Cellular Automata Beyond 100k Cores: MPI vs Fortran Coarrays

Anton Shterenlikht[1] and Luis Cebamanos[2]

[1]Mechanical Engineering Department, The University of Bristol, Bristol BS8 1TR, UK, mexas@bristol.ac.uk
[2]EPCC, The University of Edinburgh, James Clerk Maxwell Building, Peter Guthrie Tait Road, Edinburgh EH9 3FD, UK

EuroMPI 2018, 23-26 September 2018, Barcelona, Spain

Table of Contents

Fortran coarrays - native SPMD, single sided

F2008:

- coarray data objects
- allocatable coarrays
- coarrays of DT with allocatable or pointer components
- remote definitions and references
- execution segments
- image control statements
- atomics
- critical sections
- locks

F2018:

- collectives
- teams
- events
- many more atomics
- failed images

Compiler support:

- Cray
- Intel
- GCC/OpenCoarrays
- NAG (syntax only)

Implementation: (**Challenge!**)

- libpgas, DMAPP (Cray)
- MPI, OpenMP (Intel)
- MPI, GASnet (OpenCoarrays)

Coarrays primer - swap values between images

```
integer :: i[*], n, tmp[*], mype
mype = this_image()
i = mype
tmp = mype
n = num_images()
if (mype == 1) then
 sync images (n)              ! pair-wise barrier
 i = tmp[n]                   ! remote read, single sided
else if (mype == n) then
 sync images (1)              ! pair-wise barrier
 i = tmp[1]                   ! remote read, single sided
end if
print *, "on image ", mype, " i = ", i
end
```

```
$  cafrun -np 4 ./a.out
 on image 2   i = 2
 on image 3   i = 3
 on image 1   i = 4
 on image 4   i = 1
```

Coarrays - weak memory consistency model

F2018 FDIS, 11.6.2 Segments:

> *if a variable is defined or becomes undefined on an image in a segment, it shall not be referenced, defined, or become undefined in a segment on another image unless the segments are ordered*

11.6.1 Image control statements:

- SYNC ALL
- SYNC IMAGES
- SYNC MEMORY
- SYNC TEAM
- FORM TEAM
- CHANGE TEAM / END TEAM
- ALLOCATE / DEALLOCATE coarrays
- CRITICAL / END CRITICAL
- EVENT POST / EVENT WAIT
- LOCK / UNLOCK
- MOVE_ALLOC
- STOP
- END

Compiler cannot move operations across image control statements

Coarray/OpenMP usage - 3 examples

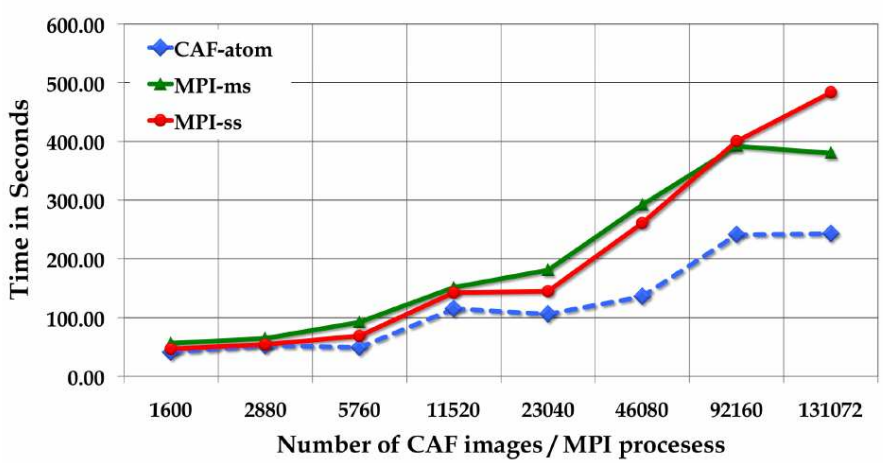
1. Gyrokinetic particle-in-cell code
2. Numerical weather prediction: European Center for Medium Range Weather Forecast, Integrated Forecasting System (ECMWF IFS)
3. Physics/engineering: Cellular Automata library for SUPercomputers, CASUP: The University of Bristol
   `https://cgpack.sourceforge.io`

H. Richardson, Coarrays from laptops to supercomputers, 2015
`http://www.fortran.bcs.org/2015/BCS_FSG_2015_HR.pdf`

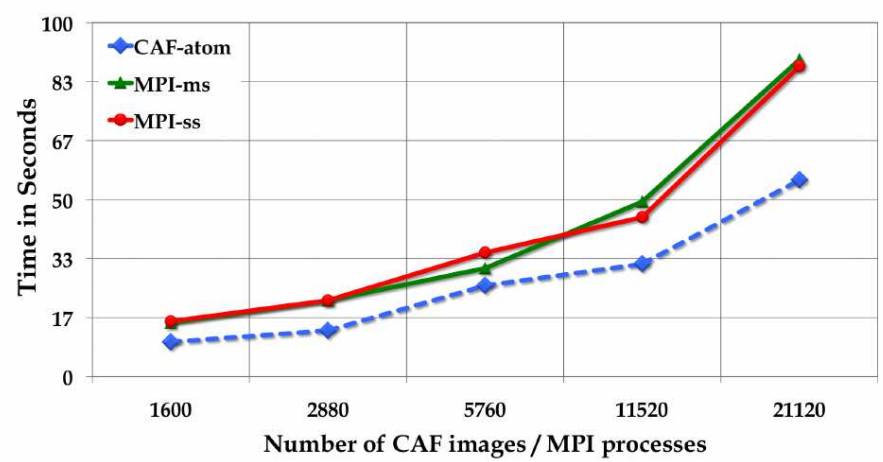
Gyrokinetic particle-in-cell

From: R. Preissl et al, Multithreaded Global Address Space Communication Techniques for Gyrokinetic Fusion Applications on Ultra-Scale Platforms, SC11.

`http://upc.lbl.gov/publications/Preissl_SC2011.pdf`



(a) 1 OpenMP thread per instance



(b) 6 OpenMP threads per instance

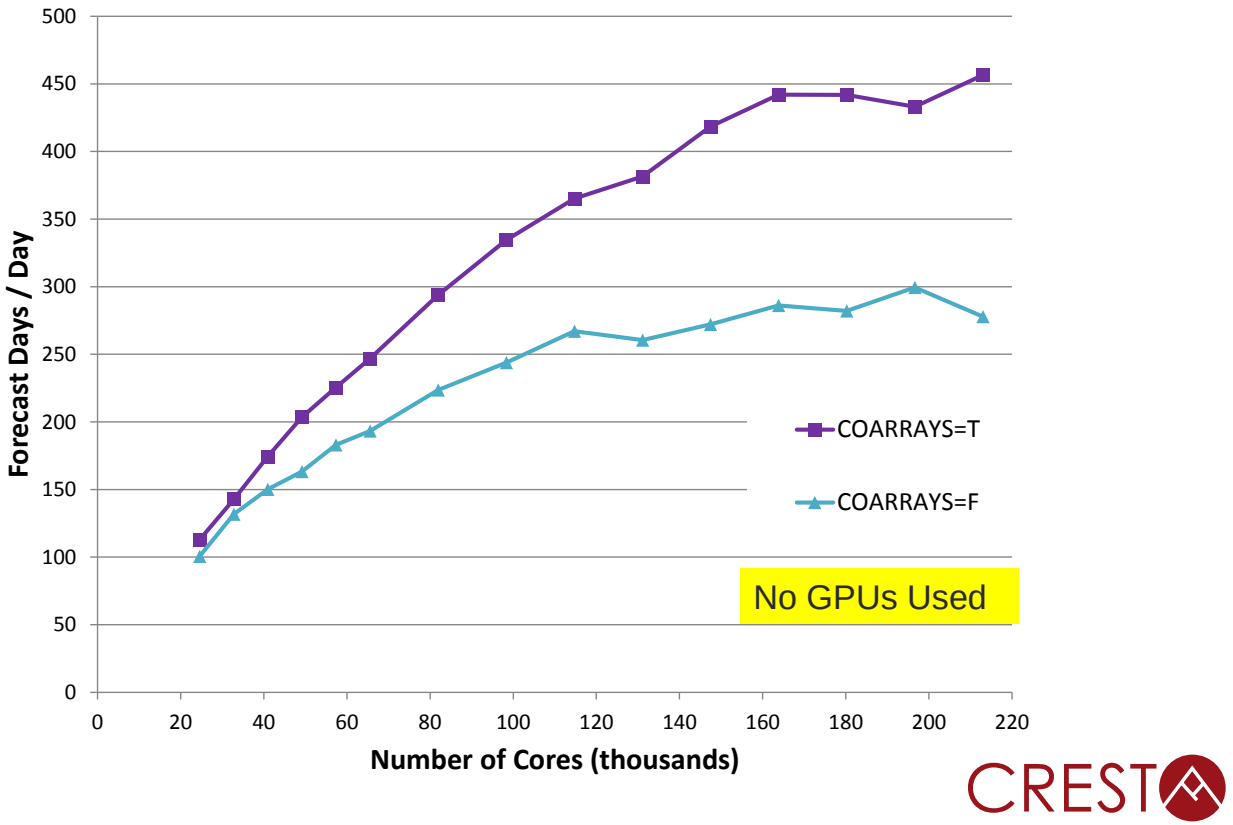
Figure 5: Weak scaling benchmarks of the CAF shifter (*CAF-atom*) and two MPI shifter (*MPI-ms*, *MPI-ss*) implementations with no (a) and full (6 OpenMP threads per NUMA node) OpenMP support (b)

