 1 )
      call ca_co_ising_energy(  space = space, hx_sub = ca_co_hx_all,  &
        iter_sub = ca_iter_tl,  kernel = ca_kernel_ising_ener,         &
        energy = energy1, magnet = magnet1 )
    case(2)
      call ca_co_run( space = space, hx_sub = ca_co_hx_all,            &
        iter_sub = ca_iter_dc,  kernel = ca_kernel_ising, niter = 1 )
      call ca_co_ising_energy(  space = space, hx_sub = ca_co_hx_all,  &
        iter_sub = ca_iter_dc,  kernel = ca_kernel_ising_ener,         &
        energy = energy1, magnet = magnet1 )
    case(3)
      call ca_co_run( space = space, hx_sub = ca_co_hx_all,            &
        iter_sub = ca_iter_omp, kernel = ca_kernel_ising, niter = 1 )
      call ca_co_ising_energy(  space = space, hx_sub = ca_co_hx_all,  &
        iter_sub = ca_iter_omp, kernel = ca_kernel_ising_ener,         &
        energy = energy1, magnet = magnet1 )
    end select

    if ( img .eq. 1 ) then
      if ( energy1 .ne. energy0 ) then
        write (*,*) "FAIL: energy0:", energy0, "energy1:", energy1
        error stop
      else
!       write (*,"(a,i0,a,es18.6)") "Magnetisation_after_iter_",      &
!         iter, ":", real(magnet1) / real(mcells)
      end if
    end if

  end do loop
```

```
  if ( img .eq. 1 ) then
    ! Stop counter
    call cpu_time( time2 )

    select case(run)
      case(1)
          write (*,*) "TIME ca_iter_tl, s:", time2-time1
      case(2)
          write (*,*) "TIME ca_iter_dc, s:", time2-time1
      case(3)
          write (*,*) "TIME ca_iter_omp,s:", time2-time1
    end select

  end if

end do main

! No halos
! to deallocate

! No need to free
! MPI types

! deallocate space
deallocate( space )
deallocate( space0 )

end program ising_perf_co
```

57 tests/ising_perf_glbar

[Unit tests]

NAME

ising_perf_glbar

SYNOPSIS

```
!$Id: ising_perf_glbar.f90 561 2018-10-14 20:48:19Z mexas $

program ising_perf_glbar
```

PURPOSE

Test performance of ising (49) magnetisation. This test uses coarray collectives + global barrier - sync all. Reproducibility on any number of images requires a serial RND for the whole model. This is too slow. Here each image uses its own RND, meaning the results are *not* reproducible when the number of images change. This test is purely for performance analysis.

DESCRIPTION

See ca_kernel_ising (1.15) and related routines for details.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

cgca

USED BY

Part of CGPACK test suite

SOURCE

```
use casup

implicit none

integer( kind=iarr ), parameter :: huge_iarr = huge(0_iarr)

real( kind=rdef ) :: &
  qual,              & ! quality
  bsz0(3),           & ! the given "box" size
  bsz(3),            & ! updated "box" size
  dm,                & ! mean grain size, linear dim, phys units
  lres,              & ! linear resolution, cells per unit of length
  res                  ! resolutions, cells per grain

integer( kind=iarr ), allocatable :: space(:,:,:),                    &
  space0(:,:,:)
```

```
integer( kind=idef ) :: ir(3), nimgs, img, ng, c(3) ! coarray dimensions

integer( kind=ilrg ) :: icells, mcells

!real, allocatable :: space_ini(:,:,:), rnd_array(:)

integer :: i, iter, seed_size, run
integer( kind=ilrg ) :: energy0, energy1, energy2, magnet0, magnet1,   &
  magnet2
integer, allocatable :: seed_array(:)

! logical :: flag

real :: time1, time2
real, allocatable :: rnd_arr(:,:,:)

!*********************************************************************72
! first executable statement

 dm = 1.0 ! Linear "size" of one spin cell
res = 1.0 ! resolution, CA cells per spin

! Read the box size from command line
call ca_cmd_real( n=3, data=bsz0 )

! When dm=res=1, then bsz0 is simply CA dimensions in cells!
!bsz0 = (/ 2.0e3, 2.0e3, 2.0e3 /) ! dimensions of the CA model
!bsz0 = (/ 3.0e3, 3.0e3, 3.0e3 /) ! dimensions of the CA model
!bsz0 = (/ 1.2e2, 1.2e2, 1.2e2 /) ! for testing on FreeBSD laptop

  img = this_image()
nimgs = num_images()

! each image calculates the coarray grid dimensions
call cgca_gdim( nimgs, ir, qual )

! calculate the resolution and the actual phys dimensions of the box
! subroutine cgca_cadim( bsz, res, dm, ir, c, lres, ng )
! c - coarray sizes
! ir - coarray grid sizes
bsz = bsz0
call cgca_cadim( bsz, res, dm, ir, c, lres, ng )

! do a check on image 1
if ( img .eq. 1 ) then
  write (*,*) "running on", nimgs, "images in a 3D grid"
  write (*,*) "iarr kind:", iarr, "huge(0_iarr):", huge_iarr

! No
! check
! for
```

```
! bad
! partition
! in
! this
! test

  ! total number of cells in a coarray
  icells = product( int( c, kind=ilrg ) )

  ! total number of cells in the model
  mcells = icells * int( nimgs, kind=ilrg )

  write ( *, "(8(a,i0),tr1,g10.3,tr1,g10.3,3(a,g10.3),a)" )            &
    "nimgs: ", nimgs, " (", c(1), "," , c(2), "," , c(3), ")[",        &
    ir(1), "," , ir(2), "," , ir(3), "] ", ng, qual, lres,             &
    " (", bsz(1), ",", bsz(2), ",", bsz(3), ")"
     write (*,'(a,i0,a)') "Each image has ",icells, " cells"
     write (*,'(a,i0,a)') "The model has ", mcells, " cells"

  ! In this test sum over all cells on an image is done, so the kind
  ! must be big enough to contain the total number of cells
  ! on an image.
  if ( icells .gt. huge_iarr ) then
    write (*,*) "ERROR: number of cells on an image:", icells,         &
      "is greater than huge(0_iarr)"
    error stop
  end if

end if

! allocate space arrays and set all values to zero
!    space - CA array to allocate, with halos!
!        c - array with space dimensions
!        d - depth of the halo layer

call ca_spalloc( space,  c, 1 )
call ca_spalloc( space0, c, 1 )

! allocate hx arrays, implicit sync all
! mask_array is set inside too.
! ir(3) - codimensions
call ca_halloc( ir )

! Init RND
!call cgca_irs( debug = .false. )
! Use a reproducible RND here for verification
call random_seed( size = seed_size )
allocate( seed_array( seed_size ) )
seed_array = (/ (i, i=1,seed_size) /)
call random_seed( put = seed_array )

! Set space arrays
```

```
allocate( rnd_arr( lbound(space, dim=1) : ubound(space, dim=1),        &
                   lbound(space, dim=2) : ubound(space, dim=2),        &
                   lbound(space, dim=3) : ubound(space, dim=3) ) )
call random_number( rnd_arr )
space = nint( rnd_arr, kind=iarr )

! MPI
! init
! in
! test_mpi_ising_perf
!
!

! MPI types
! creation

! Calculate initial energy and magnetisation

! run=1 => ca_iter_tl
! run=2 => ca_iter_dc
! run=3 => ca_iter_omp
do run=1,3
  select case(run)
  case(1)
    call ca_ising_energy_col( space = space, hx_sub = ca_hx_glbar,     &
      iter_sub = ca_iter_tl,  kernel = ca_kernel_ising_ener,           &
      energy = energy0, magnet = magnet0 )
  case(2)
    call ca_ising_energy_col( space = space, hx_sub = ca_hx_glbar,     &
      iter_sub = ca_iter_dc,  kernel = ca_kernel_ising_ener,           &
      energy = energy1, magnet = magnet1 )
  case(3)
    call ca_ising_energy_col( space = space, hx_sub = ca_hx_glbar,     &
      iter_sub = ca_iter_omp, kernel = ca_kernel_ising_ener,           &
      energy = energy2, magnet = magnet2 )
  end select
end do

if (img .eq. 1 ) then
  write (*,*) "Initial energy and magnetisation"
  write (*,*) "ca_iter_tl :", energy0, magnet0
  write (*,*) "ca_iter_dc :", energy1, magnet1
  write (*,*) "ca_iter_omp:", energy2, magnet2
  if ( energy0 .ne. energy1 .or. magnet0 .ne. magnet1 .or.             &
       energy0 .ne. energy2 .or. magnet0 .ne. magnet2 ) then
    write (*,*) "FAIL: ca_iter_tl, ca_iter_dc, ca_iter_omp differ"
    error stop
  else
    write (*,*) "PASS: ca_iter_tl, ca_iter_dc, ca_iter_omp agree"
  end if
end if
```

```
! save old space as space0
space0 = space

! run=1 => ca_iter_tl
! run=2 => ca_iter_dc
! run=3 => ca_iter_omp
main: do run=1,1

  ! Reset space to space0
  space = space0

  ! Start counter
  if ( img .eq. 1 ) call cpu_time( time1 )

  ! CA iterations
  loop: do iter = 1,100

    ! Check energy after every iter
    ! subroutine ca_run(    space, hx_sub, iter_sub, kernel, niter )
    select case(run)
    case(1)
      call ca_run(    space = space, hx_sub = ca_hx_glbar,             &
        iter_sub = ca_iter_tl,  kernel = ca_kernel_ising, niter = 1 )
      call ca_ising_energy_col( space = space, hx_sub = ca_hx_glbar,   &
        iter_sub = ca_iter_tl,  kernel = ca_kernel_ising_ener,         &
        energy = energy1, magnet = magnet1 )
    case(2)
      call ca_run(    space = space, hx_sub = ca_hx_glbar,             &
        iter_sub = ca_iter_dc,  kernel = ca_kernel_ising, niter = 1 )
      call ca_ising_energy_col( space = space, hx_sub = ca_hx_glbar,   &
        iter_sub = ca_iter_dc,  kernel = ca_kernel_ising_ener,         &
        energy = energy1, magnet = magnet1 )
    case(3)
      call ca_run(    space = space, hx_sub = ca_hx_glbar,             &
        iter_sub = ca_iter_omp, kernel = ca_kernel_ising, niter = 1 )
      call ca_ising_energy_col( space = space, hx_sub = ca_hx_glbar,   &
        iter_sub = ca_iter_omp, kernel = ca_kernel_ising_ener,         &
        energy = energy1, magnet = magnet1 )
    end select

    if ( img .eq. 1 ) then
      if ( energy1 .ne. energy0 ) then
        write (*,*) "FAIL: energy0:", energy0, "energy1:", energy1
        error stop
      else
!       write (*,"(a,i0,a,es18.6)") "Magnetisation_after_iter_",      &
!         iter, ":", real(magnet1) / real(mcells)
      end if
    end if

  end do loop
```

```
  if ( img .eq. 1 ) then
    ! Stop counter
    call cpu_time( time2 )

    select case(run)
      case(1)
          write (*,*) "TIME ca_iter_tl, s:", time2-time1
      case(2)
          write (*,*) "TIME ca_iter_dc, s:", time2-time1
      case(3)
          write (*,*) "TIME ca_iter_omp,s:", time2-time1
    end select

  end if

end do main

! deallocate halos, implicit sync all
call ca_hdalloc

! No need to free
! MPI types

! deallocate space
deallocate( space )
deallocate( space0 )

end program ising_perf_glbar
```

58 tests/Makefile-tests-bc3-ifort-shared

[Make files]

NAME

Makefile-tests-bc3-ifort-shared

SYNOPSIS

```
#$Id: Makefile-bc3-ifort-shared 382 2017-03-22 11:41:51Z mexas $

FC=             ifort
```

PURPOSE

Build CGPACK tests on University of Bristol BlueCrystal computer with Intel Fortran compiler.

DESCRIPTION

Checked with ifort 16.0.0. This file is for *shared* memory only!

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

SOURCE

```
COAR_FLAGS=     -coarray=shared
FFLAGS=         $(COAR_FLAGS) -debug full -warn all -O2 -qopt-report \
                -traceback
                #-std08 -warn stderrors -mt_mpi
FFLAGS_CA=      $(FFLAGS) -I. -I$(MODDIR)

MPIFC=          ifort-shared

CGNAME=         cg
CGLIB=          $(CGNAME)pack
LIBDIR=         $(HOME)/lib
MODDIR=         $(HOME)/mod
MODPREFIX=      cgca
LIBNAME=        lib$(CGLIB)

LIB=            $(COAR_FLAGS) -L$(LIBDIR) -l$(CGLIB) # -I$(MODDIR)

MODSRC=         testaux.f90
MODMOD=         $(MODSRC:.f90=.mod)
MODOBJ=         $(MODSRC:.f90=.o)
MOD_RPT=        $(MODSRC:.f90=.optrpt)
MOD_CLEAN=      $(MODMOD) $(MODOBJ) $(MOD_RPT)

SRC= \
testAAA.f90 testAAB.f90 testAAC.f90 testAAD.f90 testAAE.f90 \
```

```
testAAF.f90 testAAG.f90 testAAH.f90 testAAI.f90 testAAJ.f90 \
testAAK.f90 testAAL.f90 testAAM.f90 testAAN.f90 testAAO.f90 \
testAAP.f90 testAAQ.f90 testAAR.f90 testAAS.f90 testAAT.f90 \
testAAU.f90 testAAV.f90 testAAW.f90 testAAX.f90 testAAY.f90 \
testAAZ.f90 \
testABA.f90 testABB.f90 testABC.f90 testABD.f90 testABE.f90 \
testABF.f90 testABG.f90 testABH.f90 testABI.f90 testABJ.f90 \
\
testABP.f90 testABQ.f90 testABR.f90 testABS.f90 testABT.f90 \
testABU.f90                                     testABY.f90 \
testABZ.f90 \
testACA.f90 testACB.f90
# testABK.f90 testABL.f90 testABM.f90 testABN.f90
# testABO.f90 testABV.f90 testABW.f90 - co_sum not supported by ifort 15
# testABX.f90 - Cray parallel IO extensions

NON_COARRAY_SRC=test_gc.f90
NON_COARRAY_EXE=$(NON_COARRAY_SRC:.f90=.xnonca)

CA_CHK_SRC1=    ca_check1.f90
CA_CHK_EXE1=    $(CA_CHK_SRC1:.f90=.xcack)
CA_CHK_CONF1=   $(CA_CHK_SRC1:.f90=.conf)
CA_CHK_FLAGS1=  -coarray=shared \
                -debug full -warn all #-std08 -warn stderrors

CA_CHK_SRC2=    ca_check2.f90
CA_CHK_EXE2=    $(CA_CHK_SRC2:.f90=.xcack)
CA_CHK_CONF2=   $(CA_CHK_SRC2:.f90=.conf)
CA_CHK_FLAGS2=  -coarray=shared \
                -debug full -warn all #-std08 -warn stderrors

OBJ=            ${SRC:.f90=.o}
RPT=            $(SRC:.f90=.optrpt)
EXE=            ${SRC:.f90=.x} ${NON_COARRAY_EXE} $(MPI_EXE) $(CA_CHK_EXE1) $(CA_CHK_EXE2)

ALL_CLEAN=      $(MOD_CLEAN) $(OBJ) $(RPT) $(EXE)

.SUFFIXES: .f90 .o .x .mod .xnonca .xcack

all: $(OBJ) $(EXE)

.f90.o:
        $(FC) -c $< $(FFLAGS_CA)

.f90.mod:
        $(FC) -c $< $(FFLAGS_CA)

.o.x:
        $(FC) -o $@ $< $(MODOBJ) $(LIB)

.f90.xnonca:
        $(FC) -o $@ $<
```

```
.f90.xmpi:
	$(MPIFC) -o $@ $<

$(CA_CHK_EXE1): $(CA_CHK_SRC1)
	$(FC) -o $@ $(CA_CHK_FLAGS1) $(CA_CHK_SRC1)

$(CA_CHK_EXE2): $(CA_CHK_SRC2)
	$(FC) -o $@ $(CA_CHK_FLAGS2) $(CA_CHK_SRC2)

$(OBJ): $(MODMOD) $(MODDIR)/$(MODPREFIX)*.mod $(LIBDIR)/$(LIBNAME).a
$(MODMOD) $(MODOBJ): $(MODDIR)/$(MODPREFIX)*.mod $(LIBDIR)/$(LIBNAME).a
$(EXE): $(MODOBJ)

clean:
	\rm $(ALL_CLEAN)
```

59 tests/Makefile-tests-bc3-mpiifort-tau

[Make files]

NAME

Makefile-tests-bc3-mpiifort-tau

SYNOPSIS

```
#$Id: Makefile-bc3-mpiifort-tau 382 2017-03-22 11:41:51Z mexas $

FC=             tau_f90.sh
```

PURPOSE

Build CGPACK tests on University of Bristol BlueCrystal computer with Intel Fortran compiler.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See CGPACK_Copyright

SOURCE

```
# Main TAU makefile
#TAU_MAKEFILE=  $(HOME)/tau-2.25.1/x86_64/lib/Makefile.tau-icpc-papi-mpi-pdt
TAU_MAKEFILE=   $(HOME)/tau-2.25.2-intel/x86_64/lib/Makefile.tau-icpc-papi-mpi-pdt
include $(TAU_MAKEFILE)

# This file must exist only at run time.
# At build time only the name of this file
# must be specified.
CA_CONF_FILE=   xx14.conf

CGNAME=         cg
CGLIB=          $(CGNAME)pack
LIBDIR=         $(HOME)/lib
MODDIR=         $(HOME)/mod
MODPREFIX=      cgca
LIBNAME=        lib$(CGLIB)

COAR_FLAGS=     -coarray=distributed -coarray-config-file=$(CA_CONF_FILE)

FFLAGS=         -c -qopt-report -O2 -debug full -g -traceback -free -warn \
                $(COAR_FLAGS) -I$(MODDIR) $(TAU_INCLUDE) $(TAU_MPI_INCLUDE)
#-std08 -warn stderrors -mt_mpi

LDFLAGS=        -qopt-report $(COAR_FLAGS) $(USER_OPT) $(TAU_LDFLAGS)
LIBS=           -L$(LIBDIR) -l$(CGLIB) $(TAU_MPI_FLIBS) $(TAU_LIBS) $(TAU_CXXLIBS)

MODSRC=         testaux.f90
MODMOD=         $(MODSRC:.f90=.mod)
```

```
MODOBJ=         $(MODSRC:.f90=.o)
MOD_CLEAN=      $(MODMOD) $(MODOBJ) $(MOD_RPT)

SRC= \
testAAA.f90 testAAB.f90 testAAC.f90 testAAD.f90 testAAE.f90 \
testAAF.f90 testAAG.f90 testAAH.f90 testAAI.f90 testAAJ.f90 \
testAAK.f90 testAAL.f90 testAAM.f90 testAAN.f90 testAAO.f90 \
testAAP.f90 testAAQ.f90 testAAR.f90 testAAS.f90 testAAT.f90 \
testAAU.f90 testAAV.f90 testAAW.f90 testAAX.f90 testAAY.f90 \
testAAZ.f90 \
testABA.f90 testABB.f90 testABC.f90 testABD.f90 testABE.f90 \
testABF.f90 testABG.f90 testABH.f90 testABI.f90 testABJ.f90 \
                        testABM.f90                             \
testABP.f90 testABQ.f90 testABR.f90 testABS.f90 testABT.f90 \
testABU.f90             testABW.f90             testABY.f90 \
testABZ.f90 \
testACA.f90 testACB.f90
# testABK.f90 testABL.f90 testABN.f90 testABO.f90
# testABV.f90 - co_sum not supported by ifort 16
# testABX.f90 - Cray parallel IO extensions

OBJ=            ${SRC:.f90=.o}
EXE=            ${SRC:.f90=.x}

ALL_CLEAN=      $(MOD_CLEAN) $(OBJ) $(EXE) *optrpt

.SUFFIXES: .f90 .o .x .mod

all: $(OBJ) $(EXE)

.f90.o:
        $(FC) -c $< $(FFLAGS)

.f90.mod:
        $(FC) -c $< $(FFLAGS)

.o.x:
        $(FC) -o $@ $< $(MODOBJ) $(LDFLAGS) $(LIBS)

$(OBJ): $(MODMOD) $(MODDIR)/$(MODPREFIX)*.mod $(LIBDIR)/$(LIBNAME).a
$(MODMOD) $(MODOBJ): $(MODDIR)/$(MODPREFIX)*.mod $(LIBDIR)/$(LIBNAME).a
$(EXE): $(MODOBJ)

clean:
        \rm $(ALL_CLEAN)
```

60 tests/Makefile-tests-Cray

[Make files]

NAME

Makefile-tests-Cray

SYNOPSIS

```
#$Id: Makefile-Cray 560 2018-10-14 19:02:34Z mexas $

FC=             ftn
```

PURPOSE

Build CGPACK tests on HECToR with Cray compilers.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See CGPACK_Copyright

SOURCE

```
#FFLAGS=        -eacn -Rb -I. -I$(LIBDIR)
#FFLAGS=        -eacn -Rb -m1 -rl -I. -I$(LIBDIR)

#FFLAGS=                -dm -eacFn -m3 -rl -I$(LIBDIR) -g -O0 -h bounds
FFLAGS=         -dm -eacFn -m3 -rl -I$(LIBDIR) # -g -O0 -h bounds
#FFLAGS=                -dm -eacFn -m3 -rl -O3,cache3,fp4,ipa5 -I$(LIBDIR) # -g -O0 -h bounds
LDFLAGS=

casup=          casup
LIBDIR=         $(HOME)/lib
LIBNAME=        lib$(casup).a
LIB=            -L$(LIBDIR) -l$(casup)

LOGIN_NODE_FLAGS=-h cpu=x86-64 -eacn -Rb

MODSRC=         testaux.f90
MODOBJ=         $(MODSRC:.f90=.o)
MODLST=         $(MODSRC:.f90=.lst)
CLEAN+=         $(MODLST) $(MODOBJ)

SRC= \
testAAA.f90 testAAB.f90 testAAC.f90 testAAD.f90 testAAE.f90 \
testAAF.f90 testAAG.f90 testAAH.f90 testAAI.f90 testAAJ.f90 \
testAAK.f90 testAAL.f90 testAAM.f90 testAAN.f90 testAAO.f90 \
testAAP.f90 testAAQ.f90 testAAR.f90 testAAS.f90 testAAT.f90 \
testAAU.f90 testAAV.f90 testAAW.f90 testAAX.f90 testAAY.f90 \
testAAZ.f90 \
testABA.f90 testABB.f90 testABC.f90 testABD.f90 testABE.f90 \
```

```
testABF.f90 testABG.f90 testABH.f90 testABI.f90 testABJ.f90 \
            testABL.f90 testABM.f90 testABN.f90 testABO.f90 \
testABP.f90 testABQ.f90 testABR.f90 testABS.f90 testABT.f90 \
testABU.f90 testABV.f90 testABW.f90 testABX.f90 testABY.f90 \
testABZ.f90 \
testACA.f90 testACB.f90 testACC.f90 testACD.f90 testACE.f90 \
test_hdf5.f90 test_hxi.f90 test_hxir.f90 \
testACF.f90
#testABK.f90 - this test needs "module add perftools" on Cray
OBJ=            $(SRC:.f90=.o)
LST=            $(SRC:.f90=.lst)
EXE=            $(SRC:.f90=.x)
CLEAN+=         $(OBJ) $(LST) $(EXE)

NEWSRC=         hxvn.f90 hxvn_1D.f90 hxvn_co.f90 hxvn_glbar.f90 \
                hxvn_timing.f90 hxvn_timing_co.f90 hxvn_timing_mpi.f90 \
                ising.f90 ising_1D.f90 ising_co.f90 ising_col.f90 \
                ising_glbar.f90 ising_perf.f90 ising_perf_co.f90 \
                ising_perf_glbar.f90 ising_perf_1D.f90 \
                mpi_check.f90 mpi_hxvn.f90 mpi_ising.f90 mpi_ising_perf.f90 \
                future_ca_omp1.f90 future_ca_omp2.f90
NEWOBJ=         $(NEWSRC:.f90=.o)
NEWLST=         $(NEWSRC:.f90=.lst)
NEWEXE=         $(NEWSRC:.f90=.nx)
CLEAN+=         $(NEWOBJ) $(NEWLST) $(NEWEXE)

.SUFFIXES:
.SUFFIXES:      .f90 .o .x .nx

all:            old new

old:            $(EXE)
new:            $(NEWEXE)

# Extra dependencies
$(OBJ):         $(MODOBJ)
$(MODOBJ):      $(LIBDIR)/$(LIBNAME)

$(NEWOBJ):      $(LIBDIR)/$(LIBNAME)

.f90.o:
                $(FC) -c $< $(FFLAGS)

.o.x:
                $(FC) $< -o $@ $(MODOBJ) $(LDFLAGS) $(LIB)

.o.nx:
                $(FC) $< -o $@ $(LDFLAGS) $(LIB)

clean:
                rm -f $(CLEAN)
```

61 tests/Makefile-tests-Cray-wp

[Make files]

NAME

Makefile-tests-Cray-wp

SYNOPSIS

```
#$Id: Makefile-Cray-wp 533 2018-03-30 14:31:26Z mexas $

FC=             ftn
```

PURPOSE

Build CGPACK tests Cray. Whole program optimisation.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

SOURCE

```
#FFLAGS=        -eacn -Rb -I. -I$(LIBDIR)
#FFLAGS=        -eacn -Rb -m1 -rl -I. -I$(LIBDIR)

# This must be the same as in the library
HPL_DIR=        $(HOME)/cray_pl
FFLAGS=         -c -dm -eacFn -m3 -rl -I$(LIBDIR) -hwp -hpl=$(HPL_DIR)
LDFLAGS=        -hwp -hpl=$(HPL_DIR)

casup=          casup
LIBDIR=         $(HOME)/lib
LIBNAME=        lib$(casup).a
LIB=            -L$(LIBDIR) -l$(casup)

LOGIN_NODE_FLAGS=-h cpu=x86-64 -eacn -Rb

MODSRC=         testaux.f90
MODOBJ=         $(MODSRC:.f90=.o)
MODLST=         $(MODSRC:.f90=.lst)
CLEAN+=         $(MODLST) $(MODOBJ)

SRC= \
testAAA.f90 testAAB.f90 testAAC.f90 testAAD.f90 testAAE.f90 \
testAAF.f90 testAAG.f90 testAAH.f90 testAAI.f90 testAAJ.f90 \
testAAK.f90 testAAL.f90 testAAM.f90 testAAN.f90 testAAO.f90 \
testAAP.f90 testAAQ.f90 testAAR.f90 testAAS.f90 testAAT.f90 \
testAAU.f90 testAAV.f90 testAAW.f90 testAAX.f90 testAAY.f90 \
testAAZ.f90 \
testABA.f90 testABB.f90 testABC.f90 testABD.f90 testABE.f90 \
```

```
testABF.f90 testABG.f90 testABH.f90 testABI.f90 testABJ.f90 \
            testABL.f90 testABM.f90 testABN.f90 testABO.f90 \
testABP.f90 testABQ.f90 testABR.f90 testABS.f90 testABT.f90 \
testABU.f90 testABV.f90 testABW.f90 testABX.f90 testABY.f90 \
testABZ.f90 \
testACA.f90 testACB.f90 testACC.f90 testACD.f90 testACE.f90 \
test_hdf5.f90 test_hxi.f90 test_hxir.f90 \
testACF.f90
#testABK.f90 - this test needs "module add perftools" on Cray
OBJ=            $(SRC:.f90=.o)
LST=            $(SRC:.f90=.lst)
EXE=            $(SRC:.f90=.x)
CLEAN+=         $(OBJ) $(LST) $(EXE)

NEWSRC=         test_hxvn_co.f90 test_hxvn.f90 \
test_ising.f90 test_ising_col.f90 test_ising_co.f90 \
test_ising_perf.f90 test_ising_perf_co.f90 \
test_mpi_hxvn.f90 test_mpi_ising.f90 test_mpi_ising_perf.f90 \
future_ca_omp1.f90
NEWOBJ=         $(NEWSRC:.f90=.o)
NEWLST=         $(NEWSRC:.f90=.lst)
NEWEXE=         $(NEWSRC:.f90=.nx)
CLEAN+=         $(NEWOBJ) $(NEWLST) $(NEWEXE)

.SUFFIXES:
.SUFFIXES:      .f90 .o .x .nx

all:            old new

old:            $(EXE)
new:            $(NEWEXE)

# Extra dependencies
$(OBJ):         $(MODOBJ)
$(MODOBJ):      $(LIBDIR)/$(LIBNAME)

$(NEWOBJ):      $(LIBDIR)/$(LIBNAME)

.f90.o:
                $(FC) -c $< $(FFLAGS)

.o.x:
                $(FC) $< -o $@ $(MODOBJ) $(LDFLAGS) $(LIB)

.o.nx:
                $(FC) $< -o $@ $(LDFLAGS) $(LIB)

clean:
                rm -f $(CLEAN)
```

62 tests/Makefile-tests-FreeBS2

[Make files]

NAME

Makefile-tests-FreeBSD

SYNOPSIS

```
#$Id: Makefile-FreeBSD 550 2018-04-27 17:08:42Z mexas $

FC=             caf
```

PURPOSE

Build CGPACK tests on FreeBSD with GCC/OpenCoarrays.

NOTES

Adjust the inlude and link paths as needed, to make sure all *mod files and libraries are available.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

SOURCE

```
FFLAGS=         -O2 -Wall -fall-intrinsics -fopenmp\
                -fcheck-array-temporaries\
                -I$(MODDIR) -I/usr/local/include\
                -g -fbacktrace -fcheck=bounds #-Werror

casup=          casup
LIBDIR=         $(HOME)/lib
LIB=            $(LIBDIR)/lib$(casup).a
MODDIR=         $(HOME)/include
MOD=            $(MODDIR)/$(casup).mod
LDFLAGS=        -L$(LIBDIR) -l$(casup) -L/usr/local/lib \
                -lnetcdf -lnetcdff
#               -lhdf5_fortran -lhdf5 \
#               -lhdf5hl_fortran -lhdf5_hl \

MODSRC=         testaux.f90
MODMOD=         $(MODSRC:.f90=.mod)
MODOBJ=         $(MODSRC:.f90=.o)
MODSMOD=        $(MODSRC:.f90=.smod)
CLEAN+=         $(MODMOD) $(MODOBJ) $(MODSMOD)

SRC= \
testAAA.f90 testAAB.f90 testAAC.f90 testAAD.f90 testAAE.f90 \
testAAF.f90 testAAG.f90 testAAH.f90 testAAI.f90 testAAJ.f90 \
testAAK.f90 testAAL.f90 testAAM.f90 testAAN.f90 testAAO.f90 \
```

```
testAAP.f90 testAAQ.f90 testAAR.f90 testAAS.f90 testAAT.f90 \
testAAU.f90 testAAV.f90 testAAW.f90 testAAX.f90 testAAY.f90 \
testAAZ.f90 \
testABA.f90 testABB.f90 testABC.f90 testABD.f90 testABE.f90 \
testABF.f90 testABG.f90 testABH.f90 testABI.f90 testABJ.f90 \
            testABL.f90 testABM.f90 testABN.f90 testABO.f90 \
testABP.f90 testABQ.f90 testABR.f90 testABS.f90 testABT.f90 \
testABU.f90 testABV.f90 testABW.f90             testABY.f90 \
testABZ.f90 \
testACA.f90 testACB.f90 testACC.f90 testACD.f90 testACE.f90 \
test_hxi.f90 test_hxir.f90
# testACF.f90 # - need to build HDF5 with parallel support
# testABK.f90 testABX.f90 - Cray only
OBJ=            $(SRC:.f90=.o)
EXE=            $(SRC:.f90=.x)
CLEAN+=         $(OBJ) $(EXE)

NEWSRC=         test_hxvn.f90 test_hxvn_co.f90\
test_mpi_hxvn.f90 hxvn_timing.f90\
test_ising.f90 test_ising_col.f90 test_ising_co.f90\
test_ising_perf.f90 test_ising_perf_co.f90\
test_mpi_ising.f90 test_mpi_ising_perf.f90\
future_ca_omp1.f90 future_ca_omp2.f90
NEWOBJ=         $(NEWSRC:.f90=.o)
NEWEXE=         $(NEWSRC:.f90=.nx)
CLEAN+=         $(NEWOBJ) $(NEWEXE)

.SUFFIXES:
.SUFFIXES:      .f90 .o .x .nx .mod

all:            old new

old:            $(EXE)

new:            $(NEWEXE)

# Extra dependencies
$(MODOBJ):      $(MOD)
$(OBJ):         $(MODOBJ) $(MOD)
$(EXE):         $(LIB)

$(NEWOBJ):      $(MOD)
$(NEWEXE):      $(LIB)

.f90.o:
                $(FC) -c $< $(FFLAGS)

.f90.mod:
                $(FC) -c $< $(FFLAGS)

.o.x:
                $(FC) -o $@ $< $(MODOBJ) -fopenmp $(LDFLAGS)
```

```
.o.nx:
	$(FC) -o $@ $< -fopenmp $(LDFLAGS)

clean:
	rm -f $(CLEAN)
```

63 tests/Makefile-tests-gfortran

[Make files]

NAME

Makefile-tests-gfortran

SYNOPSIS

```
#$Id: Makefile-gfortran 382 2017-03-22 11:41:51Z mexas $

FC=gfortran49
```

PURPOSE

Build CGPACK tests on FreeBSD with gfortran.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See CGPACK_Copyright

SOURCE

```
FFLAGS=-fcoarray=single -Wall -Werror -I. -I$(MODDIR)

CGLIB=cgcapack
LIBDIR=$(HOME)/lib
MODDIR=$(HOME)/modules
MODPREFIX=cgca_
LIBNAME=libcgcapack

LIB=-fcoarray=single -Wall -Werror -L$(LIBDIR) -l$(CGLIB)

MODSRC=testaux.f90
MODMOD=$(MODSRC:.f90=.mod)
MODOBJ=$(MODSRC:.f90=.o)

SRC= \
testAAA.f90 testAAB.f90 testAAC.f90 testAAD.f90 testAAE.f90 \
testAAF.f90 testAAG.f90 testAAH.f90 testAAI.f90 testAAJ.f90 \
testAAK.f90 testAAL.f90 testAAM.f90 testAAN.f90 testAAO.f90 \
testAAP.f90 testAAQ.f90 testAAR.f90 testAAS.f90 testAAT.f90 \
testAAU.f90 testAAV.f90 testAAW.f90 testAAX.f90 testAAY.f90 \
testAAZ.f90 \
testABA.f90 testABB.f90 testABC.f90 testABD.f90 testABE.f90 \
testABF.f90 testABG.f90 testABH.f90 testABI.f90 testABJ.f90 \
testABK.f90 testABL.f90 testABM.f90

#testABK.f90 testABL.f90 - gfortran has no CO_SUM

NON_COARRAY_SRC=test_gc.f90
```

```
NON_COARRAY_EXE=$(NON_COARRAY_SRC:.f90=.xnonca)

OBJ=${SRC:.f90=.o}
EXE=${SRC:.f90=.x} ${NON_COARRAY_EXE}

.SUFFIXES: .f90 .o .x .mod .xnonca

all: $(OBJ) $(EXE)

.f90.o:
        $(FC) -c $< $(FFLAGS)

.f90.mod:
        $(FC) -c $< $(FFLAGS)

.o.x:
        $(FC) -o $@ $< $(MODOBJ) $(LIB)

.f90.xnonca:
        $(FC) -o $@ $<

$(OBJ): $(MODMOD) $(MODDIR)/$(MODPREFIX)*.mod $(LIBDIR)/$(LIBNAME).a
$(MODOBJ) $(MODMOD) : $(MODDIR)/$(MODPREFIX)*.mod $(LIBDIR)/$(LIBNAME).a

clean:
        \rm $(MODMOD) $(MODOBJ) $(OBJ) $(EXE)
```

64 tests/Makefile-tests-ifort

[Make files]

NAME

Makefile-tests-ifort

SYNOPSIS

```
#$Id: Makefile-ifort 525 2018-03-19 21:54:26Z mexas $

FC=             ifort
```

PURPOSE

Build CGPACK tests with with Intel MPI Fortran compiler.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

SOURCE

```
# This file must exist only at run time.
# At build time only the name of this file
# must be specified.
#CA_CONF_FILE=  xx14.conf

COAR_FLAGS=     -coarray=distributed
# -coarray-config-file=$(CA_CONF_FILE)
FFLAGS=         $(COAR_FLAGS) -debug full -warn all -O2 -qopt-report \
                -traceback
                #-std08 -warn stderrors -mt_mpi
FFLAGS_CA=      $(FFLAGS) -I. -I$(MODDIR)

CGLIB=          cgpack
LIBDIR=         $(HOME)/lib
MODDIR=         $(HOME)/include
MODFILES=       $(MODDIR)/cgca.mod
LIBNAME=        lib$(CGLIB).a

LIB=            $(COAR_FLAGS) -L$(LIBDIR) -l$(CGLIB) \
#               -L$(HOME)/soft/hdf5-1.10.1-ifort16u2-install/lib -lhdf5 -lhdf5_fortran \
#               -L$(HOME)/soft/netcdf-fortran-4.4.4-ifort-install/lib -lnetcdff
# -lhdf5_hl -lhdf5hl_fortran

MODSRC=         testaux.f90
MODMOD=         $(MODSRC:.f90=.mod)
MODOBJ=         $(MODSRC:.f90=.o)
MODRPT=         $(MODSRC:.f90=.optrpt)
CLEAN+=         $(MODMOD) $(MODOBJ) $(MODRPT)
```

```
SRC= \
testAAA.f90 testAAB.f90 testAAC.f90 testAAD.f90 testAAE.f90 \
testAAF.f90 testAAG.f90 testAAH.f90 testAAI.f90 testAAJ.f90 \
testAAK.f90 testAAL.f90 testAAM.f90 testAAN.f90 testAAO.f90 \
testAAP.f90 testAAQ.f90 testAAR.f90 testAAS.f90 testAAT.f90 \
testAAU.f90 testAAV.f90 testAAW.f90 testAAX.f90 testAAY.f90 \
testAAZ.f90 \
testABA.f90 testABB.f90 testABC.f90 testABD.f90 testABE.f90 \
testABF.f90 testABG.f90 testABH.f90 testABI.f90 testABJ.f90 \
testABP.f90 testABQ.f90 testABR.f90 testABS.f90 testABT.f90 \
testABU.f90             testABY.f90 \
testABZ.f90 \
testACA.f90 testACB.f90            testACE.f90 \
#             testACD.f90 - MPI
#             testABW.f90 - MPI used directly
#  testABM.f90 - MPI used directly
#test_hdf5.f90 test_hxir.f90 test_netcdf.f90 testACF.f90
# testABK.f90 testABL.f90 testABN.f90 testABO.f90 testABV.f90
# testACC.f90 - co_sum not supported by ifort 16
# testABX.f90 - Cray parallel IO extensions

OBJ=            $(SRC:.f90=.o)
RPT=            $(SRC:.f90=.optrpt)
EXE=            $(SRC:.f90=.x)
CLEAN+=         $(OBJ) $(RPT) $(EXE)

.SUFFIXES:
.SUFFIXES:      .f90 .o .x

all:            $(MODOBJ) $(MODMOD) $(OBJ) $(EXE)

.f90.o:
                $(FC) -c $< $(FFLAGS_CA)

.o.x:
                $(FC) -o $@ $< $(MODOBJ) $(LIB)

# Extra dependencies
$(EXE):         $(LIBDIR)/$(LIBNAME) $(MODOBJ)
$(OBJ):         $(MODMOD) $(MODSMOD) $(MODFILES)
$(MODOBJ):      $(MODFILES)

clean:
                \rm $(CLEAN)
```

65 tests/Makefile-tests-mpiifort

[Make files]

NAME

Makefile-tests-mpiifort

SYNOPSIS

```
#$Id: Makefile-mpiifort 520 2018-03-13 18:02:06Z mexas $

FC=             mpiifort
```

PURPOSE

Build CGPACK tests with with Intel MPI Fortran compiler.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

SOURCE

```
# This file must exist only at run time.
# At build time only the name of this file
# must be specified.
CA_CONF_FILE=   xx14.conf

COAR_FLAGS=     -coarray=distributed -coarray-config-file=$(CA_CONF_FILE)
FFLAGS=         $(COAR_FLAGS) -debug full -warn all -O2 -qopt-report \
                -traceback
                #-std08 -warn stderrors -mt_mpi
FFLAGS_CA=      $(FFLAGS) -I. -I$(MODDIR)

CGLIB=          cgpack
LIBDIR=         $(HOME)/lib
MODDIR=         $(HOME)/include
MODFILES=       $(MODDIR)/cgca.mod
LIBNAME=        lib$(CGLIB).a

LIB=            $(COAR_FLAGS) -L$(LIBDIR) -l$(CGLIB) \
                -L/mnt/storage/software/libraries/intel/hdf5-1.10.1-mpi/lib \
                -lhdf5 -lhdf5_fortran \
                -L/mnt/storage/software/libraries/intel/netcdf-4.4.1.1-mpi/lib \
                -lnetcdf -lnetcdff
#               -L$(HOME)/soft/hdf5-1.10.1-ifort16u2-install/lib -lhdf5 -lhdf5_fortran \
#               -L$(HOME)/soft/netcdf-fortran-4.4.4-ifort-install/lib -lnetcdff

MODSRC=         testaux.f90
MODMOD=         $(MODSRC:.f90=.mod)
MODOBJ=         $(MODSRC:.f90=.o)
```

```
MODRPT=         $(MODSRC:.f90=.optrpt)
CLEAN+=         $(MODMOD) $(MODOBJ) $(MODRPT)

SRC= \
testAAA.f90 testAAB.f90 testAAC.f90 testAAD.f90 testAAE.f90 \
testAAF.f90 testAAG.f90 testAAH.f90 testAAI.f90 testAAJ.f90 \
testAAK.f90 testAAL.f90 testAAM.f90 testAAN.f90 testAAO.f90 \
testAAP.f90 testAAQ.f90 testAAR.f90 testAAS.f90 testAAT.f90 \
testAAU.f90 testAAV.f90 testAAW.f90 testAAX.f90 testAAY.f90 \
testAAZ.f90 \
testABA.f90 testABB.f90 testABC.f90 testABD.f90 testABE.f90 \
testABF.f90 testABG.f90 testABH.f90 testABI.f90 testABJ.f90 \
                        testABM.f90                         \
testABP.f90 testABQ.f90 testABR.f90 testABS.f90 testABT.f90 \
testABU.f90             testABW.f90             testABY.f90 \
testABZ.f90 \
testACA.f90 testACB.f90             testACD.f90 testACE.f90 \
test_hdf5.f90 \
 test_netcdf.f90 \
 test_hxvn.f90
#future_ca_omp1.f90

# no co_sum in ifort 17
# testACF.f90, test_hxir.f90
# test_hxvn_co.f90 \
#test_ising.f90 test_ising_col.f90 test_ising_perf.f90 \
#test_ising_co.f90 \
#test_mpi_hxvn.f90
# test_mpi_ising.f90
# test_mpi_ising_perf.f90 \
# testABK.f90 testABL.f90 testABN.f90 testABO.f90 testABV.f90
# testACC.f90 - co_sum not supported by ifort 16
# testABX.f90 - Cray parallel IO extensions

OBJ=            $(SRC:.f90=.o)
RPT=            $(SRC:.f90=.optrpt)
EXE=            $(SRC:.f90=.x)
CLEAN+=         $(OBJ) $(RPT) $(EXE)

.SUFFIXES:
.SUFFIXES:      .f90 .o .x

all:            $(MODOBJ) $(MODMOD) $(OBJ) $(EXE)

.f90.o:
                $(FC) -c $< $(FFLAGS_CA)

.o.x:
                $(FC) -o $@ $< $(MODOBJ) $(LIB)

# Extra dependencies
$(EXE):         $(LIBDIR)/$(LIBNAME) $(MODOBJ)
```

```
$(OBJ):         $(MODMOD) $(MODSMOD) $(MODFILES)
$(MODOBJ):      $(MODFILES)

clean:
                \rm $(CLEAN)
```

66 tests/Makefile-tests-mpiifort-scorep

[Make files]

NAME

Makefile-tests-mpiifort-scorep

SYNOPSIS

```
#$Id: Makefile-mpiifort-scorep 382 2017-03-22 11:41:51Z mexas $

FC=             scorep --user mpiifort
```

PURPOSE

Build CGPACK tests on University of Bristol BlueCrystal computer with Intel Fortran compiler.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See CGPACK_Copyright

SOURCE

```
# This file must exist only at run time.
# At build time only the name of this file
# must be specified.
CA_CONF_FILE=   xx14.conf

CGNAME=         cg
CGLIB=          $(CGNAME)pack
LIBDIR=         $(HOME)/lib
MODDIR=         $(HOME)/mod
MODPREFIX=      cgca
LIBNAME=        lib$(CGLIB)

COAR_FLAGS=     -coarray=distributed -coarray-config-file=$(CA_CONF_FILE)

FFLAGS=         -c -qopt-report -O2 -debug full -g -traceback -free -warn \
                $(COAR_FLAGS) -I$(MODDIR)
#-std08 -warn stderrors -mt_mpi

LDFLAGS=        -qopt-report $(COAR_FLAGS) $(USER_OPT)
LIBS=           -L$(LIBDIR) -l$(CGLIB)

MODSRC=         testaux.f90
MODMOD=         $(MODSRC:.f90=.mod)
MODOBJ=         $(MODSRC:.f90=.o)
MOD_CLEAN=      $(MODMOD) $(MODOBJ) $(MOD_RPT)

SRC= \
testAAA.f90 testAAB.f90 testAAC.f90 testAAD.f90 testAAE.f90 \
```

```
testAAF.f90 testAAG.f90 testAAH.f90 testAAI.f90 testAAJ.f90 \
testAAK.f90 testAAL.f90 testAAM.f90 testAAN.f90 testAAO.f90 \
testAAP.f90 testAAQ.f90 testAAR.f90 testAAS.f90 testAAT.f90 \
testAAV.f90 testAAW.f90 testAAX.f90 testAAY.f90 \
testAAZ.f90 \
testABA.f90 testABB.f90 testABC.f90 testABD.f90 testABE.f90 \
testABF.f90 testABG.f90 testABH.f90 testABI.f90 testABJ.f90 \
                        testABM.f90                             \
testABP.f90 testABQ.f90 testABR.f90 testABS.f90 testABT.f90 \
testABU.f90             testABW.f90             testABY.f90 \
testABZ.f90 \
testACA.f90 testACB.f90
# testABK.f90 testABL.f90 testABN.f90 testABO.f90 testABV.f90 - co_sum not supported by ifort 15
# testABX.f90 - Cray parallel IO extensions
# testAAU.f90

OBJ=            ${SRC:.f90=.o}
EXE=            ${SRC:.f90=.x}

ALL_CLEAN=      $(MOD_CLEAN) $(OBJ) $(EXE) *optrpt

.SUFFIXES: .f90 .o .x .mod

all: $(OBJ) $(EXE)

.f90.o:
        $(FC) -c $< $(FFLAGS)

.f90.mod:
        $(FC) -c $< $(FFLAGS)

.o.x:
        $(FC) -o $@ $< $(MODOBJ) $(LDFLAGS) $(LIBS)

$(OBJ): $(MODMOD) $(MODDIR)/$(MODPREFIX)*.mod $(LIBDIR)/$(LIBNAME).a
$(MODMOD) $(MODOBJ): $(MODDIR)/$(MODPREFIX)*.mod $(LIBDIR)/$(LIBNAME).a
$(EXE): $(MODOBJ)

clean:
        \rm $(ALL_CLEAN)
```

67 tests/Makefile-tests-opencoarrays

[Make files]

NAME

Makefile-tests-opencoarrays

SYNOPSIS

```
#$Id: Makefile-opencoarrays 404 2017-05-15 15:14:21Z mexas $

FC=             caf
```

PURPOSE

Build CGPACK tests with GCC/OpenCoarrays.

NOTES

Adjust the inlude and link paths as needed, to make sure all *mod files and libraries are available.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

SOURCE

```
CGLIB=          cgpack
LIBDIR=         $(HOME)/lib
MODDIR=         $(HOME)/mod
MODFILES=       $(MODDIR)/cgca.mod
LIBNAME=        lib$(CGLIB).a
LIB=            -L$(LIBDIR) -l$(CGLIB) \

FFLAGS=         -Wall -fall-intrinsics -I. -I$(MODDIR) \
                -g -fbacktrace -fcheck-array-temporaries -O2

#               -L$(HOME)/soft/hdf5-1.10.0-patch1-install/lib \
#               -lhdf5_fortran -lhdf5 -lhdf5hl_fortran -lhdf5_hl \
#               -L$(HOME)/soft/netcdf-4.4.1.1-install/lib -lnetcdf \
#               -L$(HOME)/soft/netcdf-fortran-4.4.4-install/lib -lnetcdff

MODSRC=         testaux.f90
MODMOD=         $(MODSRC:.f90=.mod)
MODSMOD=        $(MODSRC:.f90=.smod)
MODOBJ=         $(MODSRC:.f90=.o)
CLEAN+=         $(MODMOD) $(MODOBJ) $(MODSMOD)

SRC= \
testAAA.f90 testAAB.f90 testAAC.f90 testAAD.f90 testAAE.f90 \
testAAF.f90 testAAG.f90 testAAH.f90 testAAI.f90 testAAJ.f90 \
testAAK.f90 testAAL.f90 testAAM.f90 testAAN.f90 testAAO.f90 \
```

```
testAAP.f90 testAAQ.f90 testAAR.f90 testAAS.f90 testAAT.f90 \
testAAU.f90 testAAV.f90 testAAW.f90 testAAX.f90 testAAY.f90 \
testAAZ.f90 \
testABA.f90 testABB.f90 testABC.f90 testABD.f90 testABE.f90 \
testABF.f90 testABG.f90 testABH.f90 testABI.f90 testABJ.f90 \
            testABL.f90 testABM.f90 testABN.f90 testABO.f90 \
testABP.f90 testABQ.f90 testABR.f90 testABS.f90 testABT.f90 \
testABU.f90 testABV.f90 testABW.f90             testABY.f90 \
testABZ.f90 \
testACA.f90 testACB.f90 testACC.f90 testACD.f90 testACE.f90 \
#testACF.f90 - netcdf, hdf5
# testABK.f90 testABX.f90 - Cray only
OBJ=            $(SRC:.f90=.o)
EXE=            $(SRC:.f90=.x)
CLEAN+=         $(OBJ) $(EXE)

.SUFFIXES:
.SUFFIXES:      .f90 .o .x

all:            $(MODOBJ) $(MODMOD) $(OBJ) $(EXE)

.f90.o:
                $(FC) -c $< $(FFLAGS)

.o.x:
                $(FC) -o $@ $< $(MODOBJ) $(LIB)

# Extra dependencies
$(EXE):         $(LIBDIR)/$(LIBNAME) $(MODOBJ)
$(OBJ):         $(MODMOD) $(MODSMOD) $(MODFILES)
$(MODOBJ):      $(MODFILES)

clean:
                \rm $(CLEAN)
```

68 tests/mpi_hxvn

[Unit tests]

NAME

mpi_hxvn

SYNOPSIS

```
!$Id: mpi_hxvn.f90 559 2018-10-14 17:54:00Z mexas $

program mpi_hxvn
```

PURPOSE

Test MPI HX routines: ca_mpi_halo_type_create (1.8.1), ca_mpi_hx_all (1.8.3), ca_mpi_halo_type_free (1.8.2). The kernel is a simply copy.

DESCRIPTION

Even though halo coarrays are not needed in MPI HX, they are used because they provide a logical 3D arrangement of images/MPI ranks.

