Scaling Cellular Automata beyond 100,000 cores

Coarrays vs MPI + OpenMP + do concurrent

Anton Shterenlikht
The University of Bristol

Contact Information:
Mech Eng Dept
The University of Bristol
University Walk, Bristol BS8 1TR, UK

Phone: +44 117 33 15944
Email: `mexas@bristol.ac.uk`

Abstract

CASUP (`cgpack.sf.net`) is a generic HPC cellular automata (CA) library. In this work it was applied to 3D Ising magnetisation calculations. Scaling of two halo exchange (HX) methods (Fortran coarrays, MPI) and of three CA loop routines (triple nested loop, do concurrent, OpenMP) was measured on ARCHER up to full machine capacity, 4544 nodes (109,056 cores). Ising energy was calculated with MPI_ALLREDUCE and Fortran 2018 CO_SUM collectives. Using fully populated nodes and no threading gave the highest performance, probably because the Ising model is perfectly load balanced. MPI HX scaled better than coarray HX, which is surprising because both algorithms use pair-wise ”handshake” (IRECV/ISEND/WAITALL vs SYNC IMAGES).

Introduction

CASUP solidification scaled to 32k cores on HECToR [1]. CASUP/ParaFEM (`parafem.org.uk`) cellular automata finite element (CAFE) multi-scale fracture simulation scaled to 8k cores on ARCHER [2]. In this work CASUP has been completely re-designed:

1. All synchronisation is now *hidden from the user*. CASUP automatically inserts the minimum number of sync calls to ensure data integrity. The user never needs to worry about data integrity, i.e. no sync calls in CASUP miniapps.
2. CASUP is now completely *modular*, i.e. the CA kernels, the halo exchange or collective routines can be swapped with ease, allowing performance analysis of multiple combinations of Fortran coarray, MPI, OpenMP and do concurrent (Fig. 1).