- Single sided calls
- Excellent scaling, outperforms MPI/OpenMP
- Non-standard, Cray extensions (atomics)

ECMWF IFS

ECMWF HPC in Meteorology workshop, 27-31 Oct 2014 **16**

T_c1999L137 5 km (~2024) IFS model scaling on TITAN

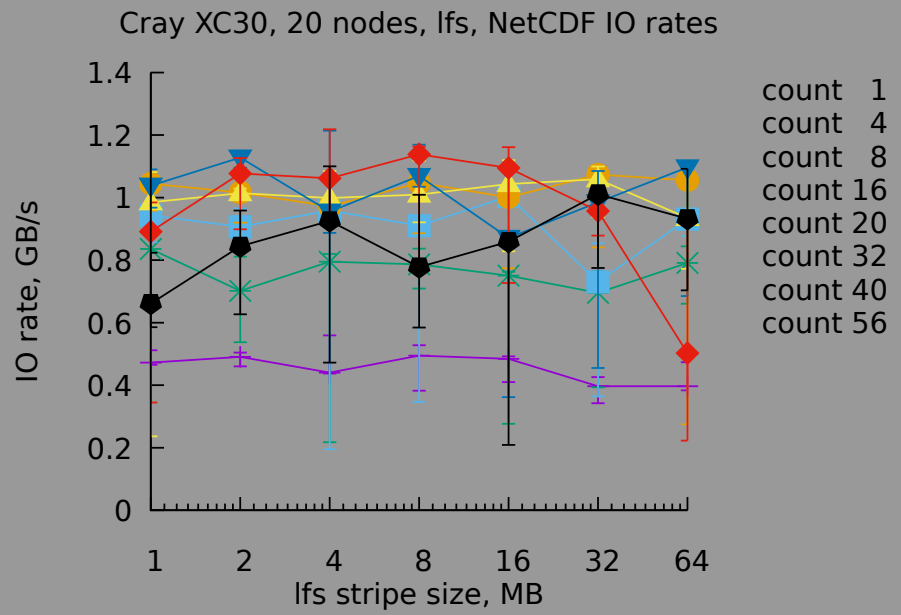


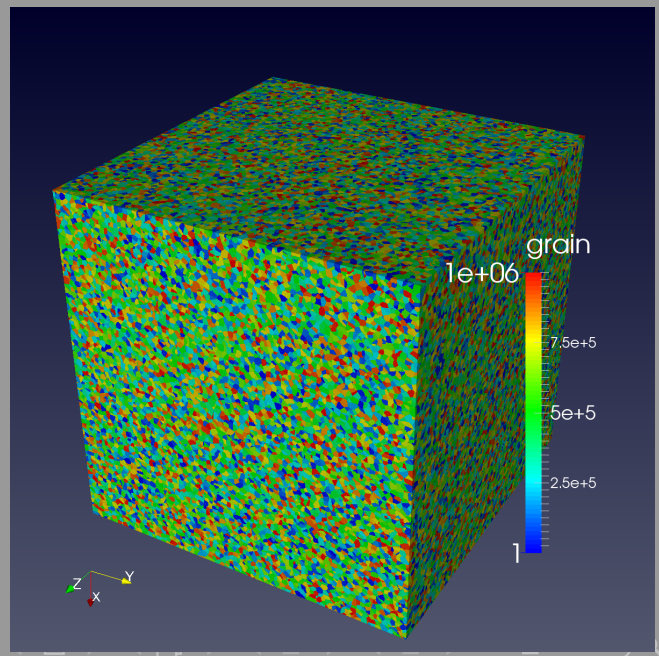
From: G. Mozdzynski et al, Challenges of getting ECMWFs weather forecast model (IFS) to the Exascale, ECMWF HPC in Meteorology workshop, 2014. PDF

Higher is better

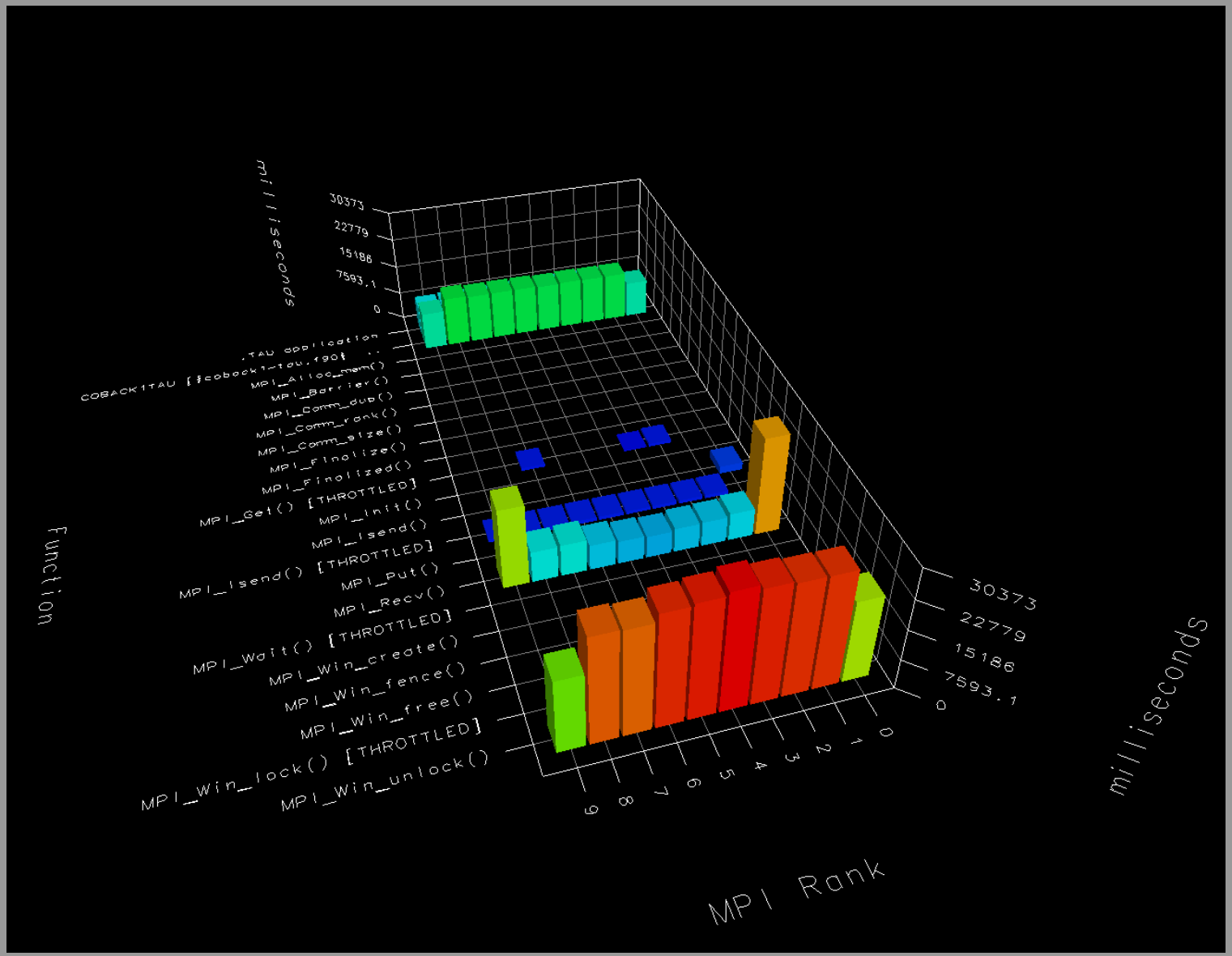
Coarray IO - no native Fortran parallel IO

- MPI/IO up to 2.3GB/s on Cray XE6 ▸ BCS talk
- MPI/IO up to 8GB/s on Cray XC30 (can reach 14GB/s)
- NetCDF 4.3, HDF5 1.8.14 - only up to 1.2GB/s on Cray XC30.
- lfs stripe count, size, number of images, file size, Cray hugepages...
- 0.5 - 1TB datasets