```
- ca_mpi_halo_type_create, ca_mpi_halo_type_free - MPI type ops
- ca_mpi_hx_all - a high level routine to do all necessary HX
```

Must work on any number of images, except when a good decomposition cannot be made. The user needs know nothing about sync.

NOTE AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

casup (3)

USED BY

Part of casup (3) test suite

SOURCE

```
use casup

implicit none

integer( kind=iarr ), parameter :: huge_iarr = huge(0_iarr)

real( kind=rdef ) :: &
  qual,              & ! quality
  bsz0(3),           & ! the given "box" size
  bsz(3),            & ! updated "box" size
  dm,                & ! mean grain size, linear dim, phys units
```

```
  lres,              & ! linear resolution, cells per unit of length
  res                  ! resolutions, cells per grain

integer( kind=iarr ), allocatable :: space(:,:,:),                    &
  space1(:,:,:)

integer( kind=idef ) :: ir(3), nimgs, img, ng, c(3) ! coarray dimensions

integer( kind=ilrg ) :: icells, mcells

integer :: ierr, d, run

logical :: flag

!*********************************************************************72
! first executable statement

!bsz0 = (/ 4.0e2, 8.0e2, 6.0e2 /) ! numbers of cells in CA space
bsz0 = (/ 4.0e1, 8.0e1, 6.0e1 /) ! for testing on FreeBSD laptop
  dm = 1.0 ! cell size
 res = 1.0 ! resolution

  img = this_image()
nimgs = num_images()

! do a check on image 1
if ( img .eq. 1 ) then
  write (*,*) "running on", nimgs, "images in a 3D grid"
  write (*,*) "iarr kind:", iarr, "huge(0_iarr):", huge_iarr

  ! In this test space is assigned image numbers - must be big enough
  ! integer kind to avoid inteter overflow.
  if ( nimgs .gt. huge_iarr ) then
    write (*,*) "ERROR: num_images(): ", nimgs,              &
      " is greater than huge(0_iarr)"
    error stop
  end if
end if

! each image calculates the coarray grid dimensions
call cgca_gdim( nimgs, ir, qual )

! calculate the resolution and the actual phys dimensions of the box
! subroutine cgca_cadim( bsz, res, dm, ir, c, lres, ng )
! c - coarray sizes
! ir - coarray grid sizes
bsz = bsz0
call cgca_cadim( bsz, res, dm, ir, c, lres, ng )

! Check that the partition is sane
if ( img .eq. 1 ) then
  if ( any(int(bsz) .ne. int(bsz0) ) ) then
```

```
    write (*,*)                                                          &
      "ERROR: bad decomposition - use a 'nicer' number of images"
    write (*,*) "ERROR: wanted         :", int(bsz0)
    write (*,*) "ERROR: but got instead:", int(bsz)
    error stop
  end if
end if

! total number of cells in a coarray
icells = int( c(1), kind=ilrg ) * int( c(2), kind=ilrg ) *             &
         int( c(3), kind=ilrg )

! total number of cells in the model
mcells = icells * int( nimgs, kind=ilrg )

if ( img .eq. 1 ) then
  write ( *, "(8(a,i0),tr1,g10.3,tr1,g10.3,3(a,g10.3),a)" )            &
    "nimgs: ", nimgs, " (", c(1), "," , c(2), "," , c(3), ")[",        &
    ir(1), "," , ir(2), "," , ir(3), "] ", ng, qual, lres,             &
    " (", bsz(1), ",", bsz(2), ",", bsz(3), ")"
     write (*,'(a,i0,a)') "Each image has ",icells, " cells"
     write (*,'(a,i0,a)') "The model has ", mcells, " cells"
end if

! Initialise MPI if not done already
call MPI_INITIALIZED( flag, ierr)
if ( .not. flag ) then
  call MPI_INIT( ierr )
  if ( img .eq. 1 ) write (*,*) "MPI not initialised, doing now!"
end if

! run=1 => ca_iter_tl
! run=2 => ca_iter_dc
! run=3 => ca_iter_omp
outer: do run=1,3

  if ( img .eq. 1 ) then
    select case( run )
    case(1)
      write (*,*) "Checking ca_iter_tl - triple loop"
    case(2)
      write (*,*) "Checking ca_iter_dc - do concurrent"
    case(3)
      write (*,*) "Checking ca_iter_omp - OpenMP"
    end select
  end if

  ! Loop over several halo depths
  ! The max halo depth is 1/4 of the min dimension
  ! of the space CA array
  main: do d=1, int( 0.25 * min( c(1), c(2), c(3) ) )
```

```
! allocate space array
!    space - CA array to allocate, with halos!
!        c - array with space dimensions
!        d - depth of the halo layer

call ca_spalloc( space,  c, d )
call ca_spalloc( space1, c, d )

! Set space to my image number
space  = int( img, kind=iarr )
space1 = space

! allocate hx arrays, implicit sync all
! ir(3) - codimensions
call ca_halloc( ir )

! Create MPI subarray types
call ca_mpi_halo_type_create( space )

! do hx, remote ops
call ca_mpi_hx_all( space )

! halo check, local ops
! space - space array, with halos
! flag - default integer
call ca_hx_check(    space=space, flag=ierr )
if ( ierr .ne. 0 ) then
  write (*,*) "ERROR: ca_hx_check    failed: img:", img,           &
              "flag:", ierr
  error stop
end if

! CA iterations
! subroutine ca_run(    space, hx_sub, iter_sub, kernel, niter )
select case( run )
case(1)
  call ca_run(    space = space, hx_sub = ca_mpi_hx_all,           &
    iter_sub = ca_iter_tl,  kernel = ca_kernel_copy, niter = 13 )
case(2)
  call ca_run(    space = space, hx_sub = ca_mpi_hx_all,           &
    iter_sub = ca_iter_dc,  kernel = ca_kernel_copy, niter = 13 )
case(3)
  call ca_run(    space = space, hx_sub = ca_mpi_hx_all,           &
    iter_sub = ca_iter_omp, kernel = ca_kernel_copy, niter = 13 )
end select

! Must be the same
if ( any( space(  1:c(1), 1:c(2), 1:c(3) ) .ne.                   &
          space1( 1:c(1), 1:c(2), 1:c(3) ) ) ) then
  write (*,*) "img:", img, "FAIL: space .ne. space1"
  error stop
end if
```

```
    ! deallocate halos, implicit sync all
    call ca_hdalloc

    ! free halo types
    call ca_mpi_halo_type_free

    ! deallocate space
    deallocate( space )
    deallocate( space1 )

    if (img .eq. 1 ) write (*,*) "PASS, halo depth:", d

  end do main

end do outer

end program mpi_hxvn
```

69 tests/mpi_ising

[Unit tests]

NAME

mpi_ising

SYNOPSIS

```
!$Id: mpi_ising.f90 559 2018-10-14 17:54:00Z mexas $

program mpi_ising
```

PURPOSE

Test ising (49) magnetisation. MPI comms.

DESCRIPTION

See ca_kernel_ising (1.15) and related routines for details. Note that I use a reproducible RND seed and generate a single sequence of RND values for the whole CA model. Thus the results must be exactly reproducible on any number of images. I include the reference value for the final magnetisation (unscaled, integer). If the test does not produce the same value, it fails. However... the ref magnetisation value is obtained here with gfortran7. It is possible (likely?) that other compliers will produce a different sequence of RND from the same seed. In such cases users need to replace the ref value accordingly.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

cgca

USED BY

Part of CGPACK test suite

SOURCE

```
use casup

implicit none

integer( kind=iarr ), parameter :: huge_iarr = huge(0_iarr)

! Reference values for different compilers for magnet_ref,
! magnetisation at the end of simulation
!integer( kind=iarr ), parameter :: magnet_ref = 863379 ! gfortran7
integer( kind=iarr ), parameter :: magnet_ref = 864070 ! Cray

real( kind=rdef ) :: &
  qual,              & ! quality
  bsz0(3),           & ! the given "box" size
```

```
  bsz(3),            & ! updated "box" size
  dm,                & ! mean grain size, linear dim, phys units
  lres,              & ! linear resolution, cells per unit of length
  res                  ! resolutions, cells per grain

integer( kind=iarr ), allocatable :: space(:,:,:), space0(:,:,:)

integer( kind=idef ) :: ir(3), nimgs, img, ng, c(3) ! coarray dimensions

integer( kind=ilrg ) :: icells, mcells

!real, allocatable :: space_ini(:,:,:), rnd_array(:)

integer :: i, iter, seed_size, run, ierr
integer( kind=ilrg ) :: energy0, energy1, energy2, magnet0, magnet1,   &
  magnet2
integer, allocatable :: seed_array(:)

logical :: flag

real :: time1, time2

!*********************************************************************72
! first executable statement

 dm = 1.0 ! Linear "size" of one spin cell
res = 1.0 ! resolution, CA cells per spin

! When dm=res=1, then bsz0 is simply CA dimensions in cells!
bsz0 = (/ 1.2e2, 1.2e2, 1.2e2 /) ! dimensions of the CA model

  img = this_image()
nimgs = num_images()

! each image calculates the coarray grid dimensions
call cgca_gdim( nimgs, ir, qual )

! calculate the resolution and the actual phys dimensions of the box
! subroutine cgca_cadim( bsz, res, dm, ir, c, lres, ng )
! c - coarray sizes
! ir - coarray grid sizes
bsz = bsz0
call cgca_cadim( bsz, res, dm, ir, c, lres, ng )

! do a check on image 1
if ( img .eq. 1 ) then
  write (*,*) "running on", nimgs, "images in a 3D grid"
  write (*,*) "iarr kind:", iarr, "huge(0_iarr):", huge_iarr

  ! Check that the partition is sane
  if ( any(int(bsz) .ne. int(bsz0) ) ) then
    write (*,*)                                                  &
```

```
      "ERROR: bad decomposition - use a 'nicer' number of images"
    write (*,*) "ERROR: wanted         :", int(bsz0)
    write (*,*) "ERROR: but got instead:", int(bsz)
    error stop
  end if

  ! total number of cells in a coarray
  icells = int( c(1), kind=ilrg ) * int( c(2), kind=ilrg ) *           &
           int( c(3), kind=ilrg )

  ! total number of cells in the model
  mcells = icells * int( nimgs, kind=ilrg )

  write ( *, "(8(a,i0),tr1,g10.3,tr1,g10.3,3(a,g10.3),a)" )            &
    "nimgs: ", nimgs, " (", c(1), "," , c(2), "," , c(3), ")[",        &
    ir(1), "," , ir(2), "," , ir(3), "] ", ng, qual, lres,             &
    " (", bsz(1), ",", bsz(2), ",", bsz(3), ")"
     write (*,'(a,i0,a)') "Each image has ",icells, " cells"
     write (*,'(a,i0,a)') "The model has ", mcells, " cells"

  ! In this test sum over all cells on an image is done, so the kind
  ! must be big enough to contain the total number of cells
  ! on an image.
  if ( icells .gt. huge_iarr ) then
    write (*,*) "ERROR: number of cells on an image:", icells,         &
      "is greater than huge(0_iarr)"
    error stop
  end if

end if

! allocate space arrays and set all values to zero
!    space - CA array to allocate, with halos!
!        c - array with space dimensions
!        d - depth of the halo layer
call ca_spalloc( space,  c, 1 )
call ca_spalloc( space0, c, 1 )

! allocate hx arrays, implicit sync all
! mask_array is set inside too.
! ir(3) - codimensions
call ca_halloc( ir )

! Init RND
!call cgca_irs( debug = .false. )
! Use a reproducible RND here for verification
call random_seed( size = seed_size )
allocate( seed_array( seed_size ) )
seed_array = (/ (i, i=1,seed_size) /)

! Set space arrays
if (img .eq. 1) write (*,*) "RND, serial IO, etc. - wait..."
```

```
call ca_set_space_rnd( seed = seed_array, frac1=0.5, space = space )

! Initialise MPI if not done already
call MPI_INITIALIZED( flag, ierr)
if ( .not. flag ) then
  call MPI_INIT( ierr )
  if ( img .eq. 1 ) write (*,*) "MPI not initialised, doing now!"
end if

! Create MPI subarray types
call ca_mpi_halo_type_create( space )

! Calculate initial energy and magnetisation

! run=1 => ca_iter_tl
! run=2 => ca_iter_dc
! run=3 => ca_iter_omp
do run=1,3
  select case(run)
  case(1)
    call ca_mpi_ising_energy( space = space, hx_sub = ca_mpi_hx_all,   &
      iter_sub = ca_iter_tl,  kernel = ca_kernel_ising_ener,           &
      energy = energy0, magnet = magnet0 )
  case(2)
    call ca_mpi_ising_energy( space = space, hx_sub = ca_mpi_hx_all,   &
      iter_sub = ca_iter_dc,  kernel = ca_kernel_ising_ener,           &
      energy = energy1, magnet = magnet1 )
  case(3)
    call ca_mpi_ising_energy( space = space, hx_sub = ca_mpi_hx_all,   &
      iter_sub = ca_iter_omp, kernel = ca_kernel_ising_ener,           &
      energy = energy2, magnet = magnet2 )
  end select
end do

if (img .eq. 1 ) then
  write (*,*) "Initial energy and magnetisation"
  write (*,*) "ca_iter_tl :", energy0, magnet0
  write (*,*) "ca_iter_dc :", energy1, magnet1
  write (*,*) "ca_iter_omp:", energy2, magnet2
  if ( energy0 .ne. energy1 .or. magnet0 .ne. magnet1 .or.             &
       energy0 .ne. energy2 .or. magnet0 .ne. magnet2 ) then
    write (*,*) "FAIL: ca_iter_tl, ca_iter_dc, ca_iter_omp differ"
    error stop
  else
    write (*,*) "PASS: ca_iter_tl, ca_iter_dc, ca_iter_omp agree"
  end if
end if

! save old space as space0
space0 = space

! run=1 => ca_iter_tl
```

```
! run=2 => ca_iter_dc
! run=3 => ca_iter_omp
main: do run=1,3

  ! Reset space to space0
  space = space0

  ! Start counter
  if ( img .eq. 1 ) call cpu_time( time1 )

  ! CA iterations
  loop: do iter = 1,100

    ! Check energy after every iter
    ! subroutine ca_run( space, hx_sub, iter_sub, kernel, niter )
    select case(run)
    case(1)
      call ca_run(    space = space, hx_sub = ca_mpi_hx_all,           &
        iter_sub = ca_iter_tl,  kernel = ca_kernel_ising, niter = 1 )
      call ca_mpi_ising_energy( space = space, hx_sub = ca_mpi_hx_all, &
        iter_sub = ca_iter_tl,  kernel = ca_kernel_ising_ener,         &
        energy = energy1, magnet = magnet1 )
    case(2)
      call ca_run(    space = space, hx_sub = ca_mpi_hx_all,           &
        iter_sub = ca_iter_dc,  kernel = ca_kernel_ising, niter = 1 )
      call ca_mpi_ising_energy( space = space, hx_sub = ca_mpi_hx_all, &
        iter_sub = ca_iter_dc,  kernel = ca_kernel_ising_ener,         &
        energy = energy1, magnet = magnet1 )
    case(3)
      call ca_run(    space = space, hx_sub = ca_mpi_hx_all,           &
        iter_sub = ca_iter_omp, kernel = ca_kernel_ising, niter = 1 )
      call ca_mpi_ising_energy( space = space, hx_sub = ca_mpi_hx_all, &
        iter_sub = ca_iter_omp, kernel = ca_kernel_ising_ener,         &
        energy = energy1, magnet = magnet1 )
    end select

    if (img .eq. 1 ) then
      if ( energy1 .ne. energy0 ) then
        write (*,*) "FAIL: energy0:", energy0, "energy1:", energy1
        error stop
      else
!       write (*,"(a,i0,a,es18.6)") "Magnetisation_after_iter_",      &
!         iter, ":", real(magnet1) / real(mcells)
      end if
    end if

  end do loop

  ! Stop counter
  if ( img .eq. 1 ) call cpu_time( time2 )

  if (img .eq. 1 ) then
```

```
    select case(run)
      case(1)
        if ( magnet1 .eq. magnet_ref ) then
          write (*,*) "PASS: ca_iter_tl : final mag:", magnet1
          write (*,*) "TIME ca_iter_tl, s :", time2-time1
        else
          write (*,"(2(a,i0))")                                          &
            "FAIL: ca_iter_tl: magnetisation ref value: ",               &
             magnet_ref, "my value:", magnet1
        end if
      case(2)
        if ( magnet1 .eq. magnet_ref ) then
          write (*,*) "PASS: ca_iter_dc : final mag:", magnet1
          write (*,*) "TIME ca_iter_dc, s :", time2-time1
        else
          write (*,"(2(a,i0))")                                          &
            "FAIL: ca_iter_dc: magnetisation ref value: ",               &
             magnet_ref, "my value:", magnet1
        end if
      case(3)
        if ( magnet1 .eq. magnet_ref ) then
          write (*,*) "PASS: ca_iter_omp: final mag:", magnet1
          write (*,*) "TIME ca_iter_omp, s:", time2-time1
        else
          write (*,"(2(a,i0))")                                          &
            "FAIL: ca_iter_omp: magnetisation ref value: ",              &
             magnet_ref, "my value:", magnet1
        end if
    end select

  end if

end do main

! deallocate halos, implicit sync all
call ca_hdalloc

! free halo types
call ca_mpi_halo_type_free

! deallocate space
deallocate( space )
deallocate( space0 )

end program mpi_ising
```

70 tests/mpi_ising_perf

[Unit tests]

NAME

mpi_ising_perf

SYNOPSIS

```
!$Id: mpi_ising_perf.f90 561 2018-10-14 20:48:19Z mexas $

program mpi_ising_perf
```

PURPOSE

Test performance of ising (49) magnetisation. MPI comms. Reproducibility on any number of images requires a serial RND for the whole model. This is too slow. Here each image uses its own RND, meaning the results are *not* reproducible when the number of images change. This test is purely for performance analysis.

DESCRIPTION

See ca_kernel_ising (1.15) and related routines for details. Note that I use a reproducible RND seed and generate a single sequence of RND values for the whole CA model. Thus the results must be exactly reproducible on any number of images. I include the reference value for the final magnetisation (unscaled, integer). If the test does not produce the same value, it fails. However... the ref magnetisation value is obtained here with gfortran7. It is possible (likely?) that other compliers will produce a different sequence of RND from the same seed. In such cases users need to replace the ref value accordingly.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

cgca

USED BY

Part of CGPACK test suite

SOURCE

```
use casup

implicit none

integer( kind=iarr ), parameter :: huge_iarr = huge(0_iarr)

real( kind=rdef ) :: &
  qual,              & ! quality
  bsz0(3),           & ! the given "box" size
  bsz(3),            & ! updated "box" size
  dm,                & ! mean grain size, linear dim, phys units
```

```
  lres,              & ! linear resolution, cells per unit of length
  res                  ! resolutions, cells per grain

integer( kind=iarr ), allocatable :: space(:,:,:),                    &
  space0(:,:,:)

integer( kind=idef ) :: ir(3), nimgs, img, ng, c(3) ! coarray dimensions

integer( kind=ilrg ) :: icells, mcells

!real, allocatable :: space_ini(:,:,:), rnd_array(:)

integer :: i, iter, seed_size, run, ierr
integer( kind=ilrg ) :: energy0, energy1, energy2, magnet0, magnet1,   &
  magnet2
integer, allocatable :: seed_array(:)

logical :: flag

real :: time1, time2
real, allocatable :: rnd_arr(:,:,:)

!*********************************************************************72
! first executable statement

 dm = 1.0 ! Linear "size" of one spin cell
res = 1.0 ! resolution, CA cells per spin

! Read the box size from command line
call ca_cmd_real( n=3, data=bsz0 )

! When dm=res=1, then bsz0 is simply CA dimensions in cells!
!bsz0 = (/ 2.0e3, 2.0e3, 2.0e3 /) ! dimensions of the CA model
!bsz0 = (/ 3.0e3, 3.0e3, 3.0e3 /) ! dimensions of the CA model
!bsz0 = (/ 1.2e2, 1.2e2, 1.2e2 /) ! for testing on FreeBSD laptop

  img = this_image()
nimgs = num_images()

! each image calculates the coarray grid dimensions
call cgca_gdim( nimgs, ir, qual )

! calculate the resolution and the actual phys dimensions of the box
! subroutine cgca_cadim( bsz, res, dm, ir, c, lres, ng )
! c - coarray sizes
! ir - coarray grid sizes
bsz = bsz0
call cgca_cadim( bsz, res, dm, ir, c, lres, ng )

! do a check on image 1
if ( img .eq. 1 ) then
  write (*,*) "running on", nimgs, "images in a 3D grid"
```

```
  write (*,*) "iarr kind:", iarr, "huge(0_iarr):", huge_iarr

! No
! check
! for
! bad
! partition
! in
! this
! test

  ! total number of cells in a coarray
  icells = product( int( c, kind=ilrg ) )

  ! total number of cells in the model
  mcells = icells * int( nimgs, kind=ilrg )

  write ( *, "(8(a,i0),tr1,g10.3,tr1,g10.3,3(a,g10.3),a)" )            &
    "nimgs: ", nimgs, " (", c(1), "," , c(2), "," , c(3), ")[",        &
    ir(1), "," , ir(2), "," , ir(3), "] ", ng, qual, lres,             &
    " (", bsz(1), ",", bsz(2), ",", bsz(3), ")"
     write (*,'(a,i0,a)') "Each image has ",icells, " cells"
     write (*,'(a,i0,a)') "The model has ", mcells, " cells"

  ! In this test sum over all cells on an image is done, so the kind
  ! must be big enough to contain the total number of cells
  ! on an image.
  if ( icells .gt. huge_iarr ) then
    write (*,*) "ERROR: number of cells on an image:", icells,         &
      "is greater than huge(0_iarr)"
    error stop
  end if

end if

! allocate space arrays and set all values to zero
!    space - CA array to allocate, with halos!
!        c - array with space dimensions
!        d - depth of the halo layer

call ca_spalloc( space,  c, 1 )
call ca_spalloc( space0, c, 1 )

! allocate hx arrays, implicit sync all
! mask_array is set inside too.
! ir(3) - codimensions
call ca_halloc( ir )

! Init RND
!call cgca_irs( debug = .false. )
! Use a reproducible RND here for verification
call random_seed( size = seed_size )
```

```
allocate( seed_array( seed_size ) )
seed_array = (/ (i, i=1,seed_size) /)
call random_seed( put = seed_array )

! Set space arrays
allocate( rnd_arr( lbound(space, dim=1) : ubound(space, dim=1),        &
                   lbound(space, dim=2) : ubound(space, dim=2),        &
                   lbound(space, dim=3) : ubound(space, dim=3) ) )
call random_number( rnd_arr )
space = nint( rnd_arr, kind=iarr )

! Initialise MPI if not done already
call MPI_INITIALIZED( flag, ierr)
if ( .not. flag ) then
  call MPI_INIT( ierr )
  if ( img .eq. 1 ) write (*,*) "MPI not initialised, doing now!"
end if

! Create MPI subarray types
call ca_mpi_halo_type_create( space )

! Calculate initial energy and magnetisation

! run=1 => ca_iter_tl
! run=2 => ca_iter_dc
! run=3 => ca_iter_omp
do run=1,3
  select case(run)
  case(1)
    call ca_mpi_ising_energy( space = space, hx_sub = ca_mpi_hx_all,   &
      iter_sub = ca_iter_tl,  kernel = ca_kernel_ising_ener,           &
      energy = energy0, magnet = magnet0 )
  case(2)
    call ca_mpi_ising_energy( space = space, hx_sub = ca_mpi_hx_all,   &
      iter_sub = ca_iter_dc,  kernel = ca_kernel_ising_ener,           &
      energy = energy1, magnet = magnet1 )
  case(3)
    call ca_mpi_ising_energy( space = space, hx_sub = ca_mpi_hx_all,   &
      iter_sub = ca_iter_omp, kernel = ca_kernel_ising_ener,           &
      energy = energy2, magnet = magnet2 )
  end select
end do

if (img .eq. 1 ) then
  write (*,*) "Initial energy and magnetisation"
  write (*,*) "ca_iter_tl :", energy0, magnet0
  write (*,*) "ca_iter_dc :", energy1, magnet1
  write (*,*) "ca_iter_omp:", energy2, magnet2
  if ( energy0 .ne. energy1 .or. magnet0 .ne. magnet1 .or.             &
       energy0 .ne. energy2 .or. magnet0 .ne. magnet2 ) then
    write (*,*) "FAIL: ca_iter_tl, ca_iter_dc, ca_iter_omp differ"
    error stop
```

```
  else
    write (*,*) "PASS: ca_iter_tl, ca_iter_dc, ca_iter_omp agree"
  end if
end if

! save old space as space0
space0 = space

! run=1 => ca_iter_tl
! run=2 => ca_iter_dc
! run=3 => ca_iter_omp
main: do run=1,1

  ! Reset space to space0
  space = space0

  ! Start counter
  if ( img .eq. 1 ) call cpu_time( time1 )

  ! CA iterations
  loop: do iter = 1,100

    ! Check energy after every iter
    ! subroutine ca_run(    space, hx_sub, iter_sub, kernel, niter )
    select case(run)
    case(1)
      call ca_run(    space = space, hx_sub = ca_mpi_hx_all,           &
        iter_sub = ca_iter_tl,  kernel = ca_kernel_ising, niter = 1 )
      call ca_mpi_ising_energy( space = space, hx_sub = ca_mpi_hx_all, &
        iter_sub = ca_iter_tl,  kernel = ca_kernel_ising_ener,         &
        energy = energy1, magnet = magnet1 )
    case(2)
      call ca_run(    space = space, hx_sub = ca_mpi_hx_all,           &
        iter_sub = ca_iter_dc,  kernel = ca_kernel_ising, niter = 1 )
      call ca_mpi_ising_energy( space = space, hx_sub = ca_mpi_hx_all, &
        iter_sub = ca_iter_dc,  kernel = ca_kernel_ising_ener,         &
        energy = energy1, magnet = magnet1 )
    case(3)
      call ca_run(    space = space, hx_sub = ca_mpi_hx_all,           &
        iter_sub = ca_iter_omp, kernel = ca_kernel_ising, niter = 1 )
      call ca_mpi_ising_energy( space = space, hx_sub = ca_mpi_hx_all, &
        iter_sub = ca_iter_omp, kernel = ca_kernel_ising_ener,         &
        energy = energy1, magnet = magnet1 )
    end select

    if ( img .eq. 1 ) then
      if ( energy1 .ne. energy0 ) then
        write (*,*) "FAIL: energy0:", energy0, "energy1:", energy1
        error stop
      else
!       write (*,"(a,i0,a,es18.6)") "Magnetisation_after_iter_",      &
!         iter, ":", real(magnet1) / real(mcells)
```

```
      end if
    end if

  end do loop

  if ( img .eq. 1 ) then
    ! Stop counter
    call cpu_time( time2 )

    select case(run)
      case(1)
          write (*,*) "TIME ca_iter_tl, s:", time2-time1
      case(2)
          write (*,*) "TIME ca_iter_dc, s:", time2-time1
      case(3)
          write (*,*) "TIME ca_iter_omp,s:", time2-time1
    end select

  end if

end do main

! deallocate halos, implicit sync all
call ca_hdalloc

! free halo types
call ca_mpi_halo_type_free

! deallocate space
deallocate( space )
deallocate( space0 )

end program mpi_ising_perf
```

71 tests/test_hdf5

[Unit tests]

NAME

test_hdf5

SYNOPSIS

```
!$Id: test_hdf5.f90 526 2018-03-25 23:44:51Z mexas $

program test_hdf5
```

PURPOSE

Test HDF5 IO timing and integrity against serial IO.

DESCRIPTION

Serial IO is cgca_swci (19.2). HDF5 IO is cgca_pswci4 (14.1).

NOTE

Should work on any file system, as long as HDF5 libraries are properly installed.

AUTHOR

Luis Cebamanos, Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

cgca testaux (133)

USED BY

Part of CGPACK test suite

SOURCE

```
use testaux

implicit none

real,parameter :: gigabyte=real(2**30)
logical( kind=ldef ), parameter :: yesdebug = .true., nodebug = .false.

  real( kind=rdef ) ::  &
       qual,            & ! quality
       bsz0(3),         & ! the given "box" size
       bsz(3),          & ! updated "box" size
       dm,              & ! mean grain size, linear dim, phys units
       lres,            & ! linear resolution, cells per unit of length
       res                ! resolutions, cells per grain

  integer( kind=idef ) :: ir(3), nimgs, img, ng, c(3) ! coarray dims
```

```
  integer( kind=iarr ), allocatable :: space(:,:,:,:)[:,:,:]

  integer( kind=ilrg ) :: icells, mcells

  double precision :: t0, t1, tdiff, fsizeb, fsizeg

!*************************************************************************72
! first executable statement

! physical dimensions of the box, assume mm
  bsz0 = (/ 1.0, 1.5, 2.0 /)

! mean grain size, linear dimension, e.g. mean grain diameter, also mm
  dm = 3.0e-1

! resolution
  res = 1.0e5

  img = this_image()
nimgs = num_images()

  ! do a check on image 1
  if ( img .eq. 1 ) then

     ! print a banner
     call banner("hdf5")

     ! print the parameter values
     call cgca_pdmp

     write (*,'(a,i0,a)') "running on ", nimgs, " images in a 3D grid"

  end if

  ! I want pdmp output appear before the rest.
  ! This might help
  sync all

  ! each image calculates the coarray grid dimensions
  call cgca_gdim( nimgs, ir, qual )

  ! calculate the resolution and the actual phys dimensions
  ! of the box
  ! subroutine cgca_cadim( bsz, res, dm, ir, c, lres, ng )
  ! c - coarray sizes
  ! ir - coarray grid sizes
  bsz = bsz0
  call cgca_cadim( bsz, res, dm, ir, c, lres, ng )

  ! total number of cells in a coarray
  icells = int( c(1), kind=ilrg ) * int( c(2), kind=ilrg ) *           &
```

```
       int( c(3), kind=ilrg )

  ! total number of cells in the model
  mcells = icells * int( nimgs, kind=ilrg )

  if ( img .eq. 1 ) then
     write ( *, "(9(a,i0),tr1,g10.3,tr1,g10.3,3(a,g10.3),a)" )         &
          "img: ", img  , " nimgs: ", nimgs, " (", c(1) ,              &
          ","    , c(2) , ","       , c(3) , ")[", ir(1),              &
          ","    , ir(2), ","       , ir(3), "] ", ng   ,              &
          qual, lres,                                                  &
          " (", bsz(1), ",", bsz(2), ",", bsz(3), ")"
     write (*,'(a,i0,a)') "Each image has ",icells, " cells"
     write (*,'(a,i0,a)') "The model has ", mcells, " cells"
  end if

  ! Total output file size, in B and in GB.
  fsizeb = real( mcells * storage_size( space, kind=ilrg ) / 8_ilrg )
  fsizeg = fsizeb / gigabyte

! allocate space coarray with a single layer
! implicit sync all
!subroutine cgca_as( l1, u1, l2, u2, l3, u3, col1, cou1, col2, cou2,   &
!                    col3, props, coarray )
call cgca_as(1, c(1), 1, c(2), 1, c(3), 1, ir(1), 1, ir(2), 1, 1, space)

! initialise coarray to image number
space = int( img, kind=iarr )

sync all

! start MPI
!  call MPI_Init(ierr)

! serial
! subroutine cgca_swci( coarray, stype, iounit, fname )
t0 = cgca_benchtime()
call cgca_swci( space, cgca_state_type_grain, 10, "seri.raw" )
t1 = cgca_benchtime()

sync all

if (img .eq. 1)  then
  tdiff = t1 - t0
  write (*,*) "Serial time: ", tdiff, "s. Rate: ", fsizeg/tdiff, "GB/s."
end if

! hdf5
! subroutine cgca_pswci4( coarray, stype, fname )
t0 = cgca_benchtime()
call cgca_pswci4( space, cgca_state_type_grain, "hdf5.dat" )
t1 = cgca_benchtime()
```

```
sync all

if (img .eq. 1)  then
  tdiff = t1 - t0
  write (*,*) "HDF5 time: ", tdiff, "s. Rate: ", fsizeg/tdiff, "GB/s."
end if

! terminate MPI
! call MPI_Finalize(ierr)

  ! deallocate all arrays
  call cgca_ds(space)

end program test_hdf5
```

72 tests/test_hxi

[Unit tests]

NAME

test_hxi

SYNOPSIS

```
!$Id: test_hxi.f90 529 2018-03-26 11:25:45Z mexas $

program test_hxi
```

PURPOSE

Test cgca_hxi (15.2), a halo exchange subroutine.

DESCRIPTION

Assign coarrays on each image the value of this_image(). Then do a hx call and check that the halos are flag.

NOTE AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

cgca testaux (133)

USED BY

Part of CGPACK test suite

SOURCE

```
use testaux

implicit none

real,parameter :: gigabyte=real(2**30)
logical( kind=ldef ), parameter :: yesdebug = .true., nodebug = .false.

real( kind=rdef ) ::   &
       qual,           & ! quality
       bsz0(3),        & ! the given "box" size
       bsz(3),         & ! updated "box" size
       dm,             & ! mean grain size, linear dim, phys units
       lres,           & ! linear resolution, cells per unit of length
       res               ! resolutions, cells per grain

integer( kind=idef ) :: ir(3), nimgs, ng, c(3) ! coarray dims

integer( kind=iarr ), allocatable :: space(:,:,:,:)[:,:,:]
```

```
integer( kind=iarr ) :: img

integer( kind=ilrg ) :: icells, mcells

integer :: flag

!**********************************************************************72
! first executable statement

! Set to an error value initially

flag = 1
! physical dimensions of the box, assume mm
  bsz0 = (/ 1.0, 1.1, 0.9 /)

! mean grain size, linear dimension, e.g. mean grain diameter, also mm
  dm = 5.0e-1

! resolution
  res = 1.0e5

  img = int( this_image(), kind=iarr )
nimgs = num_images()

  ! do a check on image 1
  if ( img .eq. 1 ) then

     ! print a banner
     call banner("hxi")

     ! print the parameter values
     call cgca_pdmp

     write (*,'(a,i0,a)') "running on ", nimgs, " images in a 3D grid"

  end if

  ! I want pdmp output appear before the rest.
  ! This might help
  sync all

  ! each image calculates the coarray grid dimensions
  call cgca_gdim( nimgs, ir, qual )

  ! calculate the resolution and the actual phys dimensions
  ! of the box
  ! subroutine cgca_cadim( bsz, res, dm, ir, c, lres, ng )
  ! c - coarray sizes
  ! ir - coarray grid sizes
  bsz = bsz0
  call cgca_cadim( bsz, res, dm, ir, c, lres, ng )
```

```
  ! total number of cells in a coarray
  icells = int( c(1), kind=ilrg ) * int( c(2), kind=ilrg ) *           &
       int( c(3), kind=ilrg )

  ! total number of cells in the model
  mcells = icells * int( nimgs, kind=ilrg )

  if ( img .eq. 1 ) then
     write ( *, "(9(a,i0),tr1,g10.3,tr1,g10.3,3(a,g10.3),a)" )         &
          "img: ", img  , " nimgs: ", nimgs, " (", c(1) ,              &
          ","    , c(2) , ","       , c(3) , ")[", ir(1),              &
          ","    , ir(2), ","       , ir(3), "] ", ng   ,              &
          qual, lres,                                                  &
          " (", bsz(1), ",", bsz(2), ",", bsz(3), ")"
     write (*,'(a,i0,a)') "Each image has ",icells, " cells"
     write (*,'(a,i0,a)') "The model has ", mcells, " cells"
  end if

! allocate space coarray with a single layer
! implicit sync all
!subroutine cgca_as( l1, u1, l2, u2, l3, u3, col1, cou1, col2, cou2,   &
!                    col3, props, coarray )
call cgca_as(1, c(1), 1, c(2), 1, c(3), 1, ir(1), 1, ir(2), 1, 1, space)

space = img
sync all

call cgca_hxi( space )
sync all

! Test that hx is flag
call cgca_hxic( space, flag )

call co_sum( flag )

if ( img .eq. 1 ) then
  if ( flag .eq. 0 )  write (*,*) "hxi PASS"
end if

sync all

space = img
sync all

call cgca_hxir( space )
sync all

! Test that hx is flag
call cgca_hxic( space, flag )

call co_sum( flag )
```

```
if ( img .eq. 1 ) then
  if ( flag .eq. 0 )  write (*,*) "hxir PASS"
end if

! deallocate all arrays
call cgca_ds( space )

end program test_hxi
```

73 tests/test_hxir

[Unit tests]

NAME

test_hxir

SYNOPSIS

```
!$Id: test_hxir.f90 526 2018-03-25 23:44:51Z mexas $

program test_hxir
```

PURPOSE

Test cgca_hxir (15.4), a halo exchange subroutine with random order of remote calls.

DESCRIPTION

Two separate solidifications are performed, which should result in bit-by-bit idendical microstructures. Both solidification runs use m3sld_hc (31.4)/cgca_sld_h (31.4.1), a routine that offers a choice of different halo exchange routines. One solidification is using cgca_hxi (15.2), a halo exchange routine with ordered, predictable sequence of remote calls, the same on all images. Another solidification is using cgca_hxir (15.4), a halo exchange routine with random order of remote calls. MPI/IO is used for speed.

NOTE AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

cgca testaux (133)

USED BY

Part of CGPACK test suite

SOURCE

```
use testaux

implicit none

real,parameter :: gigabyte=real(2**30)
logical( kind=ldef ), parameter :: yesdebug = .true., nodebug = .false.

real( kind=rdef ) ::   &
       qual,           & ! quality
       bsz0(3),        & ! the given "box" size
       bsz(3),         & ! updated "box" size
       dm,             & ! mean grain size, linear dim, phys units
       lres,           & ! linear resolution, cells per unit of length
       res               ! resolutions, cells per grain
```

```
real( kind=rdef ), allocatable :: grt(:,:,:)[:,:,:]

integer( kind=idef ) :: ir(3), nimgs, img, ng, i, ierr, funit, c(3)

integer( kind=iarr ), allocatable :: space(:,:,:,:)[:,:,:]

integer( kind=ilrg ) :: icells, mcells

logical( kind=ldef ) :: solid

character(len=5) :: fnum

double precision :: t0, t1, tdiff, fsizeb, fsizeg

!*********************************************************************72
! first executable statement

! physical dimensions of the box, assume mm
  bsz0 = (/ 1.0, 1.1, 0.9 /)

! mean grain size, linear dimension, e.g. mean grain diameter, also mm
  dm = 5.0e-1

! resolution
  res = 1.0e5

  img = this_image()
nimgs = num_images()

  ! do a check on image 1
  if ( img .eq. 1 ) then

     ! print a banner
     call banner("hxir")

     ! print the parameter values
     call cgca_pdmp

     write (*,'(a,i0,a)') "running on ", nimgs, " images in a 3D grid"

  end if

  ! I want pdmp output appear before the rest.
  ! This might help
  sync all

  ! each image calculates the coarray grid dimensions
  call cgca_gdim( nimgs, ir, qual )

  ! calculate the resolution and the actual phys dimensions
  ! of the box
  ! subroutine cgca_cadim( bsz, res, dm, ir, c, lres, ng )
```

```
  ! c - coarray sizes
  ! ir - coarray grid sizes
  bsz = bsz0
  call cgca_cadim( bsz, res, dm, ir, c, lres, ng )

  ! total number of cells in a coarray
  icells = int( c(1), kind=ilrg ) * int( c(2), kind=ilrg ) *           &
       int( c(3), kind=ilrg )

  ! total number of cells in the model
  mcells = icells * int( nimgs, kind=ilrg )

  if ( img .eq. 1 ) then
     write ( *, "(9(a,i0),tr1,g10.3,tr1,g10.3,3(a,g10.3),a)" )         &
          "img: ", img  , " nimgs: ", nimgs, " (", c(1) ,              &
          ","    , c(2) , ","       , c(3) , ")[", ir(1),              &
          ","    , ir(2), ","       , ir(3), "] ", ng   ,              &
          qual, lres,                                                  &
          " (", bsz(1), ",", bsz(2), ",", bsz(3), ")"
     write (*,'(a,i0,a)') "Each image has ",icells, " cells"
     write (*,'(a,i0,a)') "The model has ", mcells, " cells"
     write (*,*) "dataset sizes for ParaView", c*ir
  end if

  ! Total output file size, in B and in GB.
  fsizeb = real( mcells * storage_size( space, kind=ilrg ) / 8_ilrg )
  fsizeg = fsizeb / gigabyte

! allocate space coarray with a single layer
! implicit sync all
!subroutine cgca_as( l1, u1, l2, u2, l3, u3, col1, cou1, col2, cou2,   &
!                    col3, props, coarray )
call cgca_as(1, c(1), 1, c(2), 1, c(3), 1, ir(1), 1, ir(2), 1, 1, space)

! Start MPI
call MPI_Init( ierr )

! Do 2 solidifications
do i=1,2

  ! initialise the reproducible random number seed
  call cgca_ins( yesdebug )

  ! initialise space
  space = cgca_liquid_state
!  space = img

  ! allocate rotation tensors
  call cgca_art( 1, ng, 1, ir(1), 1, ir(2), 1, grt )

  ! assign rotation tensors, sync all inside
  call cgca_rt( grt )
```

```
  ! nuclei, sync all inside
  call cgca_nr( space, ng, yesdebug )

  ! Start timer
  t0 = cgca_benchtime()

  ! Solidify, implicit sync all inside
  !  module subroutine cgca_sld_h( coarray, hx, iter, heartbeat, solid )

  if      ( i .eq. 1 ) then
    call cgca_sld_h( space, cgca_hxi,  2, 1, solid )
  else if ( i .eq. 2 ) then
    call cgca_sld_h( space, cgca_hxir, 2, 1, solid )
  end if

  t1 = cgca_benchtime()

  if (img .eq. 1)  then
    tdiff = t1 - t0
    if      ( i .eq. 1 ) then
      write (*,*) "cgca_hxi  time: ", tdiff, "s"
    else if ( i .eq. 2 ) then
      write (*,*) "cgca_hxir time: ", tdiff, "s"
    end if
  end if

  sync all

  ! MPI/IO
  ! subroutine cgca_pswci2( coarray, stype, fname )
  if      ( i .eq. 1 ) then
    call cgca_pswci2( space, cgca_state_type_grain, "hxi.raw"  )
! Plain text output
write (fnum, '(i0)') img
open( newunit=funit, file="hxi_img" // fnum, status="replace" )
write (funit, *) space
close( funit )

  else if ( i .eq. 2 ) then
    call cgca_pswci2( space, cgca_state_type_grain, "hxir.raw" )
! Plain text output
write (fnum, '(i0)') img
open( newunit=funit, file="hxir_img" // fnum, status="replace" )
write (funit, *) space
close( funit )

  end if

  sync all

end do
```

```
! terminate MPI
call MPI_Finalize( ierr )

! deallocate all arrays
call cgca_ds( space )

end program test_hxir
```

74 tests/test_netcdf

[Unit tests]

NAME

test_netcdf

SYNOPSIS

```
!$Id: test_netcdf.f90 533 2018-03-30 14:31:26Z mexas $

program test_netcdf
```

PURPOSE

Test NetCDF IO timing and integrity against serial IO.

DESCRIPTION

Serial IO is cgca_swci (19.2). NetCDF IO is cgca_pswci3 (18.1).

NOTE

Should work on any file system, as long as NetCDF libraries are properly installed.

AUTHOR

Luis Cebamanos, Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

cgca testaux (133)

USED BY

Part of CGPACK test suite

SOURCE

```
use casup

implicit none

real,parameter :: gigabyte=real(2**30)
logical( kind=ldef ), parameter :: yesdebug = .true., nodebug = .false.

  real( kind=rdef ) ::  &
       qual,            & ! quality
       bsz0(3),         & ! the given "box" size
       bsz(3),          & ! updated "box" size
       dm,              & ! mean grain size, linear dim, phys units
       lres,            & ! linear resolution, cells per unit of length
       res                ! resolutions, cells per grain

  integer( kind=idef ) :: ir(3), nimgs, img, ng, c(3) ! coarray dims
```

```
  integer( kind=iarr ), allocatable :: space(:,:,:,:) [:,:,:]

  integer( kind=ilrg ) :: icells, mcells

  double precision :: t0, t1, tdiff, fsizeb, fsizeg

!*********************************************************************72
! first executable statement

! physical dimensions of the box, assume mm
  bsz0 = (/ 1.0, 1.5, 2.0 /)

! mean grain size, linear dimension, e.g. mean grain diameter, also mm
  dm = 3.0e-1

! resolution
  res = 1.0e5

  img = this_image()
nimgs = num_images()

  ! do a check on image 1
  if ( img .eq. 1 ) then

     ! print the parameter values
     call cgca_pdmp

     write (*,'(a,i0,a)') "running on ", nimgs, " images in a 3D grid"

  end if

  ! I want pdmp output appear before the rest.
  ! This might help
  sync all

  ! each image calculates the coarray grid dimensions
  call cgca_gdim( nimgs, ir, qual )

  ! calculate the resolution and the actual phys dimensions
  ! of the box
  ! subroutine cgca_cadim( bsz, res, dm, ir, c, lres, ng )
  ! c - coarray sizes
  ! ir - coarray grid sizes
  bsz = bsz0
  call cgca_cadim( bsz, res, dm, ir, c, lres, ng )

  ! total number of cells in a coarray
  icells = int( c(1), kind=ilrg ) * int( c(2), kind=ilrg ) *          &
       int( c(3), kind=ilrg )

  ! total number of cells in the model
```

```
  mcells = icells * int( nimgs, kind=ilrg )

  if ( img .eq. 1 ) then
     write ( *, "(9(a,i0),tr1,g10.3,tr1,g10.3,3(a,g10.3),a)" )         &
          "img: ", img  , " nimgs: ", nimgs, " (", c(1) ,              &
          ","    , c(2) , ","       , c(3) , ")[", ir(1),              &
          ","    , ir(2), ","       , ir(3), "] ", ng   ,              &
          qual, lres,                                                  &
          " (", bsz(1), ",", bsz(2), ",", bsz(3), ")"
     write (*,'(a,i0,a)') "Each image has ",icells, " cells"
     write (*,'(a,i0,a)') "The model has ", mcells, " cells"
  end if

  ! Total output file size, in B and in GB.
  fsizeb = real( mcells * storage_size( space, kind=ilrg ) / 8_ilrg )
  fsizeg = fsizeb / gigabyte

! allocate space coarray with a single layer
! implicit sync all
!subroutine cgca_as( l1, u1, l2, u2, l3, u3, col1, cou1, col2, cou2,   &
!                    col3, props, coarray )
call cgca_as(1, c(1), 1, c(2), 1, c(3), 1, ir(1), 1, ir(2), 1, 1, space)

! initialise coarray to image number
space = int( img, kind=iarr )

sync all

! start MPI
!  call MPI_Init(ierr)

! serial
! subroutine cgca_swci( coarray, stype, iounit, fname )
t0 = cgca_benchtime()
call cgca_swci( space, cgca_state_type_grain, 10, "seri.raw" )
t1 = cgca_benchtime()

sync all

if (img .eq. 1)  then
  tdiff = t1 - t0
  write (*,*) "Serial time: ", tdiff, "s. Rate: ", fsizeg/tdiff, "GB/s."
end if

! netcdf
! subroutine cgca_pswci3( coarray, stype, fname )
t0 = cgca_benchtime()
call cgca_pswci3( space, cgca_state_type_grain, "netcdf.ncdf" )
t1 = cgca_benchtime()

sync all
```

```
if (img .eq. 1)  then
  tdiff = t1 - t0
  write (*,*) "NetCDF time:", tdiff, "s, rate: ", fsizeg/tdiff, "GB/s."
end if

! terminate MPI
! call MPI_Finalize(ierr)

  ! deallocate all arrays
  call cgca_ds(space)

end program test_netcdf
```

75 tests/testAAA

[Unit tests]

NAME

testAAA

SYNOPSIS

```
!$Id: testAAA.f90 380 2017-03-22 11:03:09Z mexas $

program testAAA
```

PURPOSE

Checking: getcodim (133.2), cgca_as (10.2), cgca_ds (10.5)

DESCRIPTION

Testing allocating and deallocating of a coarray.

NOTES

The program must be called with 2 command line arguments, both positive integers. These are codimensions along 1 and 2. The number of images must be such that codimension3 = num_images()/(codimension1 * codimension3) is a positive integer. Example:

```
      cafrun -np 16 ./testAAA.x 2 2      ! OpenCoarrays
```

or

```
      ./testAAA.x 2 2                    ! Intel, Cray
```

which will make the third codimension equal to 16/(2*2)=4.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

cgca testaux (133)

USED BY

Part of CGPACK test suite

SOURCE

```
use testaux

implicit none

integer( kind=idef ), parameter :: size1=10, size2=10, size3=10
```

```
integer( kind=idef ) :: nimgs
integer( kind=idef ) :: codim(3)[*]
integer( kind=iarr ), allocatable :: space1( : , : , : , : ) [:,:,:]

!*********************************************************************72
! first executable statement

nimgs = num_images()

! do a check on image 1
if (this_image() .eq. 1) then
 call getcodim( nimgs, codim )
 ! print a banner
 call banner("AAA")
 ! print the parameter values
 call cgca_pdmp
 write (*, '(a,i0,a)') "running on ", nimgs, " images in a 3D grid"
 write (*, *) "codim:", codim
end if

sync all ! wait for image 1 to set codim

codim(:) = codim(:)[1]

sync all ! wait for each image to read codim from img 1.