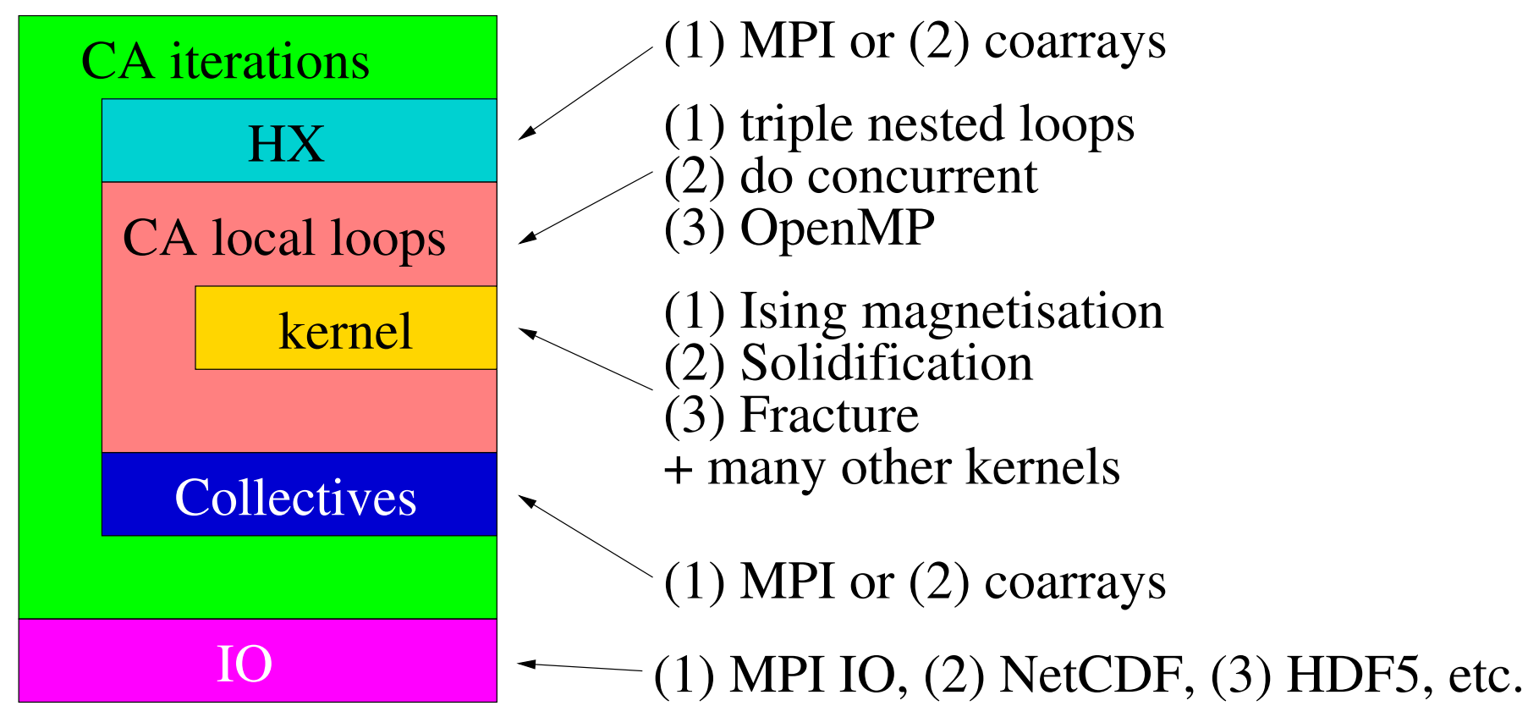

CA iterations
HX
CA local loops
kernel
Collectives
IO
(1) MPI or (2) coarrays
(1) triple nested loops
(2) do concurrent
(3) OpenMP
(1) Ising magnetisation
(2) Solidification
(3) Fracture
+ many other kernels
(1) MPI or (2) coarrays
(1) MPI IO, (2) NetCDF, (3) HDF5, etc.

Figure 1: Schematics of the modular structure of a CASUB library.

- Performance of the new CASUP miniapps is the focus of this work.
- 3D domain decomposition and von Neumann neighbourhood (6 nearest neighbours in 3D) were used.

Halo exchange (HX)

Two possibilities for HX exist when using coarrays: (1) creating the whole of the CA model of coarrays, and (2) using coarrays only for CA halos. HX in the second method involves a separate step, copying of the boundary CA data into coarray arrays, as shown in Fig. 2.

