Fortran 2008, 2018 coarrays and OpenMP

Anton Shterenlikht

Mech Eng Dept, The University of Bristol, Bristol BS8 1TR, UK
mexas@bristol.ac.uk

UK OpenMP Users' Conference 2018, Oxford, 22-MAY-2018

Table of Contents

Fortran support in OpenMP

- OMP 4.5 (2015), 5.0 (2017) support (*most of*) F2003 (2004).
  Main exclusions: OO, IEEE features.
  → *11-year lag!*
- F2008 published in 2010. F2018 will likely be published in 2018.
  → *Support in OMP 6? 7?*
- OMP ARB: alternate J3 Fortran members:
  → *Kelvin Li* (IBM), *Henry Jin* (NASA).
  `http://www.openmp.org/about/members`
- J3 Fortran: OpenMP liaison: → *Bill Long* (Cray)
  `http://j3-fortran.org/doc/standing/links/001.txt`

Who cares anyway?
Fortran coarray users!

Fortran coarrays - native SPMD

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

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 DIS, 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

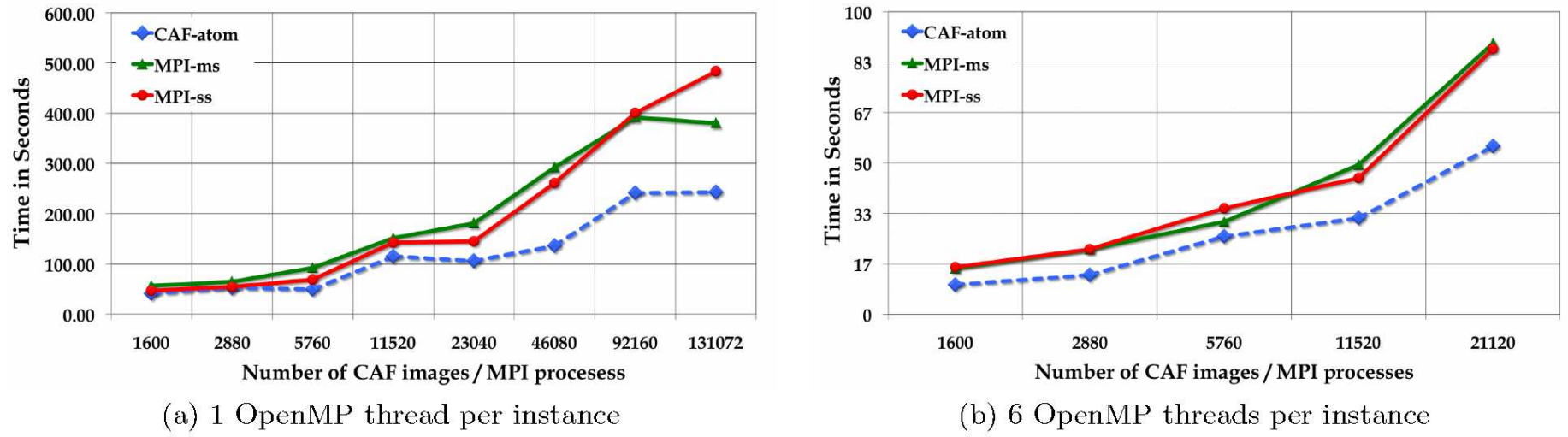
1. Gyrokinetic particle-in-cell code
2. Numerical weather prediction: European Center for Medium Range Weather Forecast, Integrated Forecasting System (ECMWF IFS)
3. Physics/engineering: Cellular Automata library for SUPercomputers, CASUP: The University of Bristol
   `https://cgpack.sourceforge.io`