! implicit sync all inside
call cgca_as( 1, size1, 1, size2, 1, size3, 1, codim(1), 1, codim(2),  &
              1, 2, space1 )

if ( allocated( space1 ) )                                              &
 write (*, "(2(a,i0), 3(a,4(i0,tr1)), 3(a,3(i0,tr1)) )")               &
  "img: ", this_image(), ". my array, size: ", size( space1 ),         &
  ". shape: " , shape(space1), ". lbound: ", lbound(space1),           &
  ". ubound:", ubound(space1), ". coar index: ", this_image( space1 ), &
 ". lcobound:", lcobound(space1), ". ucobound:", ucobound(space1)

call cgca_ds(space1)

if ( .not. allocated(space1) )                                          &
  write (*,'(a,i0,a)')"Image:",this_image(), " space1 not allocated"

end program testAAA
```

76 tests/testAAB

[Unit tests]

NAME

testAAB

SYNOPSIS

```
!$Id: testAAB.f90 529 2018-03-26 11:25:45Z mexas $

program testAAB
```

PURPOSE

Checking: cgca_as (10.2), cgca_ds (10.5), cgca_swci (19.2)

DESCRIPTION

Writing all coarrays to a file in order, as if it were a single large 3D array (super array).

NOTES

The program must be called with 2 command line arguments, both positive integers. These are codimensions along 1 and 2. The number of images must be such that codimension3 = num_images()/(codimension1 * codimension3) is a positive integer. Example:

```
      cafrun -np 16 ./testAAB.x 2 2      ! OpenCoarrays
```

or

```
      ./testAAB.x 2 2                    ! Intel, Cray
```

which will make the third codimension equal to 16/(2*2)=4.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

cgca testaux (133)

USED BY

Part of CGPACK test suite

SOURCE

```
use testaux

implicit none

 integer(kind=idef),parameter :: size1=10, size2=10, size3=10
```

```
 integer(kind=idef) :: nimages
 integer(kind=idef) :: codim(3)[*]
 integer(kind=iarr),allocatable :: space1(:,:,:,:)[:,:,:]
logical(kind=ldef) :: image1

!*********************************************************************72
! first executable statement

nimages=num_images()
image1 = .false.
if (this_image() .eq. 1) image1 = .true.

! do a check on image 1
if (image1) then
 call getcodim(nimages,codim)
 ! print a banner
 call banner("AAB")
 ! print the parameter values
 call cgca_pdmp
 write (*,'(a,i0,a)') "running on ", nimages, " images in a 3D grid"
 write (*,*) "codim:", codim
end if

sync all

if (image1) then
  write (*,*) "cgca kinds:"
  write (*,*) "default integer:", idef
  write (*,*) "coarray integer:", iarr
  write (*,*) "default logical:", ldef
end if

codim(:) = codim(:)[1]

sync all

! implicit sync all inside
call cgca_as( 1, size1, 1, size2, 1, size3, 1, codim(1), 1, codim(2),  &
              1, 2, space1 )

if (allocated(space1)) then
  write (*,'(a,i0,a)')"Image:",this_image(), " space1 allocated"
  write (*,'(a,i0,a,3(i0,tr1),a)')                                      &
    "Image: ",this_image()," is ",this_image(space1)," in the grid"
end if

space1( :, :, :, cgca_state_type_grain ) = int(this_image(), kind=iarr)
space1( :, :, :, cgca_state_type_frac ) = 0_iarr

sync all

if ( this_image() .eq. 1 )                                              &
```

```
  write (*,*) "coarrays defined, calling cgca_swci"

call cgca_swci( space1, cgca_state_type_grain, 10, 'z.raw' )

call cgca_ds(space1)

if ( .not. allocated(space1) )                                          &
  write (*,'(a,i0,a)')"Image:",this_image(), " space1 not allocated"

end program testAAB
```

77 tests/testAAC

[Unit tests]

NAME

testAAC

SYNOPSIS

```
!$Id: testAAC.f90 536 2018-04-03 12:02:13Z mexas $

program testAAC
```

PURPOSE

Checking: cgca_hxi (15.2)

DESCRIPTION

Checking internal halo calculations.

NOTES

The program must be called with 2 command line arguments, both positive integers. These are codimensions along 1 and 2. The number of images must be such that codimension3 = num_images()/(codimension1 * codimension3) is a positive integer. Example:

```
      cafrun -np 16 ./testAAC.x 2 2      ! OpenCoarrays
```

or

```
      ./testAAC.x 2 2                    ! Intel, Cray
```

which will make the third codimension equal to 16/(2*2)=4.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

cgca testaux (133)

USED BY

Part of CGPACK test suite

SOURCE

```
use testaux

implicit none

 integer( kind=idef ),parameter :: size1=10, size2=10, size3=10
```

```
 integer( kind=idef ) :: nimages,nbhd,i,j,k
 integer( kind=idef ) :: codim(3)[*]
 real :: scaling
 integer(kind=iarr),allocatable :: space1(:,:,:,:)[:,:,:], &
   space2(:,:,:,:)[:,:,:]
logical(kind=ldef) :: image1

!*********************************************************************72
! first executable statement

nimages=num_images()
image1 = .false.
if (this_image() .eq. 1) image1 = .true.

! do a check on image 1
if (image1) then
 call getcodim(nimages,codim)
 ! print a banner
 call banner("AAC")
 ! print the parameter values
 call cgca_pdmp
 write (*,'(a,i0,a)') "running on ", nimages, " images in a 3D grid"
 write (*,*) "codim:", codim
end if

sync all

nbhd=26
scaling=1.0/real(nbhd+1)

codim(:) = codim(:)[1]

if (this_image() .eq. 2) call system("sleep 1")

call cgca_as(1,size1,1,size2,1,size3,1,codim(1),1,codim(2),1,2,space1)
call cgca_as(1,size1,1,size2,1,size3,1,codim(1),1,codim(2),1,2,space2)

if (allocated(space1)) then
  write (*,'(a,i0,a)')"Image:",this_image(), " space1 allocated"
  write (*,'(a,i0,a,3(i0,tr1),a)') &
    "Image: ",this_image()," is ",this_image(space1)," in the grid"
end if

if (allocated(space2)) then
  write (*,'(a,i0,a)')"Image:",this_image(), " space2 allocated"
end if

space1(:,:,:,cgca_state_type_grain) = int( this_image(), kind=iarr )
space1(:,:,:,cgca_state_type_frac) = cgca_intact_state
space2=0

sync all
```

```
call cgca_swci(space1,cgca_state_type_grain,10,'z1.raw')
sync all

call cgca_hxi(space1)
sync all

do k=1,size3
do j=1,size2
do i=1,size1
  space2( i, j, k, cgca_state_type_grain ) =                          &
  space1( i, j, k, cgca_state_type_grain ) -                          &
    nint( scaling * sum( space1( i-1:i+1 , j-1:j+1 , k-1:k+1 ,        &
                                 cgca_state_type_grain ) ), kind=iarr )
end do
end do
end do

sync all

if (this_image() .eq. 3) call system("sleep 2")

call cgca_swci(space2,cgca_state_type_grain,10,'z2.raw')

call cgca_ds(space1)
if (.not. allocated(space1)) &
  write (*,'(a,i0,a)')"Image:",this_image(), " space1 not allocated"

call cgca_ds(space2)
if (.not. allocated(space2)) &
  write (*,'(a,i0,a)')"Image:",this_image(), " space2 not allocated"

end program testAAC
```

78 tests/testAAD

[Unit tests]

NAME

testAAD

SYNOPSIS

```
!$Id: testAAD.f90 380 2017-03-22 11:03:09Z mexas $

program testAAD
```

PURPOSE

Checking: cgca_gl (13.1), cgca_lg (13.4).

DESCRIPTION

Checking the mapping routines.

NOTES

The program must be called with 2 command line arguments, both positive integers. These are codimensions along 1 and 2. The number of images must be such that codimension3 = num_images()/(codimension1 * codimension3) is a positive integer. Example:

```
      cafrun -np 16 ./testAAD.x 2 2      ! OpenCoarrays
```

or

```
      ./testAAD.x 2 2                    ! Intel, Cray
```

which will make the third codimension equal to 16/(2*2)=4.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

cgca testaux (133)

USED BY

Part of CGPACK test suite

SOURCE

```
use testaux

implicit none

 integer(kind=idef) :: l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3, &
```

```
   nimages,super_in(3),super_out(3),imgpos(3),local(3),test,  &
   codim(3)[*]
 integer(kind=iarr),allocatable :: space1(:,:,:,:)[:,:,:]

!*********************************************************************72
! first executable statement

nimages=num_images()

! do a check on image 1
if (this_image() .eq. 1) then
 call getcodim(nimages,codim)
 ! print a banner
 call banner("AAD")
 ! print the parameter values
 call cgca_pdmp
 write (*,'(a,i0,a)') "running on ", nimages, " images in a 3D grid"
 write (*,*) "codim:", codim
end if

sync all

codim(:) = codim(:)[1]

if (this_image() .eq. 2) call system("sleep 1")

!************************************************************
! test 1

test = 1

l1=1
u1=10
l2=1
u2=10
l3=1
u3=10
col1=1
cou1=2
col2=1
cou2=2
col3=1

if (this_image() .eq. 1) then
  write (*,*) "test 1"
  write (*,'(a,2(i0,":",i0,","),i0,":",i0,")")') &
    "bounds: (",l1,u1,l2,u2,l3,u3
  write (*,'(a,2(i0,":",i0,","),i0,":",i0,")")') &
    "cobounds: (",col1,cou1,col2,cou2,col3, &
     nimages/((cou1-col1+1)*(cou2-col2+1))+col3-1
end if
```

```
call cgca_as(l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3,1,space1)

super_in=(/20,10,1/)

if (this_image() .eq. 1) then
  call cgca_gl(super_in,space1,imgpos,local)
  write (*,'(a,3(i0,tr1))') "super in ", super_in
  write (*,'(a,3(i0,tr1))') "imgpos ", imgpos
  write (*,'(a,3(i0,tr1))') "local  ", local
  call cgca_lg(imgpos,local,space1,super_out)
  write (*,'(a,3(i0,tr1))') "super out ", super_out

  if (all(super_out .eq. super_in)) then
    write (*,'(a,i0,a)') "test ", test, " passed ok"
  else
    write (*,'(a,i0,a)') "****** ERROR: TEST ", test, " FAILED"
    error stop
  end if

end if
sync all

call cgca_ds(space1)

!************************************************************
! test 2

test = 2

l1=-7
u1=-3
l2=-10
u2=10
l3=-5
u3=0
col1=-2
cou1=-1
col2=-1
cou2=0
col3=8

if (this_image() .eq. 1) then
  write (*,*) "test  2"
  write (*,'(a,2(i0,":",i0,","),i0,":",i0,")")') &
    "bounds: (",l1,u1,l2,u2,l3,u3
  write (*,'(a,2(i0,":",i0,","),i0,":",i0,")")') &
    "cobounds: (",col1,cou1,col2,cou2,col3, &
     nimages/((cou1-col1+1)*(cou2-col2+1))+col3-1
end if

call cgca_as(l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3,2,space1)
```

```
super_in=(/5,4,3/)

if (this_image() .eq. 1) then
  call cgca_gl(super_in,space1,imgpos,local)
  write (*,'(a,3(i0,tr1))') "super in ", super_in
  write (*,'(a,3(i0,tr1))') "imgpos ", imgpos
  write (*,'(a,3(i0,tr1))') "local  ", local
  call cgca_lg(imgpos,local,space1,super_out)
  write (*,'(a,3(i0,tr1))') "super out ", super_out

  if (all(super_out .eq. super_in)) then
    write (*,'(a,i0,a)') "test ", test, " passed ok"
  else
    write (*,'(a,i0,a)') "****** ERROR: TEST ", test, " FAILED"
    error stop
  end if

end if
sync all

call cgca_ds(space1)

!************************************************************
! test 3

test = 3

l1=73
u1=100
l2=15
u2=20
l3=-99
u3=-90
col1=15
cou1=16
col2=0
cou2=1
col3=-10

if (this_image() .eq. 1) then
  write (*,*) "test 3"
  write (*,'(a,2(i0,":",i0,","),i0,":",i0,")")') &
    "bounds: (",l1,u1,l2,u2,l3,u3
  write (*,'(a,2(i0,":",i0,","),i0,":",i0,")")') &
    "cobounds: (",col1,cou1,col2,cou2,col3, &
     nimages/((cou1-col1+1)*(cou2-col2+1))+col3-1
end if

call cgca_as(l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3,3,space1)

super_in=(/5,4,3/)
```

```
if (this_image() .eq. 1) then
  call cgca_gl(super_in,space1,imgpos,local)
  write (*,'(a,3(i0,tr1))') "super in ", super_in
  write (*,'(a,3(i0,tr1))') "imgpos ", imgpos
  write (*,'(a,3(i0,tr1))') "local  ", local
  call cgca_lg(imgpos,local,space1,super_out)
  write (*,'(a,3(i0,tr1))') "super out ", super_out

  if (all(super_out .eq. super_in)) then
    write (*,'(a,i0,a)') "test ", test, " passed ok"
  else
    write (*,'(a,i0,a)') "****** ERROR: TEST ", test, " FAILED"
    error stop
  end if

end if
sync all

call cgca_ds(space1)

end program testAAD
```

79 tests/testAAE

[Unit tests]

NAME

testAAE

SYNOPSIS

```
!$Id: testAAE.f90 380 2017-03-22 11:03:09Z mexas $

program testAAE
```

PURPOSE

Checking: cgca_irs (23.2), cgca_nr (29.1)

DESCRIPTION

Checking nucleation and seed initialisation.

NOTES

The program must be called with 2 command line arguments, both positive integers. These are codimensions along 1 and 2. The number of images must be such that codimension3 = num_images()/(codimension1 * codimension3) is a positive integer. Example:

```
      cafrun -np 16 ./testAAE.x 2 2      ! OpenCoarrays
```

or

```
      ./testAAE.x 2 2                    ! Intel, Cray
```

which will make the third codimension equal to 16/(2*2)=4.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

cgca testaux (133)

USED BY

Part of CGPACK test suite

SOURCE

```
use testaux
implicit none

 integer(kind=idef) :: l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3, &
   nimages,myimage,codim(3)[*]
```

```
 logical(kind=ldef),parameter :: yesdebug=.true., nodebug=.false.
 integer(kind=iarr),allocatable :: space1(:,:,:,:)[:,:,:]

!*********************************************************************72
! first executable statement

nimages=num_images()
myimage=this_image()

! do a check on image 1
if (myimage .eq. 1) then
 call getcodim(nimages,codim)
 ! print a banner
 call banner("AAE")
 ! print the parameter values
 call cgca_pdmp
 write (*,'(a,i0,a)') "running on ", nimages, " images in a 3D grid"
 write (*,*) "codim:", codim
end if

sync all

codim(:) = codim(:)[1]

if (myimage .eq. 2) call system("sleep 1")

l1=1
u1=10
l2=l1
u2=u1
l3=l1
u3=u1
col1=1
cou1=codim(1)
col2=1
cou2=codim(2)
col3=1

if (myimage .eq. 1) then
  write (*,'(a,2(i0,":",i0,","),i0,":",i0,")")') &
    "bounds: (",l1,u1,l2,u2,l3,u3
  write (*,'(a,2(i0,":",i0,","),i0,":",i0,")")') &
    "cobounds: (",col1,cou1,col2,cou2,col3, &
     nimages/((cou1-col1+1)*(cou2-col2+1))+col3-1
end if

! allocate coarray
call cgca_as(l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3,1,space1)

! initialise random number seed
call cgca_irs(yesdebug)
```

```
! initialise coarray to liquid state
space1 = cgca_liquid_state
sync all

call cgca_swci(space1,cgca_state_type_grain,10,"z1.raw")
sync all

! nucleate, sync in the routine, no need to sync in the program
call cgca_nr(space1,10,yesdebug)

call cgca_swci(space1,cgca_state_type_grain,10,"z2.raw")
sync all

call cgca_ds(space1)

end program testAAE
```

80 tests/testAAF

[Unit tests]

NAME

testAAF

SYNOPSIS

```
!$Id: testAAF.f90 380 2017-03-22 11:03:09Z mexas $

program testAAF
```

PURPOSE

Checking: cgca_sld (31.1)

DESCRIPTION

Checking solidification with a single nuclei.

NOTES

The program must be called with 2 command line arguments, both positive integers. These are codimensions along 1 and 2. The number of images must be such that codimension3 = num_images()/(codimension1 * codimension3) is a positive integer. Example:

```
      cafrun -np 16 ./testAAF.x 2 2      ! OpenCoarrays
```

or

```
      ./testAAF.x 2 2                    ! Intel, Cray
```

which will make the third codimension equal to 16/(2*2)=4.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

cgca testaux (133)

USED BY

Part of CGPACK test suite

SOURCE

```
use testaux

implicit none

 logical(kind=ldef),parameter :: yesdebug=.true., nodebug=.false., &
```

```
   noperiodicbc=.false.
 integer(kind=idef) :: l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3, &
   nimages,myimage,codim(3)[*]
 integer(kind=iarr),allocatable :: space1(:,:,:,:)[:,:,:]
 logical(kind=ldef) :: solid

!*********************************************************************72
! first executable statement

nimages=num_images()
myimage=this_image()

! do a check on image 1
if (myimage .eq. 1) then
 call getcodim(nimages,codim)
 ! print a banner
 call banner("AAF")
 ! print the parameter values
 call cgca_pdmp
 write (*,'(a,i0,a)') "running on ", nimages, " images in a 3D grid"
 write (*,*) "codim:", codim
end if

! Trying to separate the banner output from the rest
sync all

codim(:) = codim(:)[1]

if (myimage .eq. 2) call system("sleep 1")

l1=1
u1=5
l2=l1
u2=u1
l3=l1
u3=u1
col1=1
cou1=codim(1)
col2=1
cou2=codim(2)
col3=1

if (myimage .eq. 1) then
  write (*,'(a,2(i0,":",i0,","),i0,":",i0,")")') &
    "bounds: (",l1,u1,l2,u2,l3,u3
  write (*,'(a,2(i0,":",i0,","),i0,":",i0,")")') &
    "cobounds: (",col1,cou1,col2,cou2,col3, &
     nimages/((cou1-col1+1)*(cou2-col2+1))+col3-1
end if

! allocate coarray
call cgca_as(l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3,1,space1)
```

```
! initialise random number seed
call cgca_irs(nodebug)

! initialise coarray to liquid phase
space1(:,:,:,cgca_state_type_grain) = cgca_liquid_state

! nucleate, sync in the routine, no need to sync in the program
call cgca_nr(space1, 1, yesdebug)
call cgca_swci(space1,cgca_state_type_grain,10,"z1.raw")

! solidify 1
call cgca_sld(space1,noperiodicbc,10,1,solid)
call cgca_swci(space1,cgca_state_type_grain,10,"z2.raw")

! if solid, issue a message from image 1 and stop
if (myimage .eq. 1 .and. solid) write (*,*) "all solid, stop"
if (solid) stop

! solidify 2
call cgca_sld(space1,noperiodicbc,10,1,solid)
call cgca_swci(space1,cgca_state_type_grain,10,"z3.raw")

! if solid, issue a message from image 1 and stop
if (myimage .eq. 1 .and. solid) write (*,*) "all solid, stop"
if (solid) stop

! solidify 3
call cgca_sld(space1,noperiodicbc,0,1,solid)
call cgca_swci(space1,cgca_state_type_grain,10,"z9end.raw")

call cgca_ds(space1)

end program testAAF
```

81 tests/testAAG

[Unit tests]

NAME

testAAG

SYNOPSIS

```
!$Id: testAAG.f90 380 2017-03-22 11:03:09Z mexas $

program testAAG
```

PURPOSE

Checking: cgca_sld (31.1)

DESCRIPTION

Checking solidification with two nuclei.

NOTES

The program must be called with 2 command line arguments, both positive integers. These are codimensions along 1 and 2. The number of images must be such that codimension3 = num_images()/(codimension1 * codimension3) is a positive integer. Example:

```
      cafrun -np 16 ./testAAG.x 2 2      ! OpenCoarrays
```

or

```
      ./testAAG.x 2 2                    ! Intel, Cray
```

which will make the third codimension equal to 16/(2*2)=4.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

cgca testaux (133)

USED BY

Part of CGPACK test suite

SOURCE

```
use testaux
implicit none

 logical(kind=ldef),parameter :: yesdebug=.true., nodebug=.false., &
   noperiodicbc=.false.
```

```
 integer(kind=idef) :: l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3, &
   nimages,myimage,codim(3)[*]
 integer(kind=iarr),allocatable :: space1(:,:,:,:)[:,:,:]
 logical(kind=ldef) :: solid

!*********************************************************************72
! first executable statement

nimages=num_images()
myimage=this_image()

! do a check on image 1
if (myimage .eq. 1) then
 call getcodim(nimages,codim)
 ! print a banner
 call banner("AAG")
 ! print the parameter values
 call cgca_pdmp
 write (*,'(a,i0,a)') "running on ", nimages, " images in a 3D grid"
 write (*,*) "codim:", codim
end if

sync all

codim(:) = codim(:)[1]

if (myimage .eq. 2) call system("sleep 1")

l1=1
u1=10
l2=l1
u2=u1
l3=l1
u3=u1
col1=1
cou1=codim(1)
col2=1
cou2=codim(2)
col3=1

if (myimage .eq. 1) then
  write (*,'(a,2(i0,":",i0,","),i0,":",i0,")")') &
    "bounds: (",l1,u1,l2,u2,l3,u3
  write (*,'(a,2(i0,":",i0,","),i0,":",i0,")")') &
    "cobounds: (",col1,cou1,col2,cou2,col3, &
     nimages/((cou1-col1+1)*(cou2-col2+1))+col3-1
end if

! allocate coarray
call cgca_as(l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3,1,space1)

! initialise random number seed
```

```
call cgca_irs(nodebug)

! initialise coarray to liquid
space1(:,:,:,cgca_state_type_grain) = cgca_liquid_state

! these are the nuclei
space1(l1,u2,l3,cgca_state_type_grain)[cou1,1,1] = 1
space1(u1,l2,u3,cgca_state_type_grain)[cou1,cou2,1] = 2
sync all

call cgca_swci(space1,cgca_state_type_grain,10,"z1.raw")
sync all

! solidify 1
call cgca_sld(space1,noperiodicbc,10,1,solid)

call cgca_swci(space1,cgca_state_type_grain,10,"z2.raw")
sync all

! if solid, issue a message from image 1 and stop
if (myimage .eq. 1 .and. solid) write (*,*) "all solid, stop"
if (solid) stop

sync all

! solidify 2
call cgca_sld(space1,noperiodicbc,10,1,solid)

sync all

call cgca_swci(space1,cgca_state_type_grain,10,"z3.raw")
sync all

! if solid, issue a message from image 1 and stop
if (myimage .eq. 1 .and. solid) write (*,*) "all solid, stop"
if (solid) stop

sync all

! solidify 3
call cgca_sld(space1,noperiodicbc,0,1,solid)

sync all

call cgca_swci(space1,cgca_state_type_grain,10,"z9end.raw")
sync all

call cgca_ds(space1)

end program testAAG
```

82 tests/testAAH

[Unit tests]

NAME

testAAH

SYNOPSIS

```
!$Id: testAAH.f90 380 2017-03-22 11:03:09Z mexas $

program testAAH
```

PURPOSE

Checking: cgca_sld (31.1), cgca_nr (29.1)

DESCRIPTION

Checking solidification with 640 nuclei, coarray (100,100,100)[4,4,4]. This gives the desired resolution of 1e-5. Therefore run with 64 images... or try other values...

NOTES

The program must be called with 2 command line arguments, both positive integers. These are codimensions along 1 and 2. The number of images must be such that codimension3 = num_images()/(codimension1 * codimension3) is a positive integer. Example:

```
      cafrun -np 16 ./testAAH.x 2 2      ! OpenCoarrays
```

or

```
      ./testAAH.x 2 2                    ! Intel, Cray
```

which will make the third codimension equal to 16/(2*2)=4.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

cgca testaux (133)

USED BY

Part of CGPACK test suite

SOURCE

```
use testaux

implicit none
```

```
 logical(kind=ldef),parameter :: yesdebug=.true., nodebug=.false., &
   noperiodicbc=.false.
 integer(kind=idef) :: l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3, &
   nimages,myimage,codim(3)[*]
 integer(kind=iarr),allocatable :: space1(:,:,:,:)[:,:,:]
 logical(kind=ldef) :: solid

!*********************************************************************72
! first executable statement

nimages=num_images()
myimage=this_image()

! do a check on image 1
if (myimage .eq. 1) then
 call getcodim(nimages,codim)
 ! print a banner
 call banner("AAH")
 ! print the parameter values
 call cgca_pdmp
 write (*,'(a,i0,a)') "running on ", nimages, " images in a 3D grid"
 write (*,*) "codim:", codim
end if

sync all

codim(:) = codim(:)[1]

if (myimage .eq. 2) call system("sleep 1")

l1=1
u1=100
l2=l1
u2=u1
l3=l1
u3=u1
col1=1
cou1=codim(1)
col2=1
cou2=codim(2)
col3=1

if (myimage .eq. 1) then
  write (*,'(a,2(i0,":",i0,","),i0,":",i0,")")') &
    "bounds: (",l1,u1,l2,u2,l3,u3
  write (*,'(a,2(i0,":",i0,","),i0,":",i0,")")') &
    "cobounds: (",col1,cou1,col2,cou2,col3, &
     nimages/((cou1-col1+1)*(cou2-col2+1))+col3-1
end if

! allocate coarray
call cgca_as(l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3,1,space1)
```

```
! initialise coarray to liquid phase
space1(:,:,:,cgca_state_type_grain) = cgca_liquid_state

! initialise random number seed
call cgca_irs(nodebug)

sync all

! nuclei, sync all inside
call cgca_nr(space1,640,yesdebug)

call cgca_swci(space1,cgca_state_type_grain,10,"z0.raw")
sync all

! solidify 1
call cgca_sld(space1,noperiodicbc,100,10,solid)

call cgca_swci(space1,cgca_state_type_grain,10,"z100.raw")
sync all

! if solid, issue a message from image 1 and stop
if (myimage .eq. 1 .and. solid) write (*,*) "all solid, stop"
if (solid) stop

! solidify 2
call cgca_sld(space1,noperiodicbc,0,10,solid)

call cgca_swci(space1,cgca_state_type_grain,10,"z9end.raw")

call cgca_ds(space1)

end program testAAH
```

83 tests/testAAI

[Unit tests]

NAME

testAAI

SYNOPSIS

```
!$Id: testAAI.f90 380 2017-03-22 11:03:09Z mexas $

program testAAI
```

PURPOSE

Checking: cgca_sld (31.1), cgca_nr (29.1), cgca_irs (23.2), cgca_hxg (15.1)

DESCRIPTION

Checking solidification with periodic BC and a single nucleus at (l1,l2,l3)[col1,col2,col3].

NOTES

The program must be called with 2 command line arguments, both positive integers. These are codimensions along 1 and 2. The number of images must be such that codimension3 = num_images()/(codimension1 * codimension3) is a positive integer. Example:

```
      cafrun -np 16 ./testAAI.x 2 2      ! OpenCoarrays
```

or

```
      ./testAAI.x 2 2                    ! Intel, Cray
```

which will make the third codimension equal to 16/(2*2)=4.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

cgca testaux (133)

USED BY

Part of CGPACK test suite

SOURCE

```
use testaux
implicit none

 logical(kind=ldef),parameter :: yesdebug=.true., nodebug=.false., &
   periodicbc=.true.
```

```
 integer(kind=idef) :: l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3, &
   nimages,myimage,codim(3)[*]
 integer(kind=iarr),allocatable :: space1(:,:,:,:)[:,:,:]
 logical(kind=ldef) :: solid

!*****************************************************************72
! first executable statement

nimages=num_images()
myimage=this_image()

! do a check on image 1
if (myimage .eq. 1) then
 call getcodim(nimages,codim)
 ! print a banner
 call banner("AAI")
 ! print the parameter values
 call cgca_pdmp
 write (*,'(a,i0,a)') "running on ", nimages, " images in a 3D grid"
 write (*,*) "codim:", codim
end if

sync all

codim(:) = codim(:)[1]

if (myimage .eq. 2) call system("sleep 1")

l1=1
u1=10
l2=l1
u2=u1
l3=l1
u3=u1
col1=1
cou1=codim(1)
col2=1
cou2=codim(2)
col3=1

if (myimage .eq. 1) then
  write (*,'(a,2(i0,":",i0,","),i0,":",i0,")")') &
    "bounds: (",l1,u1,l2,u2,l3,u3
  write (*,'(a,2(i0,":",i0,","),i0,":",i0,")")') &
    "cobounds: (",col1,cou1,col2,cou2,col3, &
     nimages/((cou1-col1+1)*(cou2-col2+1))+col3-1
end if

! allocate coarray
call cgca_as(l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3,1,space1)

! initialise coarray to liquid
```

```
space1(:,:,:,cgca_state_type_grain) = cgca_liquid_state

! initialise random number seed
call cgca_irs(nodebug)

sync all

! single nuclei, where I want it
space1(l1,l2,l3,cgca_state_type_grain)[col1,col2,col3] = 1

call cgca_swci(space1,cgca_state_type_grain,10,"z0.raw")
sync all

! solidify 1
call cgca_sld(space1,periodicbc,10,1,solid)

call cgca_swci(space1,cgca_state_type_grain,10,"z10.raw")
sync all

! if solid, issue a message from image 1 and stop
if (myimage .eq. 1 .and. solid) write (*,*) "all solid, stop"
if (solid) stop

! solidify 2
call cgca_sld(space1,periodicbc,10,1,solid)

call cgca_swci(space1,cgca_state_type_grain,10,"z20.raw")
sync all

! if solid, issue a message from image 1 and stop
if (myimage .eq. 1 .and. solid) write (*,*) "all solid, stop"
if (solid) stop

! solidify 3
call cgca_sld(space1,periodicbc,10,1,solid)

call cgca_swci(space1,cgca_state_type_grain,10,"z30.raw")
sync all

! if solid, issue a message from image 1 and stop
if (myimage .eq. 1 .and. solid) write (*,*) "all solid, stop"
if (solid) stop

! solidify last
call cgca_sld(space1,periodicbc,0,10,solid)

call cgca_swci(space1,cgca_state_type_grain,10,"z9end.raw")

call cgca_ds(space1)

end program testAAI
```

84 tests/testAAJ

[Unit tests]

NAME

testAAJ

SYNOPSIS

```
!$Id: testAAJ.f90 380 2017-03-22 11:03:09Z mexas $

program testAAJ
```

PURPOSE

Checking: cgca_sld (31.1), cgca_nr (29.1), cgca_irs (23.2), cgca_hxg (15.1)

DESCRIPTION

Checking solidification with periodic BC and a single nucleus, where I want it:

```
space1(l1+(u1-l1)/2, l2, l3+(u3-l3)/2, cgca_state_type_grain) &
[col1+(cou1-col1)/2, col2, col3+(cou3-col3)/2] = 1
```

NOTES

The program must be called with 2 command line arguments, both positive integers. These are codimensions along 1 and 2. The number of images must be such that codimension3 = num_images()/(codimension1 * codimension3) is a positive integer. Example:

```
cafrun -np 16 ./testAAJ.x 2 2      ! OpenCoarrays
```

or

```
./testAAJ.x 2 2                    ! Intel, Cray
```

which will make the third codimension equal to 16/(2*2)=4.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

cgca testaux (133)

USED BY

Part of CGPACK test suite

SOURCE

```
use testaux
```

```
implicit none

 logical(kind=ldef),parameter :: yesdebug=.true., nodebug=.false., &
   periodicbc=.true.
 integer(kind=idef) :: l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3, &
   cou3,ncells,nimages,myimage,codim(3)[*]
 integer(kind=iarr),allocatable :: space1(:,:,:,:)[:,:,:]
 logical(kind=ldef) :: solid
 real,parameter :: gigabyte=real(2**30)
 real :: image_storage

!*********************************************************************72
! first executable statement

nimages=num_images()
myimage=this_image()

! do a check on image 1
if (myimage .eq. 1) then
 call getcodim(nimages,codim)
 ! print a banner
 call banner("AAJ")
 ! print the parameter values
 call cgca_pdmp
 write (*,'(a,i0,a)') "running on ", nimages, " images in a 3D grid"
 write (*,*) "codim:", codim
end if

sync all

codim(:) = codim(:)[1]

if (myimage .eq. 2) call system("sleep 1")

l1=1
u1=10
l2=l1
u2=u1
l3=l1
u3=u1
col1=1
cou1=codim(1)-col1+1
col2=1
cou2=codim(2)-col2+1
col3=1
cou3=codim(3)-col3+1

if (myimage .eq. 1) then
  write (*,'(a,2(i0,":",i0,","),i0,":",i0,")")') &
    "bounds: (",l1,u1,l2,u2,l3,u3
  write (*,'(a,2(i0,":",i0,","),i0,":",i0,")")') &
```

```
    "cobounds: (",col1,cou1,col2,cou2,col3,cou3

  ! total number of cells in a coarray
  ncells=(u1-l1+1)*(u2-l2+1)*(u3-l3+1)

  ! An absolute minimum of storage, in GB, per image
  image_storage = real(ncells*storage_size(space1)/8)/gigabyte

  write (*,'(a,i0,a)') "Each image has ",ncells, " cells"
  write (*,'(a,es9.2,a)') "Each image will use at least ", &
    image_storage, " GB memory"
end if

! allocate coarray
call cgca_as(l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3,1,space1)

! initialise coarray to liquid
space1(:,:,:,cgca_state_type_grain) = cgca_liquid_state

! initialise random number seed
call cgca_irs(nodebug)

sync all

! single nucleus, where I want it
space1(l1+(u1-l1)/2, l2, l3+(u3-l3)/2, cgca_state_type_grain) &
  [col1+(cou1-col1)/2, col2, col3+(cou3-col3)/2] = 1

call cgca_swci(space1,cgca_state_type_grain,10,"z0.raw")
sync all

! solidify 1
call cgca_sld(space1,periodicbc,10,1,solid)

call cgca_swci(space1,cgca_state_type_grain,10,"z10.raw")
sync all

! if solid, issue a message from image 1 and stop
if (myimage .eq. 1 .and. solid) write (*,*) "all solid, stop"
if (solid) stop

! solidify 2
call cgca_sld(space1,periodicbc,10,1,solid)

call cgca_swci(space1,cgca_state_type_grain,10,"z20.raw")
sync all

! if solid, issue a message from image 1 and stop
if (myimage .eq. 1 .and. solid) write (*,*) "all solid, stop"
if (solid) stop

! solidify 3
```

```
call cgca_sld(space1,periodicbc,10,1,solid)

call cgca_swci(space1,cgca_state_type_grain,10,"z30.raw")
sync all

! if solid, issue a message from image 1 and stop
if (myimage .eq. 1 .and. solid) write (*,*) "all solid, stop"
if (solid) stop

! solidify last
call cgca_sld(space1,periodicbc,0,10,solid)

call cgca_swci(space1,cgca_state_type_grain,10,"z9end.raw")

call cgca_ds(space1)

end program testAAJ
```

85 tests/testAAK

[Unit tests]

NAME

testAAK

SYNOPSIS

```
!$Id: testAAK.f90 529 2018-03-26 11:25:45Z mexas $

program testAAK
```

PURPOSE

Checking: cgca_sld (31.1), cgca_nr (29.1), cgca_irs (23.2), cgca_hxg (15.1)

DESCRIPTION

This program is designed to assess the max coarray dimension, assuming cubic coarray, and cubic coarray grid. Run it until it fails to fit in memory. The array is not written to disk.

Note that on HPC systems it makes sense to maximise the number of processors, and run as quickly as possible, rather then maximise the memory used by each processor. So the purpose of this test is limited to computers with limited number of processors, where achieving a big model requires using all memory.

On the other hand, CGPACK, at present, does not scale well, so using lots of memory per node and fewer nodes is best for performance for now.

NOTES

The program must be called with 2 command line arguments, both positive integers. These are codimensions along 1 and 2. The number of images must be such that codimension3 = num_images()/(codimension1 * codimension3) is a positive integer. Example:

```
      cafrun -np 16 ./testAAK.x 2 2      ! OpenCoarrays
```

or

```
      ./testAAK.x 2 2                    ! Intel, Cray
```

which will make the third codimension equal to 16/(2*2)=4.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

cgca testaux (133)

USED BY

Part of CGPACK test suite

SOURCE

```
use testaux

implicit none

 logical(kind=ldef),parameter :: yesdebug=.true., nodebug=.false., &
   periodicbc=.true.
 real,parameter :: gigabyte=real(2**30)

 integer(kind=idef) :: l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3, &
   cou3,nimages,myimage,codim(3)[*]
 integer(kind=iarr),allocatable :: space1(:,:,:,:)[:,:,:]
 integer(kind=ilrg) :: icells,mcells

 logical(kind=ldef) :: solid
 real :: image_storage

!*********************************************************************72
! first executable statement

nimages=num_images()
myimage=this_image()

! do a check on image 1
if (myimage .eq. 1) then
 call getcodim(nimages,codim)
 ! print a banner
 call banner("AAK")
 ! print the parameter values
 call cgca_pdmp
 write (*,'(a,i0,a)') "running on ", nimages, " images in a 3D grid"
 write (*,*) "codim:", codim
end if

sync all

codim(:) = codim(:)[1]

if (myimage .eq. 2) call system("sleep 1")

l1=1
l2=l1
l3=l1

u1=10
u2=u1
u3=u1

col1=1
cou1=codim(1)-col1+1
col2=1
cou2=codim(2)-col2+1
col3=1
```

```
cou3=codim(3)-col3+1

! initialise random number seed
call cgca_irs(nodebug)

main: do

if (myimage .eq. 1) then
  write (*,'(a,2(i0,":",i0,","),i0,":",i0,")")') &
    "bounds: (",l1,u1,l2,u2,l3,u3
  write (*,'(a,2(i0,":",i0,","),i0,":",i0,")")') &
    "cobounds: (",col1,cou1,col2,cou2,col3,cou3

  ! total number of cells in a coarray
  icells = int(u1-l1+1,kind=ilrg) * int(u2-l2+1,kind=ilrg) * &
    int(u3-l3+1,kind=ilrg)

  ! total number of cells in the model
  mcells = icells * int(codim(1),kind=ilrg) * int(codim(2),kind=ilrg) * &
    int(codim(3),kind=ilrg)

  ! An absolute minimum of storage, in GB, per image.
  ! A factor of 2 is used because will call _sld, which
  ! allocates another array of the same size and kind as
  ! coarray.
  image_storage = real(2*icells) * real(storage_size(space1)/8)/gigabyte

  write (*,'(a,i0,a)') "Each image has ",icells, " cells"
  write (*,'(a,i0,a)') "The model has ", mcells, " cells"
  write (*,'(a,es9.2,a)') "Each image will use at least ", &
    image_storage, " GB memory"
end if

! allocate coarray
call cgca_as(l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3,1,space1)

! initialise coarray to all solid
space1 = int( this_image(), kind=iarr )

sync all

! make only one liquid cell
space1(l1,l2,l3,cgca_state_type_grain)[col1,col2,col3] = &
 cgca_liquid_state

!call cgca_swci(space1,10,"z0.raw")
sync all

! solidify 1
call cgca_sld(space1,periodicbc,0,0,solid)

!call cgca_swci(space1,10,"z9end.raw")
```

```
call cgca_ds(space1)

! if all is fine, continue
if (myimage .eq. 1) write (*,*) "ok"

! increase the size of coarray
u1 = u1+1
u2 = u1
u3 = u1

end do main

end program testAAK
```

86 tests/testAAL

[Unit tests]

NAME

testAAL

SYNOPSIS

```
!$Id: testAAL.f90 380 2017-03-22 11:03:09Z mexas $

program testAAL
```

PURPOSE

Checking: cgca_sld (31.1), cgca_nr (29.1), cgca_irs (23.2), cgca_hxg (15.1)

DESCRIPTION

Still solidification, but this program is designed for timing analysis.

NOTES

The program must be called with 2 command line arguments, both positive integers. These are codimensions along 1 and 2. The number of images must be such that codimension3 = num_images()/(codimension1 * codimension3) is a positive integer. Example:

```
      cafrun -np 16 ./testAAL.x 2 2      ! OpenCoarrays
```

or

```
      ./testAAL.x 2 2                    ! Intel, Cray
```

which will make the third codimension equal to 16/(2*2)=4.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

cgca testaux (133)

USED BY

Part of CGPACK test suite

SOURCE

```
use testaux

implicit none

 logical(kind=ldef),parameter :: yesdebug=.true., nodebug=.false., &
```

```
   periodicbc=.true.
real, parameter :: gigabyte=real(2**30), resolution=1.0e-5

 integer(kind=idef) :: l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3, &
   cou3, &
   nuc,    & ! number of nuclei in the model
   nimages,myimage,codim(3)[*]
 integer(kind=iarr),allocatable :: space1(:,:,:,:)[:,:,:]
 integer(kind=ilrg) :: icells,mcells

 logical(kind=ldef) :: solid

 real :: image_storage

!*********************************************************************72
! first executable statement

nimages=num_images()
myimage=this_image()

! do a check on image 1
if (myimage .eq. 1) then
 call getcodim(nimages,codim)
 ! print a banner
 call banner("AAL")
 ! print the parameter values
 call cgca_pdmp
 write (*,'(a,i0,a)') "running on ", nimages, " images in a 3D grid"
 write (*,*) "codim:", codim
end if

sync all

codim(:) = codim(:)[1]

if (myimage .eq. 2) call system("sleep 1")

l1=1
l2=l1
l3=l1

! The array size is only controlled by this value
! in this program.
u1=50
u2=u1
u3=u1

col1=1
cou1=codim(1)-col1+1
col2=1
cou2=codim(2)-col2+1
col3=1
```

```
cou3=codim(3)-col3+1

! total number of cells in an image
icells = int(u1-l1+1,kind=ilrg) * int(u2-l2+1,kind=ilrg) * &
  int(u3-l3+1,kind=ilrg)

! total number of cells in the model
mcells = icells * int(codim(1),kind=ilrg) * int(codim(2),kind=ilrg) * &
  int(codim(3),kind=ilrg)

! total number of nuclei
nuc = int( resolution * real( mcells ) )

if (myimage .eq. 1) then
  write (*,'(a,2(i0,":",i0,","),i0,":",i0,")")') &
    "bounds: (",l1,u1,l2,u2,l3,u3
  write (*,'(a,2(i0,":",i0,","),i0,":",i0,")")') &
    "cobounds: (",col1,cou1,col2,cou2,col3,cou3

  ! An absolute minimum of storage, in GB, per image.
  ! A factor of 2 is used because will call _sld, which
  ! allocates another array of the same size and kind as
  ! coarray.
  image_storage = real(2 * icells*storage_size(space1)/8)/gigabyte

  write (*,'(a,i0,a)') "Each image has ",icells, " cells"
  write (*,'(a,i0,a)') "The model has ", mcells, " cells"
  write (*,'(a,i0,a)') "The model has ", nuc, " nuclei"
  write (*,'(a,es9.2,a)') "Each image will use at least ", &
    image_storage, " GB memory"
end if

! initialise random number seed
call cgca_irs(nodebug)

! allocate coarray
call cgca_as(l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3,1,space1)

! initialise coarray to liquid
space1(:,:,:,cgca_state_type_grain) = cgca_liquid_state

sync all

! populate nuclei
call cgca_nr(space1,nuc,nodebug)

!call cgca_swci(space1,10,"z0.raw")
!sync all

! solidify 1
call cgca_sld(space1,periodicbc,0,0,solid)
```

```
call cgca_swci(space1,cgca_state_type_grain,10,"z9end.raw")

call cgca_ds(space1)

end program testAAL
```

87 tests/testAAM

[Unit tests]

NAME

testAAM

SYNOPSIS

```
!$Id: testAAM.f90 380 2017-03-22 11:03:09Z mexas $

program testAAM
```

PURPOSE

Checking: cgca_av (10.3), cgca_dv (10.6)

DESCRIPTION

Checking grain volume array alloc/deallocation.

NOTES

The program must be called with 2 command line arguments, both positive integers. These are codimensions along 1 and 2. The number of images must be such that codimension3 = num_images()/(codimension1 * codimension3) is a positive integer. Example:

```
      cafrun -np 16 ./testAAM.x 2 2      ! OpenCoarrays
```

or

```
      ./testAAM.x 2 2                    ! Intel, Cray
```

which will make the third codimension equal to 16/(2*2)=4.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

cgca testaux (133)

USED BY

Part of CGPACK test suite

SOURCE

```
use testaux

implicit none

 logical(kind=ldef),parameter :: yesdebug=.true., nodebug=.false., &
```

```
   periodicbc=.true.
real, parameter :: gigabyte=real(2**30), resolution=1.0e-5

 integer(kind=idef) :: l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3, &
   cou3, &
   nuc,    & ! number of nuclei in the model
   nimages,codim(3)[*]
 integer(kind=iarr),allocatable :: space1(:,:,:,:)[:,:,:]
 integer(kind=ilrg),allocatable :: grainvol(:)[:,:,:]
 integer(kind=ilrg) :: icells,mcells

 logical(kind=ldef) :: image1

 real :: image_storage

!*****************************************************************72
! first executable statement

nimages=num_images()
image1=.false.
if (this_image().eq.1) image1=.true.