Tools: TAU

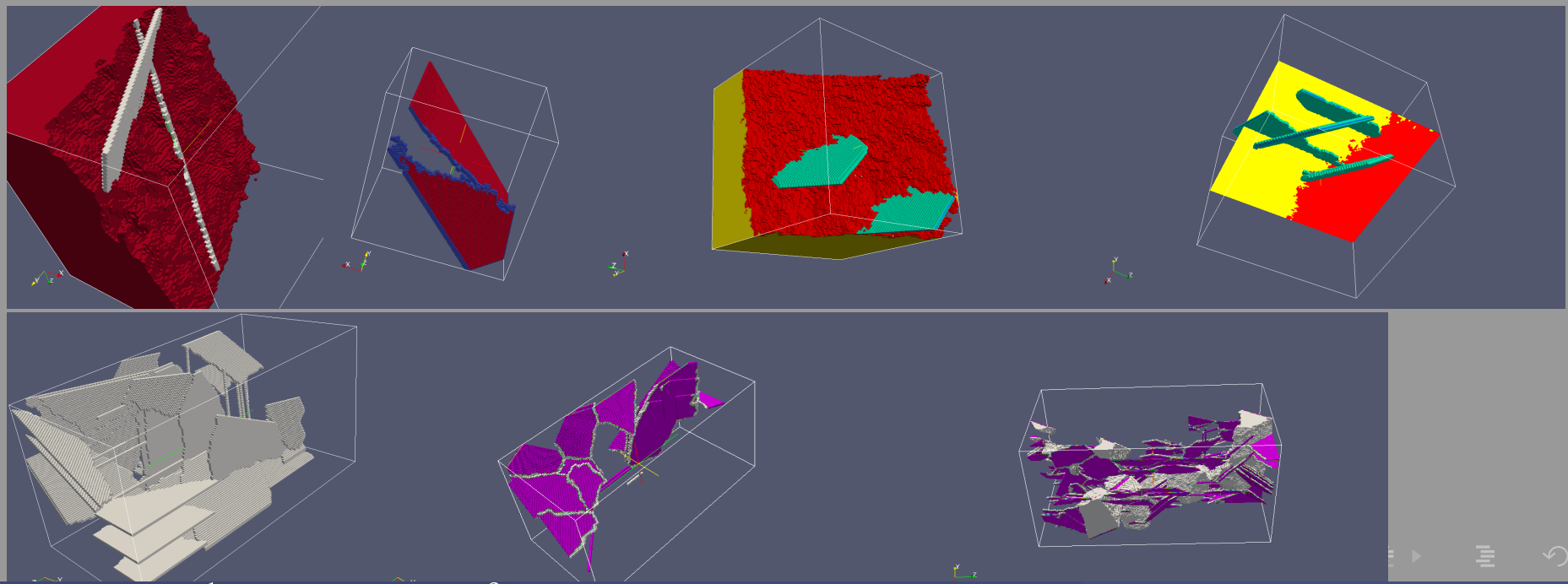


Only for coarray implementation via MPI (Intel, OpenCoarrays).

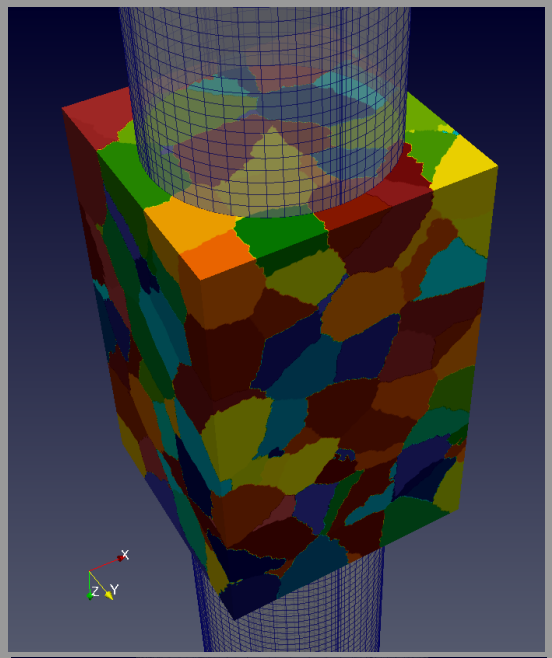
Also: CrayPAT, Scalasca, Score-P

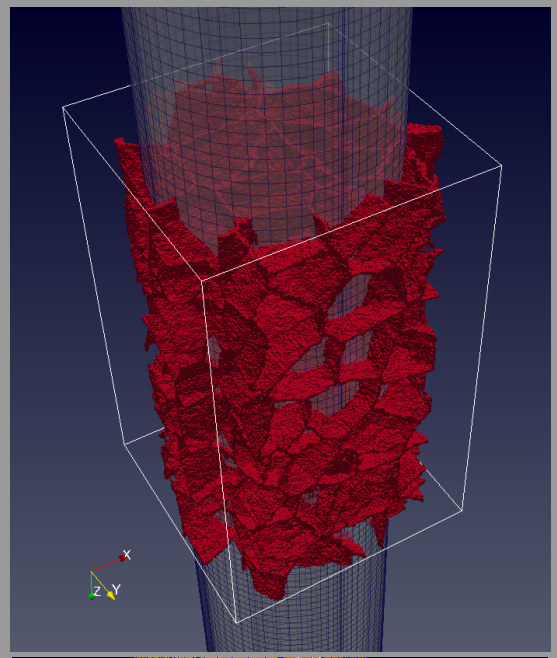
Fracture: CA + FE = CAFE multi-scale model

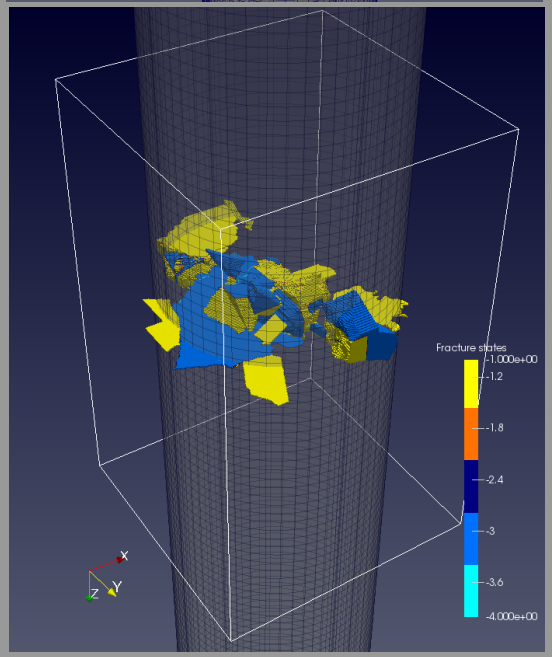
- Structured grids - CA, unstructured grids - FE
- CASUP: Cellular Automata library for SUPercomputers, `https://cgpack.sourceforge.io`
- CA via coarrays → Easy halo exchange
- CA (microstructure) + FE (continuum mechanics) = CAFE
- Transgranular cleavage - fracture stress or strain criteria
- FE → CA (localisation) - stress, strain fields
- CA → FE (homogenisation) - damage variables