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`



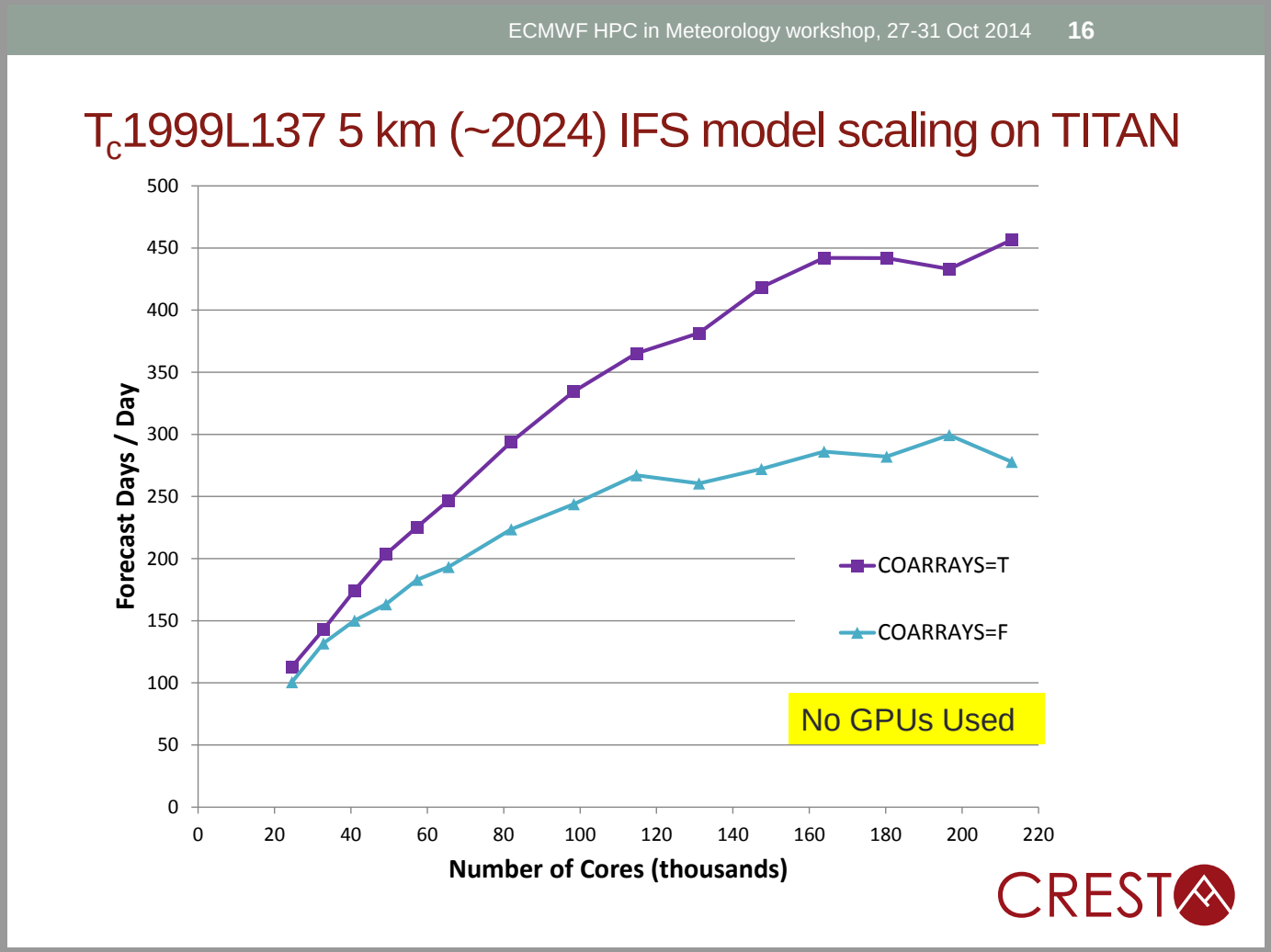
(a) 1 OpenMP thread per instance

(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



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

ECMWF IFS (cont'd) - *coarray segment order rules?*

Fortran2008 coarray (PGAS) example

```
!$OMP PARALLEL DO SCHEDULE(DYNAMIC,1) PRIVATE(JM,IM,JW,IPE,ILEN,ILENS,IOFFS,IOFFR)
DO JM=1,D%NUMP
  IM = D%MYMS(JM)
  CALL LTINV(IM,JM,KF_OUT_LT,KF_UV,KF_SCALARS,KF_SCDERS,ILEI2,IDIM1,&
    & PSPVOR,PSPDIV,PSPSCALAR ,&
    & PSPSC3A,PSPSC3B,PSPSC2 , &
    & KFLDPTRUV,KFLDPTRSC,FSPGL_PROC)
  DO JW=1,NPRTRW
    CALL SET2PE(IPE,0,0,JW,MYSETV)
    ILEN = D%NLEN_M(JW,1,JM)*IFIELD
    IF( ILEN > 0 )THEN
      IOFFS = (D%NSTAGT0B(JW)+D%NOFF_M(JW,1,JM))*IFIELD
      IOFFR = (D%NSTAGT0BW(JW,MYSETW)+D%NOFF_M(JW,1,JM))*IFIELD
      FOUBUF_C(IOFFR+1:IOFFR+ILEN)[IPE]=FOUBUF_IN(IOFFS+1:IOFFS+ILEN)
    ENDIF
    ILENS = D%NLEN_M(JW,2,JM)*IFIELD
    IF( ILENS > 0 )THEN
      IOFFS = (D%NSTAGT0B(JW)+D%NOFF_M(JW,2,JM))*IFIELD
      IOFFR = (D%NSTAGT0BW(JW,MYSETW)+D%NOFF_M(JW,2,JM))*IFIELD
      FOUBUF_C(IOFFR+1:IOFFR+ILENS)[IPE]=FOUBUF_IN(IOFFS+1:IOFFS+ILENS)
    ENDIF
