Cellular Automata Beyond 100k Cores: MPI vs Fortran Coarrays

Anton Shterenlikht\textsuperscript{1} and Luis Cebamanos\textsuperscript{2}

\textsuperscript{1}Mechanical Engineering Department, The University of Bristol, Bristol BS8 1TR, UK, mexas@bristol.ac.uk
\textsuperscript{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
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:**
- 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

```fortran
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 FD\S, 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
- MOVEALLOC
- 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


---

![Graphs showing weak scaling benchmarks](image)

**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)
Tc1999L137 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.

Higher is better
Coarray IO - no native Fortran parallel IO

- MPI/IO up to 2.3GB/s on Cray XE6
- 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.
- IFS 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: $\text{CA} + \text{FE} = \text{CAFE}$ multi-scale model

- **Structured grids** - CA, **unstructured grids** - FE
- CA via coarrays $\rightarrow$ **Easy halo exchange**
- CA (microstructure) + FE (continuum mechanics) $= \text{CAFE}$
- Transgranular cleavage - fracture stress or strain criteria
- FE $\rightarrow$ CA (localisation) - stress, strain fields
- CA $\rightarrow$ FE (homogenisation) - damage variables
Cleavage in a steel bar under tension
Modular structure of CASUP library – many combinations

- **IO**
- **Collectives**
- **CA local loops**
- **HX**
- **CA iterations**

- (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.
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:

```fortran
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):rhehend(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:

```fortran
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):rheend(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:

```fortran
 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:

```fortran
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

```fortran
!$omp parallel do default( none ) &
!$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:

```fortran
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):

```plaintext
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
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?


- 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.

from: *XC Series GNI and DMAPP API User*
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

<table>
<thead>
<tr>
<th>Group</th>
<th>HCA</th>
<th>WCA</th>
<th>MPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Triple loop + Ising kernel, %</td>
<td>19.3</td>
<td>17.9</td>
<td>27.9</td>
</tr>
<tr>
<td>Ising energy + collectives, %</td>
<td>29.6</td>
<td>33.6</td>
<td>21.2</td>
</tr>
<tr>
<td>HX, %</td>
<td>28.3</td>
<td>29.1</td>
<td>17.0</td>
</tr>
<tr>
<td>Total, %</td>
<td>77.2</td>
<td>80.6</td>
<td>66.1</td>
</tr>
<tr>
<td>Total time, s</td>
<td>191</td>
<td>210</td>
<td>120</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Group</th>
<th>HCA</th>
<th>WCA</th>
<th>MPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Triple loop + Ising kernel, %</td>
<td>4.6</td>
<td>3.2</td>
<td>5.0</td>
</tr>
<tr>
<td>Ising energy + collectives, %</td>
<td>27.1</td>
<td>26.2</td>
<td>22.2</td>
</tr>
<tr>
<td>HX, %</td>
<td>25.0</td>
<td>23.3</td>
<td>16.2</td>
</tr>
<tr>
<td>Total, %</td>
<td>56.7</td>
<td>52.7</td>
<td>43.4</td>
</tr>
<tr>
<td>Total time, s</td>
<td>59</td>
<td>82</td>
<td>44</td>
</tr>
</tbody>
</table>

- MPI → more calculation than comms.

- Coarrays → more comms than calculation.
Profiling – 100 nodes – inconclusive

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

<table>
<thead>
<tr>
<th>Samp%</th>
<th>Samp</th>
<th>Imb.</th>
<th>Imb.</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| | | | | |
| | | | | |

<table>
<thead>
<tr>
<th>Samp%</th>
<th>Samp</th>
<th>Imb.</th>
<th>Imb.</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>77.7%</td>
<td>18,256.5</td>
<td>--</td>
<td>--</td>
<td>USER</td>
</tr>
</tbody>
</table>

| | | | | |
| | | | | |

| 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_energy$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_energy$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

<table>
<thead>
<tr>
<th>Samp%</th>
<th>Samp</th>
<th>Imb.</th>
<th>Imb.</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>56.8%</td>
<td>5,693.3</td>
<td>--</td>
<td>--</td>
<td>USER</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25.3%</td>
<td>2,533.7</td>
<td>2,581.3</td>
<td>50.5%</td>
<td>ca_ising_energy_col$ca_hx_</td>
</tr>
<tr>
<td>25.0%</td>
<td>2,504.0</td>
<td>2,419.0</td>
<td>49.1%</td>
<td>ca_hx_all$ca_hx_</td>
</tr>
<tr>
<td>2.6%</td>
<td>259.9</td>
<td>66.1</td>
<td>20.3%</td>
<td>ca_kernel_ising$ca_hx_</td>
</tr>
<tr>
<td>2.0%</td>
<td>200.5</td>
<td>62.5</td>
<td>23.8%</td>
<td>ca_iter_tl$ca_hx_</td>
</tr>
<tr>
<td>1.8%</td>
<td>183.9</td>
<td>50.1</td>
<td>21.4%</td>
<td>ca_kernel_ising_ener$ca_hx_</td>
</tr>
</tbody>
</table>

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!

Just HX – massive difference!

ARCHER, Cray XC30

6 nodes

42 and 334 nodes
Global barrier – sync all – very close to MPI!

4544 nodes is the machine capacity

8bn cells

27 bn cells
Parallel efficiency – significant scatter

ARCHER, Cray XC30, 2x12-core CPUs per node

Nodes

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

Cray opt flags: -O3, cache3, fp4, ipa5

Anton Shterenlikht and Luis Cebamanos

MPI vs Fortran coarrays

EuroMPI 2018 31 / 33
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 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