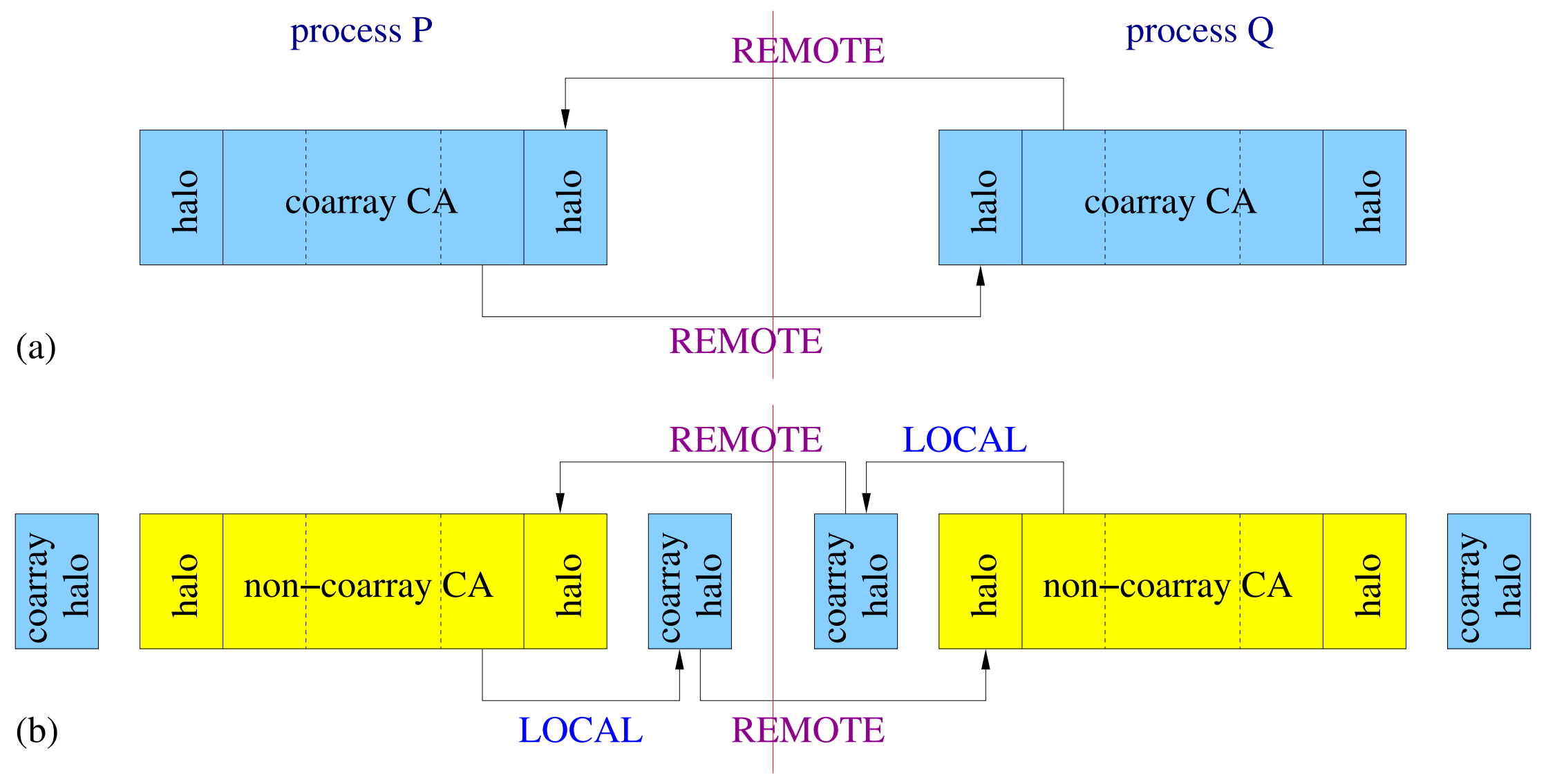


Figure 2: Schematics of HX in 1D, showing (a) a one-step algorithm (MPI and whole CA model coarrays) and (b) a two-step algorithm used when coarrays are used only for CA halos. Local copies and remote comms are shown with arrows.

- Separate halo coarrays, `space` is an non-coarray array

  Prepare coarray halos `h1minu, h1plus`, local op:

```
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
```

  Use halo coarrays, actual HX step, remote comms:

```
if ( ci(1) .ne. 1 ) then                    ! all but the  leftmost image
  sync images( nei_img_L(1) )               ! sync with the left image
  space( lhsta(1):0, 1:sub(2), 1:sub(3) ) = &              ! HX, remote op
    h1plus(:,:,:) [ nei_ci_L1(1), nei_ci_L1(2), nei_ci_L1(3) ]
end if
if ( ci(1) .ne. ucob(1) ) then              ! all but the rightmost image
  sync images( nei_img_R(1) )               ! sync with the right image
  space( rhsta(1) : rhend(1), 1:sub(2) , 1:sub(3) ) = &  ! HX, remote op
    h1minu(:,:,:) [ nei_ci_R1(1), nei_ci_R1(2), nei_ci_R1(3) ]
end if
```

- Whole model coarrays, `space` is an array coarray

```
if ( ci(1) .ne. 1 ) then                    ! all but the  leftmost image
  sync images( nei_img_L(1) )               ! sync with the left image
  space( lhsta(1):0,           1:sub(2) , 1:sub(3) ) = & ! HX, remote op
    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
if ( ci(1) .ne. ucob(1) ) then              ! all but the rightmost image
  sync images( nei_img_R(1) )               ! sync with the right image
  space( rhsta(1) : rhend(1) , 1:sub(2) , 1:sub(3) ) = & ! HX, remote op
    space( 1 : hdepth        , 1:sub(2) , 1:sub(3) )    &
      [ nei_ci_R1(1), nei_ci_R1(2), nei_ci_R1(3) ]
end if
```