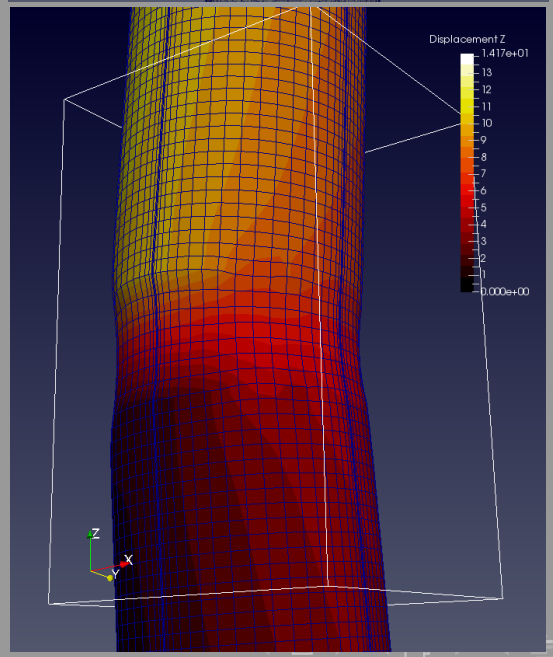


Cleavage in a steel bar under tension

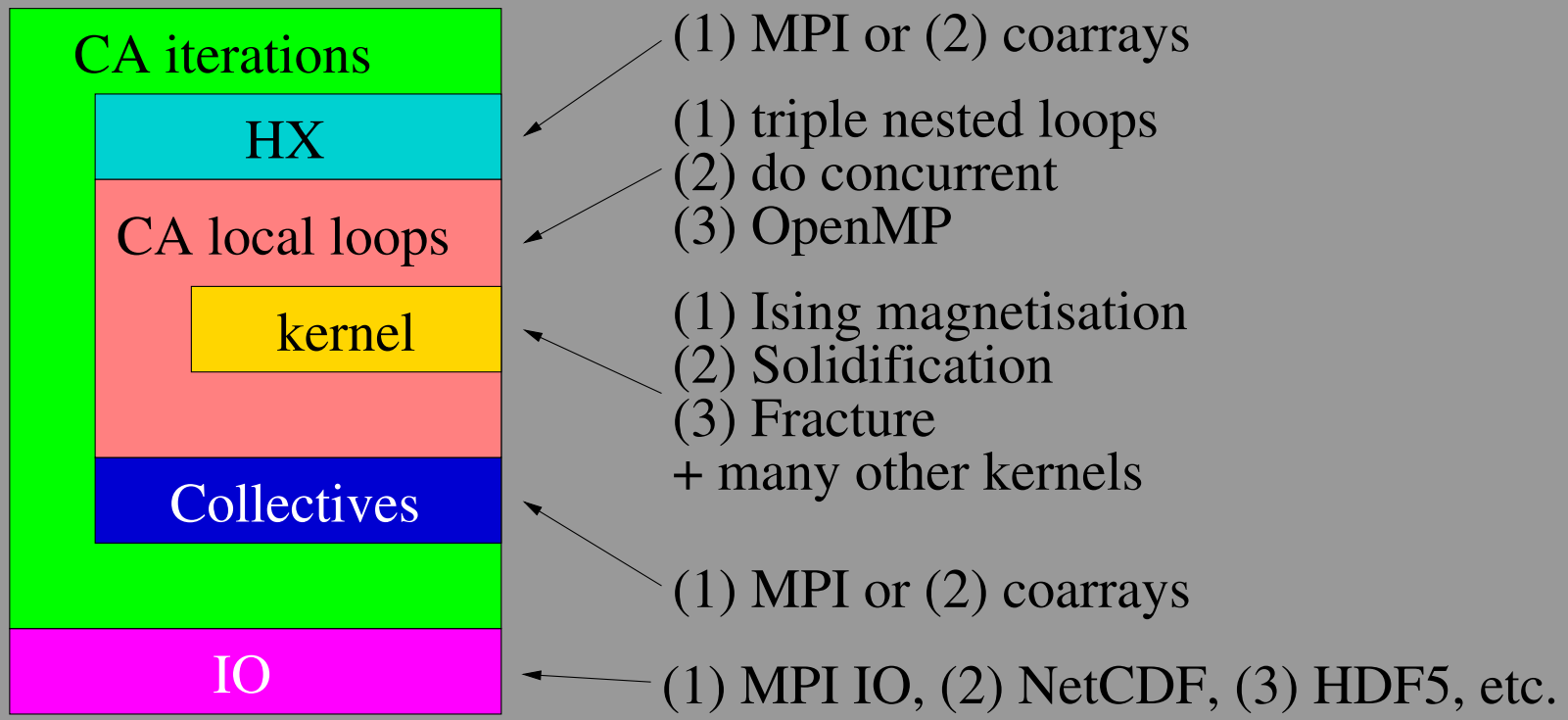




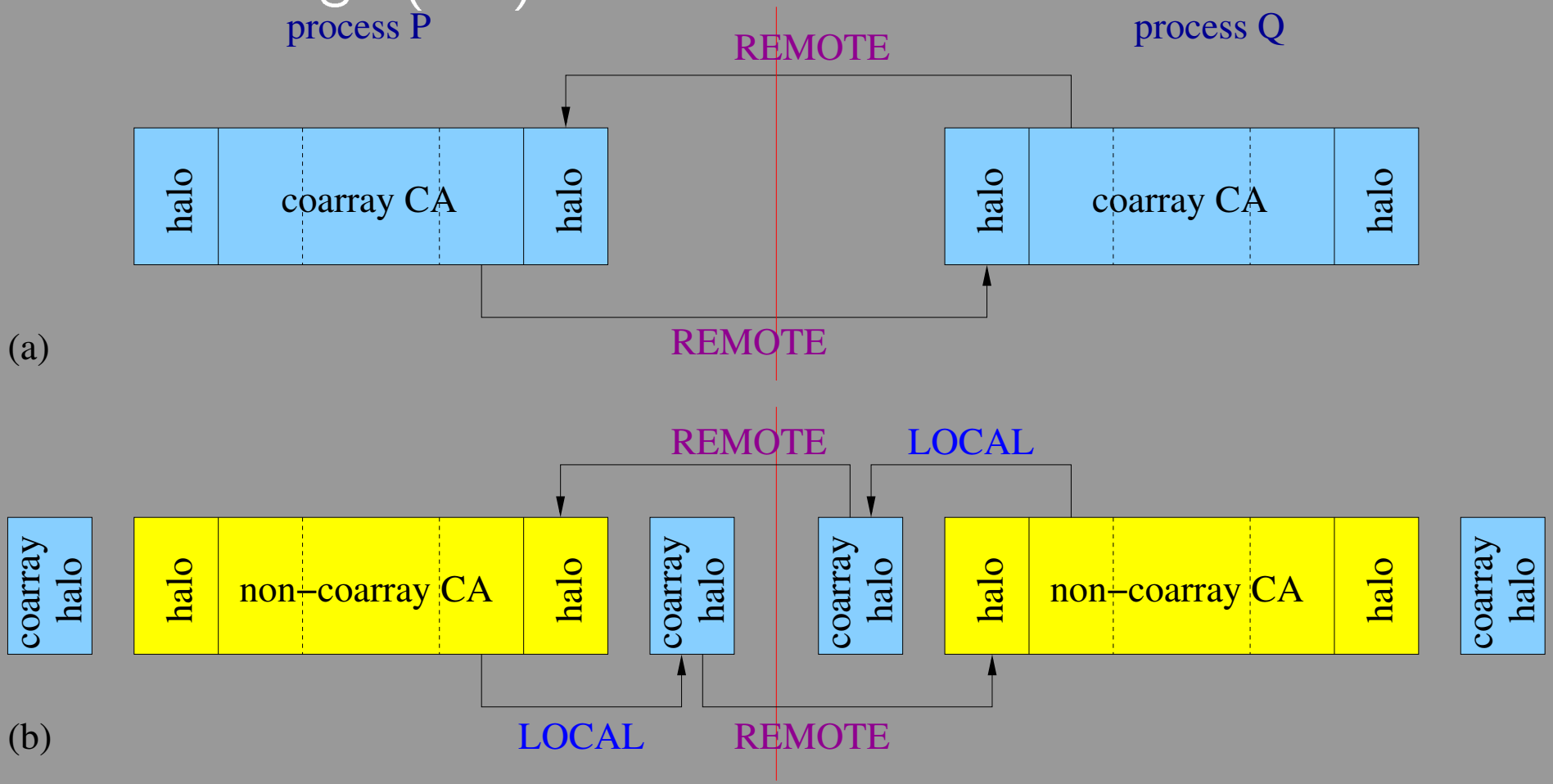




Modular structure of CASUP library – many combinations



Halo exchange (HX) in 1D



- (a) a one-step algorithm (MPI and whole CA model (WCA) coarrays)
- (b) a two-step algorithm used when coarrays are used only for CA halos (HCA).

Local copy and remote comms are shown with arrows.

HX: coarrays for halos only

(1) Copy the halo cells from to coarray arrays. (2) The actual HX:

```
if ( ci(1) .ne. 1 ) then    ! all but leftmost img
 sync images(nei_img_L(1)) ! sync with left image
 ! HX, remote op
 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
! all but the rightmost image
if ( ci(1) .ne. ucob(1) ) then
 sync images(nei_img_R(1)) ! sync with right img
 ! HX, remote op
 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
```

HX: whole model coarrays

HX can be done in a single statement:

```
if (ci(1) .ne. 1) then      ! all but leftmost img
 sync images(nei_img_L(1)) ! sync with left image
 ! HX, remote op
 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
! all but the rightmost image
if ( ci(1) .ne. ucob(1) ) then
 sync images(nei_img_R(1)) ! sync with right img
 ! HX, remote op
 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
```

HX: MPI non-blocking

Derived types, e.g. for the left halo along dimension 1, mpi_h1_LV:

```
call MPI_TYPE_CREATE_SUBARRAY( 3, sizes, subsizes, &
starts, MPI_ORDER_FORTRAN, mpi_ca_integer, mpi_h1_LV, ierr
```

Then point-to-point MPI calls are used for HX:

```
if ( ci(1) .ne. 1 ) then
 call MPI_IRECV(space,1,mpi_h1_LV,nei_img_L(1)-1,&
  TAG1R, MPI_COMM_WORLD, reqs1m(1), ierr )
 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
if ( ci(1) .ne. ucob(1) ) then
 call MPI_IRECV(space,1,mpi_h1_RV,nei_img_R(1)-1,&
  TAG1L, MPI_COMM_WORLD, reqs1p(1), ierr )
 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
```

CA iterations: OpenMP and DO CONCURRENT

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

The do concurrent version looks like this:

```
do concurrent(k=1:sub(3),j=1:sub(2), i=1:sub(1))
 tmp_space( i,j,k ) =                                &
     kernel( space, hdepth, (/ i , j , k /) )
end do
```

Cray optimisations

Unfortunately, Cray 8.6.5 Fortran compiler was unable to exploit the `do concurrent` parallelism: The compiler diagnostic for this is not specific:

```
ftn-6910: A loop was not multi-threaded
          for an unspecified reason.
```

It is well known that it is typically impossible to vectorise loops with function/subroutine calls. Therefore it is not surprising that the compiler was not able to vectorise any of the loops:

```
ftn-6287: A loop was not vectorized because
          it contains a call to function "kernel".
```

This is the price of modularity. IPO doesn't help (see "parallel efficiency" below).