! do a check on image 1
if (image1) then
 call getcodim(nimages,codim)
 ! print a banner
 call banner("AAM")
 ! print the parameter values
 call cgca_pdmp
 write (*,'(a,i0,a)') "running on ", nimages, " images in a 3D grid"
 write (*,*) "codim:", codim
end if

sync all

codim(:) = codim(:)[1]

if (image1) call system("sleep 1")

l1=1
l2=l1
l3=l1

! The array size is only controlled by this value
! in this program.
u1=50
u2=u1
u3=u1

col1=1
cou1=codim(1)-col1+1
col2=1
```

```
cou2=codim(2)-col2+1
col3=1
cou3=codim(3)-col3+1

! total number of cells in a coarray
icells = int(u1-l1+1,kind=ilrg) * int(u2-l2+1,kind=ilrg) * &
  int(u3-l3+1,kind=ilrg)

! total number of cells in the model
mcells = icells * int(codim(1),kind=ilrg) * int(codim(2),kind=ilrg) * &
  int(codim(3),kind=ilrg)

! total number of nuclei
nuc = int( resolution * real( mcells ) )

if (image1) then
  write (*,'(a,2(i0,":",i0,","),i0,":",i0,")")') &
    "bounds: (",l1,u1,l2,u2,l3,u3
  write (*,'(a,2(i0,":",i0,","),i0,":",i0,")")') &
    "cobounds: (",col1,cou1,col2,cou2,col3,cou3

  ! An absolute minimum of storage, in GB, per image.
  ! A factor of 2 is used because will call _sld, which
  ! allocates another array of the same size and kind as
  ! coarray.
  image_storage = real(2 * icells*storage_size(space1)/8)/gigabyte

  write (*,'(a,i0,a)') "Each image has ",icells, " cells"
  write (*,'(a,i0,a)') "The model has ", mcells, " cells"
  write (*,'(a,i0,a)') "The model has ", nuc, " nuclei"
  write (*,'(a,es9.2,a)') "Each image will use at least ", &
    image_storage, " GB memory"
end if

! initialise random number seed
call cgca_irs(nodebug)

! allocate coarray
call cgca_as(l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3,1,space1)

! initialise coarray to liquid
space1(:,:,:,cgca_state_type_grain) = cgca_liquid_state

! allocate grain volume
call cgca_av(0,nuc,col1,cou1,col2,cou2,col3,grainvol)

if (image1) write (*,'(a)') "grain volume coarray allocated"

! deallocate all arrays
call cgca_ds(space1)
call cgca_dv(grainvol)
if (image1) write (*,'(a)') "grain volume coarray deallocated"
```

```
end program testAAM
```

88 tests/testAAN

[Unit tests]

NAME

testAAN

SYNOPSIS

```
!$Id: testAAN.f90 529 2018-03-26 11:25:45Z mexas $

program testAAN
```

PURPOSE

Checking: cgca_av (10.3), cgca_dv (10.6), cgca_gv (25.2)

DESCRIPTION

Checking grain volume array calculation.

NOTES

The program must be called with 2 command line arguments, both positive integers. These are codimensions along 1 and 2. The number of images must be such that codimension3 = num_images()/(codimension1 * codimension3) is a positive integer. Example:

```
      cafrun -np 16 ./testAAN.x 2 2      ! OpenCoarrays
```

or

```
      ./testAAN.x 2 2                    ! Intel, Cray
```

which will make the third codimension equal to 16/(2*2)=4.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

cgca testaux (133)

USED BY

Part of CGPACK test suite

SOURCE

```
use testaux

implicit none

 logical(kind=ldef),parameter :: yesdebug=.true., nodebug=.false., &
```

```
   periodicbc=.true.
real, parameter :: gigabyte=real(2**30), resolution=1.0e-5

 integer(kind=idef) :: l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3, &
   cou3, &
   nuc,    & ! number of nuclei in the model
   nimages,codim(3)[*],i
 integer(kind=iarr),allocatable :: space(:,:,:,:)[:,:,:]
 integer(kind=ilrg),allocatable :: grainvol(:)[:,:,:]
 integer(kind=ilrg) :: icells,mcells

 logical(kind=ldef) :: solid,image1

 real :: image_storage

!*****************************************************************72
! first executable statement

nimages=num_images()
image1=.false.
if (this_image().eq.1) image1=.true.

! do a check on image 1
if (image1) then
 call getcodim(nimages,codim)
 ! print a banner
 call banner("AAN")
 ! print the parameter values
 call cgca_pdmp
 write (*,'(a,i0,a)') "running on ", nimages, " images in a 3D grid"
 write (*,*) "codim:", codim
end if

sync all

codim(:) = codim(:)[1]

if (image1) call system("sleep 1")

l1=1
l2=l1
l3=l1

! The array size is only controlled by this value
! in this program.
u1=50
u2=u1
u3=u1

col1=1
cou1=codim(1)-col1+1
col2=1
```

```
cou2=codim(2)-col2+1
col3=1
cou3=codim(3)-col3+1

! total number of cells in a coarray
icells = int(u1-l1+1,kind=ilrg) * int(u2-l2+1,kind=ilrg) * &
  int(u3-l3+1,kind=ilrg)

! total number of cells in the model
mcells = icells * int(codim(1),kind=ilrg) * int(codim(2),kind=ilrg) * &
  int(codim(3),kind=ilrg)

! total number of nuclei
nuc = int( resolution * real( mcells ) )

if (image1) then
  write (*,'(a,2(i0,":",i0,","),i0,":",i0,")")') &
    "bounds: (",l1,u1,l2,u2,l3,u3
  write (*,'(a,2(i0,":",i0,","),i0,":",i0,")")') &
    "cobounds: (",col1,cou1,col2,cou2,col3,cou3

  ! An absolute minimum of storage, in GB, per image.
  ! A factor of 2 is used because will call _sld, which
  ! allocates another array of the same size and kind as
  ! coarray.
  image_storage = real(2 * icells*storage_size(space)/8)/gigabyte

  write (*,'(a,i0,a)') "Each image has ",icells, " cells"
  write (*,'(a,i0,a)') "The model has ", mcells, " cells"
  write (*,'(a,i0,a)') "The model has ", nuc, " nuclei"
  write (*,'(a,es9.2,a)') "Each image will use at least ", &
    image_storage, " GB memory"
end if

! initialise random number seed
call cgca_irs(nodebug)

! allocate coarray
call cgca_as(l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3,1,space)

! initialise coarray
space = - int( this_image(), kind=iarr )

! allocate grain volume
call cgca_av(-num_images(),nuc,col1,cou1,col2,cou2,col3,grainvol)

if (image1) write (*,'(a)') "calling grain volume calc"

! calculate volumes
call cgca_gv(space,grainvol)

if (image1) write (*,'(a)') "grain calc done"
```

```
! dump grain volumes
if (image1) then
  do i=lbound(grainvol,dim=1),ubound(grainvol,dim=1)
    write (*,"(i0,tr1,i0)") i, grainvol(i)
  end do
end if
sync all

! re-initialise coarray to liquid
space = cgca_liquid_state
sync all

! populate nuclei
call cgca_nr(space,nuc,nodebug)

! solidify
call cgca_sld(space,periodicbc,0,1,solid)

! calculate volumes
call cgca_gv(space,grainvol)

! dump grain volumes
if (image1) then
 do i=lbound(grainvol,dim=1),ubound(grainvol,dim=1)
  write (*,"(i0,tr1,i0)") i, grainvol(i)
 end do
end if

! deallocate all arrays
call cgca_ds(space)
call cgca_dv(grainvol)

end program testAAN
```

89 tests/testAAO

[Unit tests]

NAME

testAAO

SYNOPSIS

```
!$Id: testAAO.f90 380 2017-03-22 11:03:09Z mexas $

program testAAO
```

PURPOSE

Timing cgca_sld (31.1)

DESCRIPTION

cgca_sld (31.1) is the first attempt of solidification. It uses SYNC ALL a lot. It is simple, but probably not efficient.

NOTES

This is timing test only. Use other tests to check the correctness of cgca_sld (31.1).

The program must be called with 2 command line arguments, both positive integers. These are codimensions along 1 and 2. The number of images must be such that codimension3 = num_images()/(codimension1 * codimension3) is a positive integer. Example:

```
      cafrun -np 16 ./testAAO.x 2 2      ! OpenCoarrays
```

or

```
      ./testAAO.x 2 2                    ! Intel, Cray
```

which will make the third codimension equal to 16/(2*2)=4.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

cgca testaux (133)

USED BY

Part of CGPACK test suite

SOURCE

```
use testaux

implicit none
```

```
real, parameter :: gigabyte=real(2**30), resolution=1.0e-5
logical(kind=ldef),parameter :: yesdebug = .true., nodebug = .false.,  &
 periodicbc = .true., noperiodicbc = .false.

integer(kind=idef) :: l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3,cou3, &
 nuc,    & ! number of nuclei in the model
 nimages, codim(3)[*]
integer(kind=iarr),allocatable :: space(:,:,:,:)[:,:,:]
integer(kind=ilrg) :: icells, mcells, img

logical(kind=ldef) :: solid

real :: image_storage, time1, time2

!*******************************************************************72
! first executable statement

img = this_image()
nimages = num_images()

! do a check on image 1
if ( img .eq. 1 ) then
 call getcodim(nimages,codim)
 ! print a banner
 call banner("AAO")
 ! print the parameter values
 call cgca_pdmp
 write (*,'(a,i0,a)') "running on ", nimages, " images in a 3D grid"
end if

! all images read codim from image 1
if ( img .eq. 1 ) then
 sync images(*)
else
 sync images(1)
 codim(:) = codim(:)[1]
end if

l1=1
l2=l1
l3=l1

! The array size is only controlled by this value
! in this program.
u1=16
u2=32
u3=32

col1=1
cou1=codim(1)-col1+1
col2=1
```

```
cou2=codim(2)-col2+1
col3=1
cou3=codim(3)-col3+1

! total number of cells in a coarray
icells = int(u1-l1+1,kind=ilrg) * int(u2-l2+1,kind=ilrg) * &
  int(u3-l3+1,kind=ilrg)

! total number of cells in the model
mcells = icells * int(codim(1),kind=ilrg) * int(codim(2),kind=ilrg) * &
  int(codim(3),kind=ilrg)

! total number of nuclei
nuc = int( resolution * real( mcells ) )

if ( img .eq. 1 ) then
  write (*,'(a,2(i0,":",i0,","),i0,":",i0,")")') &
    "bounds: (",l1,u1,l2,u2,l3,u3
  write (*,'(a,2(i0,":",i0,","),i0,":",i0,")")') &
    "cobounds: (",col1,cou1,col2,cou2,col3,cou3

  ! An absolute minimum of storage, in GB, per image.
  ! A factor of 2 is used because will call _sld, which
  ! allocates another array of the same size and kind as
  ! coarray.
  image_storage = real(2 * icells*storage_size(space)/8)/gigabyte

  write (*,'(a,i0,a)') "Each image has ",icells, " cells"
  write (*,'(a,i0,a)') "The model has ", mcells, " cells"
  write (*,'(a,i0,a)') "The model has ", nuc, " nuclei"
  write (*,'(a,es9.2,a)') "Each image will use at least ", &
    image_storage, " GB memory"
end if

! initialise random number seed
call cgca_irs(nodebug)

! allocate coarray
call cgca_as(l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3,1,space)

! initialise coarray to liquid
space = cgca_liquid_state

! populate nuclei
call cgca_nr(space,nuc,nodebug)

! solidify
call cpu_time(time1)
call cgca_sld(space,noperiodicbc,0,1,solid)
call cpu_time(time2)
write (*,*) "img", img, "time, s", time2-time1
```

```
! dump the model
!call cgca_swci(space,cgca_state_type_grain,10,'z.raw')

! deallocate all arrays
call cgca_ds(space)

end program testAAO
```

90 tests/testAAP

[Unit tests]

NAME

testAAP

SYNOPSIS

```
!$Id: testAAP.f90 529 2018-03-26 11:25:45Z mexas $

program testAAP
```

PURPOSE

Checking: cgca_art (10.1), cgca_drt (10.4)

DESCRIPTION

Checking rotation tensor array alloc/dealloc.

NOTES

The program must be called with 2 command line arguments, both positive integers. These are codimensions along 1 and 2. The number of images must be such that codimension3 = num_images()/(codimension1 * codimension3) is a positive integer. Example:

```
      cafrun -np 16 ./testAAP.x 2 2      ! OpenCoarrays
```

or

```
      ./testAAP.x 2 2                    ! Intel, Cray
```

which will make the third codimension equal to 16/(2*2)=4.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

cgca testaux (133)

USED BY

Part of CGPACK test suite

SOURCE

```
use testaux

implicit none

 logical(kind=ldef),parameter :: yesdebug=.true., nodebug=.false., &
```

```
   periodicbc=.true.
real, parameter :: gigabyte=real(2**30), resolution=1.0e-5

 real(kind=rdef),allocatable :: grt(:,:,:)[:,:,:]

 integer(kind=idef) :: l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3, &
   cou3,   &
   nuc,    & ! number of nuclei in the model
   nimages,codim(3)[*]
 integer(kind=iarr),allocatable :: space(:,:,:,:)[:,:,:]
 integer(kind=ilrg),allocatable :: grainvol(:)[:,:,:]
 integer(kind=ilrg) :: icells,mcells

 logical(kind=ldef) :: solid,image1

 real :: image_storage

!*********************************************************************72
! first executable statement

nimages=num_images()
image1=.false.
if (this_image().eq.1) image1=.true.

! do a check on image 1
if (image1) then
 call getcodim(nimages,codim)
 ! print a banner
 call banner("AAP")
 ! print the parameter values
 call cgca_pdmp
 write (*,'(a,i0,a)') "running on ", nimages, " images in a 3D grid"
 write (*,*) "codim:", codim
end if

sync all

codim(:) = codim(:)[1]

if (image1) call system("sleep 1")

l1=1
l2=l1
l3=l1

! The array size is only controlled by this value
! in this program.
u1=300
u2=u1
u3=u1

col1=1
```

```
cou1=codim(1)-col1+1
col2=1
cou2=codim(2)-col2+1
col3=1
cou3=codim(3)-col3+1

! total number of cells in a coarray
icells = int(u1-l1+1,kind=ilrg) * int(u2-l2+1,kind=ilrg) * &
  int(u3-l3+1,kind=ilrg)

! total number of cells in the model
mcells = icells * int(codim(1),kind=ilrg) * int(codim(2),kind=ilrg) * &
  int(codim(3),kind=ilrg)

! total number of nuclei
nuc = int( resolution * real( mcells ) )

if (image1) then
  write (*,'(a,2(i0,":",i0,","),i0,":",i0,")")') &
    "bounds: (",l1,u1,l2,u2,l3,u3
  write (*,'(a,2(i0,":",i0,","),i0,":",i0,")")') &
    "cobounds: (",col1,cou1,col2,cou2,col3,cou3

  ! An absolute minimum of storage, in GB, per image.
  ! A factor of 2 is used because will call _sld, which
  ! allocates another array of the same size and kind as
  ! coarray.
  image_storage = real(2 * icells*storage_size(space)/8)/gigabyte

  write (*,'(a,i0,a)') "Each image has ",icells, " cells"
  write (*,'(a,i0,a)') "The model has ", mcells, " cells"
  write (*,'(a,i0,a)') "The model has ", nuc, " nuclei"
  write (*,'(a,es9.2,a)') "Each image will use at least ", &
    image_storage, " GB memory"
end if

! initialise random number seed
call cgca_irs(nodebug)

! allocate coarray
call cgca_as(l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3,1,space)

! initialise coarray
space = int( this_image(), kind=iarr )
solid = .true.

! allocate grain volume
call cgca_av(1,nuc,col1,cou1,col2,cou2,col3,grainvol)

! calculate volumes, sync all inside
call cgca_gv(space,grainvol)
```

```
! allocate rotation tensors
call cgca_art(1,nuc,col1,cou1,col2,cou2,col3,grt)
if (image1) write (*,*) "successfully allocated rotation tensor coarray"

! dump grain volumes
if (image1) write (*,"(i0)") grainvol

! deallocate all arrays
call cgca_ds(space)
call cgca_dv(grainvol)
call cgca_drt(grt)
if (image1) write (*,*) "successfully deallocated rotation tensor coarray"

end program testAAP
```

91 tests/testAAQ

[Unit tests]

NAME

testAAQ

SYNOPSIS

```
!$Id: testAAQ.f90 529 2018-03-26 11:25:45Z mexas $

program testAAQ
```

PURPOSE

Checking: cgca_rt (24.6), cgca_ckrt (24.1)

DESCRIPTION

Checking rotation tensor calculation and orthogonality checking routine.

NOTES

The program must be called with 2 command line arguments, both positive integers. These are codimensions along 1 and 2. The number of images must be such that codimension3 = num_images()/(codimension1 * codimension3) is a positive integer. Example:

```
      cafrun -np 16 ./testAAQ.x 2 2      ! OpenCoarrays
```

or

```
      ./testAAQ.x 2 2                    ! Intel, Cray
```

which will make the third codimension equal to 16/(2*2)=4.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

cgca testaux (133)

USED BY

Part of CGPACK test suite

SOURCE

```
use testaux

implicit none

 logical(kind=ldef),parameter :: yesdebug=.true., nodebug=.false., &
```

```
   periodicbc=.true.
real, parameter :: gigabyte=real(2**30), resolution=1.0e-5

 real(kind=rdef),allocatable :: grt(:,:,:)[:,:,:]

 integer(kind=idef) :: l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3, &
   cou3,   &
   nuc,    & ! number of nuclei in the model
   nimages,flag,codim(3)[*],i
 integer(kind=iarr),allocatable :: space(:,:,:,:)[:,:,:]
 integer(kind=ilrg),allocatable :: grainvol(:)[:,:,:]
 integer(kind=ilrg) :: icells,mcells

 logical(kind=ldef) :: solid,image1

 real :: image_storage

!*********************************************************************72
! first executable statement

nimages=num_images()
image1=.false.
if (this_image().eq.1) image1=.true.

! do a check on image 1
if (image1) then
 call getcodim(nimages,codim)
 ! print a banner
 call banner("AAQ")
 ! print the parameter values
 call cgca_pdmp
 write (*,'(a,i0,a)') "running on ", nimages, " images in a 3D grid"
 write (*,*) "codim:", codim
end if

sync all

codim(:) = codim(:)[1]

if (image1) call system("sleep 1")

l1=1
l2=l1
l3=l1

! The array size is only controlled by this value
! in this program.
u1=100
u2=u1
u3=u1

col1=1
```

```
cou1=codim(1)-col1+1
col2=1
cou2=codim(2)-col2+1
col3=1
cou3=codim(3)-col3+1

! total number of cells in a coarray
icells = int(u1-l1+1,kind=ilrg) * int(u2-l2+1,kind=ilrg) * &
  int(u3-l3+1,kind=ilrg)

! total number of cells in the model
mcells = icells * int(codim(1),kind=ilrg) * int(codim(2),kind=ilrg) * &
  int(codim(3),kind=ilrg)

! total number of nuclei
nuc = int( resolution * real( mcells) )

if (image1) then
  write (*,'(a,2(i0,":",i0,","),i0,":",i0,")")') &
    "bounds: (",l1,u1,l2,u2,l3,u3
  write (*,'(a,2(i0,":",i0,","),i0,":",i0,")")') &
    "cobounds: (",col1,cou1,col2,cou2,col3,cou3

  ! An absolute minimum of storage, in GB, per image.
  image_storage = real( &
    ! 2nd space array is allocated in _sld
    2 * icells*storage_size(space) &
    ! grain volumes
    + nuc*storage_size(grainvol) &
    ! rotation tensors
    + nuc*9*storage_size(grt))/8/gigabyte

  write (*,'(a,i0,a)') "Each image has ",icells, " cells"
  write (*,'(a,i0,a)') "The model has ", mcells, " cells"
  write (*,'(a,i0,a)') "The model has ", nuc, " nuclei"
  write (*,'(a,es9.2,a)') "Each image will use at least ", &
    image_storage, " GB memory"
end if

! initialise random number seed
call cgca_irs(nodebug)

! allocate coarray
call cgca_as(l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3,1,space)

! initialise coarray
space = int( this_image(), kind=iarr )
solid = .true.

! allocate grain volume
call cgca_av(1,nuc,col1,cou1,col2,cou2,col3,grainvol)
```

```
! calculate volumes, sync all inside
call cgca_gv(space,grainvol)

! allocate rotation tensors
call cgca_art(1,nuc,col1,cou1,col2,cou2,col3,grt)

! assign rotation tensors, sync all inside
call cgca_rt(grt)
sync all

! check all rotation tensors on image 1
if (image1) then
  write (*,*) "rotation tensors assigned"
  do i=1,nuc
    call cgca_ckrt(grt(i,:,:),yesdebug,flag)
    if (flag .ne. 0) then
      write (*,*) "problem with grain: ", i
      write (*,*) "failed test: ", flag
      write (*,*) "stopping!"
      error stop
    end if
  end do
end if
sync all

! deallocate all arrays
call cgca_ds(space)
call cgca_dv(grainvol)
call cgca_drt(grt)
if (image1) write (*,*) "successfully deallocated rotation tensor coarray"

end program testAAQ
```

92 tests/testAAR

[Unit tests]

NAME

testAAR

SYNOPSIS

```
!$Id: testAAR.f90 529 2018-03-26 11:25:45Z mexas $

program testAAR
```

PURPOSE

Checking: cgca_mis (24.4), cgca_csym (24.2), cgca_rt (24.6)

DESCRIPTION

Checking mis-orientation calculations

NOTES

The program must be called with 2 command line arguments, both positive integers. These are codimensions along 1 and 2. The number of images must be such that codimension3 = num_images()/(codimension1 * codimension3) is a positive integer. Example:

```
      cafrun -np 16 ./testAAR.x 2 2      ! OpenCoarrays
```

or

```
      ./testAAR.x 2 2                    ! Intel, Cray
```

which will make the third codimension equal to 16/(2*2)=4.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

cgca testaux (133)

USED BY

Part of CGPACK test suite

SOURCE

```
use testaux

implicit none

 logical(kind=ldef),parameter :: yesdebug=.true., nodebug=.false., &
```

```
   periodicbc=.true.
real, parameter :: gigabyte=real(2**30), resolution=1.0e-5

 real(kind=rdef),allocatable :: grt(:,:,:)[:,:,:]
 real(kind=rdef) :: angle, minang

 integer(kind=idef) :: l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3, &
   cou3,   &
   nuc,    & ! number of nuclei in the model
   nimages,flag,codim(3)[*],i,j
 integer(kind=iarr),allocatable :: space(:,:,:,:)[:,:,:]
 integer(kind=ilrg),allocatable :: grainvol(:)[:,:,:]
 integer(kind=ilrg) :: icells,mcells

 logical(kind=ldef) :: solid,image1

 real :: image_storage

!*********************************************************************72
! first executable statement

nimages=num_images()
image1=.false.
if (this_image().eq.1) image1=.true.

! do a check on image 1
if (image1) then
 call getcodim(nimages,codim)
 ! print a banner
 call banner("AAR")
 ! print the parameter values
 call cgca_pdmp
 write (*,'(a,i0,a)') "running on ", nimages, " images in a 3D grid"
 write (*,*) "codim:", codim
end if

sync all

codim(:) = codim(:)[1]

if (image1) call system("sleep 1")

l1=1
l2=l1
l3=l1

! The array size is only controlled by this value
! in this program.
u1=100
u2=u1
u3=u1
```

```
col1=1
cou1=codim(1)-col1+1
col2=1
cou2=codim(2)-col2+1
col3=1
cou3=codim(3)-col3+1

! total number of cells in a coarray
icells = int(u1-l1+1,kind=ilrg) * int(u2-l2+1,kind=ilrg) * &
  int(u3-l3+1,kind=ilrg)

! total number of cells in the model
mcells = icells * int(codim(1),kind=ilrg) * int(codim(2),kind=ilrg) * &
  int(codim(3),kind=ilrg)

! total number of nuclei
nuc = int( resolution * real( mcells ) )

if (image1) then
  write (*,'(a,2(i0,":",i0,","),i0,":",i0,")")') &
    "bounds: (",l1,u1,l2,u2,l3,u3
  write (*,'(a,2(i0,":",i0,","),i0,":",i0,")")') &
    "cobounds: (",col1,cou1,col2,cou2,col3,cou3

  ! An absolute minimum of storage, in GB, per image.
  image_storage = real( &
    ! 2nd space array is allocated in _sld
    2 * icells*storage_size(space) &
    ! grain volumes
    + nuc*storage_size(grainvol) &
    ! rotation tensors
    + nuc*9*storage_size(grt))/8/gigabyte

  write (*,'(a,i0,a)') "Each image has ",icells, " cells"
  write (*,'(a,i0,a)') "The model has ", mcells, " cells"
  write (*,'(a,i0,a)') "The model has ", nuc, " nuclei"
  write (*,'(a,es9.2,a)') "Each image will use at least ", &
    image_storage, " GB memory"
end if

! initialise random number seed
call cgca_irs(nodebug)

! allocate coarray
call cgca_as(l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3,1,space)

! initialise coarray
space = int( this_image(), kind=iarr )
solid = .true.

! allocate grain volume
call cgca_av(1,nuc,col1,cou1,col2,cou2,col3,grainvol)
```

```
! calculate volumes, sync all inside
call cgca_gv(space,grainvol)

! allocate rotation tensors
call cgca_art(1,nuc,col1,cou1,col2,cou2,col3,grt)

! assign rotation tensors, sync all inside
call cgca_rt(grt)

! check all rotation tensors on image 1
if (image1) then
  write (*,*) "rotation tensors assigned"
  do i=1,nuc
    call cgca_ckrt(grt(i,:,:),yesdebug,flag)
    if (flag .ne. 0) then
      write (*,*) "problem with grain: ", i
      write (*,*) "failed test: ", flag
      write (*,*) "stopping!"
      error stop
    end if
  end do
  write (*,*) "all rotation tensors are fine!"
end if
sync all

! mis-orientation angle between grains

if (image1) then
 do i=1,nuc-1
  do j=i+1,nuc
   call cgca_mis(grt(i,:,:),grt(j,:,:),angle)
   call cgca_miscsym(grt(i,:,:),grt(j,:,:),minang)
   write (*,'(2(f5.2))') angle,minang
  end do
 end do
end if
sync all

! deallocate all arrays
call cgca_ds(space)
call cgca_dv(grainvol)
call cgca_drt(grt)
if (image1) write (*,*) "successfully deallocated rotation tensor coarray"

end program testAAR
```

93 tests/testAAS

[Unit tests]

NAME

testAAS

SYNOPSIS

```
!$Id: testAAS.f90 380 2017-03-22 11:03:09Z mexas $

program testAAS
```

PURPOSE

Checking: cgca_dacf (26.9)

DESCRIPTION

Checking deactivation of crack flanks. Note that in this test we need to use at least 2 state types in the space coarray.

In image1 coarray only, cells on one plane are given "crack edge" values. Then cgca_dacf (26.9) is called. As all interior cells on this planes are switched to "crack flanks".

NOTES

The program must be called with 2 command line arguments, both positive integers. These are codimensions along 1 and 2. The number of images must be such that codimension3 = num_images()/(codimension1 * codimension3) is a positive integer. Example:

```
      cafrun -np 16 ./testAAS.x 2 2      ! OpenCoarrays
```

or

```
      ./testAAS.x 2 2                    ! Intel, Cray
```

which will make the third codimension equal to 16/(2*2)=4.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

cgca testaux (133)

USED BY

Part of CGPACK test suite

SOURCE

```
use testaux
```

```
implicit none

 logical(kind=ldef),parameter :: yesdebug=.true., nodebug=.false., &
   periodicbc=.true.
 real,parameter :: gigabyte=real(2**30), resolution=1.0e-5, &

! cleavage stress on 100, 110, 111 planes for BCC,
! see the manual for derivation.
  scrit(3) = (/ 1.05e4, 1.25e4, 4.90e4 /)

 real(kind=rdef),allocatable :: grt(:,:,:)[:,:,:]
 real(kind=rdef) :: s1(3)

 integer(kind=idef) :: l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3, &
   cou3,   &
   nuc,    & ! number of nuclei in the model
   nimages,codim(3)[*]
 integer(kind=iarr),allocatable :: space(:,:,:,:)[:,:,:]
 integer(kind=ilrg),allocatable :: grainvol(:)[:,:,:], fracvol(:)[:,:,:]
 integer(kind=ilrg) :: icells,mcells

 logical(kind=ldef) :: image1

 real :: image_storage

!*********************************************************************72
! first executable statement

nimages=num_images()
image1=.false.
if (this_image().eq.1) image1=.true.

! do a check on image 1
if (image1) then
 call getcodim(nimages,codim)
 ! print a banner
 call banner("AAS")
 ! print the parameter values
 call cgca_pdmp
 write (*,'(a,i0,a)') "running on ", nimages, " images in a 3D grid"
 write (*,*) "codim:", codim
end if

sync all

codim(:) = codim(:)[1]

l1=1
l2=l1
l3=l1

! The array size is only controlled by this value
```

```
! in this program.
u1=10
u2=u1
u3=u1

col1=1
cou1=codim(1)-col1+1
col2=1
cou2=codim(2)-col2+1
col3=1
cou3=codim(3)-col3+1

! total number of cells in a coarray
icells = int(u1-l1+1,kind=ilrg) * int(u2-l2+1,kind=ilrg) * &
  int(u3-l3+1,kind=ilrg)

! total number of cells in the model
mcells = icells * int(codim(1),kind=ilrg) * int(codim(2),kind=ilrg) * &
  int(codim(3),kind=ilrg)

! total number of nuclei
!nuc = resolution*mcells
! just 1 nuclei in this test
nuc=1

if (image1) then
  write (*,'(a,2(i0,":",i0,","),i0,":",i0,")")') &
    "bounds: (",l1,u1,l2,u2,l3,u3
  write (*,'(a,2(i0,":",i0,","),i0,":",i0,")")') &
    "cobounds: (",col1,cou1,col2,cou2,col3,cou3

  ! An absolute minimum of storage, in GB, per image.
  image_storage = real( &
    ! 2nd space array is allocated in _sld
    2 * icells*storage_size(space) &
    ! grain volumes
    + nuc*storage_size(grainvol) &
    ! rotation tensors
    + nuc*9*storage_size(grt))/8/gigabyte

  write (*,'(a,i0,a)') "Each image has ",icells, " cells"
  write (*,'(a,i0,a)') "The model has ", mcells, " cells"
  write (*,'(a,i0,a)') "The model has ", nuc, " nuclei"
  write (*,'(a,es9.2,a)') "Each image will use at least ", &
    image_storage, " GB memory"
end if

! initialise random number seed
call cgca_irs(nodebug)

! allocate coarray
call cgca_as(l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3,2,space)
```

```
! initialise coarray: a single grain and no damage
space(:,:,:,cgca_state_type_grain) = 1
space(:,:,:,cgca_state_type_frac) = cgca_intact_state

! modify the fracture state array:
! set image 1 cells on some plane to an edge state
space(l1+1:u1-1,l2+1:u2-1,u3/2,cgca_state_type_frac)[col1,col2,col3] = &
 cgca_clvg_state_100_edge

! allocate grain and fracture volumes
call cgca_av(1,nuc,col1,cou1,col2,cou2,col3,grainvol)
call cgca_av(1,nuc,col1,cou1,col2,cou2,col3,fracvol)

! allocate rotation tensors
call cgca_art(1,nuc,col1,cou1,col2,cou2,col3,grt)

! assign rotation tensors, sync all inside
call cgca_rt(grt)

! set s1, although not used in this test
s1 = (/ 1.0, 1.0, 1.0 /)

! dump both the grain and the fracture arrays
call cgca_swci(space,cgca_state_type_grain,10,"zg1.raw")
call cgca_swci(space,cgca_state_type_frac,10,"zf1.raw")
sync all

! deactivate flanks
call cgca_dacf(space,yesdebug)
sync all

! dump both the grain and the fracture arrays
call cgca_swci(space,cgca_state_type_grain,10,"zg2.raw")
call cgca_swci(space,cgca_state_type_frac,10,"zf2.raw")

! deallocate all arrays
call cgca_ds(space)
call cgca_dv(grainvol)
call cgca_dv(fracvol)
call cgca_drt(grt)

end program testAAS
```

94 tests/testAAT

[Unit tests]

NAME

testAAT

SYNOPSIS

```
!$Id: testAAT.f90 380 2017-03-22 11:03:09Z mexas $

program testAAT
```

PURPOSE

Checking: cgca_dacf (26.9), cgca_hxi (15.2), cgca_hxg (15.1)

DESCRIPTION

Checking deactivation of crack flanks with halo exchange. In image1 fracture coarray only, cells on one plane are given "crack edge" values. Then cgca_dacf (26.9) is called. As all interior cells on this planes are switched to "crack flanks".

NOTES

The program must be called with 2 command line arguments, both positive integers. These are codimensions along 1 and 2. The number of images must be such that codimension3 = num_images()/(codimension1 * codimension3) is a positive integer. Example:

```
      cafrun -np 16 ./testAAT.x 2 2      ! OpenCoarrays
```

or

```
      ./testAAT.x 2 2                    ! Intel, Cray
```

which will make the third codimension equal to 16/(2*2)=4.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

cgca testaux (133)

USED BY

Part of CGPACK test suite

SOURCE

```
use testaux

implicit none
```

```
 logical(kind=ldef),parameter :: yesdebug=.true., nodebug=.false., &
   periodicbc=.true.
 real,parameter :: gigabyte=real(2**30), resolution=1.0e-5, &

! cleavage stress on 100, 110, 111 planes for BCC,
! see the manual for derivation.
  scrit(3) = (/ 1.05e4, 1.25e4, 4.90e4 /)

 real(kind=rdef),allocatable :: grt(:,:,:)[:,:,:]
 real(kind=rdef) :: s1(3)

 integer(kind=idef) :: l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3, &
   cou3,   &
   nuc,    & ! number of nuclei in the model
   nimages,codim(3)[*]
 integer(kind=iarr),allocatable :: space(:,:,:,:)[:,:,:]
 integer(kind=ilrg),allocatable :: grainvol(:)[:,:,:],fracvol(:)[:,:,:]
 integer(kind=ilrg) :: icells,mcells

 logical(kind=ldef) :: image1

 real :: image_storage

!*********************************************************************72
! first executable statement

nimages=num_images()
image1=.false.
if (this_image().eq.1) image1=.true.

! do a check on image 1
if (image1) then
 call getcodim(nimages,codim)
 ! print a banner
 call banner("AAT")
 ! print the parameter values
 call cgca_pdmp
 write (*,'(a,i0,a)') "running on ", nimages, " images in a 3D grid"
 write (*,*) "codim:", codim
end if

sync all

codim(:) = codim(:)[1]

l1=1
l2=l1
l3=l1

! The array size is only controlled by this value
! in this program.
```

```
u1=10
u2=u1
u3=u1

col1=1
cou1=codim(1)-col1+1
col2=1
cou2=codim(2)-col2+1
col3=1
cou3=codim(3)-col3+1

! total number of cells in a coarray
icells = int(u1-l1+1,kind=ilrg) * int(u2-l2+1,kind=ilrg) * &
  int(u3-l3+1,kind=ilrg)

! total number of cells in the model
mcells = icells * int(codim(1),kind=ilrg) * int(codim(2),kind=ilrg) * &
  int(codim(3),kind=ilrg)

! total number of nuclei
!nuc = resolution*mcells
! just 1 nuclei in this test
nuc=1

if (image1) then
  write (*,'(a,2(i0,":",i0,","),i0,":",i0,")")') &
    "bounds: (",l1,u1,l2,u2,l3,u3
  write (*,'(a,2(i0,":",i0,","),i0,":",i0,")")') &
    "cobounds: (",col1,cou1,col2,cou2,col3,cou3

  ! An absolute minimum of storage, in GB, per image.
  image_storage = real( &
    ! 2nd space array is allocated in _sld
    2 * icells*storage_size(space) &
    ! grain volumes
    + nuc*storage_size(grainvol) &
    ! rotation tensors
    + nuc*9*storage_size(grt))/8/gigabyte

  write (*,'(a,i0,a)') "Each image has ",icells, " cells"
  write (*,'(a,i0,a)') "The model has ", mcells, " cells"
  write (*,'(a,i0,a)') "The model has ", nuc, " nuclei"
  write (*,'(a,es9.2,a)') "Each image will use at least ", &
    image_storage, " GB memory"
end if

! initialise random number seed
call cgca_irs(nodebug)

! allocate 2 coarrays, for grains and for fracture
call cgca_as(l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3,2,space)
```

```
! initialise coarrays: a single grain and no damage
space(:,:,:,cgca_state_type_grain) = 1
space(:,:,:,cgca_state_type_frac) = cgca_intact_state

! set all image 1 cells to an edge state
space(:,:,u3/2,cgca_state_type_frac)[col1,col2,col3] = &
  cgca_clvg_state_100_edge

! Allocate grain and fracture volumes
call cgca_av(1,nuc,col1,cou1,col2,cou2,col3,grainvol)
call cgca_av(1,nuc,col1,cou1,col2,cou2,col3,fracvol)

! allocate rotation tensors
call cgca_art(1,nuc,col1,cou1,col2,cou2,col3,grt)

! assign rotation tensor, aligned with spatial coord. system
grt(1,:,:) = 0.0
grt(1,1,1) = 1.0
grt(1,2,2) = 1.0
grt(1,3,3) = 1.0

! set s1, although not used in this test
s1 = (/ 1.0, 1.0, 1.0 /)

! dump both the grain and the fracture arrays
call cgca_swci(space,cgca_state_type_grain,10,"zg1.raw")
call cgca_swci(space,cgca_state_type_frac,10,"zf1.raw")
sync all

! local and global halo exchange
call cgca_hxi(space)
call cgca_hxg(space)
sync all

! deactivate flanks
call cgca_dacf(space,yesdebug)
sync all

! local and global halo exchange
call cgca_hxi(space)
call cgca_hxg(space)
sync all

! dump both the grain and the fracture arrays
call cgca_swci(space,cgca_state_type_grain,10,"zg2.raw")
call cgca_swci(space,cgca_state_type_frac,10,"zf2.raw")
sync all

! deallocate all arrays
call cgca_ds(space)
call cgca_dv(grainvol)
call cgca_dv(fracvol)
```

```
call cgca_drt(grt)

end program testAAT
```

95 tests/testAAU

[Unit tests]

NAME

testAAU

SYNOPSIS

```
!$Id: testAAU.f90 380 2017-03-22 11:03:09Z mexas $

program testAAU
```

PURPOSE

Checking: cgca_clvgp_nocosum (26.4) (no CO_SUM version), cgca_clvgsd (26.5)

DESCRIPTION

Checking deterministic cleavage propagation with 1 cleavage nucleus in the middle of image 1. Single grain with a random rotation tensor. Single cleavage increment. Check crack flank deactivation.

NOTES

The program must be called with 2 command line arguments, both positive integers. These are codimensions along 1 and 2. The number of images must be such that codimension3 = num_images()/(codimension1 * codimension3) is a positive integer. Example:

```
      cafrun -np 16 ./testAAU.x 2 2      ! OpenCoarrays
```

or

```
      ./testAAU.x 2 2                    ! Intel, Cray
```

which will make the third codimension equal to 16/(2*2)=4.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

cgca testaux (133)

USED BY

Part of CGPACK test suite

SOURCE

```
use testaux

implicit none
```

```
 logical(kind=ldef),parameter :: yesdebug=.true., nodebug=.false., &
   periodicbc=.true.
 real,parameter :: gigabyte=real(2**30), resolution=1.0e-5, &

! cleavage stress on 100, 110, 111 planes for BCC,
! see the manual for derivation.
  scrit(3) = (/ 1.05e4, 1.25e4, 4.90e4 /)

 real(kind=rdef),allocatable :: grt(:,:,:)[:,:,:]
 real(kind=rdef) :: t(3,3)

 integer(kind=idef) :: l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3, &
   cou3,   &
   nuc,    & ! number of nuclei in the model
   nimages,codim(3)[*]
 integer(kind=iarr),allocatable :: space(:,:,:,:)[:,:,:]
 integer(kind=ilrg) :: icells,mcells

 logical(kind=ldef) :: image1

!*********************************************************************72
! first executable statement

nimages=num_images()
image1=.false.
if (this_image().eq.1) image1=.true.

! do a check on image 1
if (image1) then
 call getcodim(nimages,codim)
 ! print a banner
 call banner("AAU")
 ! print the parameter values
 call cgca_pdmp
 write (*,'(a,i0,a)') "running on ", nimages, " images in a 3D grid"
 write (*,*) "codim:", codim
end if

sync all

codim(:) = codim(:)[1]

l1=1
l2=l1
l3=l1

! The array size is only controlled by this value
! in this program.
u1=10
u2=u1
u3=u1
```

```
col1=1
cou1=codim(1)-col1+1
col2=1
cou2=codim(2)-col2+1
col3=1
cou3=codim(3)-col3+1

! total number of cells in a coarray
icells = int(u1-l1+1,kind=ilrg) * int(u2-l2+1,kind=ilrg) * &
  int(u3-l3+1,kind=ilrg)

! total number of cells in the model
mcells = icells * int(codim(1),kind=ilrg) * int(codim(2),kind=ilrg) * &
  int(codim(3),kind=ilrg)

! total number of nuclei
!nuc = resolution*mcells
! just 1 nuclei in this test
nuc=1

if (image1) then
  write (*,'(a,2(i0,":",i0,","),i0,":",i0,")")') &
    "bounds: (",l1,u1,l2,u2,l3,u3
  write (*,'(a,2(i0,":",i0,","),i0,":",i0,")")') &
    "cobounds: (",col1,cou1,col2,cou2,col3,cou3
end if

! initialise random number seed
call cgca_irs(nodebug)

! allocate coarray
call cgca_as(l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3,2,space)

! initialise coarrays: a single grain and no damage
space(:,:,:,cgca_state_type_grain) = 1
space(:,:,:,cgca_state_type_frac) = cgca_intact_state

! set a single crack nucleus somewhere in the middle of image1 array
space( u1/2, u2/2, u3/2, cgca_state_type_frac ) [ col1, col2, col3 ] = &
  cgca_clvg_state_100_edge

! dump the array
call cgca_swci( space, cgca_state_type_frac, 10, "zf0.raw" )

sync all

! allocate rotation tensors
!subroutine cgca_art(l,u,col1,cou1,col2,cou2,col3,coarray)
call cgca_art(1,nuc,col1,cou1,col2,cou2,col3,grt)

! assign rotation tensors, sync all inside
call cgca_rt(grt)
```

```
! set the stress tensor
t = 0.0
t(1,1) = 1.0e5

! propagate cleavage, sync inside
! subroutine cgca_clvgp_nocosum( coarray, rt, t, scrit, sub,
!    gcus, periodicbc, iter, heartbeat, debug )
call cgca_clvgp_nocosum( space, grt, t, scrit, cgca_clvgsd,            &
     cgca_gcupdn, .false., 1, 1, yesdebug )

! dump the array
call cgca_swci( space, cgca_state_type_frac, 10, "zf1.raw" )

! deallocate all arrays
call cgca_ds(space)
call cgca_drt(grt)

end program testAAU
```

96 tests/testAAV

[Unit tests]

NAME

testAAV

SYNOPSIS

```
!$Id: testAAV.f90 380 2017-03-22 11:03:09Z mexas $

program testAAV
```

PURPOSE

Checking: cgca_clvgp_nocosum (26.4) (no CO_SUM version), cgca_clvgsd (26.5)

DESCRIPTION

Checking deterministic cleavage propagation with 1 cleavage nucleus in the middle of image 1. Single grain with random orientation. Several cleavage increments. Check how crack propagates across image boundary.

NOTES

The program must be called with 2 command line arguments, both positive integers. These are codimensions along 1 and 2. The number of images must be such that codimension3 = num_images()/(codimension1 * codimension3) is a positive integer. Example:

```
      cafrun -np 16 ./testAAV.x 2 2      ! OpenCoarrays
```

or

```
      ./testAAV.x 2 2                    ! Intel, Cray
```

which will make the third codimension equal to 16/(2*2)=4.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

cgca testaux (133)

USED BY

Part of CGPACK test suite

SOURCE

```
use testaux

implicit none
```

```
 logical(kind=ldef),parameter :: yesdebug=.true., nodebug=.false., &
   periodicbc=.true.
 real,parameter :: gigabyte=real(2**30), resolution=1.0e-5, &

! cleavage stress on 100, 110, 111 planes for BCC,
! see the manual for derivation.
  scrit(3) = (/ 1.05e4, 1.25e4, 4.90e4 /)

 real(kind=rdef),allocatable :: grt(:,:,:)[:,:,:]
 real(kind=rdef) :: t(3,3)  ! stress tensor

 integer(kind=idef) :: l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3, &
   cou3,   &
   nuc,    & ! number of nuclei in the model
   nimages,codim(3)[*]
 integer(kind=iarr),allocatable :: space(:,:,:,:)[:,:,:]
 integer(kind=ilrg),allocatable :: grainvol(:)[:,:,:],fracvol(:)[:,:,:]
 integer(kind=ilrg) :: icells,mcells

 logical(kind=ldef) :: image1

 real :: image_storage

!*********************************************************************72
! first executable statement

nimages=num_images()
image1=.false.
if (this_image().eq.1) image1=.true.

! do a check on image 1
if (image1) then
 call getcodim(nimages,codim)
 ! print a banner
 call banner("AAV")
 ! print the parameter values
 call cgca_pdmp
 write (*,'(a,i0,a)') "running on ", nimages, " images in a 3D grid"
 write (*,*) "codim:", codim
end if

sync all

codim(:) = codim(:)[1]

l1=1
l2=l1
l3=l1

! The array size is only controlled by this value
! in this program.
```

```
u1=50
u2=u1
u3=u1

col1=1
cou1=codim(1)-col1+1
col2=1
cou2=codim(2)-col2+1
col3=1
cou3=codim(3)-col3+1

! total number of cells in a coarray
icells = int(u1-l1+1,kind=ilrg) * int(u2-l2+1,kind=ilrg) * &
  int(u3-l3+1,kind=ilrg)

! total number of cells in the model
mcells = icells * int(codim(1),kind=ilrg) * int(codim(2),kind=ilrg) * &
  int(codim(3),kind=ilrg)

! total number of nuclei
!nuc = resolution*mcells
! just 1 nuclei in this test
nuc=1

if (image1) then
  write (*,'(a,2(i0,":",i0,","),i0,":",i0,")")') &
    "bounds: (",l1,u1,l2,u2,l3,u3
  write (*,'(a,2(i0,":",i0,","),i0,":",i0,")")') &
    "cobounds: (",col1,cou1,col2,cou2,col3,cou3

  ! An absolute minimum of storage, in GB, per image.
  image_storage = real( &
    ! 2nd space array is allocated in _sld
    2 * icells*storage_size(space) &
    ! grain volumes
    + nuc*storage_size(grainvol) &
    ! rotation tensors
    + nuc*9*storage_size(grt))/8/gigabyte

  write (*,'(a,i0,a)') "Each image has ",icells, " cells"
  write (*,'(a,i0,a)') "The model has ", mcells, " cells"
  write (*,'(a,i0,a)') "The model has ", nuc, " nuclei"
  write (*,'(a,es9.2,a)') "Each image will use at least ", &
    image_storage, " GB memory"
end if

! initialise random number seed
call cgca_irs(nodebug)

! allocate coarray
call cgca_as(l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3,2,space)
```

```
! initialise coarrays: a single grain and no damage
space(:,:,:,cgca_state_type_grain) = 1
space(:,:,:,cgca_state_type_frac) = cgca_intact_state

! set a single crack nucleus at the upper end of image1 array
space(u1,u2,u3,cgca_state_type_frac)[col1,col2,col3] = &
 cgca_clvg_state_100_edge

! Allocate grain and fracture volumes
call cgca_av(1,nuc,col1,cou1,col2,cou2,col3,grainvol)
call cgca_av(1,nuc,col1,cou1,col2,cou2,col3,fracvol)

! allocate rotation tensors
call cgca_art(1,nuc,col1,cou1,col2,cou2,col3,grt)

! set the stress tensor
t = 0.0
t(1,1) = 1.0e6
t(2,2) = -1.0e6

! assign rotation tensors, sync all inside
call cgca_rt(grt)


! propagate cleavage, sync inside
! subroutine cgca_clvgp_nocosum( coarray, rt, t, scrit, sub,
!    gcus, periodicbc, iter, heartbeat, debug )
call cgca_clvgp_nocosum( space, grt, t, scrit, cgca_clvgsd,            &
     cgca_gcupdn, .false., 50, 10, nodebug )

! dump the array
call cgca_swci(space,cgca_state_type_frac,10,"zf.raw")
sync all

! calculate volumes, sync all inside
call cgca_gv(space,grainvol)

! dump grain volumes
if (image1) write (*,"(i0)") grainvol

! deallocate all arrays
call cgca_ds(space)
call cgca_dv(grainvol)
call cgca_dv(fracvol)
call cgca_drt(grt)

end program testAAV
```

97 tests/testAAW

[Unit tests]

NAME

testAAW

SYNOPSIS

```
!$Id: testAAW.f90 380 2017-03-22 11:03:09Z mexas $

program testAAW
```

PURPOSE

Checking: cgca_clvgp (26.17.1) (no CO_SUM version), cgca_clvgsp (26.7)

DESCRIPTION

Checking probabilistic cleavage propagation with 1 cleavage nucleus in the middle of image 1. Single grain with random rotation tensor. 1-2 cleavage increments. Halo exchange, so cleavage propagation across image boundary. Check crack flank deactivation.

NOTES

The program must be called with 2 command line arguments, both positive integers. These are codimensions along 1 and 2. The number of images must be such that codimension3 = num_images()/(codimension1 * codimension3) is a positive integer. Example:

```
      cafrun -np 16 ./testAAW.x 2 2      ! OpenCoarrays
```

or

```
      ./testAAW.x 2 2                    ! Intel, Cray
```

which will make the third codimension equal to 16/(2*2)=4.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

cgca testaux (133)

USED BY

Part of CGPACK test suite

SOURCE

```
use testaux

implicit none
```

```
 logical(kind=ldef),parameter :: yesdebug=.true., nodebug=.false., &
   periodicbc=.true., noperiodicbc=.false.
 real,parameter :: gigabyte=real(2**30), resolution=1.0e-5, &

! cleavage stress on 100, 110, 111 planes for BCC,
! see the manual for derivation.
  scrit(3) = (/ 1.05e4, 1.25e4, 4.90e4 /)

 real(kind=rdef),allocatable :: grt(:,:,:)[:,:,:]
 real(kind=rdef) :: t(3,3)   ! stress tensor

 integer(kind=idef) :: l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3, &
   cou3,   &
   nuc,    & ! number of nuclei in the model
   nimages,codim(3)[*]
 integer(kind=iarr),allocatable :: space(:,:,:,:)[:,:,:]
 integer(kind=ilrg),allocatable :: grainvol(:)[:,:,:],fracvol(:)[:,:,:]
 integer(kind=ilrg) :: icells,mcells

 logical(kind=ldef) :: image1

 real :: image_storage

!*********************************************************************72
! first executable statement

nimages=num_images()
image1=.false.
if (this_image().eq.1) image1=.true.