- MPI HX

  Using derived types:

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

  `space` can be a coarray or a non-coarray array:

```
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
if ( ci(2) .ne. 1 ) then
  call MPI_IRECV( space, 1, mpi_h2_LV, nei_img_L(2)-1, TAG2R,          &
    MPI_COMM_WORLD, reqs2m(1), ierr )
  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
```

3D Ising magnetisation

Extension of the Q2R Vichniac’s 2D rule [3] to 3D was implemented:

- Spins of 0 (down) or 1 (up).
- 3D mask array:

```
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
```

- A spin flips iif 3 neighbours are 0 and the other 3 are 1:

```
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 - s(i,j,k)
else
  ! Otherwise no change
  ca_kernel_ising = s(i,j,k)
end if
end associate
```

- Two-step iteration to conserve energy

```
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 - mask_array               ! Flip the mask array
end do
```

Results

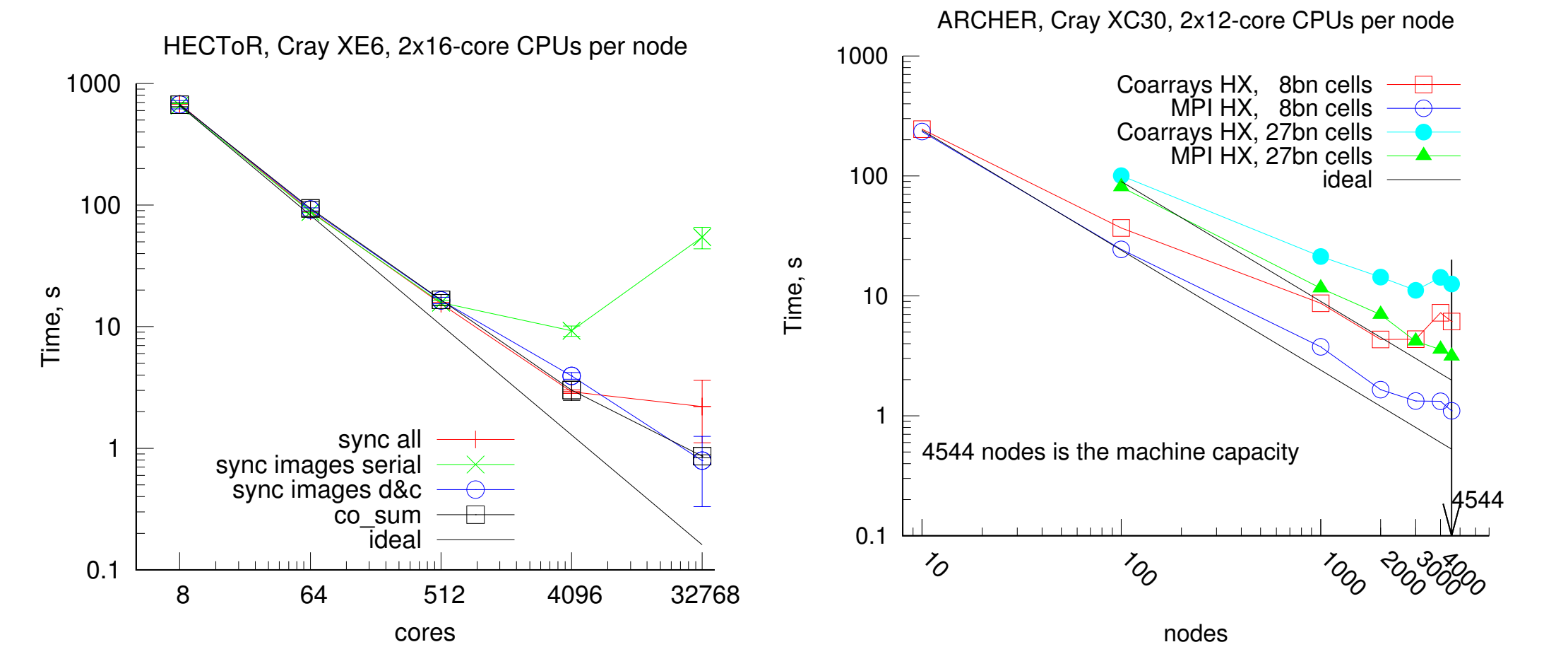


Figure 3: CASUP scaling: (left) 2013 solidification results [4] and (right) this work, Ising magnetisation.