  ENDDO
ENDDO
!$OMP END PARALLEL DO
SYNC IMAGES(D%NMYSETW)
FOUBUF(1:IBLEN)=FOUBUF_C(1:IBLEN)[MYPROC]
```

Coarrays/OMP problems

1. High level abstraction, multiple implementations, e.g. as MPI processes on coprocessors (ifort 16+).
   Clash with OMP target device specs?
2. All rules based on the image level.
   No "sub-image" (thread, fine grain) level control semantics?
3. Performance with full standard conformance.
   Fully asynchronous execution on thread level?

Coarrays *inside* OMP parallel regions - *undefined!*

Can learn from MPI/OMP: `MPICH_MAX_THREAD_SAFETY`

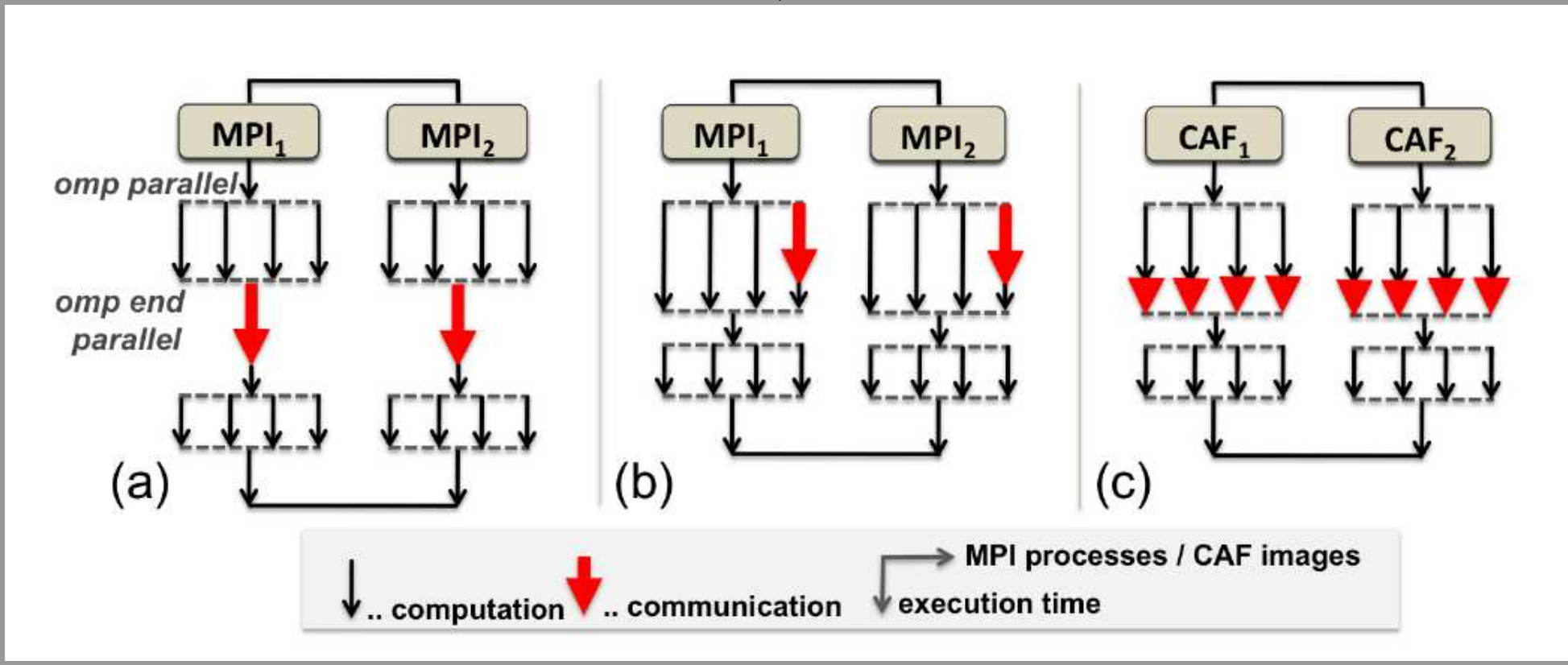
```
Specifies the maximum allowable thread-safety level that
is returned by MPI_Init_thread() in the provided argument.
This allows the user to control the maximum level of
threading allowed. The legal values are:
-------------------------------------------------------
Value              MPI_Init_thread() returns
-------------------------------------------------------
single             MPI_THREAD_SINGLE
funneled           MPI_THREAD_FUNNELED
serialized         MPI_THREAD_SERIALIZED
multiple           MPI_THREAD_MULTIPLE
-------------------------------------------------------
```