! do a check on image 1
if (image1) then
 call getcodim(nimages,codim)
 ! print a banner
 call banner("AAW")
 ! print the parameter values
 call cgca_pdmp
 write (*,'(a,i0,a)') "Running on ", nimages, " images in a 3D grid."
end if

sync all

codim(:) = codim(:)[1]

l1=1
l2=l1
l3=l1

! The array size is only controlled by this value
! in this program.
u1=10
```

```
u2=u1
u3=u1

col1=1
cou1=codim(1)-col1+1
col2=1
cou2=codim(2)-col2+1
col3=1
cou3=codim(3)-col3+1

! total number of cells in a coarray
icells = int(u1-l1+1,kind=ilrg) * int(u2-l2+1,kind=ilrg) * &
  int(u3-l3+1,kind=ilrg)

! total number of cells in the model
mcells = icells * int(codim(1),kind=ilrg) * int(codim(2),kind=ilrg) * &
  int(codim(3),kind=ilrg)

! total number of nuclei
!nuc = resolution*mcells
! just 1 nuclei in this test
nuc=1

if (image1) then
  write (*,'(a,2(i0,":",i0,","),i0,":",i0,")")') &
    "bounds: (",l1,u1,l2,u2,l3,u3
  write (*,'(a,2(i0,":",i0,","),i0,":",i0,")")') &
    "cobounds: (",col1,cou1,col2,cou2,col3,cou3

  ! An absolute minimum of storage, in GB, per image.
  image_storage = real( &
    ! 2nd space array is allocated in _sld
    2 * icells*storage_size(space) &
    ! grain volumes
    + nuc*storage_size(grainvol) &
    ! rotation tensors
    + nuc*9*storage_size(grt))/8/gigabyte

  write (*,'(a,i0,a)') "Each image has ",icells, " cells"
  write (*,'(a,i0,a)') "The model has ", mcells, " cells"
  write (*,'(a,i0,a)') "The model has ", nuc, " nuclei"
  write (*,'(a,es9.2,a)') "Each image will use at least ", &
    image_storage, " GB memory"
end if

! initialise random number seed
call cgca_irs(nodebug)

! allocate coarray
call cgca_as(l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3,2,space)

! initialise coarrays: a single grain and no damage
```

```
space(:,:,:,cgca_state_type_grain) = 1
space(:,:,:,cgca_state_type_frac) = cgca_intact_state

! set a single crack nucleus somewhere in the middle of image1 array
space(u1/2,u2/2,u3/2,cgca_state_type_frac)[col1,col2,col3] = &
 cgca_clvg_state_100_edge

! Allocate grain and fracture volumes
call cgca_av(1,nuc,col1,cou1,col2,cou2,col3,grainvol)
call cgca_av(1,nuc,col1,cou1,col2,cou2,col3,fracvol)

! allocate rotation tensors
call cgca_art(1,nuc,col1,cou1,col2,cou2,col3,grt)

! set the stress tensor
t = 0.0
t(1,1) = 1.0e6
t(2,2) = -1.0e6

! assign rotation tensors, sync all inside
call cgca_rt(grt)

! propagate cleavage, sync inside
! subroutine cgca_clvgp_nocosum( coarray, rt, t, scrit, sub,
!    gcus, periodicbc, iter, heartbeat, debug )
call cgca_clvgp_nocosum( space, grt, t, scrit, cgca_clvgsp,            &
     cgca_gcupdn, noperiodicbc, 2, 1, yesdebug )

! dump the array
call cgca_swci(space,cgca_state_type_frac,10,"zf.raw")
sync all

! deallocate all arrays
call cgca_ds(space)
call cgca_dv(grainvol)
call cgca_dv(fracvol)
call cgca_drt(grt)

end program testAAW
```

98 tests/testAAX

[Unit tests]

NAME

testAAX

SYNOPSIS

```
!$Id: testAAX.f90 380 2017-03-22 11:03:09Z mexas $

program testAAX
```

PURPOSE

Checking: cgca_clvgp (26.17.1) (no CO_SUM version), cgca_clvgsp (26.7)

DESCRIPTION

Checking probabilistic cleavage propagation with 1 cleavage nucleus in the middle of image 1. Single grain with random orientation. Multiple cleavage increments. Halo exchange, so cleavage propagation across image boundary. Check crack flank deactivation. Check formation of multiple cleavage planes.

NOTES

The program must be called with 2 command line arguments, both positive integers. These are codimensions along 1 and 2. The number of images must be such that codimension3 = num_images()/(codimension1 * codimension3) is a positive integer. Example:

```
      cafrun -np 16 ./testAAX.x 2 2      ! OpenCoarrays
```

or

```
      ./testAAX.x 2 2                    ! Intel, Cray
```

which will make the third codimension equal to 16/(2*2)=4.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

cgca testaux (133)

USED BY

Part of CGPACK test suite

SOURCE

```
use testaux

implicit none
```

```
 logical(kind=ldef),parameter :: yesdebug=.true., nodebug=.false., &
   periodicbc=.true., noperiodicbc=.false.
 real,parameter :: gigabyte=real(2**30), resolution=1.0e-5, &

! cleavage stress on 100, 110, 111 planes for BCC,
! see the manual for derivation.
  scrit(3) = (/ 1.05e4, 1.25e4, 4.90e4 /)

 real(kind=rdef),allocatable :: grt(:,:,:)[:,:,:]
 real(kind=rdef) :: t(3,3)    ! stress tensor

 integer(kind=idef) :: l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3, &
   cou3,   &
   nuc,    & ! number of nuclei in the model
   nimages,codim(3)[*]
 integer(kind=iarr),allocatable :: space(:,:,:,:)[:,:,:]
 integer(kind=ilrg),allocatable :: grainvol(:)[:,:,:],fracvol(:)[:,:,:]
 integer(kind=ilrg) :: icells,mcells

 logical(kind=ldef) :: image1

 real :: image_storage

!*********************************************************************72
! first executable statement

nimages=num_images()
image1=.false.
if (this_image().eq.1) image1=.true.

! do a check on image 1
if (image1) then
 call getcodim(nimages,codim)
 ! print a banner
 call banner("AAX")
 ! print the parameter values
 call cgca_pdmp
 write (*,'(a,i0,a)') "Running on ", nimages, " images in a 3D grid."
end if

sync all

codim(:) = codim(:)[1]

l1=1
l2=l1
l3=l1

! The array size is only controlled by this value
! in this program.
u1=50
```

```
u2=u1
u3=u1

col1=1
cou1=codim(1)-col1+1
col2=1
cou2=codim(2)-col2+1
col3=1
cou3=codim(3)-col3+1

! total number of cells in a coarray
icells = int(u1-l1+1,kind=ilrg) * int(u2-l2+1,kind=ilrg) * &
  int(u3-l3+1,kind=ilrg)

! total number of cells in the model
mcells = icells * int(codim(1),kind=ilrg) * int(codim(2),kind=ilrg) * &
  int(codim(3),kind=ilrg)

! total number of nuclei
!nuc = resolution*mcells
! just 1 nuclei in this test
nuc=1

if (image1) then
  write (*,'(a,2(i0,":",i0,","),i0,":",i0,")")') &
    "bounds: (",l1,u1,l2,u2,l3,u3
  write (*,'(a,2(i0,":",i0,","),i0,":",i0,")")') &
    "cobounds: (",col1,cou1,col2,cou2,col3,cou3

  ! An absolute minimum of storage, in GB, per image.
  image_storage = real( &
    ! 2nd space array is allocated in _sld
    2 * icells*storage_size(space) &
    ! grain volumes
    + nuc*storage_size(grainvol) &
    ! rotation tensors
    + nuc*9*storage_size(grt))/8/gigabyte

  write (*,'(a,i0,a)') "Each image has ",icells, " cells"
  write (*,'(a,i0,a)') "The model has ", mcells, " cells"
  write (*,'(a,i0,a)') "The model has ", nuc, " nuclei"
  write (*,'(a,es9.2,a)') "Each image will use at least ", &
    image_storage, " GB memory"
end if

! initialise random number seed
call cgca_irs(nodebug)

! allocate coarray
call cgca_as(l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3,2,space)

! initialise coarrays: a single grain and no damage
```

```
space(:,:,:,cgca_state_type_grain) = 1
space(:,:,:,cgca_state_type_frac) = cgca_intact_state

! set a single crack nucleus somewhere in the middle of image1 array
space(u1/2,u2/2,u3/2,cgca_state_type_frac)[col1,col2,col3] = &
 cgca_clvg_state_100_edge

! Allocate grain and fracture volumes
call cgca_av(1,nuc,col1,cou1,col2,cou2,col3,grainvol)
call cgca_av(1,nuc,col1,cou1,col2,cou2,col3,fracvol)

! allocate rotation tensors
call cgca_art(1,nuc,col1,cou1,col2,cou2,col3,grt)

! set the stress tensor
t = 0.0
t(1,1) = 1.0e6
t(2,2) = -1.0e6

! assign rotation tensors, sync all inside
call cgca_rt(grt)

! propagate cleavage, sync inside
! subroutine cgca_clvgp_nocosum( coarray, rt, t, scrit, sub,
!    gcus, periodicbc, iter, heartbeat, debug )
call cgca_clvgp_nocosum( space, grt, t, scrit, cgca_clvgsp,            &
     cgca_gcupdn, noperiodicbc, 70, 10, nodebug )

! dump the array
call cgca_swci(space,cgca_state_type_frac,10,"zf.raw")
sync all

! calculate volumes, sync all inside
call cgca_gv(space,grainvol)

! dump grain volumes
if (image1) write (*,"(i0)") grainvol

! deallocate all arrays
call cgca_ds(space)
call cgca_dv(grainvol)
call cgca_dv(fracvol)
call cgca_drt(grt)

end program testAAX
```

99 tests/testAAY

[Unit tests]

NAME

testAAY

SYNOPSIS

```
!$Id: testAAY.f90 380 2017-03-22 11:03:09Z mexas $

program testAAY
```

PURPOSE

Checking: cgca_gcu (11.9), cgca_gcp (11.6)

DESCRIPTION

Checking the grain connectivity array routines: updating (recalculation really) and printing (dump on stdout).

NOTES

The program must be called with 2 command line arguments, both positive integers. These are codimensions along 1 and 2. The number of images must be such that codimension3 = num_images()/(codimension1 * codimension3) is a positive integer. Example:

```
      cafrun -np 16 ./testAAY.x 2 2      ! OpenCoarrays
```

or

```
      ./testAAY.x 2 2                    ! Intel, Cray
```

which will make the third codimension equal to 16/(2*2)=4.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

cgca testaux (133)

USED BY

Part of CGPACK test suite

SOURCE

```
use testaux

implicit none
```

```
real,parameter :: gigabyte=real(2**30), resolution=1.0e-5
logical(kind=ldef),parameter :: yesdebug=.true., nodebug=.false., &
  periodicbc=.true.,  noperiodicbc=.false.

integer(kind=idef) :: l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3, &
 cou3, nuc,nimages,codim(3)[*]
integer(kind=iarr),allocatable :: space(:,:,:,:)[:,:,:]
integer(kind=ilrg) :: icells,mcells

real :: image_storage

logical(kind=ldef) :: solid,image1
character(6) :: image

!*********************************************************************72
! first executable statement

nimages = num_images()
 image1 = .false.
if (this_image() .eq. 1) image1 = .true.

! do a check on image 1
if (image1) then
 call getcodim(nimages,codim)
 ! print a banner
 call banner("AAY")
 ! print the parameter values
 call cgca_pdmp
 write (*,'(a,i0,a)') "running on ", nimages, " images in a 3D grid"
 write (*,*) "codim:", codim
end if

sync all

codim(:) = codim(:)[1]

l1 = 1
l2 = l1
l3 = l1

u1 = 2**6 ! 64
u2 = u1
u3 = u1

col1 = 1
cou1 = codim(1)-col1+1
col2 = 1
cou2 = codim(2)-col2+1
col3 = 1
cou3 = codim(3)-col3+1

! total number of cells in a coarray
```

```
icells = int(u1-l1+1,kind=ilrg) * int(u2-l2+1,kind=ilrg) * &
  int(u3-l3+1,kind=ilrg)

! total number of cells in the model
mcells = icells * int(codim(1),kind=ilrg)  &
                * int(codim(2),kind=ilrg)  &
                * int(codim(3),kind=ilrg)

! total number of nuclei
nuc = 3 ! int( resolution * mcells )

100 format( a,3(i0,":",i0,a) )

if (image1) then
  write (*,100) "bounds: (", l1,u1,",", l2,u2,",", l3,u3, ")"
  write (*,100) "cobounds: [", &
    col1,cou1,",", col2,cou2,",", col3,cou3, "]"

  ! An absolute minimum of storage, in GB, per image.
  ! A factor of 2 is used because will call _sld, which
  ! allocates another array of the same size and kind as
  ! coarray.
  image_storage = real(2 * icells*storage_size(space)/8)/gigabyte

  write (*,'(a,i0,a)') "Each image has ",icells, " cells"
  write (*,'(a,i0,a)') "The model has ", mcells, " cells"
  write (*,'(a,i0,a)') "The model has ", nuc, " nuclei"
  write (*,'(a,es9.2,a)') "Each image will use at least ", &
    image_storage, " GB memory"
end if

! allocate space
call cgca_as(l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3,1,space)

! initialise random number seed
call cgca_irs(nodebug)

! initialise coarray to liquid
space(:,:,:,cgca_state_type_grain) = cgca_liquid_state

sync all

! nuclei, sync all inside
call cgca_nr(space,nuc,yesdebug)

! solidify, implicit sync all inside
!subroutine cgca_sld(coarray,periodicbc,iter,heartbeat,solid)
call cgca_sld(space,noperiodicbc,0,10,solid)

! dump space array to file
call cgca_swci(space,cgca_state_type_grain,10,"zg1.raw")
sync all
```

```
! update grain connectivity
call cgca_gcu(space)
sync all

! dump grain connectivity to files
write (image,"(i0)") this_image()
call cgca_gcp(ounit=10,fname="z_gc_"//image)

! deallocate all arrays
call cgca_ds(space)
call cgca_dgc

end program testAAY
```

100 tests/testAAZ

[Unit tests]

NAME

testAAZ

SYNOPSIS

```
!$Id: testAAZ.f90 526 2018-03-25 23:44:51Z mexas $

program testAAZ
```

PURPOSE

Checking: cgca_gcf (11.4), cgca_gcr (11.7)

DESCRIPTION

Checking the grain connectivity boundary status.

NOTES

The program must be called with 2 command line arguments, both positive integers. These are codimensions along 1 and 2. The number of images must be such that codimension3 = num_images()/(codimension1 * codimension3) is a positive integer. Example:

```
      cafrun -np 16 ./testAAZ.x 2 2      ! OpenCoarrays
```

or

```
      ./testAAZ.x 2 2                    ! Intel, Cray
```

which will make the third codimension equal to 16/(2*2)=4.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

cgca testaux (133)

USED BY

Part of CGPACK test suite

SOURCE

```
use testaux

implicit none

real,parameter :: gigabyte=real(2**30), resolution=1.0e-5
```

```
logical(kind=ldef),parameter :: yesdebug=.true., nodebug=.false., &
  periodicbc=.true.,  noperiodicbc=.false.

integer(kind=idef) :: l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3, &
 cou3, nuc,nimages,codim(3)[*]
integer(kind=iarr),allocatable :: space(:,:,:,:)[:,:,:]
integer(kind=ilrg) :: icells,mcells

real :: image_storage

logical(kind=ldef) :: solid,image1, intact
character(6) :: image

!*********************************************************************72
! first executable statement

nimages = num_images()
 image1 = .false.
if (this_image() .eq. 1) image1 = .true.

! do a check on image 1
if (image1) then
 call getcodim(nimages,codim)
 ! print a banner
 call banner("AAZ")
 ! print the parameter values
 call cgca_pdmp
 write (*,'(a,i0,a)') "running on ", nimages, " images in a 3D grid"
end if

sync all

codim(:) = codim(:)[1]

l1 = 1
l2 = l1
l3 = l1

u1 = 50
u2 = u1
u3 = u1

col1 = 1
cou1 = codim(1)-col1+1
col2 = 1
cou2 = codim(2)-col2+1
col3 = 1
cou3 = codim(3)-col3+1

! total number of cells in a coarray
icells = int(u1-l1+1,kind=ilrg) * int(u2-l2+1,kind=ilrg) * &
  int(u3-l3+1,kind=ilrg)
```

```
! total number of cells in the model
mcells = icells * int(codim(1),kind=ilrg) * int(codim(2),kind=ilrg) * &
  int(codim(3),kind=ilrg)

! total number of nuclei
nuc = int( resolution * real( mcells ) )

100 format( a,3(i0,":",i0,a) )

if (image1) then
  write (*,100) "bounds: (", l1,u1,",", l2,u2,",", l3,u3, ")"
  write (*,100) "cobounds: [", &
    col1,cou1,",", col2,cou2,",", col3,cou3, "]"

  ! An absolute minimum of storage, in GB, per image.
  ! A factor of 2 is used because will call _sld, which
  ! allocates another array of the same size and kind as
  ! coarray.
  image_storage = real(2 * icells*storage_size(space)/8)/gigabyte

  write (*,'(a,i0,a)') "Each image has ",icells, " cells"
  write (*,'(a,i0,a)') "The model has ", mcells, " cells"
  write (*,'(a,i0,a)') "The model has ", nuc, " nuclei"
  write (*,'(a,es9.2,a)') "Each image will use at least ", &
    image_storage, " GB memory"
end if

! allocate space
call cgca_as(l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3,1,space)

! initialise random number seed
call cgca_irs(nodebug)

! initialise the grain coarray to liquid
space(:,:,:,cgca_state_type_grain) = cgca_liquid_state

sync all

! nuclei, sync all inside
call cgca_nr(space,nuc,yesdebug)

! solidify, implicit sync all inside
call cgca_sld(space,noperiodicbc,0,1,solid)

! update grain connectivity
call cgca_gcu(space)
sync all

! mark gb between two grains as intact
! read the value
call cgca_gcr( 3_iarr, 2_iarr, intact )
```

```
if (.not. intact) write (*,*) "image", this_image(), intact
! set the value
call cgca_gcf( 3_iarr, 2_iarr )
! read the value
call cgca_gcr( 3_iarr, 2_iarr, intact)
if (.not. intact) write (*,*) "image", this_image(), intact
sync all

! dump grain connectivity to files
write (image,"(i0)") this_image()
call cgca_gcp(ounit=10,fname="cgca_gcp_out"//image)

! dump space array to file
call cgca_swci(space,cgca_state_type_grain,10,"z9end.raw")
sync all

! deallocate all arrays
call cgca_ds(space)
call cgca_dgc

end program testAAZ
```

101 tests/testABA

[Unit tests]

NAME

testABA

SYNOPSIS

```
!$Id: testABA.f90 380 2017-03-22 11:03:09Z mexas $

program testABA
```

PURPOSE

Checking: cgca_clvgp (26.17.1) (no CO_SUM version) , cgca_m2gb (11), cgca_clvgsd (26.5)

DESCRIPTION

Checking cleavage propagation across grain boundary with two grains, i.e. a single grain boundary.

With no grain boundary smoothing, crack finds it very hard to propagate across a grain boundary. This is because the GB is locally very irregular, and it is likely the first cell in the new grain will find itself in some sort of corner or a tunnel, from where it cannot see enough neighbours of the same grains to propagate into. Hence use cgca_gbs (11.3), at least once, possibly multiple iterations.

NOTES

The program must be called with 2 command line arguments, both positive integers. These are codimensions along 1 and 2. The number of images must be such that codimension3 = num_images()/(codimension1 * codimension3) is a positive integer. Example:

```
      cafrun -np 16 ./testABA.x 2 2      ! OpenCoarrays
```

or

```
      ./testABA.x 2 2                    ! Intel, Cray
```

which will make the third codimension equal to 16/(2*2)=4.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

cgca testaux (133)

USED BY

Part of CGPACK test suite

SOURCE

```
use testaux
```

```
implicit none

logical( kind=ldef ), parameter :: yesdebug=.true., nodebug=.false., &
  periodicbc=.true.,  noperiodicbc=.false.

real,parameter :: gigabyte=real(2**30), resolution=1.0e-5, &
! cleavage stress on 100, 110, 111 planes for BCC,
! see the manual for derivation.
  scrit(3) = (/ 1.05e4, 1.25e4, 4.90e4 /)

real( kind=rdef ), allocatable :: grt(:,:,:)[:,:,:]
real( kind=rdef ) :: t(3,3)   ! stress tensor

integer( kind=idef ) :: l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3, &
                        cou3, nuc, nimages, codim(3)[*], i
integer( kind=iarr ), allocatable :: space(:,:,:,:)[:,:,:]
integer( kind=ilrg ) :: icells,mcells

real :: image_storage

logical( kind=ldef ) :: solid, image1=.false.
character(6) :: image

!*********************************************************************72
! first executable statement

nimages = num_images()
 image1 = .false.
if (this_image() .eq. 1) image1 = .true.

! do a check on image 1
if (image1) then
 call getcodim(nimages,codim)
 ! print a banner
 call banner("ABA")
 ! print the parameter values
 call cgca_pdmp
 write (*,'(a,i0,a)') "running on ", nimages, " images in a 3D grid"
end if

sync all

codim(:) = codim(:)[1]

l1 = 1
l2 = l1
l3 = l1

u1 = 2**6 ! 64
u2 = u1
u3 = u1
```

```
col1 = 1
cou1 = codim(1)-col1+1
col2 = 1
cou2 = codim(2)-col2+1
col3 = 1
cou3 = codim(3)-col3+1

! total number of cells in a coarray
icells = int( u1-l1+1, kind=ilrg ) * &
         int( u2-l2+1, kind=ilrg ) * &
         int( u3-l3+1, kind=ilrg )

! total number of cells in the model
mcells = icells * int( codim(1), kind=ilrg ) * &
                  int( codim(2), kind=ilrg ) * &
                  int( codim(3), kind=ilrg )

! total number of nuclei
!nuc = resolution*mcells
nuc = 2

100 format( a,3(i0,":",i0,a) )

if (image1) then
  write (*,100) "bounds: (", l1,u1,",", l2,u2,",", l3,u3, ")"
  write (*,100) "cobounds: [", &
    col1,cou1,",", col2,cou2,",", col3,cou3, "]"

  ! An absolute minimum of storage, in GB, per image.
  ! A factor of 2 is used because will call _sld, which
  ! allocates another array of the same size and kind as
  ! coarray.
  image_storage = real(2 * icells*storage_size(space)/8)/gigabyte

  write (*,'(a,i0,a)') "Each image has ",icells, " cells"
  write (*,'(a,i0,a)') "The model has ", mcells, " cells"
  write (*,'(a,i0,a)') "The model has ", nuc, " nuclei"
  write (*,'(a,es9.2,a)') "Each image will use at least ", &
    image_storage, " GB memory"
end if

! initialise random number seed
call cgca_irs(nodebug)

! allocate space with two layers
call cgca_as(l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3,2,space)

! allocate rotation tensors
call cgca_art(1,nuc,col1,cou1,col2,cou2,col3,grt)

! initialise space
```

```
space(:,:,:,cgca_state_type_grain) = cgca_liquid_state
space(:,:,:,cgca_state_type_frac) = cgca_intact_state

! nuclei, sync all inside
call cgca_nr(space,nuc,yesdebug)

! assign rotation tensors, sync all inside
call cgca_rt(grt)

! dump the rotation tensors to stdout
if (image1) then
 do i=1,nuc
  write (*,*) "grain=", i, grt(i,:,:)
 end do
end if

! set a single crack nucleus somewhere in the middle of image1 array
space( u1/2, u2/2, u3/2, cgca_state_type_frac)[col1,col2,col3] = &
  cgca_clvg_state_100_edge

! solidify, implicit sync all inside
call cgca_sld(space,noperiodicbc,0,10,solid)

! smoothen the GB, single iteration, sync needed
call cgca_gbs( space )
sync all

! halo exchange, following smoothing
call cgca_hxi( space )
sync all

! update grain connectivity, local routine, no sync needed
call cgca_gcu(space)

! dump grain connectivity to files, local routine, no sync needed
write (image,"(i0)") this_image()
call cgca_gcp(ounit=10,fname="z_gc_1_"//image)

if (image1) write (*,*) "dumping model to files"

! dump space arrays to files, only image 1 does it, all others
! wait at the barrier, hence sync needed
call cgca_swci( space, cgca_state_type_grain, 10, "zg1.raw" )
call cgca_swci( space, cgca_state_type_frac,  10, "zf1.raw" )

if (image1) write (*,*) "finished dumping model to files"

sync all

! set the stress tensor
t = 0.0
t(1,1) = 1.0e6
```

```
t(2,2) = -1.0e6

! propagate cleavage, sync inside
! subroutine cgca_clvgp_nocosum( coarray, rt, t, scrit, sub,
!    gcus, periodicbc, iter, heartbeat, debug )
call cgca_clvgp_nocosum( space, grt, t, scrit, cgca_clvgsd,            &
     cgca_gcupdn, noperiodicbc, 200, 10, yesdebug )

! dump grain connectivity to files, local routine, no sync needed.
write (image,"(i0)") this_image()
call cgca_gcp( ounit=10, fname="z_gc_2_"//image )

! dump the fracture space array to files, only image 1 does it,
! all others wait at the barrier, hence sync needed
call cgca_swci( space, cgca_state_type_frac, 10, "zf2.raw" )

! However, since there's nothing more to do, no sync is needed.

! deallocate all arrays, implicit sync all.
call cgca_ds(space)
call cgca_dgc
call cgca_drt(grt)

end program testABA
```

102 tests/testABB

[Unit tests]

NAME

testABB

SYNOPSIS

```
!$Id: testABB.f90 380 2017-03-22 11:03:09Z mexas $

program testABB
```

PURPOSE

Checking: cgca_clvgp (26.17.1) (no CO_SUM version), cgca_clvgsd (26.5)

DESCRIPTION

Checking cleavage propagation across grain boundary with many grains. Still a single cleavage nucleus. Put it at the centre of one of the faces of the model.

With no grain boundary smoothing, crack finds it very hard to propagate across a grain boundary. This is because the GB is locally very irregular, and it is likely the first cell in the new grain will find itself in some sort of corner or a tunnel, from where it cannot see enough neighbours of the same grains to propagate into. Hence use cgca_gbs (11.3), at least once, possibly multiple iterations.

NOTES

The program must be called with 2 command line arguments, both positive integers. These are codimensions along 1 and 2. The number of images must be such that codimension3 = num_images()/(codimension1 * codimension3) is a positive integer. Example:

```
cafrun -np 16 ./testABB.x 2 2      ! OpenCoarrays
```

or

```
./testABB.x 2 2                    ! Intel, Cray
```

which will make the third codimension equal to 16/(2*2)=4.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

cgca testaux (133)

USED BY

Part of CGPACK test suite

SOURCE

```
use testaux

implicit none

logical( kind=ldef ), parameter :: yesdebug=.true., nodebug=.false., &
 periodicbc=.true.,  noperiodicbc=.false.

real, parameter :: gigabyte=real(2**30), resolution=1.0e-5,          &
! cleavage stress on 100, 110, 111 planes for BCC,
! see the manual for derivation.
 scrit(3) = (/ 1.05e4, 1.25e4, 4.90e4 /)

real( kind=rdef ), allocatable :: grt(:,:,:)[:,:,:]
real( kind=rdef ) :: t(3,3)   ! stress tensor

integer( kind=idef ) :: l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3,  &
 cou3, nuc, nimages, codim(3)[*], iter
integer( kind=iarr ), allocatable :: space(:,:,:,:)[:,:,:]
integer( kind=ilrg ) :: icells,mcells

real :: image_storage

logical( kind=ldef ) :: solid, image1
character(6) :: image

!*********************************************************************72
! first executable statement

nimages = num_images()
 image1 = .false.
if (this_image() .eq. 1) image1 = .true.

! do a check on image 1
if (image1) then
 call getcodim(nimages,codim)
 ! print a banner
 call banner("ABB")
 ! print the parameter values
 call cgca_pdmp
 write (*,'(a,i0,a)') "running on ", nimages, " images in a 3D grid"
end if

sync all

codim(:) = codim(:)[1]

l1 = 1
l2 = l1
l3 = l1

u1 = 2**5 ! 32
u2 = u1
```

```
u3 = u1

col1 = 1
cou1 = codim(1) - col1 + 1
col2 = 1
cou2 = codim(2) - col2 + 1
col3 = 1
cou3 = codim(3) - col3 + 1

! total number of cells in a coarray
icells = int( u1-l1+1, kind=ilrg ) *    &
         int( u2-l2+1, kind=ilrg ) *    &
         int( u3-l3+1, kind=ilrg )

! total number of cells in the model
mcells = icells * int( nimages, kind=ilrg )

! total number of nuclei
! nuc should not exceed resolution*mcells
nuc = 20

100 format( a,3(i0,":",i0,a) )

if (image1) then
  write (*,100) "bounds: (", l1,u1,",", l2,u2,",", l3,u3, ")"
  write (*,100) "cobounds: [", &
    col1,cou1,",", col2,cou2,",", col3,cou3, "]"

  ! An absolute minimum of storage, in GB, per image.
  ! A factor of 2 is used because will call _sld, which
  ! allocates another array of the same size and kind as
  ! coarray.
  image_storage = real(2 * icells*storage_size(space)/8)/gigabyte

  write (*,'(a,i0,a)') "Each image has ",icells, " cells"
  write (*,'(a,i0,a)') "The model has ", mcells, " cells"
  write (*,'(a,i0,a)') "The model has ", nuc, " nuclei"
  write (*,'(a,es9.2,a)') "Each image will use at least ",       &
                          image_storage, " GB memory"
end if

! initialise random number seed
call cgca_irs( nodebug )

! allocate space with two layers
call cgca_as( l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3,2,space )

! allocate rotation tensors
call cgca_art( 1, nuc, col1, cou1, col2, cou2, col3, grt )

! initialise space
space( :, :, :, cgca_state_type_grain ) = cgca_liquid_state
```

```
space( :, :, :, cgca_state_type_frac  ) = cgca_intact_state

! nuclei, sync all inside
call cgca_nr( space, nuc, yesdebug )

! assign rotation tensors, sync all inside
call cgca_rt( grt )

! set a single crack nucleus in the centre of the front face
space( u1/2, u2/2, u3, cgca_state_type_frac )[ cou1/2, cou2/2, cou3] = &
 cgca_clvg_state_100_edge

! solidify, implicit sync all inside
call cgca_sld( space, noperiodicbc, 0, 10, solid )

! smoothen the GB, several iterations, sync needed
do iter=1,2
  call cgca_gbs( space )
  sync all

  ! halo exchange, following smoothing
  call cgca_hxi( space )
  sync all
end do

! update grain connectivity, local routine, no sync needed
call cgca_gcu( space )

! dump grain connectivity to files, local routine, no sync needed
write ( image, "(i0)" ) this_image()
call cgca_gcp( ounit=10, fname="z_gc_1_"//image )

if ( image1 ) write (*,*) "dumping model to files"

! dump space arrays to files, only image 1 does it, all others
! wait at the barrier, hence sync needed
call cgca_swci( space, cgca_state_type_grain, 10, "zg1.raw" )
call cgca_swci( space, cgca_state_type_frac,  10, "zf1.raw" )

if ( image1 ) write (*,*) "finished dumping model to files"

sync all

! set the stress tensor
t(1,1) = 1.0e6
t(2,2) = -1.0e6

! propagate cleavage, sync inside
! subroutine cgca_clvgp_nocosum( coarray, rt, t, scrit, sub,
!    gcus, periodicbc, iter, heartbeat, debug )
call cgca_clvgp_nocosum( space, grt, t, scrit, cgca_clvgsd,            &
     cgca_gcupdn, noperiodicbc, 30, 10, yesdebug )
```

```
! dump grain connectivity to files, local routine, no sync needed.
write ( image, "(i0)" ) this_image()
call cgca_gcp( ounit=10, fname="z_gc_2_"//image )

! dump the fracture space array to files, only image 1 does it,
! all others wait at the barrier, hence sync needed
call cgca_swci( space, cgca_state_type_frac, 10, "zf2.raw" )

! However, since there's nothing more to do, no sync is needed.

! deallocate all arrays, implicit sync all.
call cgca_ds( space )
call cgca_dgc
call cgca_drt( grt )

end program testABB
```

103 tests/testABC

[Unit tests]

NAME

testABC

SYNOPSIS

```
!$Id: testABC.f90 380 2017-03-22 11:03:09Z mexas $

program testABC
```

PURPOSE

Checking: cgca_pdmp (20.1)

DESCRIPTION

Dump the global CGPACK parameters to stdout. Any or all images can call this routine. Here only image 1 does it.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

cgca testaux (133)

USED BY

Part of CGPACK test suite

SOURCE

```
use testaux
implicit none

! only image 1 works

if ( this_image() .eq. 1) then
 call banner("ABC") ! print a banner
 call cgca_pdmp     ! print parameter values
end if

end program testABC
```

104 tests/testABD

[Unit tests]

NAME

testABD

SYNOPSIS

```
!$Id: testABD.f90 380 2017-03-22 11:03:09Z mexas $

program testABD
```

PURPOSE

Checking: cgca_gbf1f (28.1), cgca_clvgp_nocosum (26.4) (no CO_SUM version), cgca_clvgsd (26.5)

DESCRIPTION

Checking grain boundary fracture propagation for a single iteration with fixed model BC.

NOTES

The program must be called with 2 command line arguments, both positive integers. These are codimensions along 1 and 2. The number of images must be such that codimension3 = num_images()/(codimension1 * codimension3) is a positive integer. Example:

```
      cafrun -np 16 ./testABD.x 2 2      ! OpenCoarrays
```

or

```
      ./testABD.x 2 2                    ! Intel, Cray
```

which will make the third codimension equal to 16/(2*2)=4.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

cgca testaux (133)

USED BY

Part of CGPACK test suite

SOURCE

```
use testaux

implicit none

logical(kind=ldef),parameter :: yesdebug=.true., nodebug=.false., &
```

```
  periodicbc=.true.,  noperiodicbc=.false.

real,parameter :: gigabyte=real(2**30), resolution=1.0e-5, &
! cleavage stress on 100, 110, 111 planes for BCC,
! see the manual for derivation.
  scrit(3) = (/ 1.05e4, 1.25e4, 4.90e4 /)

 real(kind=rdef),allocatable :: grt(:,:,:)[:,:,:]
 real(kind=rdef) :: t(3,3)   ! stress tensor

integer(kind=idef) :: l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3, &
 cou3, nuc,nimages,codim(3)[*]
integer(kind=iarr),allocatable :: space(:,:,:,:)[:,:,:]
integer(kind=ilrg) :: icells,mcells

real :: image_storage

logical(kind=ldef) :: solid, image1
character(6) :: image

!*********************************************************************72
! first executable statement

nimages = num_images()
 image1 = .false.
if (this_image() .eq. 1) image1 = .true.

! do a check on image 1
if (image1) then
 call getcodim(nimages,codim)
 ! print a banner
 call banner("ABD")
 ! print the parameter values
 call cgca_pdmp
 write (*,'(a,i0,a)') "Running on ", nimages, " images in a 3D grid."
end if

sync all

codim(:) = codim(:)[1]

l1=1
l2=l1
l3=l1

u1=50
u2=u1
u3=u1

col1=1
cou1=codim(1)-col1+1
col2=1
```

```
cou2=codim(2)-col2+1
col3=1
cou3=codim(3)-col3+1

! total number of cells in a coarray
icells = int(u1-l1+1,kind=ilrg) * int(u2-l2+1,kind=ilrg) * &
  int(u3-l3+1,kind=ilrg)

! total number of cells in the model
mcells = icells * int(codim(1),kind=ilrg) * int(codim(2),kind=ilrg) * &
  int(codim(3),kind=ilrg)

! total number of nuclei
!nuc = resolution*mcells
nuc = 2

if (image1) then
  write (*,'(a,2(i0,":",i0,","),i0,":",i0,")")') &
    "bounds: (",l1,u1,l2,u2,l3,u3
  write (*,'(a,2(i0,":",i0,","),i0,":",i0,")")') &
    "cobounds: (",col1,cou1,col2,cou2,col3,cou3

  ! An absolute minimum of storage, in GB, per image.
  ! A factor of 2 is used because will call _sld, which
  ! allocates another array of the same size and kind as
  ! coarray.
  image_storage = real(2 * icells*storage_size(space)/8)/gigabyte

  write (*,'(a,i0,a)') "Each image has ",icells, " cells"
  write (*,'(a,i0,a)') "The model has ", mcells, " cells"
  write (*,'(a,i0,a)') "The model has ", nuc, " nuclei"
  write (*,'(a,es9.2,a)') "Each image will use at least ", &
    image_storage, " GB memory"
end if

! initialise random number seed
call cgca_irs(nodebug)

! allocate space with two layers
call cgca_as(l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3,2,space)

! allocate rotation tensors
call cgca_art(1,nuc,col1,cou1,col2,cou2,col3,grt)

! initialise space
space(:,:,:,cgca_state_type_grain) = cgca_liquid_state
space(:,:,:,cgca_state_type_frac) = cgca_intact_state

! nuclei, sync all inside
call cgca_nr(space,nuc,yesdebug)

! assign rotation tensors, sync all inside
```

```
call cgca_rt(grt)

! set a single crack nucleus somewhere in the middle of image1 array
space(u1/2,u2/2,u3/2,cgca_state_type_frac)[col1,col2,col3] = &
 cgca_clvg_state_100_edge

! solidify, implicit sync all inside
call cgca_sld(space,noperiodicbc,0,10,solid)

! smoothen the grain boundaries, sync is required
call cgca_gbs(space)
sync all

! update grain connectivity, local routine, no sync needed
call cgca_gcu(space)

! dump grain connectivity to files, local routine, no sync needed
write (image,"(i0)") this_image()
call cgca_gcp(ounit=10,fname="z_gc_1_"//image)

if (image1) write (*,*) "dumping model to files"

! dump space arrays to files, only image 1 does it, all others
! wait at the barrier, hence sync needed
call cgca_swci(space,cgca_state_type_grain,10,"zg1.raw")
call cgca_swci(space,cgca_state_type_frac,10,"zfb.raw")

if (image1) write (*,*) "finished dumping model to files"

sync all

! set the stress tensor
t = 0.0
t(1,1) = 1.0e6
t(2,2) = -1.0e6

! propagate cleavage, sync inside
! subroutine cgca_clvgp_nocosum( coarray, rt, t, scrit, sub,
!    gcus, periodicbc, iter, heartbeat, debug )
call cgca_clvgp_nocosum( space, grt, t, scrit, cgca_clvgsd,            &
     cgca_gcupdn, noperiodicbc, 70, 10, yesdebug )

! dump grain connectivity to files, local routine, no sync needed.
write (image,"(i0)") this_image()
call cgca_gcp(ounit=10,fname="z_gc_2_"//image)

! Do a single iteration of grain boundary fracture
! Calling the GB routine for *fixed* BC
! no sync inside it, so need to sync all after
call cgca_gbf1f(space)
sync all
call cgca_hxi(space)
```

```
sync all

! dump the fracture space array to files, only image 1 does it,
! all others wait at the barrier, hence sync needed
call cgca_swci(space,cgca_state_type_frac,10,"zfe.raw")

! However, since there's nothing more to do, no sync is needed.

! deallocate all arrays, implicit sync all.
call cgca_ds(space)
call cgca_dgc
call cgca_drt(grt)

end program testABD
```

105 tests/testABE

[Unit tests]

NAME

testABE

SYNOPSIS

```
!$Id: testABE.f90 380 2017-03-22 11:03:09Z mexas $

program testABE
```

PURPOSE

Checking: cgca_gbs (11.3)

DESCRIPTION

Checking grain boundary smoothing routine

NOTES

The program must be called with 2 command line arguments, both positive integers. These are codimensions along 1 and 2. The number of images must be such that codimension3 = num_images()/(codimension1 * codimension3) is a positive integer. Example:

```
      cafrun -np 16 ./testABE.x 2 2      ! OpenCoarrays
```

or

```
      ./testABE.x 2 2                    ! Intel, Cray
```

which will make the third codimension equal to 16/(2*2)=4.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

cgca testaux (133)

USED BY

Part of CGPACK test suite

SOURCE

```
use testaux

implicit none

logical(kind=ldef),parameter :: yesdebug=.true., nodebug=.false., &
```

```
  periodicbc=.true.,  noperiodicbc=.false.

real,parameter :: gigabyte=real(2**30), resolution=1.0e-5, &
! cleavage stress on 100, 110, 111 planes for BCC,
! see the manual for derivation.
  scrit(3) = (/ 1.05e4, 1.25e4, 4.90e4 /)

 real(kind=rdef),allocatable :: grt(:,:,:)[:,:,:]

integer(kind=idef) :: l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3, &
 cou3, nuc,nimages,codim(3)[*], i
integer(kind=iarr),allocatable :: space(:,:,:,:)[:,:,:]
integer(kind=ilrg) :: icells,mcells

real :: image_storage

logical(kind=ldef) :: solid, image1

!*********************************************************************72
! first executable statement

nimages = num_images()
 image1 = .false.
if (this_image() .eq. 1) image1 = .true.

! do a check on image 1
if (image1) then
 call getcodim(nimages,codim)
 ! print a banner
 call banner("ABE")
 ! print the parameter values
 call cgca_pdmp
 write (*,'(a,i0,a)') "Running on ", nimages, " images in a 3D grid."
end if

sync all

codim(:) = codim(:)[1]

l1 = 1
l2 = l1
l3 = l1

u1 = 10
u2 = u1
u3 = u1

col1 = 1
cou1 = codim(1)-col1+1
col2 = 1
cou2 = codim(2)-col2+1
col3 = 1
```

```
cou3 = codim(3)-col3+1

! total number of cells in a coarray
icells = int(u1-l1+1,kind=ilrg) * int(u2-l2+1,kind=ilrg) * &
  int(u3-l3+1,kind=ilrg)

! total number of cells in the model
mcells = icells * int(codim(1),kind=ilrg) * int(codim(2),kind=ilrg) * &
  int(codim(3),kind=ilrg)

! total number of nuclei
!nuc = resolution*mcells
nuc = 2

if (image1) then
  write (*,'(a,2(i0,":",i0,","),i0,":",i0,")")') &
    "bounds: (",l1,u1,l2,u2,l3,u3
  write (*,'(a,2(i0,":",i0,","),i0,":",i0,")")') &
    "cobounds: (",col1,cou1,col2,cou2,col3,cou3

  ! An absolute minimum of storage, in GB, per image.
  ! A factor of 2 is used because will call _sld, which
  ! allocates another array of the same size and kind as
  ! coarray.
  image_storage = real(2 * icells*storage_size(space)/8)/gigabyte

  write (*,'(a,i0,a)') "Each image has ",icells, " cells"
  write (*,'(a,i0,a)') "The model has ", mcells, " cells"
  write (*,'(a,i0,a)') "The model has ", nuc, " nuclei"
  write (*,'(a,es9.2,a)') "Each image will use at least ", &
    image_storage, " GB memory"
end if

! initialise random number seed
call cgca_irs(nodebug)

! allocate space with two layers
call cgca_as(l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3,2,space)

! allocate rotation tensors
call cgca_art(1,nuc,col1,cou1,col2,cou2,col3,grt)

! initialise space
space(:,:,:,cgca_state_type_grain) = cgca_liquid_state
space(:,:,:,cgca_state_type_frac ) = cgca_intact_state

! nuclei, sync all inside
call cgca_nr(space,nuc,yesdebug)

! solidify, implicit sync all inside
call cgca_sld(space,noperiodicbc,0,10,solid)
```

```
! dump space grain layer to file
call cgca_swci(space,cgca_state_type_grain,10,"zg0.raw")
sync all

! smooth the grain boundary, many iterations
do i=1,16
call cgca_gbs(space)
sync all

if (image1) write (*,*) "done GB smoothing, iter=",i

if (i .eq. 1) call cgca_swci(space,cgca_state_type_grain,10,"zg1.raw")
if (i .eq. 2) call cgca_swci(space,cgca_state_type_grain,10,"zg2.raw")
if (i .eq. 4) call cgca_swci(space,cgca_state_type_grain,10,"zg4.raw")
if (i .eq. 8) call cgca_swci(space,cgca_state_type_grain,10,"zg8.raw")
! no sync here, because the halo exchange changes
! only *local* coarray values.

! internal halo exchange
call cgca_hxi(space)
sync all
end do

! dump space grain layer to file
! all others wait at the barrier, hence sync needed
call cgca_swci(space,cgca_state_type_grain,10,"zg16.raw")

! However, since there's nothing more to do, no sync is needed.

! deallocate all arrays, implicit sync all.
call cgca_ds(space)

end program testABE
```

106 tests/testABF

[Unit tests]

NAME

testABF

SYNOPSIS

```
!$Id: testABF.f90 380 2017-03-22 11:03:09Z mexas $

program testABF
```

PURPOSE

Checking: cgca_gbf1f (28.1), cgca_cgvgp_nocosum (no CO_SUM version)

DESCRIPTION

Checking grain boundary fracture propagation for a several iterations *after* all cleavage iterations. Fixed model BC used.

NOTES

The program must be called with 2 command line arguments, both positive integers. These are codimensions along 1 and 2. The number of images must be such that codimension3 = num_images()/(codimension1 * codimension3) is a positive integer. Example:

```
      cafrun -np 16 ./testABF.x 2 2      ! OpenCoarrays
```

or

```
      ./testABF.x 2 2                    ! Intel, Cray
```

which will make the third codimension equal to 16/(2*2)=4.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

cgca testaux (133)

USED BY

Part of CGPACK test suite

SOURCE

```
use testaux

implicit none
```

```
logical(kind=ldef),parameter :: yesdebug=.true., nodebug=.false., &
  periodicbc=.true.,  noperiodicbc=.false.

real,parameter :: gigabyte=real(2**30), resolution=1.0e-5, &
! cleavage stress on 100, 110, 111 planes for BCC,
! see the manual for derivation.
  scrit(3) = (/ 1.05e4, 1.25e4, 4.90e4 /)

 real(kind=rdef),allocatable :: grt(:,:,:)[:,:,:]
 real(kind=rdef) :: t(3,3)   ! stress tensor

integer(kind=idef) :: l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3, &
 cou3, nuc,nimages,codim(3)[*], i
integer(kind=iarr),allocatable :: space(:,:,:,:)[:,:,:]
integer(kind=ilrg) :: icells,mcells

real :: image_storage

logical(kind=ldef) :: solid, image1
character(6) :: image

!*********************************************************************72
! first executable statement

nimages=num_images()
image1=.false.
if (this_image() .eq. 1) image1 = .true.