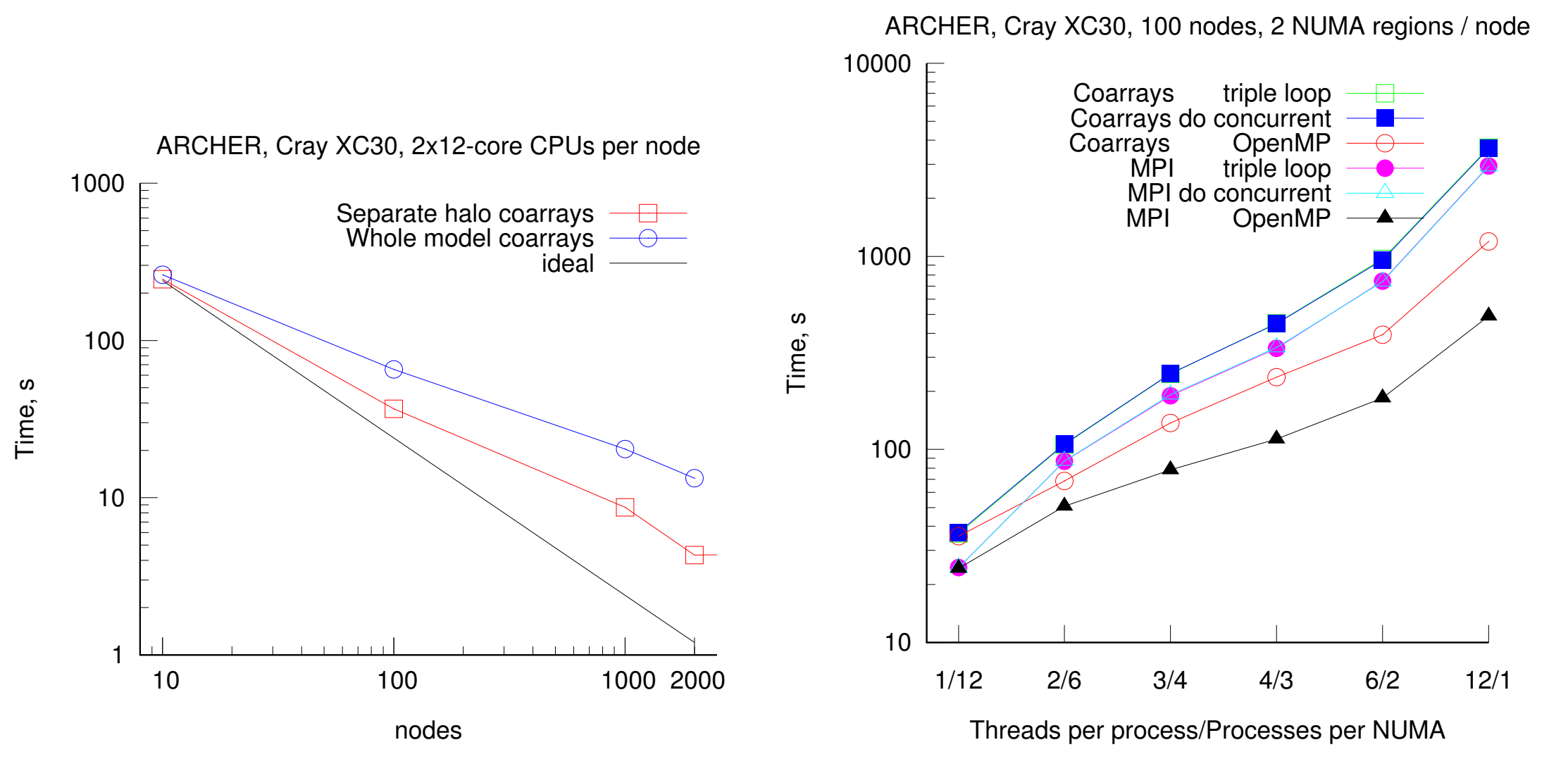


Figure 4: Performance of Ising magnetisation miniapps: (a) Whole model coarrays vs using coarrays only for halos; (b) Addition of threading.