from Cray intro_mpi(3)

Coarrays/OMP can work up to SERIALIZED ?

COARRAYS_THREAD_*?

single | funneled/serialised | multiple



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

- COARRAYS_THREAD_* - support level by OpenMP?

Coarray comms *outside* OMP parallel regions

```
integer :: a(100,100,100)[*]
main: do iter=1,niter
  ! pairwise sync, e.g. sync images
  call halo_exchange(a) ! Remote calls
  !$omp parallel do shared(a) num_threads(...)
  do i=1,n
   ! Update/use "a" on my image. No remote calls
  end do
  !$omp end parallel do
end do main
```

- Each image spawns threads
- Same or different number of threads per image
- Similar to `MPI_THREAD_SINGLE` mode

COARRAYS_THREAD_SERIALIZED?

```
integer :: a(0:n+1)[*], b(0:n+1), img, tmp
!$omp parallel do private(i,tmp) shared(img,a,b)
loop: do i=1, n
 if (img .eq. 1 .and. i .eq. n ) then
  sync images (2) ! The thread that has i=n on img 1
  a(n+1)=a(1) [2] ! will sync with img 2 and pull a(1).
 end if
 if (img .eq. 2 .and. i .eq. 1 ) then
  sync images (1) ! The thread that has i=1 on img 2
  a(0) = a(n) [1] ! will sync with img 1 and pull a(n)
 end if
 ! kernel function: b(i) = fun( a )
end do loop
!$omp end parallel do
```

Coarrays/OMP serialised - complete program

```
integer, parameter :: n=10 ! Assume 2 images!
integer :: a(0:n+1)[*], b(0:n+1)=0, img, iter
img = this_image()
if (img .eq. 2 ) b(n+1) = 1                    ! Single non-zero element
main: do iter = 1, 2*n
  a = b                                        ! Halo defined outside OMP
  !$omp parallel do default(none) private(i,tmp) &
  !$omp shared(img,a,b)
  loop: do i=1, n
    if (img .eq. 1 .and. i .eq. n ) then ! hx on img 1
      sync images   (2)                        ! on thread which
      a(n+1) = a(1) [2]                        ! has i=n.
    end if
    if (img .eq. 2 .and. i .eq. 1 ) then ! hx on img 2
      sync images   (1)                        ! on thread which
      a(0)   = a(n) [1]                        ! has i=1.
    end if
    b(i) = a(i+1)                              ! kernel function
  end do loop
  if (img .eq. 2 ) b(n+1) = a(0)
  !$omp end parallel do
end do main
end
```

Coarrays/OMP serialised - results - all ok!