3D Ising magnetisation - mask array

- 3D extension of the Q2R Vichniac's 2D rule – first ever?
- A CA cell is a magnetic spin – 0 (down) or 1 (up).
- Energy conservation: CA cells are split into 2 groups according to a 3D chess-like pattern. Either all 'white' or all 'black' cells updated in a single iteration.
- 3D chess-like mask array (extension of the 2D case):

```
ci = this_image( halo_array )
 c = ucobound( space ) - halo_depth
do concurrent( i=1:c(1), j=1:c(2), k=1:c(3) )
 mask_array(i,j,k)=mod( (i+j+k + (ci(1)-1)*c(1)+ &
  (ci(2)-1)*c(2) + (ci(3)-1)*c(3) ) , 2 )
end do
```

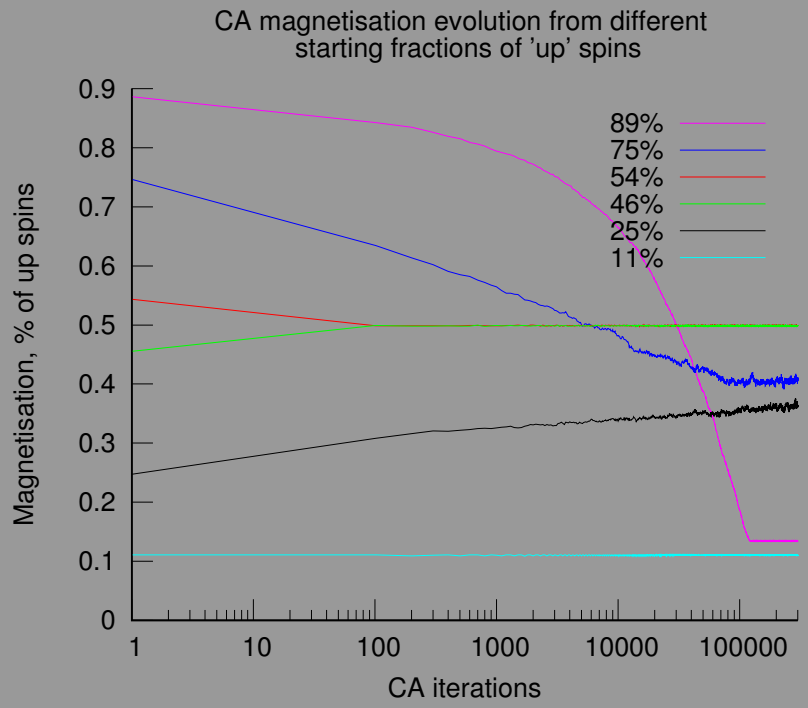
3D Ising magnetisation - energy

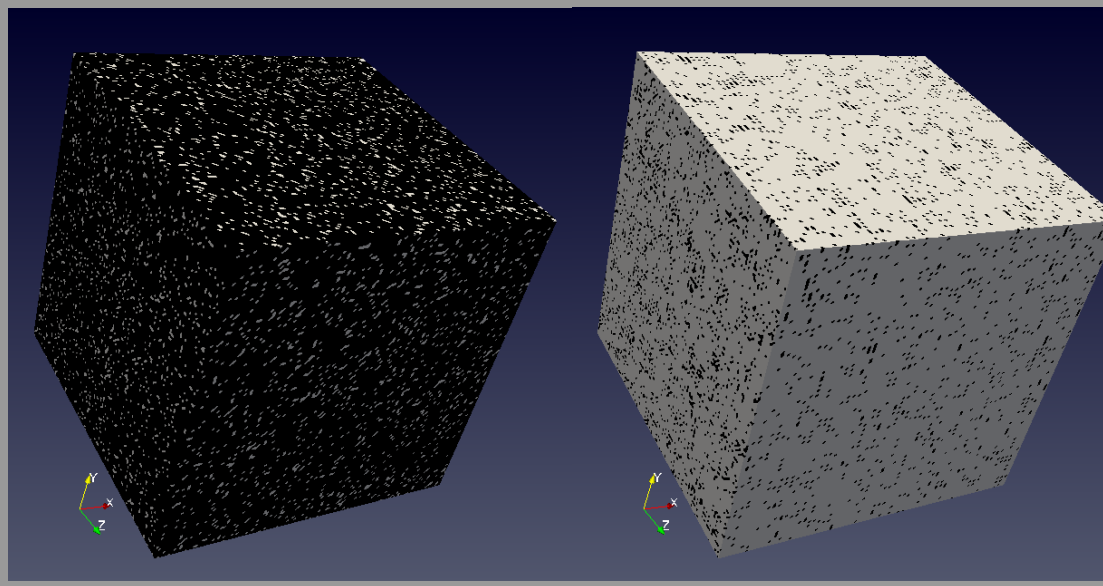
```
! sum of the spins of the 6 neighbours
n = s(i-1,j,k) + s(i+1,j,k) + s(i,j-1,k) +        &
    s(i,j+1,k) + s(i,j,k-1) + s(i,j,k+1)
if (n.eq.3 .and. mask_array(i,j,k).eq.1 ) then
 ! If the sum of 6 neighbours is exactly 3
 ! and the mask value is 1 then flip the state.
  ca_kernel_ising = 1 - s(i,j,k)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

A two-step CA iteration, with energy conservation:

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

3D Ising magnetisation - useful physical results

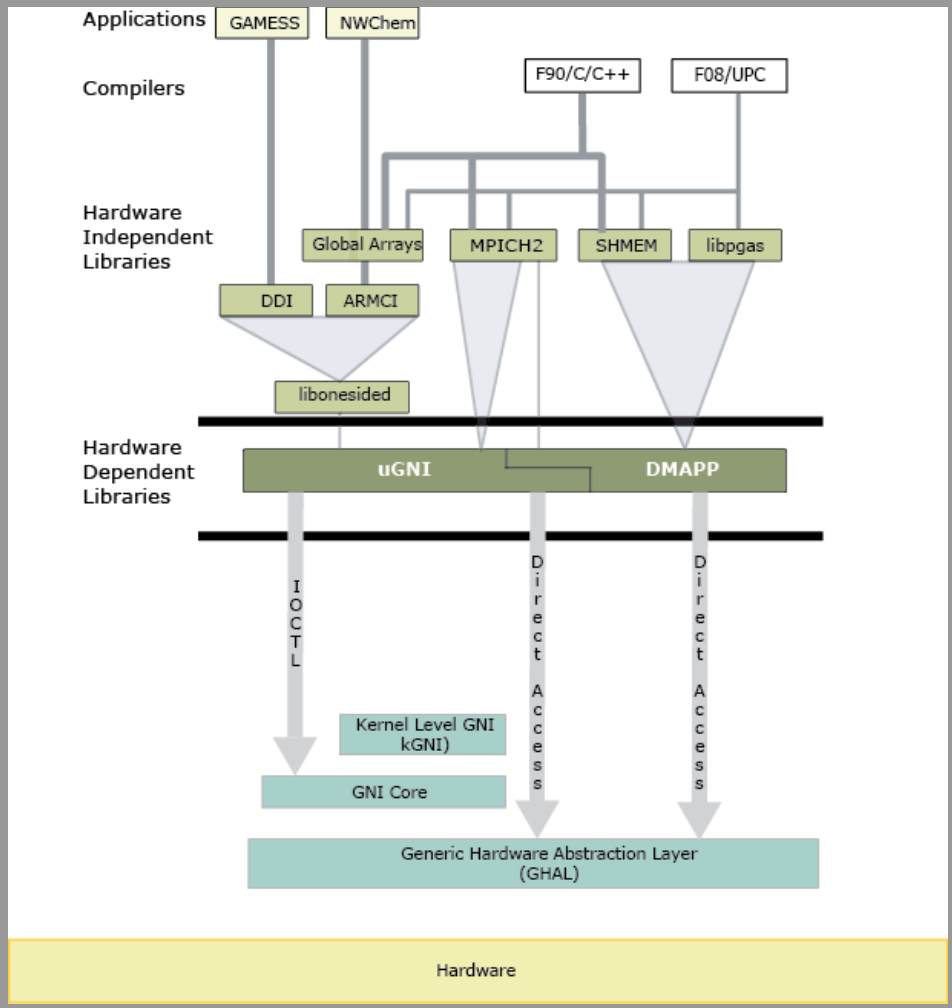




CA with 90% of up spins (left) evolves to 13% of 'up' spins in ∼100k iterations (right).

- Black - up spins
- White - down spins

Cray software stack – different libraries?