! do a check on image 1
if (image1) then
 call getcodim(nimages,codim)
 ! print a banner
 call banner("ABF")
 ! print the parameter values
 call cgca_pdmp
 write (*,'(a,i0,a)') "Running on ", nimages, " images in a 3D grid."
end if

sync all

codim(:) = codim(:)[1]

l1=1
l2=l1
l3=l1

u1=50
u2=u1
u3=u1

col1=1
cou1=codim(1)-col1+1
```

```
col2=1
cou2=codim(2)-col2+1
col3=1
cou3=codim(3)-col3+1

! total number of cells in a coarray
icells = int(u1-l1+1,kind=ilrg) * int(u2-l2+1,kind=ilrg) * &
  int(u3-l3+1,kind=ilrg)

! total number of cells in the model
mcells = icells * int(codim(1),kind=ilrg) * int(codim(2),kind=ilrg) * &
  int(codim(3),kind=ilrg)

! total number of nuclei
!nuc = resolution*mcells
nuc = 2

if (image1) then
  write (*,'(a,2(i0,":",i0,","),i0,":",i0,")")') &
    "bounds: (",l1,u1,l2,u2,l3,u3
  write (*,'(a,2(i0,":",i0,","),i0,":",i0,")")') &
    "cobounds: (",col1,cou1,col2,cou2,col3,cou3

  ! An absolute minimum of storage, in GB, per image.
  ! A factor of 2 is used because will call _sld, which
  ! allocates another array of the same size and kind as
  ! coarray.
  image_storage = real(2 * icells*storage_size(space)/8)/gigabyte

  write (*,'(a,i0,a)') "Each image has ",icells, " cells"
  write (*,'(a,i0,a)') "The model has ", mcells, " cells"
  write (*,'(a,i0,a)') "The model has ", nuc, " nuclei"
  write (*,'(a,es9.2,a)') "Each image will use at least ", &
    image_storage, " GB memory"
end if

! initialise random number seed
call cgca_irs(nodebug)

! allocate space with two layers
call cgca_as(l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3,2,space)

! allocate rotation tensors
call cgca_art(1,nuc,col1,cou1,col2,cou2,col3,grt)

! initialise space
space(:,:,:,cgca_state_type_grain) = cgca_liquid_state
space(:,:,:,cgca_state_type_frac) = cgca_intact_state

! nuclei, sync all inside
call cgca_nr(space,nuc,yesdebug)
```

```
! assign rotation tensors, sync all inside
call cgca_rt(grt)

! set a single crack nucleus somewhere in the middle of image1 array
space(u1/2,u2/2,u3/2,cgca_state_type_frac)[col1,col2,col3] = &
 cgca_clvg_state_100_edge

! solidify, implicit sync all inside
call cgca_sld(space,noperiodicbc,0,10,solid)

! smoothen the grain boundaries, sync is required
call cgca_gbs(space)
sync all

! update grain connectivity, local routine, no sync needed
call cgca_gcu(space)

! dump grain connectivity to files, local routine, no sync needed
write (image,"(i0)") this_image()
call cgca_gcp(ounit=10,fname="z_gc_1_"//image)

if (image1) write (*,*) "dumping model to files"

! dump space arrays to files, only image 1 does it, all others
! wait at the barrier, hence sync needed
call cgca_swci(space,cgca_state_type_grain,10,"zg1.raw")
call cgca_swci(space,cgca_state_type_frac,10,"zf1.raw")

if (image1) write (*,*) "finished dumping model to files"

sync all

! set the max. principal stress
t = 0.0
t(1,1) = 1.0e6
t(2,2) = -1.0e6

! propagate cleavage, sync inside
! subroutine cgca_clvgp_nocosum( coarray, rt, t, scrit, sub,
!    gcus, periodicbc, iter, heartbeat, debug )
call cgca_clvgp_nocosum( space, grt, t, scrit, cgca_clvgsd,            &
     cgca_gcupdn, noperiodicbc, 70, 10, nodebug )

! dump grain connectivity to files, local routine, no sync needed.
write (image,"(i0)") this_image()
call cgca_gcp(ounit=10,fname="z_gc_2_"//image)

! do several iteration of grain boundary fracture
! no sync inside it, so need to sync all after
do i=1,10
 call cgca_gbf1f(space)
 sync all
```

```
 if (image1) write (*,*) "done ",i," GB fracture iterations"

 call cgca_hxi(space)
 sync all
end do

! dump the fracture space array to files, only image 1 does it,
! all others wait at the barrier, hence sync needed
call cgca_swci(space,cgca_state_type_frac,10,"zf2.raw")

! However, since there's nothing more to do, no sync is needed.

! deallocate all arrays, implicit sync all.
call cgca_ds(space)
call cgca_dgc
call cgca_drt(grt)

end program testABF
```

107 tests/testABG

[Unit tests]

NAME

testABG

SYNOPSIS

```
!$Id: testABG.f90 380 2017-03-22 11:03:09Z mexas $

program testABG
```

PURPOSE

Timing cgca_sld1 (31.2)

DESCRIPTION

cgca_sld1 (31.2) is the second attempt of solidification. It uses SYNC IMAGES to enforce the order of a collective operation. However, like cgca_sld (31.1), it still does it one image at a time. It is more complex to program than cgca_sld (31.1), but hopefully slightly more efficient.

This is timing test only. Use other tests to check the correctness of cgca_sld1 (31.2).

NOTES

The program must be called with 2 command line arguments, both positive integers. These are codimensions along 1 and 2. The number of images must be such that codimension3 = num_images()/(codimension1 * codimension3) is a positive integer. Example:

```
      cafrun -np 16 ./testABG.x 2 2      ! OpenCoarrays
```

or

```
      ./testABG.x 2 2                    ! Intel, Cray
```

which will make the third codimension equal to 16/(2*2)=4.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

cgca testaux (133)

USED BY

Part of CGPACK test suite

SOURCE

```
use testaux
```

```
implicit none

real,parameter :: gigabyte=real(2**30), resolution=1.0e-5
logical(kind=ldef),parameter :: yesdebug = .true., nodebug = .false.

integer(kind=idef) :: l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3,cou3, &
 nuc,    & ! number of nuclei in the model
 nimages, codim(3)[*]
integer(kind=iarr),allocatable :: space(:,:,:,:)[:,:,:]
integer(kind=ilrg) :: icells, mcells, img

logical(kind=ldef) :: solid

real :: image_storage, time1, time2

!*****************************************************************72
! first executable statement

img = this_image()
nimages = num_images()

! do a check on image 1
if ( img .eq. 1 ) then
 call getcodim(nimages,codim)
 ! print a banner
 call banner("ABG")
 ! print the parameter values
 call cgca_pdmp
 write (*,'(a,i0,a)') "running on ", nimages, " images in a 3D grid"
end if

! all images read codim from image 1
if ( img .eq. 1 ) then
 sync images(*)
else
 sync images(1)
 codim(:) = codim(:)[1]
end if

l1=1
l2=l1
l3=l1

! The array size is only controlled by this value
! in this program.
u1=512
u2=u1
u3=u1

col1=1
cou1=codim(1)-col1+1
col2=1
```

```
cou2=codim(2)-col2+1
col3=1
cou3=codim(3)-col3+1

! total number of cells in a coarray
icells = int(u1-l1+1,kind=ilrg) * int(u2-l2+1,kind=ilrg) * &
  int(u3-l3+1,kind=ilrg)

! total number of cells in the model
mcells = icells * int(codim(1),kind=ilrg) * int(codim(2),kind=ilrg) * &
  int(codim(3),kind=ilrg)

! total number of nuclei
nuc = int( resolution * real( mcells ) )

if ( img .eq. 1 ) then
  write (*,'(a,2(i0,":",i0,","),i0,":",i0,")")') &
    "bounds: (",l1,u1,l2,u2,l3,u3
  write (*,'(a,2(i0,":",i0,","),i0,":",i0,")")') &
    "cobounds: (",col1,cou1,col2,cou2,col3,cou3

  ! An absolute minimum of storage, in GB, per image.
  ! A factor of 2 is used because will call _sld, which
  ! allocates another array of the same size and kind as
  ! coarray.
  image_storage = real(2 * icells*storage_size(space)/8)/gigabyte

  write (*,'(a,i0,a)') "Each image has ",icells, " cells"
  write (*,'(a,i0,a)') "The model has ", mcells, " cells"
  write (*,'(a,i0,a)') "The model has ", nuc, " nuclei"
  write (*,'(a,es9.2,a)') "Each image will use at least ", &
    image_storage, " GB memory"
end if

! initialise random number seed
call cgca_irs(nodebug)

! allocate coarray
call cgca_as(l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3,1,space)

! initialise coarray to liquid
space = cgca_liquid_state

! populate nuclei
call cgca_nr(space,nuc,nodebug)

! solidify
call cpu_time(time1)
call cgca_sld1(space,0,1,solid)
call cpu_time(time2)
write (*,*) "img", img, "time, s", time2-time1
```

```
! dump the model
!call cgca_swci(space,cgca_state_type_grain,10,'z.raw')

! deallocate all arrays
call cgca_ds(space)

end program testABG
```

108 tests/testABH

[Unit tests]

NAME

testABH

SYNOPSIS

```
!$Id: testABH.f90 389 2017-03-22 16:31:21Z mexas $

program testABH
```

PURPOSE

Checking: cgca_redand (22.1), part of cgca_m2red (22)

DESCRIPTION

Checking collective AND reduction over a logical coarray. Works only when the number of images is 2**p, where p is an integer, so use 2, 4, 8, 16, 32, etc. images.

NOTES

The program must be called with 2 command line arguments, both positive integers. These are codimensions along 1 and 2. The number of images must be such that codimension3 = num_images()/(codimension1 * codimension3) is a positive integer. Example:

```
      cafrun -np 16 ./testABH.x 2 2      ! OpenCoarrays
```

or

```
      ./testABH.x 2 2                    ! Intel, Cray
```

which will make the third codimension equal to 16/(2*2)=4.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

cgca testaux (133)

USED BY

Part of CGPACK test suite

SOURCE

```
use testaux

implicit none
```

```
real, parameter :: l2 = log(real(2))
logical, parameter :: nodebug = .false.
real :: num
integer(kind=idef) :: p, nimages, img, codim(3)[*]
logical(kind=ldef) :: z[*]

!*********************************************************************72
! first executable statement

nimages=num_images()
img = this_image()

! check than n is a power of 2
p = nint(log(real(nimages))/l2)
if ( 2**p .ne. nimages)                                                 &
      error stop "number of images is not a power of 2"

! do a check on image 1
if (img .eq. 1) then
 call getcodim(nimages,codim)
 ! print a banner
 call banner("ABH")
 write (*,'(a,i0,a)') "running on ", nimages, " images in a 3D grid"
end if

! Trying to separate the output
sync all

! initialise random number seed
call cgca_irs(nodebug)

! assign z
call random_number(num)

if (num .gt. 0.5) then
 z = .true.
else
 z = .false.
end if

z = .true.
if (img .eq. nimages) z = .false.

write (*,*) "image", img, "z", z

! Trying to separate the output
sync all

! call collective AND
call cgca_redand(z,p)

write (*,*) "image", img, "answer", z
```

```
end program testABH
```

109 tests/testABI

[Unit tests]

NAME

testABI

SYNOPSIS

```
!$Id: testABI.f90 380 2017-03-22 11:03:09Z mexas $

program testABI
```

PURPOSE

Timing cgca_sld2 (31.3)

DESCRIPTION

Timing solidification where the check for the complete solidification of the whole model is done with a divide and conquer algorithm, cgca_redall, which does a collective AND reduction over a logical array.

This is timing test only. Use testABH (108) to check the correctness of cgca_sld2 (31.3).

NOTES

The program must be called with 2 command line arguments, both positive integers. These are codimensions along 1 and 2. The number of images must be such that codimension3 = num_images()/(codimension1 * codimension3) is a positive integer. Example:

```
      cafrun -np 16 ./testABI.x 2 2      ! OpenCoarrays
```

or

```
      ./testABI.x 2 2                    ! Intel, Cray
```

which will make the third codimension equal to 16/(2*2)=4.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

cgca testaux (133)

USED BY

Part of CGPACK test suite

SOURCE

```
use testaux

implicit none
```

```
real,parameter :: gigabyte=real(2**30), resolution=1.0e-5, &
 loge2 = log(real(2))
logical(kind=ldef),parameter :: yesdebug = .true., nodebug = .false.

integer(kind=idef) :: l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3,cou3, &
 nuc,    & ! number of nuclei in the model
 nimages, codim(3)[*], p
integer(kind=iarr),allocatable :: space(:,:,:,:)[:,:,:]
integer(kind=ilrg) :: icells, mcells, img

logical(kind=ldef) :: solid

real :: image_storage, time1, time2

!*********************************************************************72
! first executable statement

img = this_image()
nimages = num_images()

! check than nimages is a power of 2
p = nint(log(real(nimages))/loge2)
if ( 2**p .ne. nimages) error stop "number of images is not a power of 2"

! do a check on image 1
if ( img .eq. 1 ) then
 call getcodim(nimages,codim)
 ! print a banner
 call banner("ABI")
 ! print the parameter values
 call cgca_pdmp
 write (*,'(a,i0,a)') "running on ", nimages, " images in a 3D grid"
 ! dump the value of p
 write (*,"(a,i0)") "p=",p
end if

! all images read codim from image 1
if ( img .eq. 1 ) then
 sync images(*)
else
 sync images(1)
 codim(:) = codim(:)[1]
end if

l1=1
l2=l1
l3=l1

! The array size is only controlled by this value
! in this program.
u1=512
```

```
u2=u1
u3=u1

col1=1
cou1=codim(1)-col1+1
col2=1
cou2=codim(2)-col2+1
col3=1
cou3=codim(3)-col3+1

! total number of cells in a coarray
icells = int(u1-l1+1,kind=ilrg) * int(u2-l2+1,kind=ilrg) * &
  int(u3-l3+1,kind=ilrg)

! total number of cells in the model
mcells = icells * int(codim(1),kind=ilrg) * int(codim(2),kind=ilrg) * &
  int(codim(3),kind=ilrg)

! total number of nuclei
nuc = int( resolution * real( mcells ) )

if ( img .eq. 1 ) then
  write (*,'(a,2(i0,":",i0,","),i0,":",i0,")")') &
    "bounds: (",l1,u1,l2,u2,l3,u3
  write (*,'(a,2(i0,":",i0,","),i0,":",i0,")")') &
    "cobounds: (",col1,cou1,col2,cou2,col3,cou3

  ! An absolute minimum of storage, in GB, per image.
  ! A factor of 2 is used because will call _sld, which
  ! allocates another array of the same size and kind as
  ! coarray.
  image_storage = real(2 * icells*storage_size(space)/8)/gigabyte

  write (*,'(a,i0,a)') "Each image has ",icells, " cells"
  write (*,'(a,i0,a)') "The model has ", mcells, " cells"
  write (*,'(a,i0,a)') "The model has ", nuc, " nuclei"
  write (*,'(a,es9.2,a)') "Each image will use at least ", &
    image_storage, " GB memory"
end if

! initialise random number seed
call cgca_irs(nodebug)

! allocate coarray
call cgca_as(l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3,1,space)

! initialise coarray to liquid
space = cgca_liquid_state

! populate nuclei
call cgca_nr(space,nuc,nodebug)
```

```
! solidify
call cpu_time(time1)
call cgca_sld2(space,p,0,1,solid)
call cpu_time(time2)
write (*,*) "img", img, "time, s", time2-time1

! dump the model
!call cgca_swci(space,cgca_state_type_grain,10,'z.raw')

! deallocate all arrays
call cgca_ds(space)

end program testABI
```

110 tests/testABJ

[Unit tests]

NAME

testABJ

SYNOPSIS

```
!$Id: testABJ.f90 380 2017-03-22 11:03:09Z mexas $

program testABJ
```

PURPOSE

Testing and timing dumping the local coarray from each image into its own local binary stream file.

DESCRIPTION

Some postprocessing program (not written yet) should be used to check the integrity of the resulting files, e.g. by reading them all and writing all data into a single file for analysis with Paraview.

The timing of this test should be compared against a test using cgca_swci (19.2).

NOTES

The program must be called with 2 command line arguments, both positive integers. These are codimensions along 1 and 2. The number of images must be such that codimension3 = num_images()/(codimension1 * codimension3) is a positive integer. Example:

```
      cafrun -np 16 ./testABJ.x 2 2      ! OpenCoarrays
```

or

```
      ./testABJ.x 2 2                    ! Intel, Cray
```

which will make the third codimension equal to 16/(2*2)=4.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

cgca testaux (133)

USED BY

Part of CGPACK test suite

SOURCE

```
use testaux

implicit none
```

```
 logical(kind=ldef),parameter :: yesdebug=.true., nodebug=.false., &
   periodicbc=.true.
 real,parameter :: gigabyte=real(2**30), resolution=1.0e-5

 integer(kind=idef) :: l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3, &
   cou3, thisimage, lb(4), ub(4), &
   nuc,    & ! number of nuclei in the model
   nimages,codim(3)[*]
 integer(kind=iarr),allocatable :: space(:,:,:,:)[:,:,:]
 integer(kind=ilrg) :: icells,mcells

 logical(kind=ldef) :: solid,image1=.false.

 real :: image_storage

 character(len=10) :: fnum

!*********************************************************************72
! first executable statement

thisimage = this_image()
nimages=num_images()
image1=.false.
if (this_image().eq.1) image1=.true.

! do a check on image 1
if (image1) then
 call getcodim(nimages,codim)
 ! print a banner
 call banner("ABJ")
 ! print the parameter values
 call cgca_pdmp
 write (*,'(a,i0,a)') "running on ", nimages, " images in a 3D grid"
end if

sync all

codim(:) = codim(:)[1]

!if (image1) call system("sleep 1")

l1=1
l2=l1
l3=l1

! The array size is only controlled by this value
! in this program.
u1=128
u2=u1
u3=u1
```

```
col1=1
cou1=codim(1)-col1+1
col2=1
cou2=codim(2)-col2+1
col3=1
cou3=codim(3)-col3+1

! total number of cells in a coarray
icells = int(u1-l1+1,kind=ilrg) * int(u2-l2+1,kind=ilrg) * &
  int(u3-l3+1,kind=ilrg)

! total number of cells in the model
mcells = icells * int(codim(1),kind=ilrg) * int(codim(2),kind=ilrg) * &
  int(codim(3),kind=ilrg)

! total number of nuclei
nuc = int( resolution * real( mcells ) )

if (image1) then
  write (*,'(a,2(i0,":",i0,","),i0,":",i0,")")') &
    "bounds: (",l1,u1,l2,u2,l3,u3
  write (*,'(a,2(i0,":",i0,","),i0,":",i0,")")') &
    "cobounds: (",col1,cou1,col2,cou2,col3,cou3

  ! An absolute minimum of storage, in GB, per image.
  ! A factor of 2 is used because will call _sld, which
  ! allocates another array of the same size and kind as
  ! coarray.
  image_storage = real(2 * icells*storage_size(space)/8)/gigabyte

  write (*,'(a,i0,a)') "Each image has ",icells, " cells"
  write (*,'(a,i0,a)') "The model has ", mcells, " cells"
  write (*,'(a,i0,a)') "The model has ", nuc, " nuclei"
  write (*,'(a,es9.2,a)') "Each image will use at least ", &
    image_storage, " GB memory"
end if

! initialise random number seed
call cgca_irs(nodebug)

! allocate coarray
call cgca_as(l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3,1,space)

! initialise coarray to liquid
space = cgca_liquid_state

! populate nuclei
call cgca_nr(space,nuc,nodebug)

! solidify, fixed boundaries
call cgca_sld1(space,0,1,solid)
```

```
! dump each local coarray into its own binary stream file

lb=lbound(space)+1
ub=ubound(space)-1

write (fnum,"(i0)") thisimage
open (unit=10, file="out"//fnum, form="unformatted", access="stream", status="replace")
write (10) space(lb(1):ub(1), lb(2):ub(2), lb(3):ub(3), cgca_state_type_grain)
close (10)

! deallocate all arrays
call cgca_ds(space)

end program testABJ
```

111 tests/testABK

[Unit tests]

NAME

testABK

SYNOPSIS

```
!$Id: testABK.f90 380 2017-03-22 11:03:09Z mexas $

program testABK
```

PURPOSE

Checking: cgca_av (10.3), cgca_dv (10.6), cgca_gv (25.2)

DESCRIPTION

Checking and timing grain volume array calculation on Cray. Use with CRAY only!

NOTES

The program must be called with 2 command line arguments, both positive integers. These are codimensions along 1 and 2. The number of images must be such that codimension3 = num_images()/(codimension1 * codimension3) is a positive integer. Example:

```
./testABK.x 2 2
```

which will make the third codimension equal to 16/(2*2)=4.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

cgca testaux (133)

USED BY

Part of CGPACK test suite

SOURCE

```
use testaux

implicit none

 logical(kind=ldef),parameter :: yesdebug=.true., nodebug=.false., &
   periodicbc=.true.
 real,parameter :: gigabyte=real(2**30), resolution=1.0e-5

 integer(kind=idef) :: l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3,cou3, &
   nuc,    & ! number of nuclei in the model
```

```
   nimages, img, codim(3)[*], i
 integer(kind=iarr),allocatable :: space(:,:,:,:)[:,:,:]
 integer(kind=ilrg),allocatable :: grainvol(:)[:,:,:]
 integer(kind=ilrg) :: icells, mcells

 logical(kind=ldef) :: solid

 real :: image_storage

 integer :: pat_status

!*********************************************************************72
! first executable statement

nimages = num_images()
img = this_image()

! do a check on image 1
if ( img .eq. 1 ) then
 call getcodim(nimages,codim)
 ! print a banner
 call banner("ABK")
 ! print the parameter values
 call cgca_pdmp
 write (*,'(a,i0,a)') "running on ", nimages, " images in a 3D grid"
end if

! all images read codim from image 1
if ( img .eq. 1 ) then
 sync images(*)
else
 sync images(1)
 codim(:) = codim(:)[1]
end if

l1=1
l2=l1
l3=l1

! The array size is only controlled by this value
! in this program.
u1=128
u2=u1
u3=u1

col1=1
cou1=codim(1)-col1+1
col2=1
cou2=codim(2)-col2+1
col3=1
cou3=codim(3)-col3+1
```

```
! total number of cells in a coarray
icells = int(u1-l1+1,kind=ilrg) * int(u2-l2+1,kind=ilrg) * &
  int(u3-l3+1,kind=ilrg)

! total number of cells in the model
mcells = icells * int(codim(1),kind=ilrg) * int(codim(2),kind=ilrg) * &
  int(codim(3),kind=ilrg)

! total number of nuclei
nuc = int( resolution * real(mcells) )

if ( img .eq. 1 ) then
  write (*,'(a,2(i0,":",i0,","),i0,":",i0,")")') &
    "bounds: (",l1,u1,l2,u2,l3,u3
  write (*,'(a,2(i0,":",i0,","),i0,":",i0,")")') &
    "cobounds: (",col1,cou1,col2,cou2,col3,cou3

  ! An absolute minimum of storage, in GB, per image.
  ! A factor of 2 is used because will call _sld, which
  ! allocates another array of the same size and kind as
  ! coarray.
  image_storage = real(2 * icells*storage_size(space)/8)/gigabyte

  write (*,'(a,i0,a)') "Each image has ",icells, " cells"
  write (*,'(a,i0,a)') "The model has ", mcells, " cells"
  write (*,'(a,i0,a)') "The model has ", nuc, " nuclei"
  write (*,'(a,es9.2,a)') "Each image will use at least ", &
    image_storage, " GB memory"
end if

! initialise random number seed
call cgca_irs(nodebug)

! allocate space and grain volume coarrays
call cgca_as(l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3,1,space)
call cgca_av(1,nuc,col1,cou1,col2,cou2,col3,grainvol)

! initialise coarray
space = cgca_liquid_state

! populate nuclei
call cgca_nr(space,nuc,nodebug)

! solidify
call cgca_sld(space,periodicbc,0,1,solid)

! calculate volumes with my routine
call pat_region_begin(1,"_gv",pat_status)
call cgca_gv(space,grainvol)
call pat_region_end(1,pat_status)

! dump grain volumes
```

```
if ( img .eq. 1 ) then
 write (*,*) "results from cgca_gv"
 do i=lbound(grainvol,dim=1),ubound(grainvol,dim=1)
  write (*,"(i0,tr1,i0)") i, grainvol(i)
 end do
end if

! now calculate volumes using CO_SUM intrinsic
call pat_region_begin(2,"_gvl+co_sum",pat_status)
call cgca_gvl(space,grainvol)
call co_sum(grainvol)
call pat_region_end(2,pat_status)

! dump grain volumes
if ( img .eq. 1 ) then
 write (*,*) "results from cgca_gvl + co_sum"
 do i=lbound(grainvol,dim=1),ubound(grainvol,dim=1)
  write (*,"(i0,tr1,i0)") i, grainvol(i)
 end do
end if

! deallocate all arrays
call cgca_ds(space)
call cgca_dv(grainvol)

end program testABK
```

112 tests/testABL

[Unit tests]

NAME

testABL

SYNOPSIS

```
!$Id: testABL.f90 380 2017-03-22 11:03:09Z mexas $

program testABL
```

PURPOSE

Timing and checking cgca_sld3 (31.5.1)

DESCRIPTION

Timing solidification where the check for the complete solidification of the whole model is done with CO_SUM.

NOTES

The program must be called with 2 command line arguments, both positive integers. These are codimensions along 1 and 2. The number of images must be such that codimension3 = num_images()/(codimension1 * codimension3) is a positive integer. Example:

```
      cafrun -np 16 ./testABL.x 2 2      ! OpenCoarrays
```

or

```
      ./testABL.x 2 2                    ! Intel, Cray
```

which will make the third codimension equal to 16/(2*2)=4.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

cgca testaux (133)

USED BY

Part of CGPACK test suite

SOURCE

```
use testaux

implicit none
```

```
real,parameter :: gigabyte=real(2**30), resolution=1.0e-5, &
 loge2 = log(real(2))
logical(kind=ldef),parameter :: yesdebug = .true., nodebug = .false.

integer(kind=idef) :: l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3,cou3, &
 nuc,    & ! number of nuclei in the model
 nimages, codim(3)[*], p
integer(kind=iarr),allocatable :: space(:,:,:,:)[:,:,:]
integer(kind=ilrg) :: icells, mcells, img

logical(kind=ldef) :: solid

real :: image_storage, time1, time2

!*********************************************************************72
! first executable statement

img = this_image()
nimages = num_images()

! check than nimages is a power of 2
p = nint(log(real(nimages))/loge2)
if ( 2**p .ne. nimages) error stop "number of images is not a power of 2"

! do a check on image 1
if ( img .eq. 1 ) then
 call getcodim(nimages,codim)
 ! print a banner
 call banner("ABL")
 ! print the parameter values
 call cgca_pdmp
 write (*,'(a,i0,a)') "running on ", nimages, " images in a 3D grid"
 ! dump the value of p
 write (*,"(a,i0)") "p=",p
end if

! all images read codim from image 1
if ( img .eq. 1 ) then
 sync images(*)
else
 sync images(1)
 codim(:) = codim(:)[1]
end if

l1=1
l2=l1
l3=l1

! The array size is only controlled by this value
! in this program.
u1=16
u2=32
```

```
u3=32

col1=1
cou1=codim(1)-col1+1
col2=1
cou2=codim(2)-col2+1
col3=1
cou3=codim(3)-col3+1

! total number of cells in a coarray
icells = int(u1-l1+1,kind=ilrg) * int(u2-l2+1,kind=ilrg) * &
  int(u3-l3+1,kind=ilrg)

! total number of cells in the model
mcells = icells * int(codim(1),kind=ilrg) * int(codim(2),kind=ilrg) * &
  int(codim(3),kind=ilrg)

! total number of nuclei
nuc = int( resolution * real( mcells ) )

if ( img .eq. 1 ) then
  write (*,'(a,2(i0,":",i0,","),i0,":",i0,")")') &
    "bounds: (",l1,u1,l2,u2,l3,u3
  write (*,'(a,2(i0,":",i0,","),i0,":",i0,")")') &
    "cobounds: (",col1,cou1,col2,cou2,col3,cou3

  ! An absolute minimum of storage, in GB, per image.
  ! A factor of 2 is used because will call _sld, which
  ! allocates another array of the same size and kind as
  ! coarray.
  image_storage = real(2 * icells*storage_size(space)/8)/gigabyte

  write (*,'(a,i0,a)') "Each image has ",icells, " cells"
  write (*,'(a,i0,a)') "The model has ", mcells, " cells"
  write (*,'(a,i0,a)') "The model has ", nuc, " nuclei"
  write (*,'(a,es9.2,a)') "Each image will use at least ", &
    image_storage, " GB memory"
end if

! initialise random number seed
call cgca_irs(nodebug)

! allocate coarray
call cgca_as(l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3,1,space)

! initialise coarray to liquid
space = cgca_liquid_state

! populate nuclei
call cgca_nr(space,nuc,nodebug)

! solidify
```

```
call cpu_time(time1)
call cgca_sld3(space,0,1,solid)
call cpu_time(time2)
write (*,*) "img", img, "time, s", time2-time1

! dump the model
!call cgca_swci(space,cgca_state_type_grain,10,'z.raw')

! deallocate all arrays
call cgca_ds(space)

end program testABL
```

113 tests/testABM

[Unit tests]

NAME

testABM

SYNOPSIS

```
!$Id: testABM.f90 530 2018-03-26 16:10:00Z mexas $

program testABM
```

PURPOSE

Testing MPI/IO, cgca_m2mpiio (17)/cgca_pswci (19.4.1)

DESCRIPTION

Timing output of MPI/IO (cgca_pswci (19.4.1)) against the serial version (cgca_swci (19.2)).

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

cgca testaux (133)

USED BY

Part of CGPACK test suite

SOURCE

```
use testaux
use mpi

implicit none

real,parameter :: gigabyte=real(2**30), resolution=1.0e-5,            &
 loge2 = log(real(2))
logical(kind=ldef),parameter :: yesdebug = .true., nodebug = .false.

real( kind=rdef ) ::    &
 qual,                  & ! quality
 bsz0(3),               & ! the given "box" size
 bsz(3),                & ! updated "box" size
 dm,                    & ! mean grain size, linear dim, phys units
 lres,                  & ! linear resolution, cells per unit of length
 res                      ! resolutions, cells per grain

integer :: c(3), ir(3), nimgs, img, ng
integer( kind=iarr ), allocatable :: space(:,:,:,:)[:,:,:]
```

```
integer( kind=ilrg ) :: icells, mcells

integer :: ierr

real :: time1, time2, fsizeb, fsizeg, tdiff

logical :: iflag

!*********************************************************************72
! first executable statement

! physical dimensions of the box, assume mm
bsz0 = (/ 2.0, 3.0, 1.0 /)

! mean grain size, linear dimension, e.g. mean grain diameter, also mm
dm = 1.0e-1

! resolution
res = 1.0e5

    img = this_image()
nimgs = num_images()

! do a check on image 1
if ( img .eq. 1 ) then
 ! print a banner
 call banner("ABM")
 ! print the parameter values
 call cgca_pdmp
 write (*,'(a,i0,a)') "running on ", nimgs, " images in a 3D grid"
end if

! I want pdmp output appear before the rest.
! This might help
sync all

! each image calculates the coarray grid dimensions
call cgca_gdim( nimgs, ir, qual )

! calculate the resolution and the actual phys dimensions
! of the box
! subroutine cgca_cadim( bsz, res, dm, ir, c, lres, ng )
! c - coarray sizes
! ir - coarray grid sizes
bsz = bsz0
call cgca_cadim( bsz, res, dm, ir, c, lres, ng )

! total number of cells in a coarray
icells = int( c(1), kind=ilrg ) * int( c(2), kind=ilrg ) *             &
         int( c(3), kind=ilrg )
```

```
! total number of cells in the model
mcells = icells * int( nimgs, kind=ilrg )

if ( img .eq. 1 ) then
  write ( *, "(9(a,i0),tr1,g10.3,tr1,g10.3,3(a,g10.3),a)" )            &
    "img: ", img  , " nimgs: ", nimgs, " (", c(1) ,                    &
    ","    , c(2) , ","       , c(3) , ")[", ir(1),                    &
    ","    , ir(2), ","       , ir(3), "] ", ng   ,                    &
    qual, lres,                                                        &
    " (", bsz(1), ",", bsz(2), ",", bsz(3), ")"
  write (*,'(a,i0,a)') "Each image has ",icells, " cells"
  write (*,'(a,i0,a)') "The model has ", mcells, " cells"
end if

! Total output file size, in B and in GB.
fsizeb = real( mcells * storage_size( space, kind=ilrg ) / 8_ilrg )
fsizeg = fsizeb / gigabyte

! allocate space coarray with a single layer
! implicit sync all
!subroutine cgca_as( l1, u1, l2, u2, l3, u3, col1, cou1, col2, cou2,   &
!                    col3, props, coarray )
call cgca_as(1, c(1), 1, c(2), 1, c(3), 1, ir(1), 1, ir(2), 1, 1, space)

! initialise coarray to image number
space = int( img, kind=iarr )

! dump the model, serial
call cpu_time( time1 )
call cgca_swci( space, cgca_state_type_grain, 10, 'serial.raw' )
call cpu_time( time2 )
tdiff = time2-time1
if (img .eq. 1)                                                        &
  write (*,*) "Serial IO: ", tdiff, "s, rate: ", fsizeg/tdiff, "GB/s."

! All images start MPI, used only for I/O here, so no need to start
! earlier. Note that OpenCoarrays doesn't like MPI_Init.
! Probably it's called automatically by the runtime, so
! I put a check for it.
call MPI_Initialized( iflag, ierr )
if ( .not. iflag ) call MPI_Init(ierr)

! dump the model, MPI/IO
call cpu_time( time1 )
call cgca_pswci( space, cgca_state_type_grain, 'mpiio.raw' )
call cpu_time( time2 )
tdiff = time2-time1
if (img .eq. 1)                                                        &
  write (*,*) "MPI/IO: ", tdiff, "s, rate: ", fsizeg/tdiff, "GB/s."

! terminate MPI
! See the note above on MPI_Init
```

```
!call MPI_Finalized( iflag, ierr)
!if ( .not. iflag ) call MPI_Finalize(ierr)

! deallocate all arrays
call cgca_ds(space)

end program testABM
```

114 tests/testABN

[Unit tests]

NAME

testABN

SYNOPSIS

```
!$Id: testABN.f90 380 2017-03-22 11:03:09Z mexas $

program testABN
```

PURPOSE

Testing solidification with MPI/IO, cgca_m2mpiio (17)/cgca_pswci (19.4.1)

DESCRIPTION

Solidification with Cray reduction, CO_SUM (cgca_sld3 (31.5.1)). Timing output of MPI/IO (cgca_pswci (19.4.1)) against the serial version (cgca_swci (19.2)).

NOTES

The program must be called with 2 command line arguments, both positive integers. These are codimensions along 1 and 2. The number of images must be such that codimension3 = num_images()/(codimension1 * codimension3) is a positive integer. Example:

```
./testABN.x 2 2
```

which will make the third codimension equal to 16/(2*2)=4.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

cgca testaux (133)

USED BY

Part of CGPACK test suite

SOURCE

```
use testaux

implicit none

real,parameter :: gigabyte=real(2**30), resolution=1.0e-5, &
 loge2 = log(real(2))
logical(kind=ldef),parameter :: yesdebug = .true., nodebug = .false.

integer(kind=idef) :: l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3,cou3, &
```

```
 nuc,    & ! number of nuclei in the model
 nimages, codim(3)[*], p
integer(kind=iarr),allocatable :: space(:,:,:,:)[:,:,:]
integer(kind=ilrg) :: icells, mcells, img

real :: time1, time2, fsizeb, fsizeg, tdiff

logical(kind=ldef) :: solid

!*****************************************************************72
! first executable statement

img = this_image()
nimages = num_images()

! do a check on image 1
if ( img .eq. 1 ) then
 call getcodim(nimages,codim)
 ! print a banner
 call banner("ABN")
 ! print the parameter values
 call cgca_pdmp
 write (*,'(a,i0,a)') "running on ", nimages, " images in a 3D grid"
 ! dump the value of p
 write (*,"(a,i0)") "p=",p
end if

! all images read codim from image 1
sync all
codim(:) = codim(:)[1]

l1 = 1
l2 = l1
l3 = l1

! The array size is only controlled by this value in this program.
u1 = 2**7 ! 128
u2 = u1
u3 = u1

col1 = 1
col2 = col1
col3 = col1

cou1 = codim(1)-col1+1
cou2 = codim(2)-col2+1
cou3 = codim(3)-col3+1

! total number of cells in a coarray
icells = int(u1-l1+1,kind=ilrg) * int(u2-l2+1,kind=ilrg) *           &
 int(u3-l3+1,kind=ilrg)
```

```
! total number of cells in the model
mcells = icells * int(codim(1),kind=ilrg) * int(codim(2),kind=ilrg) * &
  int(codim(3),kind=ilrg)

! total number of nuclei
nuc = int( resolution * real( mcells ) )

if ( img .eq. 1 ) then
  write (*,'(a,2(i0,":",i0,","),i0,":",i0,")")') &
    "bounds: (",l1,u1,l2,u2,l3,u3
  write (*,'(a,2(i0,":",i0,","),i0,":",i0,")")') &
    "cobounds: (",col1,cou1,col2,cou2,col3,cou3

! Total output file size, in B and in GB.
fsizeb = real( mcells*storage_size(space,kind=ilrg)/8_ilrg )
fsizeg = fsizeb / gigabyte

  write (*,'(a,i0,a)') "Each image has ",icells, " cells"
  write (*,'(a,i0,a)') "The model has ", mcells, " cells"
  write (*,'(a,i0,a)') "The model has ", nuc, " nuclei"
  write (*,'(2(a,es9.2),a)') "The output file size is ", fsizeb, &
   " B, or ", fsizeg, "GB."
end if

! calculate file size in MB
! allocate coarray
call cgca_as(l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3,1,space)

! initialise coarray to liquid
space = cgca_liquid_state

! populate nuclei
call cgca_nr(space,nuc,nodebug)

! solidify
!subroutine cgca_sld3(coarray,iter,heartbeat,solid)
call cgca_sld3(space,0,1,solid)

! dump the model
call cpu_time(time1)
call cgca_pswci(space, cgca_state_type_grain, 'mpiio.raw')
call cpu_time(time2)
tdiff = time2-time1
if (img .eq. 1) write (*,*) "MPI/IO: ", tdiff, "s, rate: ", fsizeg/tdiff, "GB/s."

call cpu_time(time1)
call cgca_swci (space, cgca_state_type_grain, 10, 'serial.raw')
call cpu_time(time2)
tdiff = time2-time1
if (img .eq. 1) write (*,*) "Serial IO: ", tdiff, "s, rate: ", fsizeg/tdiff, "GB/s."

! deallocate all arrays
```

```
call cgca_ds(space)

end program testABN
```

115 tests/testABO

[Unit tests]

NAME

testABO

SYNOPSIS

```
!$Id: testABO.f90 418 2017-06-07 14:02:10Z mexas $

program testABO
```

PURPOSE

Checking: cgca_clvgp (26.17.1), cgca_clvgsd (26.5), cgca_pswci (19.4.1), cgca_sld3 (31.5.1)

DESCRIPTION

Checking cleavage propagation across grain boundary with many grains. Still a single cleavage nucleus. Put it at the centre of one of the faces of the model. Solidification is done with Cray reduction, CO_SUM (cgca_sld3 (31.5.1)). Output is done with MPI/IO (cgca_pswci (19.4.1)) and with the serial version (cgca_swci (19.2)). The timings of both IO routines are done for comparison.

NOTES

The program must be called with 2 command line arguments, both positive integers. These are codimensions along 1 and 2. The number of images must be such that codimension3 = num_images()/(codimension1 * codimension3) is a positive integer. Example:

```
./testABO.x 2 2
```

which will make the third codimension equal to 16/(2*2)=4.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

cgca testaux (133)

USED BY

Part of CGPACK test suite

SOURCE

```
use testaux

implicit none

logical( kind=ldef ), parameter :: yesdebug=.true., nodebug=.false., &
  periodicbc=.true.,  noperiodicbc=.false.
```

```
real, parameter :: gigabyte=real(2**30), resolution=1.0e-5,          &
! cleavage stress on 100, 110, 111 planes for BCC,
! see the manual for derivation.
  scrit(3) = (/ 1.05e4, 1.25e4, 4.90e4 /)

real( kind=rdef ), allocatable :: grt(:,:,:)[:,:,:]
real( kind=rdef ) :: t(3,3)   ! stress tensor

integer( kind=idef ) :: l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3,  &
 cou3, nuc,nimages,codim(3)[*]
integer( kind=iarr ), allocatable :: space(:,:,:,:)[:,:,:]
integer( kind=ilrg ) :: icells, mcells

real :: time1, time2, fsizeb, fsizeg, tdiff

logical( kind=ldef ) :: solid, image1

!*********************************************************************72
! first executable statement

nimages=num_images()
image1=.false.
if (this_image() .eq. 1) image1 = .true.

! do a check on image 1
if ( image1 ) then
 call getcodim(nimages,codim)
 ! print a banner
 call banner("ABO")
 ! print the parameter values
 call cgca_pdmp
 write (*,'(a,i0,a)') "running on ", nimages, " images in a 3D grid"
end if

sync all

codim(:) = codim(:)[1]

l1 = 1
l2 = l1
l3 = l1

! The array size is only controlled by this value in this program.
u1 = 2**7 ! 128
u2 = u1
u3 = u1

col1 = 1
cou1 = codim(1) - col1 + 1
col2 = 1
cou2 = codim(2) - col2 + 1
col3 = 1
```

```
cou3 = codim(3) - col3 + 1

! total number of cells in a coarray
icells = int( u1-l1+1, kind=ilrg ) *    &
         int( u2-l2+1, kind=ilrg ) *    &
         int( u3-l3+1, kind=ilrg )

! total number of cells in the model
mcells = icells * int( nimages, kind=ilrg )

! total number of nuclei
! nuc should not exceed resolution*mcells
nuc = 100

! total number of nuclei
nuc = int( resolution * real( mcells ) )

if ( image1 ) then
  write (*,'(a,2(i0,":",i0,","),i0,":",i0,")")') &
    "bounds: (",l1,u1,l2,u2,l3,u3
  write (*,'(a,2(i0,":",i0,","),i0,":",i0,")")') &
    "cobounds: (",col1,cou1,col2,cou2,col3,cou3

! Total output file size, in B and in GB.
fsizeb = real( mcells*storage_size(space,kind=ilrg)/8_ilrg )
fsizeg = fsizeb / gigabyte

  write (*,'(a,i0,a)') "Each image has ",icells, " cells"
  write (*,'(a,i0,a)') "The model has ", mcells, " cells"
  write (*,'(a,i0,a)') "The model has ", nuc, " nuclei"
  write (*,'(2(a,es9.2),a)') "The output file size is ", fsizeb, &
   " B, or ", fsizeg, "GB."
end if

! initialise random number seed
call cgca_irs( nodebug )

! allocate space with two layers
call cgca_as( l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3,2,space )

! allocate rotation tensors
call cgca_art( 1, nuc, col1, cou1, col2, cou2, col3, grt )

! initialise space
space( :, :, :, cgca_state_type_grain ) = cgca_liquid_state
space( :, :, :, cgca_state_type_frac ) = cgca_intact_state

! nuclei, sync all inside
call cgca_nr( space, nuc, nodebug )

! assign rotation tensors, sync all inside
call cgca_rt( grt )
```

```
! set a single crack nucleus in the centre of the front face
space( u1/2, u2/2, u3, cgca_state_type_frac )[ cou1/2, cou2/2, cou3] = &
 cgca_clvg_state_100_edge

! solidify
! subroutine cgca_sld3(coarray,iter,heartbeat,solid)
call cgca_sld3( space, 0, 1, solid )

! update grain connectivity, local routine, no sync needed
call cgca_gcu( space )

! global sync needed to wait for cgca_gcu to complete on all images
sync all

! set the stress tensor
t = 0.0
t(1,1) = 1.0e6
t(2,2) = -1.0e6

! propagate cleavage, sync inside
! subroutine cgca_clvgp( coarray, rt, t, scrit, sub, gcus,
!    periodicbc, iter, heartbeat, debug )
call cgca_clvgp( space, grt, t, scrit, cgca_clvgsd, cgca_gcupdn,       &
     noperiodicbc, 200, 10, nodebug )

! dump the model
call cpu_time(time1)
call cgca_pswci( space, cgca_state_type_frac, 'frac_mpi.raw' )
call cpu_time(time2)
tdiff = time2-time1
if ( image1 ) write (*,*) "MPI/IO: ", tdiff, "s, rate: ", fsizeg/tdiff, "GB/s."

call cpu_time(time1)
call cgca_swci( space, cgca_state_type_frac, 10, 'frac_ser.raw' )
call cpu_time(time2)
tdiff = time2-time1
if ( image1 ) write (*,*) "Serial IO: ", tdiff, "s, rate: ", fsizeg/tdiff, "GB/s."

! However, since there's nothing more to do, no sync is needed.

! deallocate all arrays, implicit sync all.
call cgca_ds( space )
call cgca_dgc
call cgca_drt( grt )

end program testABO
```

116 tests/testABP

[Unit tests]

NAME

testABP

SYNOPSIS

```
!$Id: testABP.f90 380 2017-03-22 11:03:09Z mexas $

program testABP
```

PURPOSE

Checking: cgca_tchk (26.16.1)

DESCRIPTION

Checking the MAXMIN value of the dot product between an arbitrary cleavage plane normal and all 26 unit vectors connecting the central cell with its neighbours. cgca_tchk (26.16.1) is a serial routine, so can make only image 1 call it, or even better, make all images execute it, to increase the search space. No sync required.

NOTES

The program must be called with 2 command line arguments, both positive integers. These are codimensions along 1 and 2. The number of images must be such that codimension3 = num_images()/(codimension1 * codimension3) is a positive integer. Example:

```
./testABP.x 2 2
```

which will make the third codimension equal to 16/(2*2)=4.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

cgca testaux (133)

USED BY

Part of CGPACK test suite

SOURCE

```
use testaux

implicit none

integer( kind=idef ) :: nimages, codim(3)[*], image
logical( kind=ldef ) :: image1
real(    kind=rlrg ) :: maxmin, minmax
```

```
!*********************************************************************72
! first executable statement

nimages = num_images()
  image = this_image()
 image1 = .false.
if (this_image().eq.1) image1 = .true.

! do a check on image 1
if (image1) then
 call getcodim( nimages, codim )
 ! print a banner
 call banner("ABP")
 ! print the parameter values
 call cgca_pdmp
 write (*,'(a,i0,a)') "running on ", nimages, " images in a 3D grid"
 write (*,*) "codim:", codim
end if

sync all

codim(:) = codim(:)[1]

! initialise random seed
call cgca_irs( debug = .false. )

! check threshold t
call cgca_tchk( 2_ilrg**32, maxmin, minmax ) ! 4,294,967,296
write( *, "(a, i0, 2(a,es20.10))" ) "image: ", image, " maxmin: ",      &
                         maxmin, " minmax: ", minmax

end program testABP
```

117 tests/testABQ

[Unit tests]

NAME

testABQ

SYNOPSIS

```
!$Id: testABQ.f90 380 2017-03-22 11:03:09Z mexas $

program testABQ
```

PURPOSE

Checking: cgca_igb (11.10)

DESCRIPTION

Checking inialisation of the grain boundary cells. Right after the solidification and grain boundary smoothing call cgca_igb (11.10), and actually create the intact GB cells in the fracture array.

NOTES

The program must be called with 2 command line arguments, both positive integers. These are codimensions along 1 and 2. The number of images must be such that codimension3 = num_images()/(codimension1 * codimension3) is a positive integer. Example:

```
./testABQ.x 2 2
```

which will make the third codimension equal to 16/(2*2)=4.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

cgca testaux (133)

USED BY

Part of CGPACK test suite

SOURCE

```
use testaux

implicit none

logical( kind=ldef ), parameter :: yesdebug=.true., nodebug=.false., &
 periodicbc=.true.,  noperiodicbc=.false.

real, parameter :: gigabyte=real(2**30), resolution=1.0e-5
```

```
real( kind=rdef ), allocatable :: grt(:,:,:)[:,:,:]

integer( kind=idef ) :: l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3,  &
 cou3, nuc, nimages, codim(3)[*], iter
integer( kind=iarr ), allocatable :: space(:,:,:,:)[:,:,:]
integer( kind=ilrg ) :: icells,mcells

real :: image_storage

logical( kind=ldef ) :: solid, image1

!******************************************************************72
! first executable statement

nimages = num_images()
 image1 = .false.
if (this_image() .eq. 1) image1 = .true.