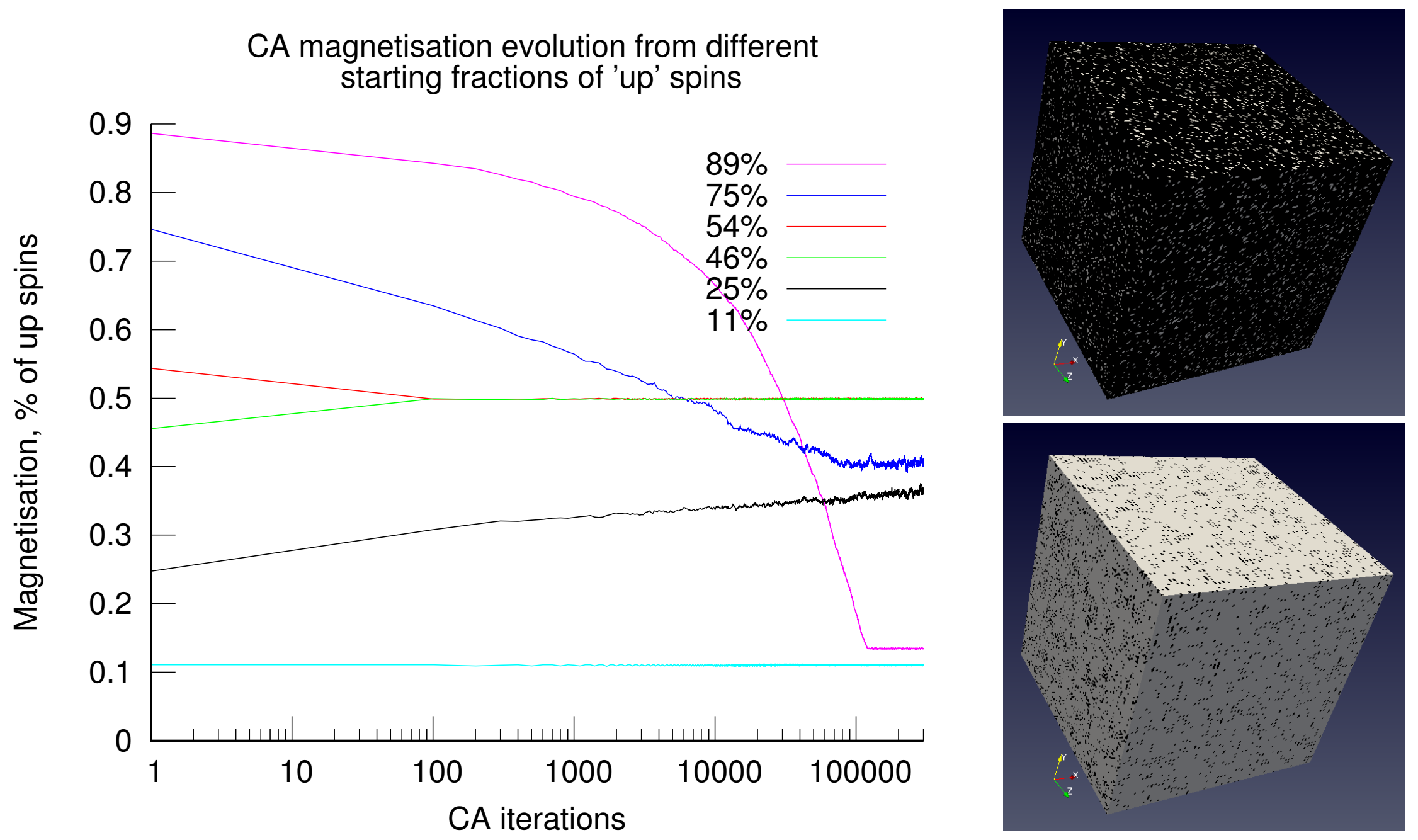


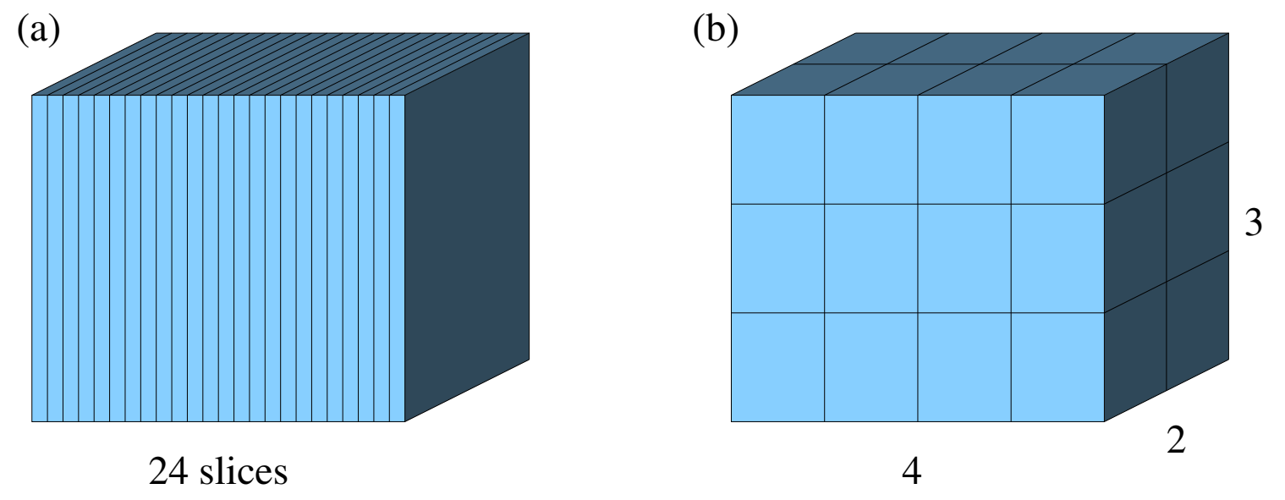
Figure 5: Ising magnetisation: (left) 3 distinct stable regions identified; (right) CA with 90% of ‘up’ spins (top) evolves to 13% of ‘up’ spins in ∼100k iterations (bottom). ‘Up’ spin cells are are black. ‘Down’ spin cells are white.

Conclusions

- MPI HX performs better than Fortran coarrays HX.
- Virtuous loop: more coarray usage → better support by vendors → higher performance → more coarray usage, etc.
- Whole model coarrays perform worse than when coarrays used only for halos. This is unexpected.
- Any amount of threading lowers performance. This is likely due to very little load imbalance in Ising magnetisation CA.

Future

- 1D vs 3D domain decomposition will be explored. 3D decomposition uses smaller messages, but ×3 more messages:



- Asynchronous coarrays HX using F2018 EVENTs + atomics. This might improve performance, particularly if load balancing is poor.
- Allocatable vs F2008 CONTIGUOUS assumed shape arrays.

Acknowledgments

This work used the ARCHER UK National Supercomputing Service (`http://www.archer.ac.uk`).

References

[1] A. Shterenlikht and L. Margetts. *Proc. Roy. Soc. A*, 471:20150039, 2015. DOI: 10.1098/rspa.2015.0039.

[2] A. Shterenlikht *et al* . In *PGAS Appl. Workshop, SC17*. DOI: 10.1109/PAW.2016.006.

[3] B. Chopard and M. Droz. *Cellular Automata Modelling of Physical Systems*. Cambridge, 1998.

[4] A. Shterenlikht. In *Proc. 7th PGAS Conf., Edinburgh*, 2013. http://www.pgas2013.org.uk/programme.