from: *XC Series GNI and DMAPP API User Guide (CLE 6.0.UP06) S-2446, Cray Inc. 2018.*

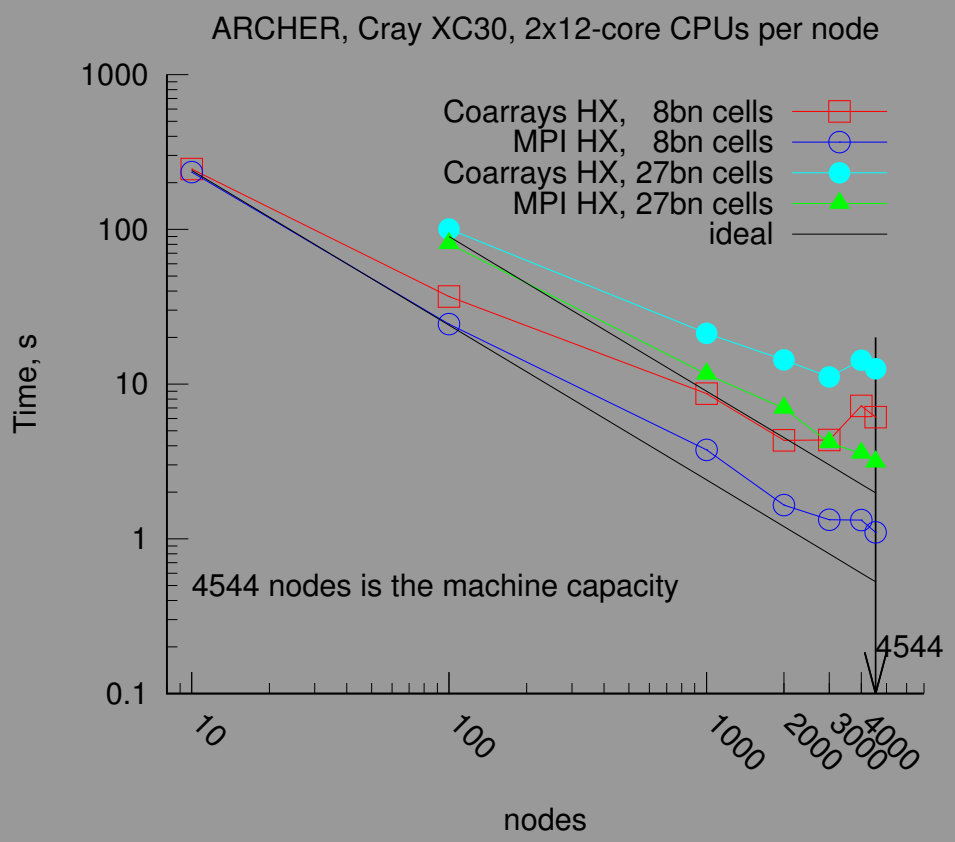
- MPI → MPICH2 → uGNI.
- coarrays → libpgas → DMAPP.

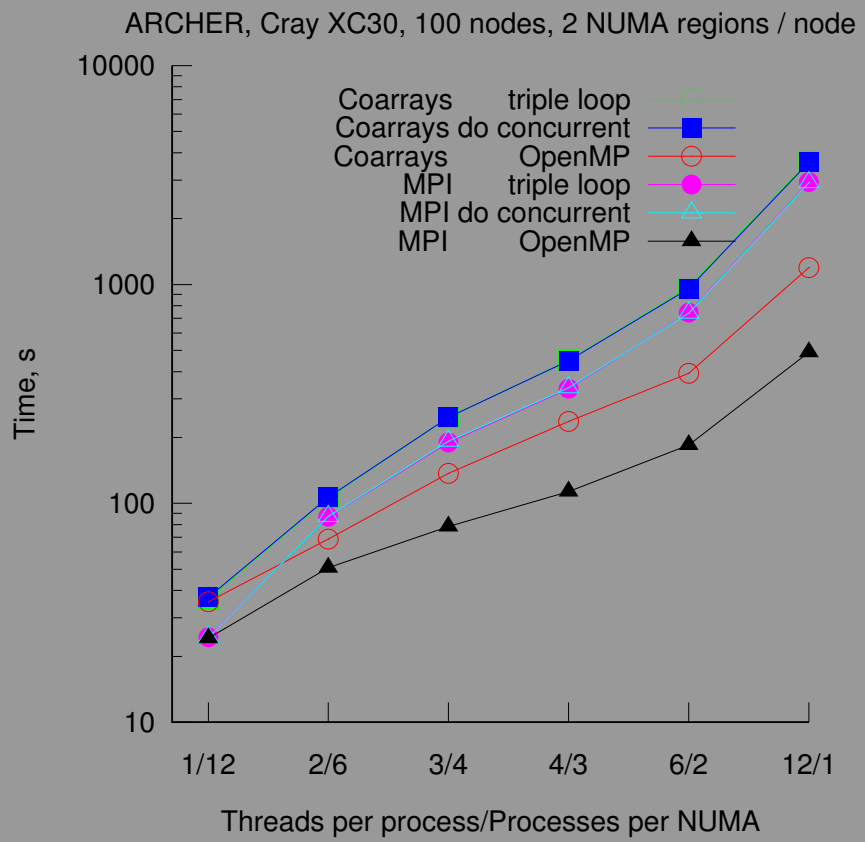
System: ARCHER, the UK national HPC system, Cray XC30 with 2×12-core Ivy Bridge CPUs/node.

aprun:

- 1 thread/core
- `-ss` – 'strict memory containment per NUMA node' – no memory allocation off local NUMA region.

Strong scaling





- MPI beats coarrays! Scales to nearly 110k cores.
- Threading makes things worse - good load balance?

Profiling - time spent in different parts of miniapps

100 nodes			
Group	HCA	WCA	MPI
Triple loop + Ising kernel, %	19.3	17.9	27.9
Ising energy + collectives, %	29.6	33.6	21.2
HX, %	28.3	29.1	17.0
Total, %	77.2	80.6	66.1
Total time, s	191	210	120

1000 nodes			
Group	HCA	WCA	MPI
Triple loop + Ising kernel, %	4.6	3.2	5.0
Ising energy + collectives, %	27.1	26.2	22.2
HX, %	25.0	23.3	16.2
Total, %	56.7	52.7	43.4
Total time, s	59	82	44

- MPI $\rightarrow$ more calculation than comms.
- Coarrays $\rightarrow$ more comms than calculation.

Profiling – 100 nodes – inconclusive

Fortran coarrays (HCA) – comms are in USER functions!

Samp%	Samp	Imb. Samp	Imb. Samp%	Group Function
77.7%	18,256.5	--	--	USER
28.3%	6,650.9	4,960.1	42.7%	ca_hx_all$ca_hx_
21.8%	5,119.2	5,456.8	51.6%	ca_ising_energy_col$ca_hx_
10.7%	2,519.5	144.5	5.4%	ca_kernel_ising$ca_hx_
8.6%	2,011.1	138.9	6.5%	ca_iter_tl$ca_hx_