! do a check on image 1
if (image1) then
 call getcodim(nimages,codim)
 ! print a banner
 call banner("ABQ")
 ! print the parameter values
 call cgca_pdmp
 write (*,'(a,i0,a)') "running on ", nimages, " images in a 3D grid"
end if

sync all

codim(:) = codim(:)[1]

l1 = 1
l2 = l1
l3 = l1

u1 = 2**6 ! 64
u2 = u1
u3 = u1

col1 = 1
cou1 = codim(1) - col1 + 1
col2 = 1
cou2 = codim(2) - col2 + 1
col3 = 1
cou3 = codim(3) - col3 + 1

! total number of cells in a coarray
icells = int( u1-l1+1, kind=ilrg ) *    &
         int( u2-l2+1, kind=ilrg ) *    &
         int( u3-l3+1, kind=ilrg )
```

```
! total number of cells in the model
mcells = icells * int( nimages, kind=ilrg )

! total number of nuclei
! nuc should not exceed resolution*mcells
nuc = 20

100 format( a,3(i0,":",i0,a) )

if (image1) then
  write (*,100) "bounds: (", l1,u1,",", l2,u2,",", l3,u3, ")"
  write (*,100) "cobounds: [", &
    col1,cou1,",", col2,cou2,",", col3,cou3, "]"

  ! An absolute minimum of storage, in GB, per image.
  ! A factor of 2 is used because will call _sld, which
  ! allocates another array of the same size and kind as
  ! coarray.
  image_storage = real(2 * icells*storage_size(space)/8)/gigabyte

  write (*,'(a,i0,a)') "Each image has ",icells, " cells"
  write (*,'(a,i0,a)') "The model has ", mcells, " cells"
  write (*,'(a,i0,a)') "The model has ", nuc, " nuclei"
  write (*,'(a,es9.2,a)') "Each image will use at least ",       &
                          image_storage, " GB memory"
end if

! initialise random number seed
call cgca_irs( nodebug )

! allocate space with two layers
call cgca_as( l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3,2,space )

! allocate rotation tensors
call cgca_art( 1, nuc, col1, cou1, col2, cou2, col3, grt )

! initialise space
space( :, :, :, cgca_state_type_grain ) = cgca_liquid_state
space( :, :, :, cgca_state_type_frac  ) = cgca_intact_state

! nuclei, sync all inside
call cgca_nr( space, nuc, yesdebug )

! assign rotation tensors, sync all inside
call cgca_rt( grt )

! set a single crack nucleus in the centre of the front face
space( u1/2, u2/2, u3, cgca_state_type_frac )[ cou1/2, cou2/2, cou3] = &
 cgca_clvg_state_100_edge

! solidify, implicit sync all inside
call cgca_sld( space, noperiodicbc, 0, 10, solid )
```

```
! smoothen the GB, several iterations, sync needed
do iter=1,2
  call cgca_gbs( space )
  sync all

  ! halo exchange, following smoothing
  call cgca_hxi( space )
  sync all
end do

! update grain connectivity, local routine, no sync needed
call cgca_igb( space )

if ( image1 ) write (*,*) "dumping model to files"

! dump space arrays to files, only image 1 does it, all others
! wait at the barrier, hence sync needed
call cgca_swci( space, cgca_state_type_grain, 10, "zg1.raw" )
call cgca_swci( space, cgca_state_type_frac,  10, "zf1.raw" )

if ( image1 ) write (*,*) "finished dumping model to files"

! deallocate all arrays, implicit sync all.
call cgca_ds( space )
call cgca_drt( grt )

end program testABQ
```

118 tests/testABR

[Unit tests]

NAME

testABR

SYNOPSIS

```
!$Id: testABR.f90 380 2017-03-22 11:03:09Z mexas $

program testABR
```

PURPOSE

Checking: cgca_igb (11.10), cgca_clvgp_nocosum (26.4) (no CO_SUM version),

```
cgca_clvgsd
```

DESCRIPTION

Checking cleavage propagation across grain boundary with many grains. Grain boundaries are initiated before cleavage. The intention is to compare with test ABB, to see if explicit GB make any difference. Put a single crack nucleus at the centre of one of the faces of the model.

With no grain boundary smoothing, crack finds it very hard to propagate across a grain boundary. This is because the GB is locally very irregular, and it is likely the first cell in the new grain will find itself in some sort of corner or a tunnel, from where it cannot see enough neighbours of the same grains to propagate into. Hence use cgca_gbs (11.3), at least once, possibly multiple iterations.

NOTES

The program must be called with 2 command line arguments, both positive integers. These are codimensions along 1 and 2. The number of images must be such that codimension3 = num_images()/(codimension1 * codimension3) is a positive integer. Example:

```
./testABR.x 2 2
```

which will make the third codimension equal to 16/(2*2)=4.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

cgca testaux (133)

USED BY

Part of CGPACK test suite

SOURCE

```
use testaux
```

```
implicit none

logical( kind=ldef ), parameter :: yesdebug=.true., nodebug=.false., &
 periodicbc=.true.,  noperiodicbc=.false.

real, parameter :: gigabyte=real(2**30), resolution=1.0e-5,          &
! cleavage stress on 100, 110, 111 planes for BCC,
! see the manual for derivation.
 scrit(3) = (/ 1.05e4, 1.25e4, 4.90e4 /)

real( kind=rdef ), allocatable :: grt(:,:,:)[:,:,:]
real( kind=rdef ) :: t(3,3)   ! stress tensor

integer( kind=idef ) :: l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3,  &
 cou3, nuc, nimages, codim(3)[*], iter
integer( kind=iarr ), allocatable :: space(:,:,:,:)[:,:,:]
integer( kind=ilrg ) :: icells,mcells

real :: image_storage

logical( kind=ldef ) :: solid, image1
character(6) :: image

!*********************************************************************72
! first executable statement

nimages = num_images()
 image1 = .false.
if (this_image() .eq. 1) image1 = .true.

! do a check on image 1
if (image1) then
 call getcodim(nimages,codim)
 ! print a banner
 call banner("ABR")
 ! print the parameter values
 call cgca_pdmp
 write (*,'(a,i0,a)') "running on ", nimages, " images in a 3D grid"
end if

sync all

codim(:) = codim(:)[1]

l1 = 1
l2 = l1
l3 = l1

u1 = 2**7 ! 128
u2 = u1
u3 = u1
```

```
col1 = 1
cou1 = codim(1) - col1 + 1
col2 = 1
cou2 = codim(2) - col2 + 1
col3 = 1
cou3 = codim(3) - col3 + 1

! total number of cells in a coarray
icells = int( u1-l1+1, kind=ilrg ) *    &
         int( u2-l2+1, kind=ilrg ) *    &
         int( u3-l3+1, kind=ilrg )

! total number of cells in the model
mcells = icells * int( nimages, kind=ilrg )

! total number of nuclei
! nuc should not exceed resolution*mcells
nuc = 20

100 format( a,3(i0,":",i0,a) )

if (image1) then
  write (*,100) "bounds: (", l1,u1,",", l2,u2,",", l3,u3, ")"
  write (*,100) "cobounds: [", &
    col1,cou1,",", col2,cou2,",", col3,cou3, "]"

  ! An absolute minimum of storage, in GB, per image.
  ! A factor of 2 is used because will call _sld, which
  ! allocates another array of the same size and kind as
  ! coarray.
  image_storage = real(2 * icells*storage_size(space)/8)/gigabyte

  write (*,'(a,i0,a)') "Each image has ",icells, " cells"
  write (*,'(a,i0,a)') "The model has ", mcells, " cells"
  write (*,'(a,i0)') "Max recommended nuclei ",                         &
                     int( resolution* real( mcells ) )
  write (*,'(a,i0,a)') "The model has ", nuc, " nuclei"
  write (*,'(a,es9.2,a)') "Each image will use at least ",              &
                          image_storage, " GB memory"
end if

! initialise random number seed
call cgca_irs( nodebug )

! allocate space with two layers
call cgca_as( l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3,2,space )

! allocate rotation tensors
call cgca_art( 1, nuc, col1, cou1, col2, cou2, col3, grt )

! initialise space
```

```
space( :, :, :, cgca_state_type_grain ) = cgca_liquid_state
space( :, :, :, cgca_state_type_frac  ) = cgca_intact_state

! nuclei, sync all inside
call cgca_nr( space, nuc, yesdebug )

! assign rotation tensors, sync all inside
call cgca_rt( grt )

! set a single crack nucleus in the centre of the front face
space( u1/2, u2/2, u3, cgca_state_type_frac )[ cou1/2, cou2/2, cou3] = &
 cgca_clvg_state_100_edge

! solidify, implicit sync all inside
call cgca_sld( space, noperiodicbc, 0, 10, solid )

! smoothen the GB, several iterations, sync needed
do iter=1,2
  call cgca_gbs( space )
  sync all

  ! halo exchange, following smoothing
  call cgca_hxi( space )
  sync all
end do

! update grain connectivity, local routine, no sync needed
call cgca_gcu( space )

! initiate grain boundaries
call cgca_igb( space )

! dump grain connectivity to files, local routine, no sync needed
write ( image, "(i0)" ) this_image()
call cgca_gcp( ounit=10, fname="z_gc_1_"//image )

if ( image1 ) write (*,*) "dumping model to files"

! dump space arrays to files, only image 1 does it, all others
! wait at the barrier, hence sync needed
call cgca_swci( space, cgca_state_type_grain, 10, "zg1.raw" )
call cgca_swci( space, cgca_state_type_frac,  10, "zf1.raw" )

if ( image1 ) write (*,*) "finished dumping model to files"

sync all

! set the stress tensor
t = 0.0
t(1,1) = 1.0e6
t(2,2) = -1.0e6
```

```
! propagate cleavage, sync inside
! subroutine cgca_clvgp_nocosum( coarray, rt, t, scrit, sub,
!    gcus, periodicbc, iter, heartbeat, debug )
call cgca_clvgp_nocosum( space, grt, t, scrit, cgca_clvgsd,            &
     cgca_gcupdn, noperiodicbc, 300, 10, yesdebug )

! dump grain connectivity to files, local routine, no sync needed.
write ( image, "(i0)" ) this_image()
call cgca_gcp( ounit=10, fname="z_gc_2_"//image )

! dump the fracture space array to files, only image 1 does it,
! all others wait at the barrier, hence sync needed
call cgca_swci( space, cgca_state_type_frac, 10, "zf2.raw" )

! However, since there's nothing more to do, no sync is needed.

! deallocate all arrays, implicit sync all.
call cgca_ds( space )
call cgca_dgc
call cgca_drt( grt )

if ( image1 ) write (*,"(a)") "Test ABR completed sucessfully"

end program testABR
```

119 tests/testABS

[Unit tests]

NAME

testABS

SYNOPSIS

```
!$Id: testABS.f90 380 2017-03-22 11:03:09Z mexas $

program testABS
```

PURPOSE

Checking: cgca_igb (11.10), cgca_clvgp_nocosum (26.4) (no CO_SUM version),

```
cgca_clvgsd, dumping
```

crack state at regular intervals to make crack growth animation.

DESCRIPTION

Checking cleavage propagation across grain boundary with many grains. Grain boundaries are initiated before cleavage. A single crack nucleus in the middle of x3=0 face.

With no grain boundary smoothing, crack finds it very hard to propagate across a grain boundary. This is because the GB is locally very irregular, and it is likely the first cell in the new grain will find itself in some sort of corner or a tunnel, from where it cannot see enough neighbours of the same grains to propagate into. Hence use cgca_gbs (11.3), at least once, possibly multiple iterations.

NOTES

The program must be called with 2 command line arguments, both positive integers. These are codimensions along 1 and 2. The number of images must be such that codimension3 = num_images()/(codimension1 * codimension3) is a positive integer. Example:

```
./testABN.x 2 2
```

which will make the third codimension equal to 16/(2*2)=4.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

cgca testaux (133)

USED BY

Part of CGPACK test suite

SOURCE

```
use testaux

implicit none

logical( kind=ldef ), parameter :: yesdebug=.true., nodebug=.false., &
 periodicbc=.true.,  noperiodicbc=.false.

! specify the total number of cleavage propagation iterations,
! and the number of times the fracture array will be dumped along
! the way
integer( kind=idef ), parameter :: itot = 300_idef, idmp = 10_idef

real, parameter :: gigabyte=real(2**30), resolution=1.0e-5,          &
! cleavage stress on 100, 110, 111 planes for BCC,
! see the manual for derivation.
 scrit(3) = (/ 1.05e4, 1.25e4, 4.90e4 /)

real( kind=rdef ), allocatable :: grt(:,:,:)[:,:,:]
real( kind=rdef ) :: t(3,3)    ! stress tensor

integer( kind=idef ) :: l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3,  &
 cou3, nuc, nimages, codim(3)[*], iter, fiter
integer( kind=iarr ), allocatable :: space(:,:,:,:)[:,:,:]
integer( kind=ilrg ) :: icells, mcells

real :: image_storage

logical( kind=ldef ) :: solid, image1
character(6) :: image, citer

!*****************************************************************72
! first executable statement

nimages = num_images()
 image1 = .false.
if (this_image() .eq. 1) image1 = .true.

! do a check on image 1
if (image1) then
 call getcodim(nimages,codim)
 ! print a banner
 call banner("ABS")
 ! print the parameter values
 call cgca_pdmp
 write (*,'(a,i0,a)') "running on ", nimages, " images in a 3D grid"
end if

sync all

codim(:) = codim(:)[1]

l1 = 1
```

```
l2 = l1
l3 = l1

!u1 = 2**6 ! 64
u1=100
u2 = u1
u3 = u1

col1 = 1
cou1 = codim(1) - col1 + 1
col2 = 1
cou2 = codim(2) - col2 + 1
col3 = 1
cou3 = codim(3) - col3 + 1

! total number of cells in a coarray
icells = int( u1-l1+1, kind=ilrg ) *    &
         int( u2-l2+1, kind=ilrg ) *    &
         int( u3-l3+1, kind=ilrg )

! total number of cells in the model
mcells = icells * int( nimages, kind=ilrg )

! total number of nuclei
! nuc should not exceed resolution*mcells
nuc = 100

100 format( a,3(i0,":",i0,a) )

if (image1) then
  write (*,100) "bounds: (", l1,u1,",", l2,u2,",", l3,u3, ")"
  write (*,100) "cobounds: [", &
    col1,cou1,",", col2,cou2,",", col3,cou3, "]"

  ! An absolute minimum of storage, in GB, per image.
  ! A factor of 2 is used because will call _sld, which
  ! allocates another array of the same size and kind as
  ! coarray.
  image_storage = real(2 * icells*storage_size(space)/8)/gigabyte

  write (*,'(a,i0,a)') "Each image has ",icells, " cells"
  write (*,'(a,i0,a)') "The model has ", mcells, " cells"
  write (*,'(a,i0)') "Max recommended nuclei ",                         &
                     int( resolution * real( mcells ) )
  write (*,'(a,i0,a)') "The model has ", nuc, " nuclei"
  write (*,'(a,es9.2,a)') "Each image will use at least ",              &
                          image_storage, " GB memory"
end if

! initialise random number seed
call cgca_irs( nodebug )
```

```
! allocate space with two layers
call cgca_as( l1,u1,l2,u2,l3,u3,col1,cou1,col2,cou2,col3,2,space )

! allocate rotation tensors
call cgca_art( 1, nuc, col1, cou1, col2, cou2, col3, grt )

! initialise space
space( :, :, :, cgca_state_type_grain ) = cgca_liquid_state
space( :, :, :, cgca_state_type_frac  ) = cgca_intact_state

! nuclei, sync all inside
call cgca_nr( space, nuc, yesdebug )

! assign rotation tensors, sync all inside
call cgca_rt( grt )

! set a single crack nucleus in the centre of the x3=0 face
space( u1/2, u2/2, l3, cgca_state_type_frac )[ cou1/2, cou2/2, col3] = &
 cgca_clvg_state_100_edge

! solidify, implicit sync all inside
call cgca_sld( space, noperiodicbc, 0, 10, solid )

! smoothen the GB, several iterations, sync needed
do iter=1,2
  call cgca_gbs( space )
  sync all

  ! halo exchange, following smoothing
  call cgca_hxi( space )
  sync all
end do

! update grain connectivity, local routine, no sync needed
call cgca_gcu( space )

! initiate grain boundaries
call cgca_igb( space )

! dump grain connectivity to files, local routine, no sync needed
write ( image, "(i0)" ) this_image()
call cgca_gcp( ounit=10, fname="z_gc_1_"//image )

if ( image1 ) write (*,*) "dumping model to files"

! dump space arrays to files, only image 1 does it, all others
! wait at the barrier, hence sync needed
call cgca_swci( space, cgca_state_type_grain, 10, "zg0.raw" )
call cgca_swci( space, cgca_state_type_frac,  10, "zf0.raw" )

if ( image1 ) write (*,*) "finished dumping model to files"
```

```
sync all

! set the stress tensor
t = 0.0
t(1,1) = 1.0e6
t(2,2) = -1.0e6

! propagate cleavage, sync inside, dump fracture arrays
! every certain number of increments.
do iter = 1, idmp

 ! Propagate cleavage, sync inside
 ! Run for fiter iterations
 ! subroutine cgca_clvgp( coarray, rt, t, scrit, sub, periodicbc,    &
 !                        iter, heartbeat, debug )
 fiter = itot/idmp

 ! subroutine cgca_clvgp_nocosum( coarray, rt, t, scrit, sub,
 !    gcus, periodicbc, iter, heartbeat, debug )
 call cgca_clvgp_nocosum( space, grt, t, scrit, cgca_clvgsd,           &
      cgca_gcupdn, noperiodicbc, fiter, 10, yesdebug )

 ! dump the fracture space array to files, only image 1 does it,
 ! all others wait at the barrier, hence sync needed.
 ! citer is the total number of cleavage fracture iterations,
 ! "c" for character date type.
 write ( citer, "(i0)" ) iter*fiter
 call cgca_swci( space, cgca_state_type_frac, 10,  &
                 "zf"//trim(citer)//".raw" )

 sync all

 if ( image1 ) write (*,"(a)") &
  "Completed "//trim(citer)//" cleavage iterations"

end do

! dump grain connectivity to files, local routine, no sync needed.
write ( image, "(i0)" ) this_image()
call cgca_gcp( ounit=10, fname="z_gc_2_"//image )

! deallocate all arrays, implicit sync all.
call cgca_ds( space )
call cgca_dgc
call cgca_drt( grt )

if ( image1 ) write (*,"(a)") "Test ABS completed sucessfully"

end program testABS
```

120 tests/testABT

[Unit tests]

NAME

testABT

SYNOPSIS

```
!$Id: testABT.f90 526 2018-03-25 23:44:51Z mexas $

program testABT
```

PURPOSE

Checking: cgca_gdim (21.2), cgca_cadim (21.1)

DESCRIPTION

cgca_gdim (21.2) finds the optimum coarray grid layout for a given total number of images. It also reports the quality of this optimum, from 0 - worst, to 1 - best. cgca_cadim (21.1) then calculates the coarray dimensions the new updated box size.

NOTE

Both cgca_gdim (21.2) and cgca_cadim (21.1) are serial routines. It makes no sence to run this test at high numbers of images. A single image is enough to test the routines.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

cgca testaux (133)

USED BY

Part of CGPACK test suite

SOURCE

```
use testaux

implicit none

integer( kind=idef ) :: n, ir(3), nimgs, &
  ng,     & ! number of grains in the whole model
  c(3)      ! coarray dimensions
logical( kind=ldef ) :: image1
real( kind=rdef ) ::    &
 qual,                  & ! quality
 bsz0(3),               & ! the given "box" size
 bsz(3),                & ! updated "box" size
 dm,                    & ! mean grain size, linear dim, phys units
```

```
 res,                   & ! resolutions, cells per grain
! tmprnd(3),             & ! array of random numbers
 lres                     ! linear resolution

!*********************************************************************72
! first executable statement

! physical dimensions of the box, assume mm
bsz0 = (/ 10.0, 20.0, 30.0 /)

! mean grain size, also mm
dm = 5.0e-1

! resolution
res = 1.0e5

 nimgs = num_images()
image1 = .false.
if (this_image() .eq. 1) image1 = .true.

! do a check on image 1
if (image1) then

 ! print a banner
 call banner("ABT")

 ! print the parameter values
 call cgca_pdmp
 write (*,'(a,i0,a)') "running on ", nimgs, " images in a 3D grid"

 ! calculate the coarray grid dimensions
 do n = 1, 2**15
   call cgca_gdim( n, ir, qual )

!   ! choose box sizes at random, max 30 in any dimension
!   call random_number( tmprnd )
!   bsz0 = tmprnd * 30.0

   ! subroutine cgca_cadim( bsz, res, dm, ir, c, lres, ng )
   bsz = bsz0
   call cgca_cadim( bsz, res, dm, ir, c, lres, ng )

   write ( *, "(8(i0,a),es9.2,tr1,es9.2,3(a,es9.2),a)" )              &
    n, "(", c(1), ",", c(2), ",", c(3), ")[" ,                        &
    ir(1), ",", ir(2), ",", ir(3), "] ", ng, " ",                     &
    qual, lres,                                                       &
    " (", bsz(1), ",", bsz(2), ",", bsz(3), ")"
 end do

end if

sync all
```

```
end program testABT
```

121 tests/testABU

[Unit tests]

NAME

testABU

SYNOPSIS

```
!$Id: testABU.f90 526 2018-03-25 23:44:51Z mexas $

program testABU
```

PURPOSE

Checking: cgca_imco (21.3)

DESCRIPTION

First need to call cgca_gdim (21.2), cgca_cadim (21.1) to calculate all parameters of coarray space.

NOTE

cgca_gdim (21.2) and cgca_cadim (21.1) can be called by any or all images. Their results do not depend on the index of the invoking image. However, cgca_imco (21.3) must be called by every image, So it makes sense to call all three routines by every image.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

cgca testaux (133)

USED BY

Part of CGPACK test suite

SOURCE

```
use testaux

implicit none

integer( kind=idef ) :: ir(3), img, nimgs,                              &
  ng,    & ! number of grains in the whole model
  c(3)     ! coarray dimensions
integer( kind=iarr ), allocatable :: space(:,:,:,:) [:,:,:]

real( kind=rdef ), parameter :: zero = 0.0_rdef, one = 1.0_rdef
real( kind=rdef ) ::    &
 lres,                  & ! linear resolution
 qual,                  & ! quality
 bsz0(3),               & ! the given "box" size
```

```
 bsz(3),                & ! updated "box" size
 origin(3),             & ! origin of the "box" cs, in FE cs
 rot(3,3),              & ! rotation tensor *from* FE cs *to* CA cs
 dm,                    & ! mean grain size, linear dim, phys units
 res,                   & ! resolutions, cells per grain
 bcol(3), bcou(3)         ! lower and upper phys. coord of the coarray
                          ! on each image

!*****************************************************************72
! first executable statement

! physical dimensions of the box, assume mm
bsz0 = (/ 1.0, 2.0, 3.0 /)

! origin of the box cs, assume mm
origin = (/ 10.0, 11.0, 12.0 /)

! rotation tensor *from* FE cs *to* CA cs.
! The box cs is aligned with the box.
rot = zero
rot(1,1) = one
rot(2,2) = one
rot(3,3) = one

! mean grain size, also mm
dm = 1.0e-1

! resolution
res = 1.0e5

! In this test set the number of images via the env var
! the code must be able to cope with any value >= 1.
   img = this_image()
 nimgs = num_images()

! do a check on image 1
if ( img .eq. 1 ) then

 ! print a banner
 call banner("ABU")

 ! print the parameter values
 call cgca_pdmp
 write (*,'(a,i0,a)') "running on ", nimgs, " images in a 3D grid"

end if

! want to sync here to make sure the banner is
! printed before the rest.
sync all

! each image calculates the coarray grid dimensions
```

```
call cgca_gdim( nimgs, ir, qual )

! calculate the resolution and the actual phys dimensions
! of the box
! subroutine cgca_cadim( bsz, res, dm, ir, c, lres, ng )
bsz = bsz0
call cgca_cadim( bsz, res, dm, ir, c, lres, ng )

write ( *, "(9(a,i0),tr1,g10.3,tr1,i0,3(a,g10.3),a)" )           &
    "img: ", img, " nimgs: ", nimgs,                             &
     " (", c(1), ",", c(2), ",", c(3),                           &
     ")[", ir(1), ",", ir(2), ",", ir(3), "] ", ng,              &
    qual, lres,                                                  &
    " (", bsz(1), ",", bsz(2), ",", bsz(3), ")"

! allocate space coarray with a single layer
call cgca_as(1, c(1), 1, c(2), 1, c(3), 1, ir(1), 1, ir(2), 1, 1,space)

! calculate the extremeties of the box, in the CA cs, on each image
!subroutine cgca_imco( space, lres, bcol, bcou )
call cgca_imco( space, lres, bcol, bcou )

write ( *,"(a,i0,2(a,3(g10.3,tr1)),a)" ) "img: ", img,           &
  " CA bcol: (", bcol, ") CA bcou: (", bcou, ")"

! and now in FE cs:
write ( *,"(a,i0,2(a,3(g10.3,tr1)),a)" ) "img: ", img,           &
   " FE bcol: (", matmul( transpose( rot ),bcol ) + origin,      &
  ") FE bcou: (", matmul( transpose( rot ),bcou ) + origin, ")"

! deallocate space
call cgca_ds( space )

end program testABU
```

122 tests/testABV

[Unit tests]

NAME

testABV

SYNOPSIS

```
!$Id: testABV.f90 526 2018-03-25 23:44:51Z mexas $

program testABV
```

PURPOSE

Scaling analysis of solidification and cleavage. MPI/IO and F2015 collectives are used.

DESCRIPTION

The model is defined by (1) the CA box size, (2) the mean grain size and (3) the spatial resolution. First need to call cgca_gdim (21.2), cgca_cadim (21.1) to calculate all parameters of coarray space, including the number of nuclei. Then call solidification and then cleavage. The parameters are chosen to give the biggest model that can fit on a single XC40 node in Hazel Hen (HLRS, PRACE, Tier-0 system). Then strong scaling can be investigated, including pure computation, computation + IO, pure IO etc. for solidification and fracture simulations.

NOTE AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

cgca testaux (133)

USED BY

Part of CGPACK test suite

SOURCE

```
use testaux

implicit none

logical( kind=ldef ), parameter :: yesdebug=.true., nodebug=.false.,   &
 periodicbc=.true.,  noperiodicbc=.false.

! specify the total number of cleavage propagation iterations,
! and the number of times the fracture array will be dumped along
! the way
integer( kind=idef ), parameter ::                                     &
 itot = 300_idef,       & ! total cleavage iterations to do
 idmp = 10_idef           ! number of dumps to do along the way
```

```
real, parameter :: gigabyte=real(2**30),                                &
! cleavage stress on 100, 110, 111 planes for BCC,
! see the manual for derivation.
 scrit(3) = (/ 1.05e4, 1.25e4, 4.90e4 /)

!integer :: ierr ! default integer needed for MPI

integer( kind=idef ) :: iter, fiter, ir(3), img, nimgs,                 &
  ng,    & ! number of grains in the whole model
  c(3)     ! coarray dimensions
integer( kind=iarr ), allocatable :: space(:,:,:,:) [:,:,:]

integer( kind=ilrg ) :: icells, mcells

real( kind=rdef ) ::    &
 t(3,3),                & ! stress tensor
 qual,                  & ! quality
 bsz0(3),               & ! the given "box" size
 bsz(3),                & ! updated "box" size
 dm,                    & ! mean grain size, linear dim, phys units
 lres,                  & ! linear resolution, cells per unit of length
 res                      ! resolutions, cells per grain
real( kind=rdef ), allocatable :: grt(:,:,:)[:,:,:]

logical( kind=ldef ) :: solid
character(6) :: citer
real :: time1, time2

!*********************************************************************72
! first executable statement

! physical dimensions of the box, assume mm
bsz0 = (/ 4.0, 5.0, 5.0 /)

! mean grain size, linear dimension, e.g. mean grain diameter, also mm
dm = 1.0e-1

! resolution
res = 1.0e5

! In this test set the number of images via the env var
! the code must be able to cope with any value >= 1.
   img = this_image()
 nimgs = num_images()

! do a check on image 1
if ( img .eq. 1 ) then

 ! print a banner
 call banner("ABV")

 ! print the parameter values
```

```
 call cgca_pdmp
 write (*,'(a,i0,a)') "running on ", nimgs, " images in a 3D grid"

end if

! want to sync here to make sure the banner is
! printed before the rest.
sync all

! each image calculates the coarray grid dimensions
call cgca_gdim( nimgs, ir, qual )

! calculate the resolution and the actual phys dimensions
! of the box
! subroutine cgca_cadim( bsz, res, dm, ir, c, lres, ng )
! c - coarray sizes
! ir - coarray grid sizes
bsz = bsz0
call cgca_cadim( bsz, res, dm, ir, c, lres, ng )

! total number of cells in a coarray
icells = int( c(1), kind=ilrg ) *    &
         int( c(2), kind=ilrg ) *    &
         int( c(3), kind=ilrg )

! total number of cells in the model
mcells = icells * int( nimgs, kind=ilrg )

if ( img .eq. 1 ) then
  write ( *, "(9(a,i0), g11.3, g11.3, 3(a,g11.3), a )" )           &
    "img: ", img  , " nimgs: ", nimgs, " (", c(1) ,                &
    ","    , c(2) , ","       , c(3) , ")[", ir(1),                &
    ","    , ir(2), ","       , ir(3), "] ", ng   ,                &
    qual, lres,                                                    &
    " (", bsz(1), ",", bsz(2), ",", bsz(3), ")"
  write (*,'(a,i0,a)') "Each image has ",icells, " cells"
  write (*,'(a,i0,a)') "The model has ", mcells, " cells"
end if

! allocate space coarray with two layers
! implicit sync all
call cgca_as(1, c(1), 1, c(2), 1, c(3), 1, ir(1), 1, ir(2), 1, 2, space)

! initialise random number seed
call cgca_irs( nodebug )

! allocate rotation tensors
call cgca_art( 1, ng, 1, ir(1), 1, ir(2), 1, grt )

! initialise space
space( :, :, :, cgca_state_type_grain ) = cgca_liquid_state
space( :, :, :, cgca_state_type_frac  ) = cgca_intact_state
```

```
! nuclei, sync all inside
call cgca_nr( space, ng, nodebug )

! assign rotation tensors, sync all inside
call cgca_rt( grt )

! solidify, implicit sync all inside
!subroutine cgca_sld( coarray, periodicbc, iter, heartbeat, solid )
!call cgca_sld( space, noperiodicbc, 0, 10, solid )

! subroutine cgca_sld3( coarray, iter, heartbeat, solid )
call cpu_time(time1)
call cgca_sld3( space, 0, 10, solid )
call cpu_time(time2)
if ( img .eq. 1 ) write (*,*) "time, s", time2-time1

! Stop at solidification
stop

! initiate grain boundaries
call cgca_igb( space )

! smoothen the GB, several iterations, sync needed
! halo exchange, following smoothing
call cgca_gbs( space )
call cgca_hxi( space )
call cgca_gbs( space )
call cgca_hxi( space )
sync all

! update grain connectivity, local routine, no sync needed
call cgca_gcu( space )

! set a single crack nucleus in the centre of the x3=max(x3) face
space( c(1)/2, c(2)/2, c(3), cgca_state_type_frac )  &
        [ ir(1)/2, ir(2)/2, ir(3) ] = cgca_clvg_state_100_edge

! all images start MPI, used only for I/O here, so
! no need to start earlier
!call MPI_Init(ierr)

if ( img .eq. 1 ) write (*,*) "dumping model to files"

  ! dump space arrays to files, only image 1 does it, all others
  ! wait at the barrier, hence sync needed
  call cgca_pswci( space, cgca_state_type_grain, "zg0.raw" )
  call cgca_pswci( space, cgca_state_type_frac,  "zf0.raw" )
  call cgca_swci( space, cgca_state_type_grain, 10, "zg0-ser.raw" )
  call cgca_swci( space, cgca_state_type_frac, 10, "zf0-ser.raw" )

if ( img .eq. 1) write (*,*) "finished dumping model to files"
```

```
sync all

! set the stress tensor
t = 0.0
t(1,1) = 1.0e6
t(2,2) = -1.0e6

! number of cleavage iterations between file dumps
fiter = itot/idmp

! propagate cleavage, sync inside, dump fracture arrays
! every certain number of increments.
do iter = 1, idmp

 ! Propagate cleavage, sync inside
 ! Run for fiter iterations
 ! subroutine cgca_clvgp( coarray, rt, t, scrit, sub, gcus,
 ! periodicbc, iter, heartbeat, debug )
 call cgca_clvgp( space, grt, t, scrit, cgca_clvgsd, cgca_gcupdn,      &
      noperiodicbc, fiter, 10, yesdebug )

 ! dump the fracture space array to files, only image 1 does it,
 ! all others wait at the barrier, hence sync needed.
 ! citer is the total number of cleavage fracture iterations,
 ! "c" for character date type.
 write ( citer, "(i0)" ) iter*fiter
 call cgca_pswci( space, cgca_state_type_frac, &
                  "zf"//trim(citer)//".raw" )
 call cgca_swci( space, cgca_state_type_frac, 10, &
                  "zf"//trim(citer)//"-ser.raw" )

 sync all

 if ( img .eq. 1 ) write (*,"(a)") &
  "Completed "//trim(citer)//" cleavage iterations"

end do

! deallocate all arrays, implicit sync all.
call cgca_ds( space )
call cgca_dgc
call cgca_drt( grt )

! terminate MPI
!call MPI_Finalize(ierr)

if ( img .eq. 1 ) write (*,*) "Test ABV completed sucessfully"

end program testABV
```

123 tests/testABW

[Unit tests]

NAME

testABW

SYNOPSIS

```
!$Id: testABW.f90 526 2018-03-25 23:44:51Z mexas $

program testABW
```

PURPOSE

Check cgca_clvgp_nocosum (26.4) (no CO_SUM version). Cleavage propagation in a model with the given "box" size, mean grain size and the microstructure resolution values. Using parallel I/O here!

DESCRIPTION

First need to call cgca_gdim (21.2), cgca_cadim (21.1) to calculate all parameters of coarray space, including the number of nuclei. Then call solidification and then cleavage.

NOTE

Using 2 cleavage nuclei to see how the two macro-cracks interact.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

cgca testaux (133)

USED BY

Part of CGPACK test suite

SOURCE

```
use testaux

implicit none

logical( kind=ldef ), parameter :: yesdebug=.true., nodebug=.false.,   &
 periodicbc=.true.,  noperiodicbc=.false.

! specify the total number of cleavage propagation iterations,
! and the number of times the fracture array will be dumped along
! the way
integer( kind=idef ), parameter ::                                     &
 itot = 150_idef,       & ! total cleavage iterations to do
 idmp = 3_idef           ! number of dumps to do along the way
```

```
real, parameter :: gigabyte=real(2**30),                                &
! cleavage stress on 100, 110, 111 planes for BCC,
! see the manual for derivation.
 scrit(3) = (/ 1.05e4, 1.25e4, 4.90e4 /)

!integer :: ierr ! default integer needed for MPI

integer( kind=idef ) :: iter, fiter, ir(3), img, nimgs,                 &
  ng,    & ! number of grains in the whole model
  c(3)     ! coarray dimensions
integer( kind=iarr ), allocatable :: space(:,:,:,:) [:,:,:]

real( kind=rdef ) ::    &
 t(3,3),                & ! stress tensor
 qual,                  & ! quality
 bsz0(3),               & ! the given "box" size
 bsz(3),                & ! updated "box" size
 dm,                    & ! mean grain size, linear dim, phys units
 res,                   & ! resolutions, cells per grain
 icells,                & ! total number of cells on each image
 lres                     ! linear resolution, cells per unit of length
real( kind=rdef ), allocatable :: grt(:,:,:)[:,:,:]

logical( kind=ldef ) :: solid
character(6) :: citer

!*********************************************************************72
! first executable statement

! physical dimensions of the box, assume mm
bsz0 = (/ 1.0, 1.0, 1.0 /)

! mean grain size, linear dimension, e.g. mean grain diameter, also mm
dm = 3.0e-1

! resolution
res = 1.0e5

! In this test set the number of images via the env var
! the code must be able to cope with any value >= 1.
   img = this_image()
 nimgs = num_images()

! do a check on image 1
if ( img .eq. 1 ) then

 ! print a banner
 call banner("ABW")

 ! print the parameter values
 call cgca_pdmp
 write (*,'(a,i0,a)') "running on ", nimgs, " images in a 3D grid"
```

```
end if

! want to sync here to make sure the banner is
! printed before the rest.
sync all

! each image calculates the coarray grid dimensions
call cgca_gdim( nimgs, ir, qual )

! calculate the resolution and the actual phys dimensions
! of the box
! subroutine cgca_cadim( bsz, res, dm, ir, c, lres, ng )
! c - coarray sizes
! ir - coarray grid sizes
bsz = bsz0
call cgca_cadim( bsz, res, dm, ir, c, lres, ng )

! total number of cells in a coarray
icells = product( real( c ) )

! dump some stats from image 1
if ( img .eq. 1 ) then
  write ( *, "(9(a,i0), 2(tr1, es9.2), 3(a, es9.2), a)" )             &
    "img: ", img  , " nimgs: ", nimgs, " (", c(1) ,                   &
    ","    , c(2) , ","       , c(3) , ")[", ir(1),                   &
    ","    , ir(2), ","       , ir(3), "] ", ng   ,                   &
    qual, lres,                                                       &
    " (", bsz(1), ",", bsz(2), ",", bsz(3), ")"
  write (*,'(a,es9.2,a)') "Cells on each image: ", icells
  write (*,*) "dataset sizes for ParaView", c*ir
  write (*,"(a, es10.2, a, i0)") "Total cells in the model (real): ",  &
    product( real(c) * real(ir) ), " (int): ",                        &
    product( int(c, kind=ilrg) * int(ir, kind=ilrg) )
end if

! allocate space coarray with two layers
! implicit sync all
call cgca_as(1, c(1), 1, c(2), 1, c(3), 1, ir(1), 1, ir(2), 1, 2, space)

! initialise random number seed
call cgca_irs( nodebug )

! allocate rotation tensors
call cgca_art( 1, ng, 1, ir(1), 1, ir(2), 1, grt )

! initialise space
space( :, :, :, cgca_state_type_grain ) = cgca_liquid_state
space( :, :, :, cgca_state_type_frac  ) = cgca_intact_state

! assign rotation tensors, sync all inside
call cgca_rt( grt )
```

```
! nuclei, sync all inside
call cgca_nr( space, ng, nodebug )

! solidify, implicit sync all inside
call cgca_sld( space, noperiodicbc, 0, 10, solid )

! initiate grain boundaries
call cgca_igb( space )
sync all
call cgca_hxi( space )

! smoothen the GB, several iterations, sync needed
! halo exchange, following smoothing
call cgca_gbs( space )
sync all
call cgca_hxi( space )
sync all
call cgca_gbs( space )
sync all
call cgca_hxi( space )
sync all

! update grain connectivity, local routine, no sync needed
call cgca_gcu( space )

! set a crack nucleus in the centre of the x3=max(x3) face
space( c(1)/2, c(2)/2, c(3), cgca_state_type_frac )  &
        [ ir(1)/2+1, ir(2)/2+1, ir(3) ] = cgca_clvg_state_100_edge

! set a crack nucleus in the centre of the x3=1 face
space( c(1)/2, c(2)/2, 1, cgca_state_type_frac )  &
        [ ir(1)/2+1, ir(2)/2+1, 1 ] = cgca_clvg_state_100_edge

sync all

! all images start MPI, used only for I/O here, so
! no need to start earlier
!call MPI_Init(ierr)

if ( img .eq. 1 ) write (*,*) "dumping model to files"

  ! dump space arrays to files, only image 1 does it, all others
  ! wait at the barrier, hence sync needed
  call cgca_pswci( space, cgca_state_type_grain, "zg0.raw" )
  call cgca_pswci( space, cgca_state_type_frac,  "zf0.raw" )

if ( img .eq. 1) write (*,*) "finished dumping model to files"

sync all

! set the stress tensor
```

```
t = 0.0
t(1,1) = 1.0e6
!t(2,2) = -1.0e6

! number of cleavage iterations between file dumps
fiter = itot/idmp

! propagate cleavage, sync inside, dump fracture arrays
! every certain number of increments.
do iter = 1, idmp

 ! Propagate cleavage, sync inside
 ! Run for fiter iterations
 ! subroutine cgca_clvgp_nocosum( coarray, rt, t, scrit, sub,
 !    gcus, periodicbc, iter, heartbeat, debug )
 call cgca_clvgp_nocosum( space, grt, t, scrit, cgca_clvgsd,           &
      cgca_gcupdn, noperiodicbc, fiter, 10, yesdebug )

 ! dump the fracture space array to files, only image 1 does it,
 ! all others wait at the barrier, hence sync needed.
 ! citer is the total number of cleavage fracture iterations,
 ! "c" for character date type.
 write ( citer, "(i0)" ) iter*fiter
 call cgca_pswci( space, cgca_state_type_frac,                         &
                               "zf"//trim(citer)//".raw" )

 sync all

 if ( img .eq. 1 ) write (*,"(a)") &
  "Completed "//trim(citer)//" cleavage iterations"

end do

! deallocate all arrays, implicit sync all.
call cgca_ds( space )
call cgca_dgc
call cgca_drt( grt )

! terminate MPI
!call MPI_Finalize(ierr)

if ( img .eq. 1 ) write (*,*) "Test ABW completed sucessfully"

end program testABW
```

124 tests/testABX

[Unit tests]

NAME

testABX

SYNOPSIS

```
!$Id: testABX.f90 526 2018-03-25 23:44:51Z mexas $

program testABX
```

PURPOSE

Testing Cray parallel direct access file IO, aka "assign" environment, routine cgca_m2out (19)/cgca_pc (19.3.1).

DESCRIPTION

Verify data integrity and compare timings of cgca_swci (19.2) and cgca_pc (19.3.1) - i.e. a serial writer from image 1 and a parallel direct access shared writer. The latter is a non standard Cray extension. So don't try to run this on non-Cray machines.

NOTE

Compile only on Cray! Don't try to compile with other compilers!

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

cgca testaux (133)

USED BY

Part of CGPACK test suite

SOURCE

```
use testaux

implicit none

real, parameter :: gigabyte=real(2**30)
logical( kind=ldef ), parameter :: yesdebug = .true., nodebug = .false.

real :: time1, time2, fsizeb, fsizeg, tdiff

integer( kind=idef ) :: ir(3), img, nimgs,                              &
  ng,   & ! number of grains in the whole model
  c(3)    ! coarray dimensions
integer( kind=iarr ), allocatable :: space(:,:,:,:) [:,:,:]
```

```
integer( kind=ilrg ) :: icells, mcells

real( kind=rdef ) ::    &
 qual,                  & ! quality
 bsz0(3),               & ! the given "box" size
 bsz(3),                & ! updated "box" size
 dm,                    & ! mean grain size, linear dim, phys units
 lres,                  & ! linear resolution, cells per unit of length
 res                      ! resolutions, cells per grain

logical( kind=ldef ) :: solid

!**********************************************************************72
! first executable statement

! physical dimensions of the box, assume mm
bsz0 = (/ 1.0, 2.0, 3.0 /)

! mean grain size, linear dimension, e.g. mean grain diameter, also mm
dm = 1.0e-1

! resolution
res = 1.0e5

! In this test set the number of images via the env var
! the code must be able to cope with any value >= 1.
   img = this_image()
 nimgs = num_images()

! do a check on image 1
if ( img .eq. 1 ) then
  ! print a banner
  call banner("ABX")
  ! print the parameter values
  call cgca_pdmp
  write (*,'(a,i0,a)') "running on ", nimgs, " images in a 3D grid"
end if

! want to sync here to make sure the banner is
! printed before the rest.
sync all

! each image calculates the coarray grid dimensions
call cgca_gdim( nimgs, ir, qual )

! calculate the resolution and the actual phys dimensions
! of the box
! subroutine cgca_cadim( bsz, res, dm, ir, c, lres, ng )
! c - coarray sizes
! ir - coarray grid sizes
bsz = bsz0
```

```
call cgca_cadim( bsz, res, dm, ir, c, lres, ng )

! total number of cells in a coarray
icells = int( c(1), kind=ilrg ) * int( c(2), kind=ilrg ) *             &
         int( c(3), kind=ilrg )

! total number of cells in the model
mcells = icells * int( nimgs, kind=ilrg )

if ( img .eq. 1 ) then
  write ( *, "(9(a,i0),tr1,g10.3,tr1,i0,3(a,g10.3),a)" )               &
    "img: ", img  , " nimgs: ", nimgs, " (", c(1) ,                    &
    ","    , c(2) , ","       , c(3) , ")[", ir(1),                    &
    ","    , ir(2), ","       , ir(3), "] ", ng   ,                    &
    qual, lres,                                                        &
    " (", bsz(1), ",", bsz(2), ",", bsz(3), ")"
  write (*,'(a,i0,a)') "Each image has ",icells, " cells"
  write (*,'(a,i0,a)') "The model has ", mcells, " cells"
end if

! Total output file size, in B and in GB.
fsizeb = real( mcells * storage_size( space, kind=ilrg ) / 8_ilrg )
fsizeg = fsizeb / gigabyte

! allocate space coarray with a single layer
! implicit sync all
!subroutine cgca_as( l1, u1, l2, u2, l3, u3, col1, cou1, col2, cou2,   &
!                    col3, props, coarray )
call cgca_as(1, c(1), 1, c(2), 1, c(3), 1, ir(1), 1, ir(2), 1, 1, space)

! initialise random number seed
call cgca_irs( nodebug )

! initialise space
space( :, :, :, cgca_state_type_grain ) = cgca_liquid_state

! nuclei, sync all inside
call cgca_nr( space, ng, nodebug )

! solidify with co_sum, implicit sync all inside
! subroutine cgca_sld3( coarray, iter, heartbeat, solid )
call cgca_sld3( space, 0, 10, solid )

! dump the model
call cpu_time(time1)
call cgca_pc( space, cgca_state_type_grain, 'craypario.raw' )
call cpu_time(time2)
tdiff = time2-time1
if (img .eq. 1) then
 write (*,*) "File size:", fsizeg, "GB"
 write (*,*) "Cray assign -m on IO: ", tdiff, "s, rate: ",             &
             fsizeg/tdiff, "GB/s."
```

```
end if

call cpu_time(time1)
call cgca_swci( space, cgca_state_type_grain, 10, 'serial.raw' )
call cpu_time(time2)
tdiff = time2-time1
if (img .eq. 1) write (*,*) "Serial IO: ", tdiff, "s, rate: ",         &
   fsizeg/tdiff, "GB/s."