- Kernel: copy the value from the neighbour on the right:

```
b(i) = a(i+1)
```

- 2 images, 4 threads/image
- HX on image 1 → always thread 3. HX on image 2 → always thread 0. *Expected but irrelevant - could be any thread!*
- Array section `b(1:10)` (no halos) initially:

```
00000 00000 | 00000 00001
<- img 1 -> | <- img 2 ->
```

- `b(1:10)` after 10 iterations:

```
00000 00000 | 11111 11111
<- img 1 -> | <- img 2 ->
```

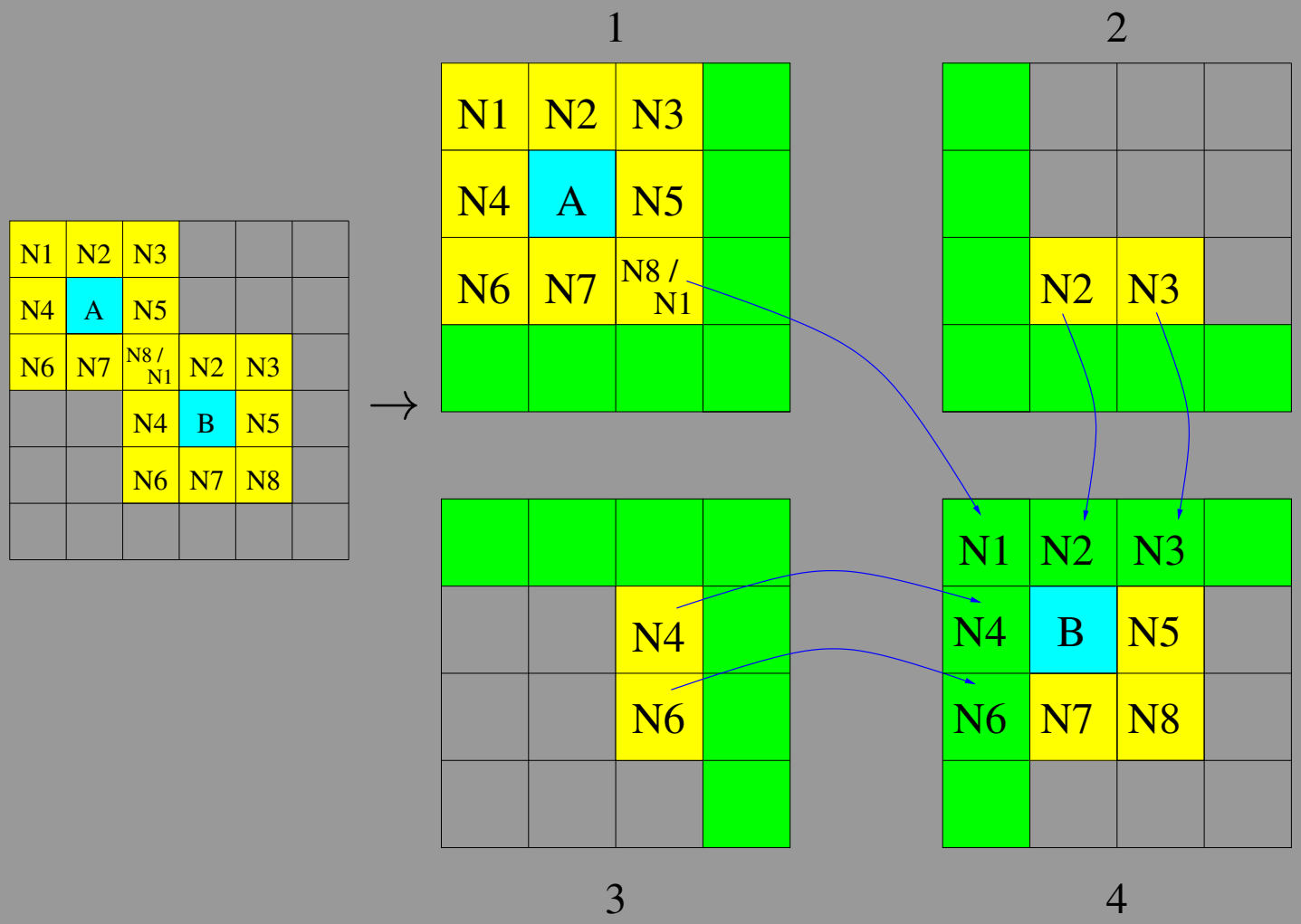
- `b(1:n 10)` after 20 iterations:

```
11111 11111 | 11111 11111
<- img 1 -> | <- img 2 ->
```

Coarrays/OMP serialised - conclusions

- Ok for 1D arrays - single thread (single, funneled, serialised) - single element copy
- *Will not work for 2D+ arrays*

Coarrays/OMP multiple - *problems!*



- Naive: every thread does its own HX.
- Unsafe with multiple threads - *might break sync rules* → deadlocks!