7.8%	1,829.4	122.6	6.3%	ca_kernel_ising_ener$ca_hx_

MPI – very low imbalance in calculations

40.5%	6,574.7	--	--	USER
15.4%	2,501.5	152.5	5.7%	ca_kernel_ising$ca_hx_
12.5%	2,024.0	139.0	6.4%	ca_iter_tl$ca_hx_
11.4%	1,845.5	104.5	5.4%	ca_kernel_ising_ener$ca_hx_
27.3%	4,425.4	--	--	MPI
14.9%	2,419.9	1,589.1	39.7%	mpi_waitall
9.8%	1,592.8	1,832.2	53.5%	MPI_ALLREDUCE
2.1%	347.3	119.7	25.6%	mpi_isend

Profiling – 1000 nodes – still inconclusive

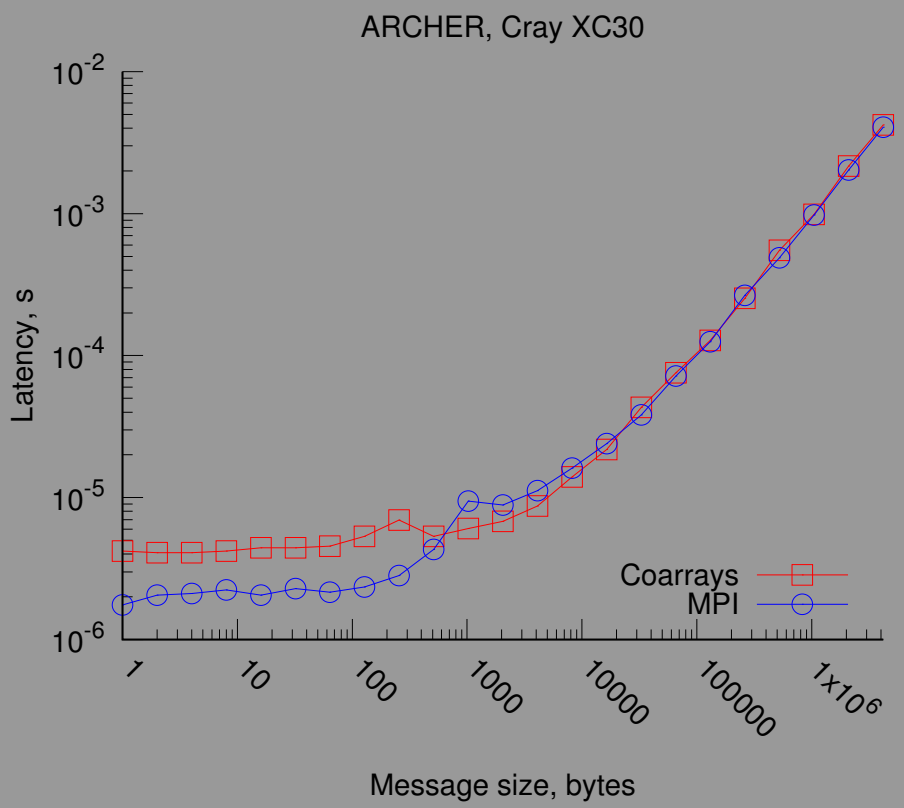
Fortran coarrays (HCA) – user % dropped!, % PGAS time rose, more imbalance

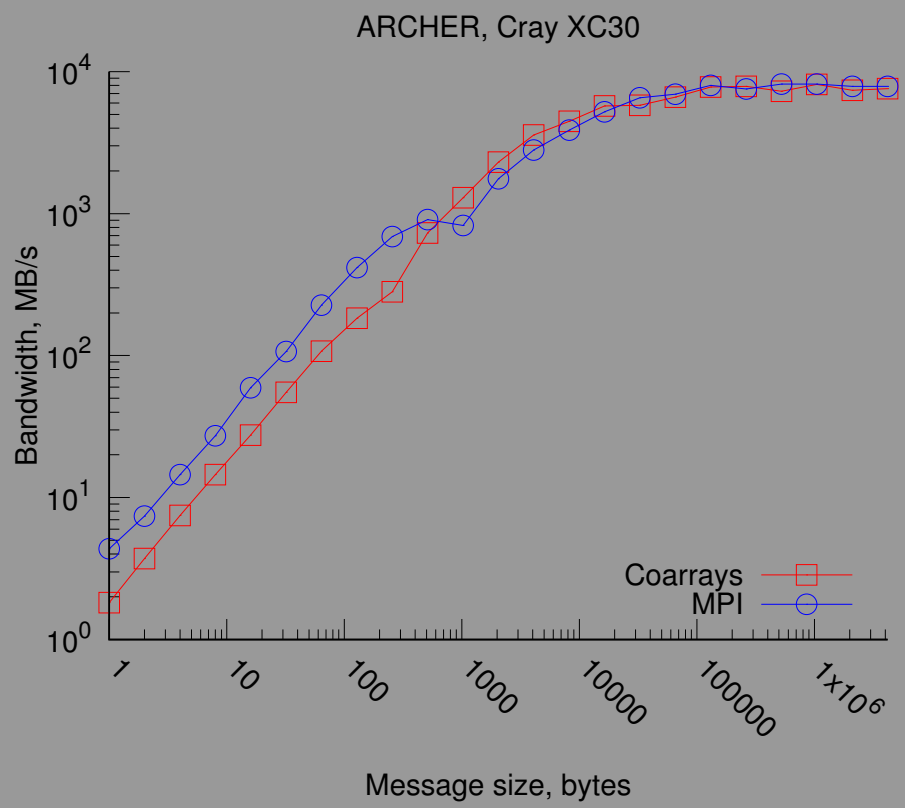
Samp%	Samp	Imb. Samp	Imb. Samp%	Group Function
56.8%	5,693.3	--	--	USER
25.3%	2,533.7	2,581.3	50.5%	ca_ising_energy_col$ca_hx_
25.0%	2,504.0	2,419.0	49.1%	ca_hx_all$ca_hx_
2.6%	259.9	66.1	20.3%	ca_kernel_ising$ca_hx_
2.0%	200.5	62.5	23.8%	ca_iter_tl$ca_hx_
1.8%	183.9	50.1	21.4%	ca_kernel_ising_ener$ca_hx_

MPI – comms dominate, more imbalance

37.4%	3,436.1	--	--	MPI
20.2%	1,858.8	1,633.2	46.8%	MPI_ALLREDUCE
16.5%	1,516.0	1,604.0	51.4%	mpi_waitall
7.1%	653.3	--	--	USER
2.8%	254.4	45.6	15.2%	ca_kernel_ising$ca_hx_
2.2%	198.3	54.7	21.6%	ca_iter_tl$ca_hx_
2.0%	181.5	32.5	15.2%	ca_kernel_ising_ener$ca_hx_

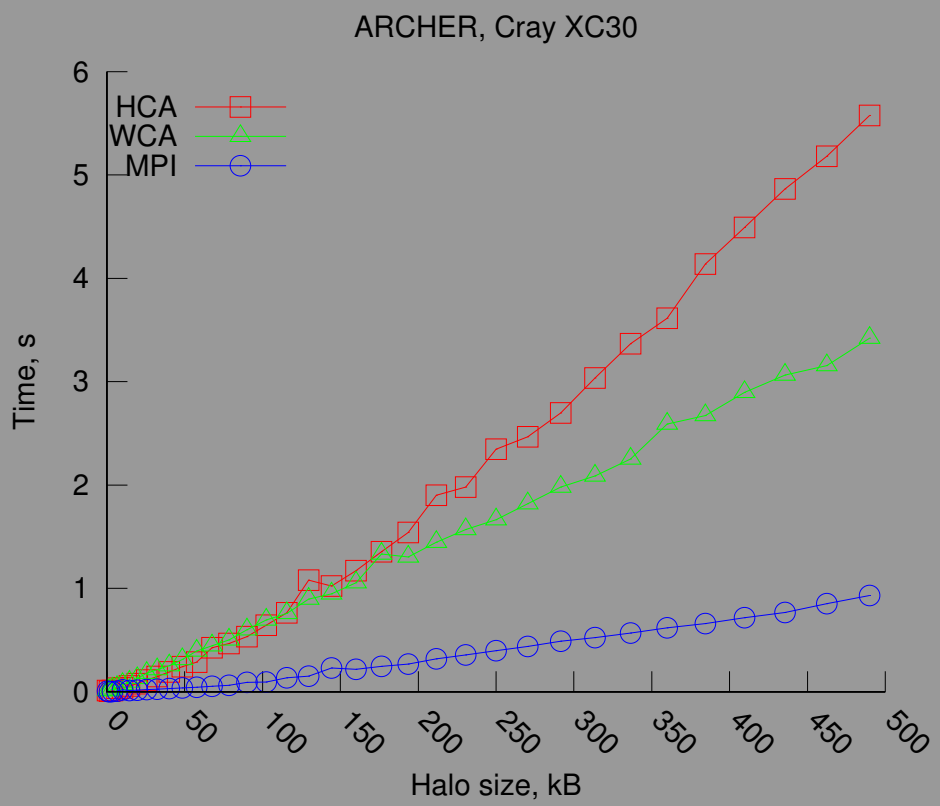
Ping-pong – no difference!



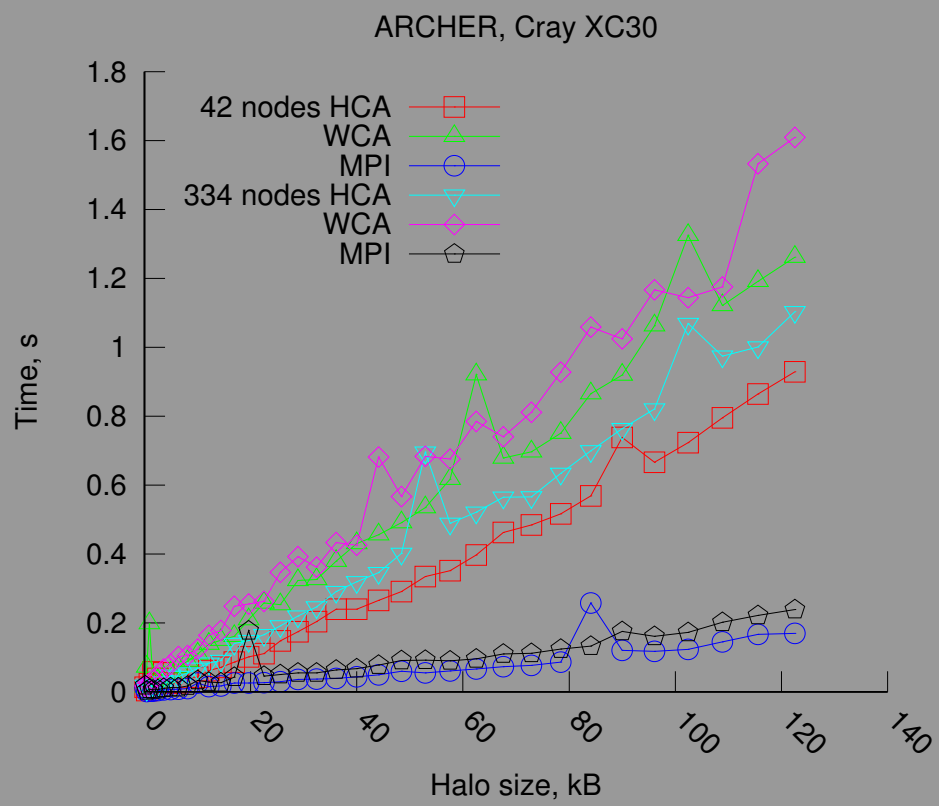


Fortran coarrays data using benchmarks from: I. Bethune et al, A Parallel Benchmark Suite for Fortran Coarrays, Applications, Tools and Techniques on the Road to Exascale Computing, IOS Press, 22:281-288,2012, https://doi.org/10.3233/978-1-61499-041-3-281.