! deallocate all arrays
call cgca_ds( space )

end program testABX
```

125 tests/testABY

[Unit tests]

NAME

testABY

SYNOPSIS

```
!$Id: testABY.f90 380 2017-03-22 11:03:09Z mexas $

program testABY
```

PURPOSE

Testing routines from the linked list module, cgca_m2lnklst (16).

DESCRIPTION

Verify data integrity and compare timings of cgca_swci (19.2) and cgca_pc (19.3.1) - i.e. a serial writer from image 1 and a parallel direct access shared writer. The latter is a non standard Cray extension. So don't try to run this on non-Cray machines.

NOTE

All routines are serial, so no need to use multiple images. A single image will do.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

cgca testaux (133)

USED BY

Part of CGPACK test suite

SOURCE

```
use testaux
implicit none

type( cgca_lnklst_node ), pointer :: head, tmp
type( cgca_lnklst_tpayld ) :: box
integer :: img, i, stat

! Only image 1 works
img = this_image()
main: if ( img .eq. 1 ) then

! print a banner
call banner("ABY")
```

```
! initialise head
box%lwr = (/ 1, 2, 3 /)
box%upr = (/ 4, 5, 6 /)
call cgca_inithead( head, box )

! dump the list
write (*,*) "The list after initialising head"
call cgca_lstdmp( head )

! add several nodes on top of head
box%lwr = (/ 7, 8, 9 /)
box%upr = (/ 10, 11, 12 /)
call cgca_addhead( head, box )
box%lwr = (/ 13, 14, 15 /)
box%upr = (/ 16, 17, 18 /)
call cgca_addhead( head, box )
box%lwr = (/ 19, 20, 21 /)
box%upr = (/ 22, 23, 24 /)
call cgca_addhead( head, box )
box%lwr = (/ 25, 26, 27 /)
box%upr = (/ 28, 29, 30 /)
call cgca_addhead( head, box )
box%lwr = (/ 31, 32, 33 /)
box%upr = (/ 34, 35, 36 /)
call cgca_addhead( head, box )

! dump the list
write (*,*) "The list after adding 5 nodes on top of head"
call cgca_lstdmp( head )

! add few nodes 3 levels lower than head
tmp => head
do i=1,2
  tmp => tmp%next
end do
box%lwr = (/ 101, 102, 103 /)
box%upr = (/ 104, 105, 106 /)
call cgca_addmiddle( tmp, box )
box%lwr = (/ 107, 108, 109 /)
box%upr = (/ 110, 111, 112 /)
call cgca_addmiddle( tmp, box )
box%lwr = (/ 113, 114, 115 /)
box%upr = (/ 116, 117, 118 /)
call cgca_addmiddle( tmp, box )
box%lwr = (/ 119, 120, 121 /)
box%upr = (/ 122, 123, 124 /)
call cgca_addmiddle( tmp, box )

! dump the list
write (*,*) "The list after adding 4 nodes 3 levels lower than head"
call cgca_lstdmp( head )
```

```
! remove the head node several times
do i=1,2
  call cgca_rmhead( head, stat )
  if ( stat .eq. 1 ) then
    write (*,*) "Reached NULL"
    exit
  end if
end do

! dump the list
write (*,*) "Removed two head nodes, now the list is:"
call cgca_lstdmp( head )

! remove few middle nodes 3 levels below head
tmp => head
do i=1,2
  tmp => tmp%next
end do
do i = 1,3
  call cgca_rmmiddle( tmp, stat )
  if ( stat .eq. 1 ) write (*,*) "WARN: cgca_rmmiddle: Reached NULL"
end do

! dump the list
write (*,*) "Removed 3 middle nodes 3 levels below head, the new list:"
call cgca_lstdmp( head )

! continue removing till NULL has been reached
write (*,*) "Continue removing from that point, till NULL has been reached."
remove: do
  call cgca_rmmiddle( tmp, stat )
!write (*,*) "stat: ", stat
  if ( stat .eq. 1 ) then
    write (*,*) "Reached NULL"
    exit remove
  end if
end do remove

! dump the list
write (*,*) "The list after removing all nodes till NULL"
call cgca_lstdmp( head )

! add 3 more nodes on top of head
box%lwr = (/ 2201, 2201, 2201 /)
box%upr = (/ 2333, 2333, 2333 /)
call cgca_addhead( head, box )
box%lwr = (/ 3201, 3201, 3201 /)
box%upr = (/ 3333, 3333, 3333 /)
call cgca_addhead( head, box )
box%lwr = (/ 4201, 4201, 4201 /)
box%upr = (/ 4333, 4333, 4333 /)
call cgca_addhead( head, box )
```

```
! dump the list
write (*,*) "The list after adding 3 more nodes on top of head"
call cgca_lstdmp( head )

! remove all head nodes till NULL has been reached
write (*,*) "Removing all head nodes"
do
  call cgca_rmhead( head, stat )
  if ( stat .eq. 1 ) then
    write (*,*) "Reached NULL. associated( head ) = ", associated( head )
    exit
  end if
  if ( .not. associated( head%next ) ) then
    write (*,*) "The list when head%next is not associated, just a single node left"
    call cgca_lstdmp( head )
  end if
end do

! dump the list
write (*,*) "The list after removing all head nodes"
call cgca_lstdmp( head )

end if main

end program testABY
```

126 tests/testABZ

[Unit tests]

NAME

testABZ

SYNOPSIS

```
!$Id: testABZ.f90 380 2017-03-22 11:03:09Z mexas $

program testABZ
```

PURPOSE

Testing cgca_boxsplit (12.1) from module cgca_m2geom (12).

DESCRIPTION

cgca_boxsplit (12.1) splits a box, given by 2 corner coordinates, all integers, into two smaller boxes, along the biggest dimension of the old box.

NOTE

Serial routine, a single image will do.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

cgca testaux (133)

USED BY

Part of CGPACK test suite

SOURCE

```
use testaux
implicit none

integer( kind=idef) :: lwr(3), upr(3), lwr1(3), upr1(3), lwr2(3),      &
   upr2(3), img, i

! Only image 1 works
img = this_image()
main: if ( img .eq. 1 ) then

! print a banner
call banner("ABZ")

! set some values
lwr = (/ 1, 1, 1 /)
```

```
upr = (/ 123, 456, 789 /)

call cgca_boxsplit( lwr, upr, lwr1, upr1, lwr2, upr2 )
write (*,101) "lwr: ", lwr, "upr:", upr
write (*,*) "split into:"
write (*,101) "lwr1: ", lwr1, "upr1:", upr1
write (*,101) "lwr2: ", lwr2, "upr2:", upr2

do i=1,100

  if ( mod(i,2) .eq. 0 ) then
    ! even i
    write (*,*) "choose box 1"
    lwr = lwr1
    upr = upr1
  else
    ! odd i
    write (*,*) "choose box 2"
    lwr = lwr2
    upr = upr2
  end if

  call cgca_boxsplit( lwr, upr, lwr1, upr1, lwr2, upr2 )
  write (*,101) "lwr: ", lwr, "upr:", upr
  write (*,*) "split into:"
  write (*,101) "lwr1: ", lwr1, "upr1:", upr1
  write (*,101) "lwr2: ", lwr2, "upr2:", upr2

  if ( all( lwr1 .eq. upr1 ) .and. all( lwr2 .eq. upr2 ) ) exit

end do

end if main

101 format (2(a5,3(i5),tr1))

end program testABZ
```

127 tests/testACA

[Unit tests]

NAME

testACA

SYNOPSIS

```
!$Id: testACA.f90 526 2018-03-25 23:44:51Z mexas $

program testACA
```

PURPOSE

Testing cgca_m2gl/cgca_ico (13.2).

DESCRIPTION

cgca_ico (13.2) converts some image index into its cosubscripts. Lots of data has to be set prior to calling cgca_ico (13.2). The test can be run on any number of images. space coarray is established and the cgca_ico (13.2) is called using its cosubscripts.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

cgca testaux (133)

USED BY

Part of CGPACK test suite

SOURCE

```
use testaux

implicit none

integer( kind=idef ) :: ir( cgca_scodim ), img, nimgs, flag, ind,      &
 ng,      & ! number of grains in the whole model
 cosub( cgca_scodim ), c( cgca_scodim )  ! coarray dimensions

integer( kind=iarr ), allocatable :: space(:,:,:,:) [:,:,:]

integer( kind=ilrg ) :: icells, mcells

real( kind=rdef ) ::    &
 qual,                  & ! quality
 bsz0(3),               & ! the given "box" size
 bsz(3),                & ! updated "box" size
 dm,                    & ! mean grain size, linear dim, phys units
```

```
 lres,                  & ! linear resolution, cells per unit of length
 res                      ! resolutions, cells per grain

!*****************************************************************72
! first executable statement

! physical dimensions of the box, assume mm
bsz0 = (/ 1.0, 2.0, 3.0 /)

! mean grain size, linear dimension, e.g. mean grain diameter, also mm
dm = 5.0e-1

! resolution
res = 1.0e5

! In this test set the number of images via the env var
! the code must be able to cope with any value >= 1.
   img = this_image()
 nimgs = num_images()

! do a check on image 1
if ( img .eq. 1 ) then
  ! print a banner
  call banner("ACA")
  ! print the parameter values
  call cgca_pdmp
  write (*,'(a,i0,a)') "running on ", nimgs, " images in a 3D grid"
end if

! want to sync here to make sure the banner is
! printed before the rest.
sync all

! each image calculates the coarray grid dimensions
call cgca_gdim( nimgs, ir, qual )

! calculate the resolution and the actual phys dimensions
! of the box
! subroutine cgca_cadim( bsz, res, dm, ir, c, lres, ng )
! c - coarray sizes
! ir - coarray grid sizes
bsz = bsz0
call cgca_cadim( bsz, res, dm, ir, c, lres, ng )

! total number of cells in a coarray
icells = int( c(1), kind=ilrg ) * int( c(2), kind=ilrg ) *             &
         int( c(3), kind=ilrg )

! total number of cells in the model
mcells = icells * int( nimgs, kind=ilrg )

if ( img .eq. 1 ) then
```

```
  write ( *, "(9(a,i0),tr1,g10.3,tr1,g10.3,3(a,g10.3),a)" )           &
    "img: ", img  , " nimgs: ", nimgs, " (", c(1) ,                   &
    ","    , c(2) , ","       , c(3) , ")[", ir(1),                   &
    ","    , ir(2), ","       , ir(3), "] ", ng   ,                   &
    qual, lres,                                                       &
    " (", bsz(1), ",", bsz(2), ",", bsz(3), ")"
  write (*,'(a,i0,a)') "Each image has ",icells, " cells"
  write (*,'(a,i0,a)') "The model has ", mcells, " cells"
end if

! allocate space coarray with a single layer
! implicit sync all
!subroutine cgca_as( l1, u1, l2, u2, l3, u3, col1, cou1, col2, cou2,   &
!                    col3, props, coarray )
call cgca_as(1, c(1), 1, c(2), 1, c(3), 1, ir(1), 1, ir(2), 1, 1, space)

! check cgca_ico on the last image
! make only image 1 do this.
if ( img .eq. 1 ) then

  ! Dump lower and upper cobounds
  write (*,*) "cgca_slcob:", cgca_slcob
  write (*,*) "cgca_sucob:", cgca_sucob

  ! Calculate subscripts for all indices and check
  do ind = 1, num_images()
    call cgca_ico( ind, cosub, flag )
    if ( flag .ne. 0 ) then
      write (*,*) "ERROR: cgca_ico returned flag:", flag
      stop
    end if

    write (*,*) "index=", ind, "cosub:", cosub

    ! check
    if ( image_index( space, cosub ) .ne. ind ) then
      write (*,*) "ERROR: testACA failed: cgca_ico calculated wrong values"
    end if

  end do

end if

sync all

! deallocate all arrays
call cgca_ds( space )

end program testACA
```

128 tests/testACB

[Unit tests]

NAME

testACB

SYNOPSIS

```
!$Id: testACB.f90 526 2018-03-25 23:44:51Z mexas $

program testACB
```

PURPOSE

Testing cgca_m2gl/cgca_ico2 (13.3).

DESCRIPTION

cgca_ico2 (13.3) converts some image index into its cosubscripts. Lots of data has to be set prior to calling cgca_ico2 (13.3). The test can be run on any number of images. space coarray is established and the cgca_ico2 (13.3) is called using its cosubscripts.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

cgca testaux (133)

USED BY

Part of CGPACK test suite

SOURCE

```
use testaux

implicit none

integer( kind=idef ) :: ir( cgca_scodim ), img, nimgs, ind,           &
 ng,      & ! number of grains in the whole model
 cosub( cgca_scodim ), c( cgca_scodim )  ! coarray dimensions

integer( kind=iarr ), allocatable :: space(:,:,:,:) [:,:,:]

integer( kind=ilrg ) :: icells, mcells

real( kind=rdef ) ::    &
 qual,                  & ! quality
 bsz0(3),               & ! the given "box" size
 bsz(3),                & ! updated "box" size
 dm,                    & ! mean grain size, linear dim, phys units
```

```
 lres,                  & ! linear resolution, cells per unit of length
 res                      ! resolutions, cells per grain

!*****************************************************************72
! first executable statement

! physical dimensions of the box, assume mm
bsz0 = (/ 1.0, 2.0, 3.0 /)

! mean grain size, linear dimension, e.g. mean grain diameter, also mm
dm = 5.0e-1

! resolution
res = 1.0e5

! In this test set the number of images via the env var
! the code must be able to cope with any value >= 1.
   img = this_image()
 nimgs = num_images()

! do a check on image 1
if ( img .eq. 1 ) then
  ! print a banner
  call banner("ACB")
  ! print the parameter values
  call cgca_pdmp
  write (*,'(a,i0,a)') "running on ", nimgs, " images in a 3D grid"
end if

! want to sync here to make sure the banner is
! printed before the rest.
sync all

! each image calculates the coarray grid dimensions
call cgca_gdim( nimgs, ir, qual )

! calculate the resolution and the actual phys dimensions
! of the box
! subroutine cgca_cadim( bsz, res, dm, ir, c, lres, ng )
! c - coarray sizes
! ir - coarray grid sizes
bsz = bsz0
call cgca_cadim( bsz, res, dm, ir, c, lres, ng )

! total number of cells in a coarray
icells = int( c(1), kind=ilrg ) * int( c(2), kind=ilrg ) *             &
         int( c(3), kind=ilrg )

! total number of cells in the model
mcells = icells * int( nimgs, kind=ilrg )

if ( img .eq. 1 ) then
```

```
  write ( *, "(9(a,i0),tr1,g10.3,tr1,g10.3,3(a,g10.3),a)" )           &
    "img: ", img  , " nimgs: ", nimgs, " (", c(1) ,                   &
    ","    , c(2) , ","       , c(3) , ")[", ir(1),                   &
    ","    , ir(2), ","       , ir(3), "] ", ng   ,                   &
    qual, lres,                                                       &
    " (", bsz(1), ",", bsz(2), ",", bsz(3), ")"
  write (*,'(a,i0,a)') "Each image has ",icells, " cells"
  write (*,'(a,i0,a)') "The model has ", mcells, " cells"
end if

! allocate space coarray with a single layer
! implicit sync all
!subroutine cgca_as( l1, u1, l2, u2, l3, u3, col1, cou1, col2, cou2,   &
!                    col3, props, coarray )
call cgca_as(1, c(1), 1, c(2), 1, c(3), 1, ir(1), 1, ir(2), 1, 1, space)

! check cgca_ico2 on the last image
! make only image 1 do this.
if ( img .eq. 1 ) then

  ! Dump lower and upper cobounds
  write (*,*) "cgca_slcob:", cgca_slcob
  write (*,*) "cgca_sucob:", cgca_sucob

  ! Calculate subscripts for all indices and check
  do ind = 1, num_images()
    call cgca_ico2( cgca_slcob, cgca_sucob, ind, cosub )

    write (*,*) "index=", ind, "cosub:", cosub

    ! check
    if ( image_index( space, cosub ) .ne. ind ) then
      write (*,*) "ERROR: testACB failed: cgca_ico2 calculated wrong values"
    end if

  end do

end if

sync all

! deallocate all arrays
call cgca_ds( space )

end program testACB
```

129 tests/testACC

[Unit tests]

NAME

testACC

SYNOPSIS

```
!$Id: testACC.f90 526 2018-03-25 23:44:51Z mexas $

program testACC
```

PURPOSE

Test the reproducible random seed routine cgca_ins (23.1). The idea is that if the number of images is kept constant, this program should produce the same fracture results on every run.

DESCRIPTION

Cleavage propagation in a model with the given "box" size, mean grain size and the microstructure resolution values. MPI I/O is used for writing out grain and crack arrays. First need to call cgca_gdim (21.2), cgca_cadim (21.1) to calculate all parameters of coarray space, including the number of nuclei. Then call solidification and then cleavage.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

cgca testaux (133)

USED BY

Part of CGPACK test suite

SOURCE

```
use testaux

implicit none

logical( kind=ldef ), parameter :: yesdebug=.true., nodebug=.false.,   &
 periodicbc=.true.,  noperiodicbc=.false.

! specify the total number of cleavage propagation iterations,
! and the number of times the fracture array will be dumped along
! the way
integer( kind=idef ), parameter ::                                     &
 itot = 300_idef,       & ! total cleavage iterations to do
 idmp = 10_idef           ! number of dumps to do along the way

real, parameter :: gigabyte=real(2**30),                               &
```

```
! cleavage stress on 100, 110, 111 planes for BCC,
! see the manual for derivation.
 scrit(3) = (/ 1.05e4, 1.25e4, 4.90e4 /)

! integer :: ierr ! default integer needed for MPI

integer( kind=idef ) :: iter, fiter, ir(3), img, nimgs,                &
  ng,   & ! number of grains in the whole model
  c(3)    ! coarray dimensions

integer( kind=iarr ), allocatable :: space(:,:,:,:) [:,:,:]

integer( kind=ilrg ) :: icells, mcells

real( kind=rdef ) ::    &
 t(3,3),                & ! stress tensor
 qual,                  & ! quality
 bsz0(3),               & ! the given "box" size
 bsz(3),                & ! updated "box" size
 dm,                    & ! mean grain size, linear dim, phys units
 lres,                  & ! linear resolution, cells per unit of length
 res                      ! resolutions, cells per grain
real( kind=rdef ), allocatable :: grt(:,:,:)[:,:,:]

logical( kind=ldef ) :: solid
character(6) :: citer

!*********************************************************************72
! first executable statement

! physical dimensions of the box, assume mm
bsz0 = (/ 1.0, 1.0, 1.0 /)

! mean grain size, linear dimension, e.g. mean grain diameter, also mm
dm = 3.0e-1

! resolution
res = 1.0e5

! In this test set the number of images via the env var
! the code must be able to cope with any value >= 1.
   img = this_image()
 nimgs = num_images()

! do a check on image 1
if ( img .eq. 1 ) then

 ! print a banner
 call banner("ACC")

 ! print the parameter values
 call cgca_pdmp
```

```
 write (*,'(a,i0,a)') "running on ", nimgs, " images in a 3D grid"

end if

! want to sync here to make sure the banner is
! printed before the rest.
sync all

! each image calculates the coarray grid dimensions
call cgca_gdim( nimgs, ir, qual )

! calculate the resolution and the actual phys dimensions
! of the box
! subroutine cgca_cadim( bsz, res, dm, ir, c, lres, ng )
! c - coarray sizes
! ir - coarray grid sizes
bsz = bsz0
call cgca_cadim( bsz, res, dm, ir, c, lres, ng )

! total number of cells in a coarray
icells = int( c(1), kind=ilrg ) *    &
         int( c(2), kind=ilrg ) *    &
         int( c(3), kind=ilrg )

! total number of cells in the model
mcells = icells * int( nimgs, kind=ilrg )

if ( img .eq. 1 ) then
  write ( *, "(9(a,i0), 2(tr1, es9.2), 3(a, es9.2), a)" )              &
    "img: ", img  , " nimgs: ", nimgs, " (", c(1) ,                    &
    ","    , c(2) , ","       , c(3) , ")[", ir(1),                    &
    ","    , ir(2), ","       , ir(3), "] ", ng   ,                    &
    qual, lres,                                                        &
    " (", bsz(1), ",", bsz(2), ",", bsz(3), ")"
  write (*,*) "dataset sizes for ParaView", c*ir
  write (*,'(a,i0)') "Cells on each image: ", icells
  write (*,"(a,es10.2)") "Total cells in the model",                   &
              product( real(c) * real(ir) )
end if

! allocate space coarray with two layers
! implicit sync all
call cgca_as(1, c(1), 1, c(2), 1, c(3), 1, ir(1), 1, ir(2), 1, 2, space)

! initialise the reproducible random number seed
call cgca_ins( yesdebug )

! allocate rotation tensors
call cgca_art( 1, ng, 1, ir(1), 1, ir(2), 1, grt )

! initialise space
space( :, :, :, cgca_state_type_grain ) = cgca_liquid_state
```

```
space( :, :, :, cgca_state_type_frac  ) = cgca_intact_state

! nuclei, sync all inside
call cgca_nr( space, ng, yesdebug )

! assign rotation tensors, sync all inside
call cgca_rt( grt )

! solidify, implicit sync all inside
!subroutine cgca_sld( coarray, periodicbc, iter, heartbeat, solid )
call cgca_sld( space, noperiodicbc, 0, 10, solid )

! initiate grain boundaries
call cgca_igb( space )

! smoothen the GB, several iterations, sync needed
! halo exchange, following smoothing
call cgca_gbs( space )
sync all
call cgca_hxi( space )
sync all
call cgca_gbs( space )
sync all
call cgca_hxi( space )
sync all

! update grain connectivity, local routine, no sync needed
call cgca_gcu( space )

! set a single crack nucleus in the centre of the x3=max(x3) face
space( c(1)/2, c(2)/2, c(3), cgca_state_type_frac )  &
        [ ir(1)/2, ir(2)/2, ir(3) ] = cgca_clvg_state_100_edge

! all images start MPI, used only for I/O here, so
! no need to start earlier
!call MPI_Init(ierr)

if ( img .eq. 1 ) write (*,*) "dumping model to files"

  ! dump space arrays to files, only image 1 does it, all others
  ! wait at the barrier, hence sync needed
  call cgca_swci( space, cgca_state_type_grain, 10, "zg0.raw" )
  call cgca_swci( space, cgca_state_type_frac,  10, "zf0.raw" )

if ( img .eq. 1) write (*,*) "finished dumping model to files"

sync all

! set the stress tensor
t = 0.0
t(1,1) = 1.0e6
t(2,2) = -1.0e6
```

```
! number of cleavage iterations between file dumps
fiter = itot/idmp

! propagate cleavage, sync inside, dump fracture arrays
! every certain number of increments.
do iter = 1, idmp

 ! Propagate cleavage, sync inside
 ! Run for fiter iterations
 ! subroutine cgca_clvgp( coarray, rt, t, scrit, sub, gcus,
 ! periodicbc, iter, heartbeat, debug )
 call cgca_clvgp( space, grt, t, scrit, cgca_clvgsd, cgca_gcupdn,      &
      noperiodicbc, fiter, 10, yesdebug )

 ! dump the fracture space array to files, only image 1 does it,
 ! all others wait at the barrier, hence sync needed.
 ! citer is the total number of cleavage fracture iterations,
 ! "c" for character date type.
 write ( citer, "(i0)" ) iter*fiter
 call cgca_swci( space, cgca_state_type_frac, 10, &
                 "zf"//trim(citer)//".raw" )

 sync all

 if ( img .eq. 1 ) write (*,"(a)") &
  "Completed "//trim(citer)//" cleavage iterations"

end do

! deallocate all arrays, implicit sync all.
call cgca_ds( space )
call cgca_dgc
call cgca_drt( grt )

! terminate MPI
!call MPI_Finalize(ierr)

if ( img .eq. 1 ) write (*,*) "Test ACC completed sucessfully"

end program testACC
```

130 tests/testACD

[Unit tests]

NAME

testACD

SYNOPSIS

```
!$Id: testACD.f90 526 2018-03-25 23:44:51Z mexas $

program testACD
```

PURPOSE

Test the reproducible random seed routine cgca_ins (23.1). The idea is that if the number of images is kept constant, this program should produce the same fracture results on every run.

DESCRIPTION

Cleavage propagation in a model with the given "box" size, mean grain size and the microstructure resolution values. MPI I/O is used for writing out grain and crack arrays. First need to call cgca_gdim (21.2), cgca_cadim (21.1) to calculate all parameters of coarray space, including the number of nuclei. Then call solidification and then cleavage.

NOTE

This test is for platforms which do not support collectives yet, e.g. Intel 16. This test uses cgca_clvgp_nocosum (26.4).

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

cgca testaux (133)

USED BY

Part of CGPACK test suite

SOURCE

```
use testaux

implicit none

logical( kind=ldef ), parameter :: yesdebug=.true., nodebug=.false.,   &
 periodicbc=.true.,  noperiodicbc=.false.

! specify the total number of cleavage propagation iterations,
! and the number of times the fracture array will be dumped along
! the way
integer( kind=idef ), parameter ::                                     &
```

```
 itot = 300_idef,       & ! total cleavage iterations to do
 idmp = 10_idef           ! number of dumps to do along the way

real, parameter :: gigabyte=real(2**30),                              &
! cleavage stress on 100, 110, 111 planes for BCC,
! see the manual for derivation.
 scrit(3) = (/ 1.05e4, 1.25e4, 4.90e4 /)

!integer :: ierr ! default integer needed for MPI

integer( kind=idef ) :: iter, fiter, ir(3), img, nimgs,               &
 ng,    & ! number of grains in the whole model
 c(3)     ! coarray dimensions

integer( kind=iarr ), allocatable :: space(:,:,:,:) [:,:,:]

integer( kind=ilrg ) :: icells, mcells

real( kind=rdef ) ::    &
 t(3,3),                & ! stress tensor
 qual,                  & ! quality
 bsz0(3),               & ! the given "box" size
 bsz(3),                & ! updated "box" size
 dm,                    & ! mean grain size, linear dim, phys units
 lres,                  & ! linear resolution, cells per unit of length
 res                      ! resolutions, cells per grain
real( kind=rdef ), allocatable :: grt(:,:,:)[:,:,:]

logical( kind=ldef ) :: solid
character(6) :: citer

!*********************************************************************72
! first executable statement

! physical dimensions of the box, assume mm
bsz0 = (/ 1.0, 1.0, 1.0 /)

! mean grain size, linear dimension, e.g. mean grain diameter, also mm
dm = 5.0e-1

! resolution
res = 1.0e5

! In this test set the number of images via the env var
! the code must be able to cope with any value >= 1.
   img = this_image()
 nimgs = num_images()

! do a check on image 1
if ( img .eq. 1 ) then

 ! print a banner
```

```
 call banner("ACD")

 ! print the parameter values
 call cgca_pdmp
 write (*,'(a,i0,a)') "running on ", nimgs, " images in a 3D grid"

end if

! want to sync here to make sure the banner is
! printed before the rest.
sync all

! each image calculates the coarray grid dimensions
call cgca_gdim( nimgs, ir, qual )

! calculate the resolution and the actual phys dimensions
! of the box
! subroutine cgca_cadim( bsz, res, dm, ir, c, lres, ng )
! c - coarray sizes
! ir - coarray grid sizes
bsz = bsz0
call cgca_cadim( bsz, res, dm, ir, c, lres, ng )

! total number of cells in a coarray
icells = int( c(1), kind=ilrg ) *    &
         int( c(2), kind=ilrg ) *    &
         int( c(3), kind=ilrg )

! total number of cells in the model
mcells = icells * int( nimgs, kind=ilrg )

if ( img .eq. 1 ) then
  write ( *, "(9(a,i0), 2(tr1, es9.2), 3(a, es9.2), a)" )              &
    "img: ", img  , " nimgs: ", nimgs, " (", c(1) ,                    &
    ","    , c(2) , ","       , c(3) , ")[", ir(1),                    &
    ","    , ir(2), ","       , ir(3), "] ", ng   ,                    &
    qual, lres,                                                        &
    " (", bsz(1), ",", bsz(2), ",", bsz(3), ")"
  write (*,*) "dataset sizes for ParaView", c*ir
  write (*,'(a,i0)') "Cells on each image: ", icells
  write (*,"(a, es10.2, a, i0)") "Total cells in the model (real): ",  &
    product( real(c) * real(ir) ), " (int): ", mcells
end if

! allocate space coarray with two layers
! implicit sync all
call cgca_as(1, c(1), 1, c(2), 1, c(3), 1, ir(1), 1, ir(2), 1, 2, space)

! initialise the reproducible random number seed
call cgca_ins( yesdebug )

! allocate rotation tensors
```

```
call cgca_art( 1, ng, 1, ir(1), 1, ir(2), 1, grt )

! initialise space
space( :, :, :, cgca_state_type_grain ) = cgca_liquid_state
space( :, :, :, cgca_state_type_frac  ) = cgca_intact_state

sync all

! nuclei, sync all inside
call cgca_nr( space, ng, yesdebug )

! assign rotation tensors, sync all inside
call cgca_rt( grt )

! solidify, implicit sync all inside
!subroutine cgca_sld( coarray, periodicbc, iter, heartbeat, solid )
call cgca_sld( space, noperiodicbc, 0, 10, solid )

! initiate grain boundaries
call cgca_igb( space )
sync all
call cgca_hxi( space )

! smoothen the GB, several iterations, sync needed
! halo exchange, following smoothing
call cgca_gbs( space )
sync all
call cgca_hxi( space )
sync all
call cgca_gbs( space )
sync all
call cgca_hxi( space )
sync all

! update grain connectivity, local routine, no sync needed
call cgca_gcu( space )

! set a single crack nucleus in the centre of the x3=max(x3) face
space( c(1)/2, c(2)/2, c(3), cgca_state_type_frac )  &
        [ ir(1)/2, ir(2)/2, ir(3) ] = cgca_clvg_state_100_edge

! all images start MPI, used only for I/O here, so
! no need to start earlier
!call MPI_Init(ierr)

if ( img .eq. 1 ) write (*,*) "dumping model to files"

  ! dump space arrays to files, only image 1 does it, all others
  ! wait at the barrier, hence sync needed
  call cgca_pswci( space, cgca_state_type_grain, "zg0.raw" )
  call cgca_pswci( space, cgca_state_type_frac,  "zf0.raw" )
```

```
if ( img .eq. 1) write (*,*) "finished dumping model to files"

sync all

! set the stress tensor
t = 0.0
t(1,1) = 1.0e6
t(2,2) = -1.0e6

! number of cleavage iterations between file dumps
fiter = itot/idmp

! propagate cleavage, sync inside, dump fracture arrays
! every certain number of increments.
do iter = 1, idmp

 ! Propagate cleavage, sync inside
 ! Run for fiter iterations
 ! subroutine cgca_clvgp( coarray, rt, t, scrit, sub, gcus,
 ! periodicbc, iter, heartbeat, debug )
 call cgca_clvgp_nocosum( space, grt, t, scrit, cgca_clvgsd,           &
   cgca_gcupdn, noperiodicbc, fiter, 10, yesdebug )

 ! dump the fracture space array to files, only image 1 does it,
 ! all others wait at the barrier, hence sync needed.
 ! citer is the total number of cleavage fracture iterations,
 ! "c" for character date type.
 write ( citer, "(i0)" ) iter*fiter
 call cgca_pswci( space, cgca_state_type_frac, &
                  "zf"//trim(citer)//".raw" )

 sync all

 if ( img .eq. 1 ) write (*,"(a)") &
  "Completed "//trim(citer)//" cleavage iterations"

end do

! deallocate all arrays, implicit sync all.
call cgca_ds( space )
call cgca_dgc
call cgca_drt( grt )

! terminate MPI
!call MPI_Finalize(ierr)

if ( img .eq. 1 ) write (*,*) "Test ACD completed sucessfully"

end program testACD
```

131 tests/testACE

[Unit tests]

NAME

testACE

SYNOPSIS

```
!$Id: testACE.f90 526 2018-03-25 23:44:51Z mexas $

program testACE
```

PURPOSE

Test cgca_fwci (19.1)/cgca_m2out (19) - a debugging output routine.

DESCRIPTION

cgca_fwci (19.1)/cgca_m2out (19) collects coarray data from all images and writes it out from image 1 in a formatted text file. A single line contains the state of a single cells with its full local global coordinates, i.e. its position in the coarray on this image, and the image position in the coarray grid. The data is dumped out as soon as the solidificaiton is finished.

Both grain and fracture layers are used and dumped.

NOTE

The program uses cgca_ins (23.1) RND seed, to obtain a reproducible model data. If this test is re-run on the same platform with the same number of images, the results must be the same.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

cgca testaux (133)

USED BY

Part of CGPACK test suite

SOURCE

```
use testaux

implicit none

logical( kind=ldef ), parameter :: yesdebug=.true., nodebug=.false.,   &
 periodicbc=.true.,  noperiodicbc=.false.

real, parameter :: gigabyte=real(2**30)

integer( kind=idef ) :: ir(3), img, nimgs,                             &
  ng,  & ! number of grains in the whole model
```

```
  c(3)   ! coarray dimensions

integer( kind=iarr ), allocatable :: space(:,:,:,:) [:,:,:]

integer( kind=ilrg ) :: icells, mcells

real( kind=rdef ) ::    &
 qual,                  & ! quality
 bsz0(3),               & ! the given "box" size
 bsz(3),                & ! updated "box" size
 dm,                    & ! mean grain size, linear dim, phys units
 lres,                  & ! linear resolution, cells per unit of length
 res                      ! resolutions, cells per grain
real( kind=rdef ), allocatable :: grt(:,:,:)[:,:,:]

logical( kind=ldef ) :: solid

!*********************************************************************72
! first executable statement

! physical dimensions of the box, assume mm
bsz0 = (/ 1.0, 1.0, 1.0 /)

! mean grain size, linear dimension, e.g. mean grain diameter, also mm
dm = 5.0e-1

! resolution
res = 1.0e5

! In this test set the number of images via the env var
! the code must be able to cope with any value >= 1.
   img = this_image()
 nimgs = num_images()

! do a check on image 1
if ( img .eq. 1 ) then

 ! print a banner
 call banner("ACE")

 ! print the parameter values
 call cgca_pdmp
 write (*,'(a,i0,a)') "running on ", nimgs, " images in a 3D grid"

end if

! want to sync here to make sure the banner is
! printed before the rest.
sync all

! each image calculates the coarray grid dimensions
call cgca_gdim( nimgs, ir, qual )
```

```
! calculate the resolution and the actual phys dimensions
! of the box
! subroutine cgca_cadim( bsz, res, dm, ir, c, lres, ng )
! c - coarray sizes
! ir - coarray grid sizes
bsz = bsz0
call cgca_cadim( bsz, res, dm, ir, c, lres, ng )

! total number of cells in a coarray
icells = int( c(1), kind=ilrg ) *    &
         int( c(2), kind=ilrg ) *    &
         int( c(3), kind=ilrg )

! total number of cells in the model
mcells = icells * int( nimgs, kind=ilrg )

if ( img .eq. 1 ) then
  write ( *, "(9(a,i0), 2(tr1, es9.2), 3(a, es9.2), a)" )              &
    "img: ", img  , " nimgs: ", nimgs, " (", c(1) ,                    &
    ","    , c(2) , ","       , c(3) , ")[", ir(1),                    &
    ","    , ir(2), ","       , ir(3), "] ", ng   ,                    &
    qual, lres,                                                        &
    " (", bsz(1), ",", bsz(2), ",", bsz(3), ")"
  write (*,*) "dataset sizes for ParaView", c*ir
  write (*,'(a,i0)') "Cells on each image: ", icells
  write (*,"(a, es10.2, a, i0)") "Total cells in the model (real): ",  &
    product( real(c) * real(ir) ), " (int): ", mcells
end if

! allocate space coarray with two layers, implicit SYNC ALL inside
call cgca_as(1, c(1), 1, c(2), 1, c(3), 1, ir(1), 1, ir(2), 1, 2, space)

! initialise the reproducible random number seed
call cgca_ins( yesdebug )

! Set initial values to all layers of space
space( :, :, :, cgca_state_type_grain ) = cgca_liquid_state
space( :, :, :, cgca_state_type_frac  ) = cgca_intact_state

! Allocate rotation tensors, implicit SYNC ALL inside
call cgca_art( 1, ng, 1, ir(1), 1, ir(2), 1, grt )

! Image 1 sets crack nuclei
if ( img .eq. 1 ) then

  ! set a single crack nucleus in the centre of the x3=max(x3) face
  space( c(1)/2, c(2)/2, c(3), cgca_state_type_frac )                  &
        [ ir(1)/2, ir(2)/2, ir(3) ] = cgca_clvg_state_100_edge
end if

! Set grain nuclei, SYNC ALL inside
```

```
call cgca_nr( space, ng, yesdebug )

! assign rotation tensors, SYNC ALL inside
call cgca_rt( grt )

! solidify, implicit SYNC ALL inside
!subroutine cgca_sld( coarray, periodicbc, iter, heartbeat, solid )
call cgca_sld( space, noperiodicbc, 0, 10, solid )

! Initiate grain boundaries. cgca_igb has no remote comms. Halo
! exchange is needed to update other images.
call cgca_igb( space )
sync all
call cgca_hxi( space )
sync all

! Smoothen the GB, several iterations.
! cgca_gbs has no remote comms.
call cgca_gbs( space )
sync all
call cgca_hxi( space )
sync all
call cgca_gbs( space )
sync all
call cgca_hxi( space )
sync all

if ( img .eq. 1 ) write (*,*) "dumping model to files"

  ! dump space arrays to files, only image 1 does it, all others
  ! wait at the barrier, hence sync needed
  call cgca_fwci( space, cgca_state_type_grain, "zg0.raw" )
  call cgca_fwci( space, cgca_state_type_grain, "zf0.raw" )

if ( img .eq. 1) write (*,*) "finished dumping model to files"

sync all

! deallocate all coaarrays, implicit sync all.
call cgca_ds( space )
call cgca_drt( grt )

if ( img .eq. 1 ) write (*,*) "Test ACE completed sucessfully"

end program testACE
```

132 tests/testACF

[Unit tests]

NAME

testACF

SYNOPSIS

```
!$Id: testACF.f90 526 2018-03-25 23:44:51Z mexas $

program testACF
```

PURPOSE

Testing MPI/IO, hdf5 and netCDF on Lustre FS

DESCRIPTION

Timing output of MPI/IO (cgca_pswci2 (17.1)) against the netcCDF version (cgca_pswci3 (18.1)) and hdf5 (cgca_hdf5).

NOTE

Works only on lustre file system!

AUTHOR

Luis Cebamanos, Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

cgca testaux (133)

USED BY

Part of CGPACK test suite

SOURCE

```
  use testaux

  implicit none

  integer, parameter :: maxlen = 64
  real,parameter :: gigabyte=real(2**30), resolution=1.0e-5,            &
       loge2 = log(real(2))
  logical(kind=ldef),parameter :: yesdebug = .true., nodebug = .false.

  real( kind=rdef ) ::    &
       qual,                  & ! quality
       bsz0(3),               & ! the given "box" size
       bsz(3),                & ! updated "box" size
       dm,                    & ! mean grain size, linear dim, phys units
       lres,                  & ! linear resolution, cells per unit of length
```

```
      res                     ! resolutions, cells per grain

  integer( kind=idef ) :: ir(3), nimgs, img, ng, c(3)  ! coarray dimensions

  integer( kind=iarr ), allocatable :: space(:,:,:,:)[:,:,:]

  integer( kind=ilrg ) :: icells, mcells

  !################################
  character*(maxlen) :: filename
!  integer, parameter :: numiolayer = 4
!  integer, parameter :: totdim = 4, arrdim = totdim-1, coardim = 3
!  integer, parameter :: numstriping = 3
!  character*(maxlen), dimension(numstriping) :: stripestring
  character*(maxlen), dimension(3)  :: iolayername
!  integer ::  comm, ierr=0, rank=0, mpisize=0, filetype,      &
!       mpi_subarray, fh, funit, i, j, k
  integer ::  ierr=0, i, j, k, errstat

! Anton >>> I think we don't need these vars?
!  integer, dimension(totdim) :: asizehal
!  integer, dimension(arrdim) :: arrsize, arstart, artsize
!  integer, dimension(coardim) :: coarsize, copos

! Add trailing blanks to keep all elements of the array of the same
! length. Max stripe count on ARCHER is 56.
  character( len=maxlen), dimension(9) :: stripe_count = (/           &
    "-c-1 ", "-c0  ", "-c1  ", "-c4  ", "-c8  ",                      &
    "-c16 ", "-c20 ", "-c32 ", "-c40 " /)
  character( len=maxlen), dimension(7) :: stripe_size = (/            &
    "-S1m ", "-S2m ", "-S4m ", "-S8m ", "-S16m",                      &
    "-S32m", "-S64m" /)
  character( len=2*maxlen ) :: dir
  character( len=120 ) :: errmsg

  !################################
  double precision :: t0, t1, tdiff, fsizeb, fsizeg
!*****************************************************************72
  ! first executable statement
  iolayername(1) = 'mpiio.dat'
  iolayername(2) = 'netcdf.dat'
  iolayername(3) = 'hdf5.dat'
! These not used yet
!  iolayername(3) = 'serial.dat'

!  stripestring(1) = 'unstriped'
!  stripestring(2) = 'striped'
!  stripestring(3) = 'defstriped'

  ! physical dimensions of the box, assume mm
  bsz0 = (/ 2.0, 2.0, 3.0 /)
```

```
! mean grain size, linear dimension, e.g. mean grain diameter, also mm
dm = 1.0e-1
!dm = 1.0e0

! resolution
res = 1.0e5

  img = this_image()
nimgs = num_images()

! do a check on image 1
if ( img .eq. 1 ) then
   ! print a banner
   call banner("ACF")
   ! print the parameter values
   call cgca_pdmp
   write (*,'(a,i0,a)') "running on ", nimgs, " images in a 3D grid"
end if

! I want pdmp output appear before the rest.
! This might help
sync all

! each image calculates the coarray grid dimensions
call cgca_gdim( nimgs, ir, qual )

! calculate the resolution and the actual phys dimensions
! of the box
! subroutine cgca_cadim( bsz, res, dm, ir, c, lres, ng )
! c - coarray sizes
! ir - coarray grid sizes
bsz = bsz0
call cgca_cadim( bsz, res, dm, ir, c, lres, ng )

! total number of cells in a coarray
icells = int( c(1), kind=ilrg ) * int( c(2), kind=ilrg ) *             &
     int( c(3), kind=ilrg )

! total number of cells in the model
mcells = icells * int( nimgs, kind=ilrg )

if ( img .eq. 1 ) then
   write ( *, "(9(a,i0),tr1,g10.3,tr1,g10.3,3(a,g10.3),a)" )            &
        "img: ", img  , " nimgs: ", nimgs, " (", c(1) ,                    &
        ","    , c(2) , ","       , c(3) , ")[", ir(1),                    &
        ","    , ir(2), ","       , ir(3), "] ", ng   ,                    &
        qual, lres,                                                        &
        " (", bsz(1), ",", bsz(2), ",", bsz(3), ")"
   write (*,'(a,i0,a)') "Each image has ",icells, " cells"
   write (*,'(a,i0,a)') "The model has ", mcells, " cells"
end if
```

```
  ! Total output file size, in B and in GB.
  fsizeb = real( mcells * storage_size( space, kind=ilrg ) / 8_ilrg )
  fsizeg = fsizeb / gigabyte

  ! allocate space coarray with a single layer
  ! implicit sync all
  !subroutine cgca_as( l1, u1, l2, u2, l3, u3, col1, cou1, col2, cou2,   &
  !                    col3, props, coarray )
  call cgca_as(1, c(1), 1, c(2), 1, c(3), 1, ir(1), 1, ir(2), 1, 1, space)

  ! initialise coarray to image number
  space = int( img, kind=iarr )

  ! start MPI
  call MPI_Init(ierr)

  ! Loop over lfs stripe counts
  do i = 1, size( stripe_count )

    ! Loop over lfs stripe sizes
    do j = 1, size( stripe_size )

      dir = "lfs" // trim( stripe_count(i) ) // trim( stripe_size(j) )

      ! Image 1 makes a dir with desired lfs settings
      if ( img .eq. 1 ) then

write (*,*) "before mkdir"

        ! Make the dir
        errmsg = ""
        call execute_command_line( command = "mkdir " // trim(dir),    &
          wait = .true. , exitstat = ierr, cmdstat = errstat,          &
          cmdmsg = errmsg )

write (*,*) "after mkdir, exitstat:", ierr, "cmdstat:", errstat,       &
            "cmdmsg:", errmsg
if ( ierr .ne. 0 ) error stop

write (*,*) "before lfs setstripe"

        ! Set lfs parameters
        call execute_command_line( command = "lfs setstripe " //       &
          stripe_count(i) // " " // stripe_size(j) // " " //           &
          trim(dir) )

      end if

      ! Loop over IO layers
      do k = 1, size( iolayername )

        filename = trim(dir) // "/" // trim( iolayername(k) )
```

```
        sync all

        t0 = cgca_benchtime()

        if ( k .eq. 1 ) then
          ! MPI/IO
          call cgca_pswci2( space, cgca_state_type_grain, filename )
        else if ( k .eq. 2 ) then
          ! NetCDF
          call cgca_pswci3( space, cgca_state_type_grain, filename )
        else if ( k .eq. 3 ) then
          ! HDF5
         call cgca_pswci4( space, cgca_state_type_grain, filename )
        end if

        t1 = cgca_benchtime()

        sync all

        if (img .eq. 1)  then
          tdiff = t1 - t0
          write (*,*) trim( iolayername(k) ), " ",                      &
            trim( stripe_count(i) ), " ", trim( stripe_size(j) ), " ", &
            fsizeg/tdiff
!            tdiff, "s, rate: ", fsizeg/tdiff, "GB/s."
        end if

        sync all

      end do
    end do
  end do

  ! terminate MPI
  call MPI_Finalize(ierr)

  ! deallocate all arrays
  call cgca_ds(space)


contains

subroutine fdelete(filename)

  implicit none

  character *(*) :: filename
  integer :: stat, funit=0

  open(newunit=funit, iostat=stat, file=filename, status='old')
  if (stat.eq.0) close(unit=funit, status='delete')
```

```
end subroutine fdelete

end program testACF
```

133 tests/testaux

[Modules]

NAME

testaux

SYNOPSIS

```
!$Id: testaux.f90 533 2018-03-30 14:31:26Z mexas $

module testaux
```

DESCRIPTION

A helper module to use with the tests. This is not a part of CGCA. Use of this module is not required.

AUTHOR

Anton Shterenlikht

COPYRIGHT

CONTAINS

getcodim (133.2), banner (133.1)

USES

cgca

USED BY

Part of CGPACK test suite

SOURCE

```
use casup
implicit none

contains
```

133.1 testaux/banner

[testaux] [Subroutines]

NAME

banner

SYNOPSIS

```
subroutine banner( test )
```

INPUT

```
character( len=3 ), intent( in ) :: test
```

SIDE EFFECTS

A banner with the test name is printed on sdtout

USES

none

USED BY

all tests

SOURCE

```
write (*,'(a)') "******************************************************"
write (*,'(a)') "******                                          ******"
write (*,'(a)') "******                TEST "//test//"                ******"
write (*,'(a)') "******                                          ******"
write (*,'(a)') "******************************************************"

end subroutine banner
```

133.2 testaux/getcodim

[testaux] [Subroutines]

NAME

getcodim

SYNOPSIS

```
subroutine getcodim(nimages,codim)
```

INPUT

```
integer( kind=idef ), intent( in ) :: nimages
```

OUTPUTS

```
integer( kind=idef ), intent( out ) :: codim(3)
```

DESCRIPTION

Subroutine to get coarray codimensions from command line. Call this routine from all tests.

USES

none

USED BY

all tests

SOURCE

```
integer, parameter :: maxarglen=20
character( len=maxarglen ) :: value, fmt
integer :: arglen=0, errstat=0, i

do i = 1, 2
   call get_command_argument( i, value, arglen, errstat )

   if (errstat .gt. 0) then
     write (*,'(a,i0,a)') "ERROR: argument ", i, " cannot be retrieved"
     error stop
   elseif (errstat .eq. -1) then
     write (*,'(a,i0,a)') "ERROR: argument ", i, &
       " length is longer than the string to store it"
     error stop
   elseif (errstat .lt. -1) then
     write (*,'(a,i0)') &
       "ERROR: get_command_argument(", i, " returned ",errstat
     write (*,'(a)') "ERROR: unknown error, should never end up here"
     error stop
   end if

   write(fmt,'(a,i0,a)') "(i", arglen, ")"
```

```
   read (value,fmt) codim(i)

   if ( codim(i) .le. 0 ) then
     write (*, "(a,i0,a)") "ERROR: testaux/getcodim: codimension ", i, &
       "is negative"
     error stop
   end if

end do

codim(3) = nimages / ( codim(1) * codim(2) )

if ( codim(1) * codim(2) * codim(3) .ne. nimages ) then
  write (*,*) "ERROR: testaux/getcodim: codimension 3 wrong"
  error stop
end if

end subroutine getcodim
```

134 tests/testgc

[Unit tests]

NAME

testgc

SYNOPSIS

```
!$Id: test_gc.f90 380 2017-03-22 11:03:09Z mexas $

program testgc
```

PURPOSE

Checking: grain connectivity, cgca_m2gb (11)

DESCRIPTION

This program reads the global grain connectivity from stdin and dumps a number of neighbours for each grain to stdout.

AUTHOR

Anton Shterenlikht

COPYRIGHT

See LICENSE (33)

USES

none

USED BY

Part of CGPACK test suite

SOURCE

```
implicit none

integer :: errstat=0, data1=0, data2=0, value=0, total=0, grain=0

read (unit=*,fmt=*,iostat=errstat) data1
if (errstat .ne. 0) stop "Cannot read data1"
value=1
total=1
grain=1

! The first value must always be grain 1.
! Stop with an error, if not.
if (data1 .ne. grain) stop "The first grain must be number 1"

do
 read (unit=*,fmt=*,iostat=errstat) data2
 if (errstat .ne. 0) then
  ! this means end of file
```

```
  write (*,*) value
  exit
 end if
 if ( data2 .eq. data1 ) then
  ! same grain
  value = value+1
 else
  write (*,*) value
  value=1
  grain=grain+1
  ! The value must match grain number. Stop if not.
  if (data2 .ne. grain) then
   write (*,*) "This should have been grain", grain
   write (*,*) "Instead I read", data2
   stop "This is an error, and I can do no more for you..."
  end if
 data1 = data2
 end if
 total = total +1
end do

write (*,*) "Total values read: ", total
write (*,*) "Total grains: ", grain

end program testgc
```