Just HX – massive difference!

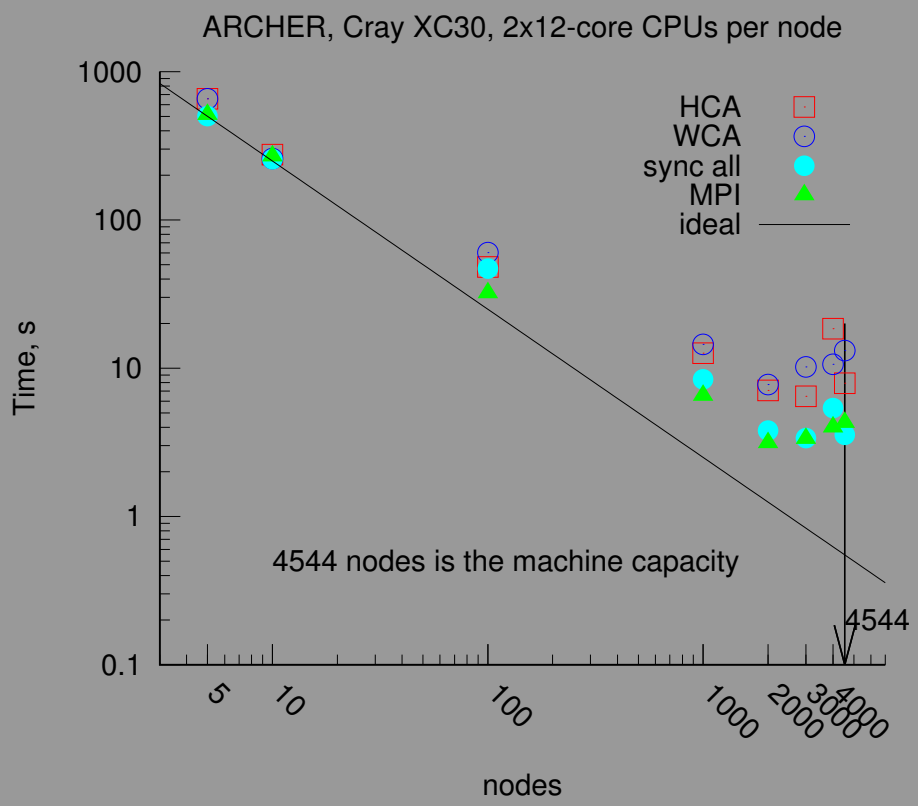


6 nodes

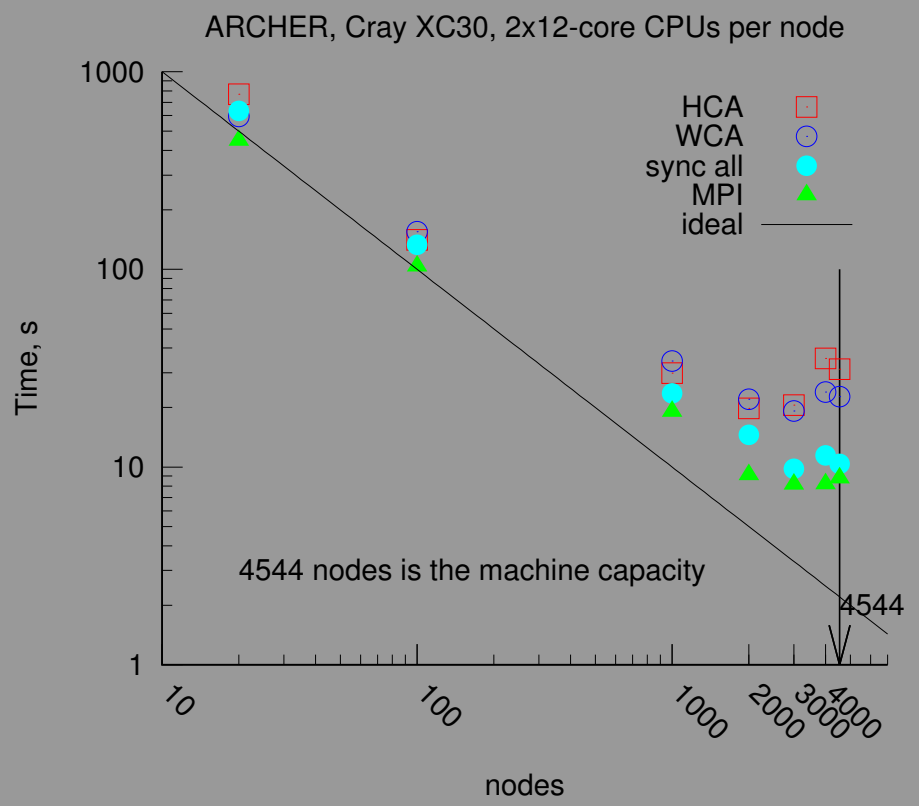


42 and 334 nodes

Global barrier – sync all – very close to MPI!

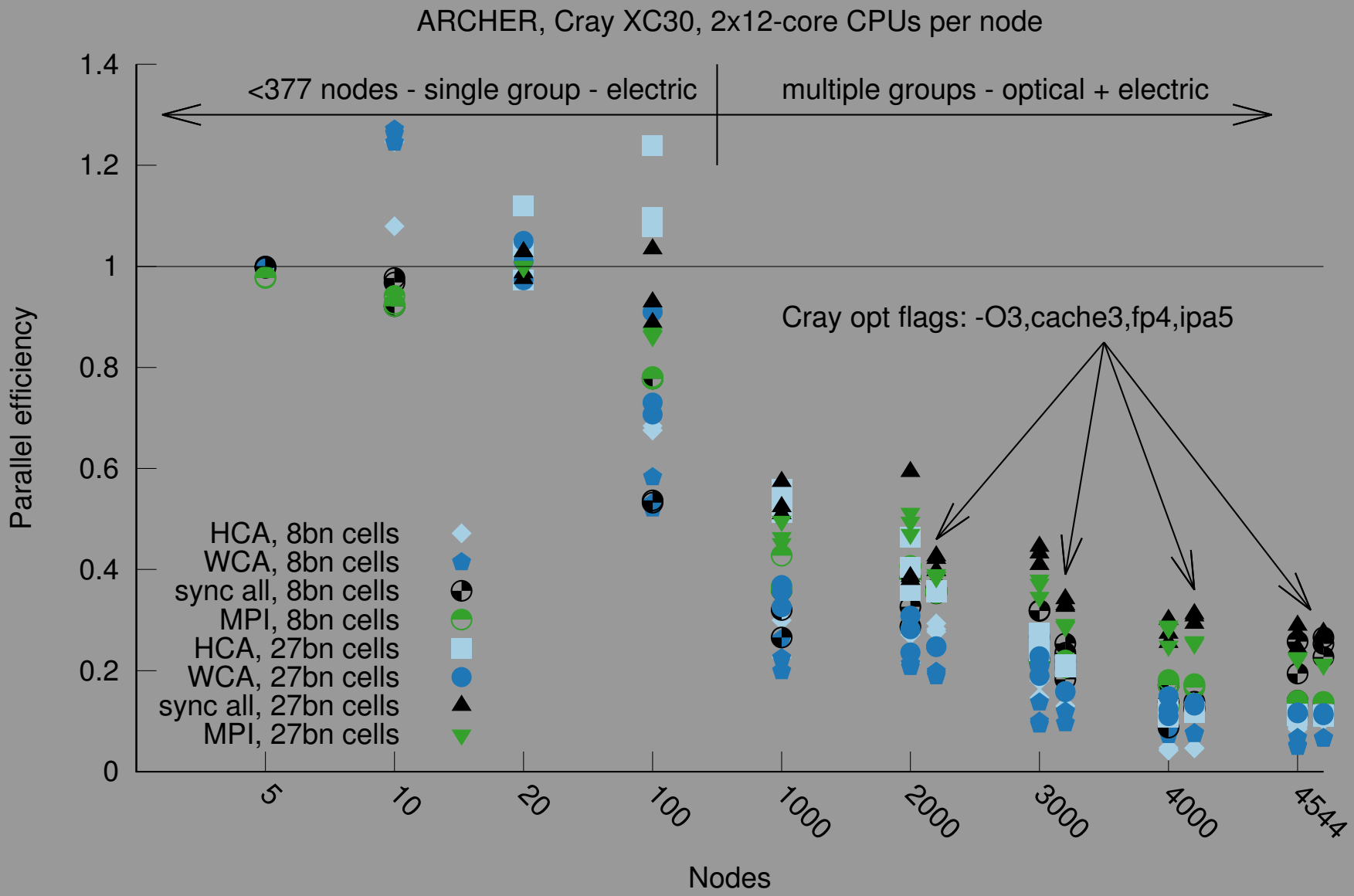


8bn cells



27 bn cells

Parallel efficiency – significant scatter



Baseline for 8bn cells model is 5 nodes; for 27bn cells model is 20 nodes.

Conclusions and future

- Scaling beyond 100k cores is achieved
- CASUP coarrays library is a useful application benchmark

Want fully asynchronous flexible portable library, but...

- Modular library $\rightarrow$ poor optimisation $\rightarrow$ poor scaling.
- Coarrays global barrier – SYNC ALL – (nearly) as good as MPI
- Coarrays SYNC IMAGES worse than MPI ISEND/IRECV
- Future: 1D partition – fewer large messages in HX
- Future: coarray EVENTs + atomics – better possibility for async algorithms

Looking for collaborations – async algorithms, single sided comms for exascale, further comparisons of coarrays vs ...

Acknowledgements

We acknowledge financial and other support from the the following organisations.

- EPSRC grants EP/R013047/1, EP/P034446/1.
- ARCHER UK National Supercomputing Service, `http://www.archer.ac.uk`, projects eCSE05-05, e560
- Advanced Computing Research Centre, University of Bristol, `https://www.acrc.bris.ac.uk`