Coarrays/OMP multiple - SYNC IMAGES - no luck!

> *Executions of SYNC IMAGES statements on images M and T correspond if the number of times image M has executed a SYNC IMAGES statement in the current team with T in its image set is the same as the number of times image T has executed a SYNC IMAGES statement with M in its image set in this team.*

To avoid <u>races</u> and <u>deadlocks</u>, need to ensure:

1. Segment ordering rules are *not broken*, and
2. SYNC IMAGES statements are *corresponding*

Coarrays/OMP multiple - *deadlocks?*

image 1

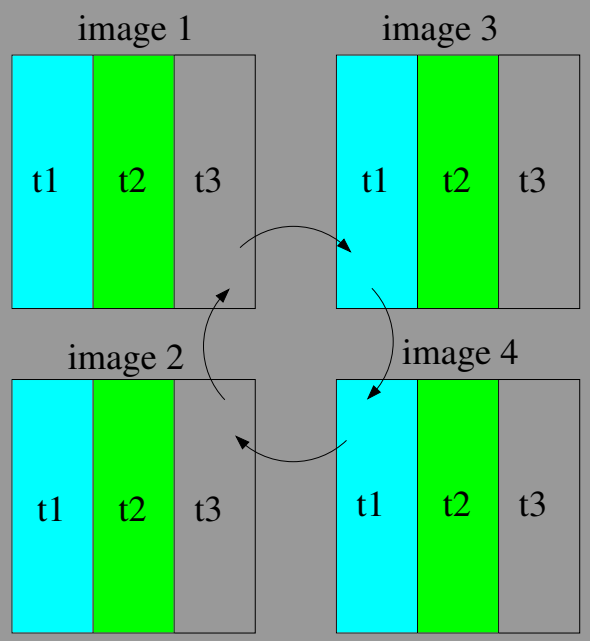
```
if (this_image() .eq. 1) then
 sync images(2) ! then HX
end if                  ! thread 1
```

```
if (this_image() .eq. 1) then
 sync images(3) ! then HX
end if                  ! thread 2
```

image 3

```
if (this_image() .eq. 3) then
 sync images(1) ! then HX
end if                  ! thread 2
```

```
if (this_image() .eq. 3) then
 sync images(4) ! then HX
end if                  ! thread 1
```



- Order of SYNC IMAGES invocation unpredictable
- Circular dependency
- All wait → deadlocks.
- *Cannot use SYNC IMAGES in Multiple mode?*
- Need a finer grain sync mechanism

Coarrays/OMP multiple - Locks? Critical? Events? - *same problem*

Also unsuitable:

- LOCK / UNLOCK - image control statement
- CRITICAL / END CRITICAL - image control statement
- EVENT POST / EVENT WAIT - image control statement

Sync too coarse. Not thread-safe! Breaks OMP rules:

> *All library, intrinsic and built-in routines provided by the base language must be thread-safe in a compliant implementation. In addition, the implementation of the base language must also be thread-safe.*

Async thread execution is likely to *break the segment ordering rules*. Need "*sub-image*" level sync.

Coarrays/OMP multiple - no image control statements!

Any thread that needs a halo element does a remote coarray read.

```
real :: a(0:n+1,0:n+1)[2,*], b(0:n+1,0:n+1)
integer :: grid(2), i, j, img, iter
 grid = this_image( a )
main: do iter = 1, niter
 a = b
 sync all ! multiple SYNC IMAGES in production code
 !$omp parallel do default(none) shared(n,a,b,grid)
 do j = 1, n                          ! Any thread can do remote
 do i = 1, n                          ! coarray reads
  if (i.eq.n.and.grid(1).ne.2) a(n+1,j)=a(1,j)[grid(1)+1, grid(2)]
  if (i.eq.1.and.grid(1).ne.1) a(0,  j)=a(n,j)[grid(1)-1, grid(2)]
  if (j.eq.n.and.grid(2).ne.2) a(i,n+1)=a(i,1)[grid(1), grid(2)+1]
  if (j.eq.1.and.grid(2).ne.1) a(i,  0)=a(i,n)[grid(1), grid(2)-1]
  b(i,j) = 0.25 * ( a(i-1,j) + a(i+1,j) + a(i,j-1) + a(i,j+1) )
 end do
 end do
 !$omp end parallel do
end do main
```

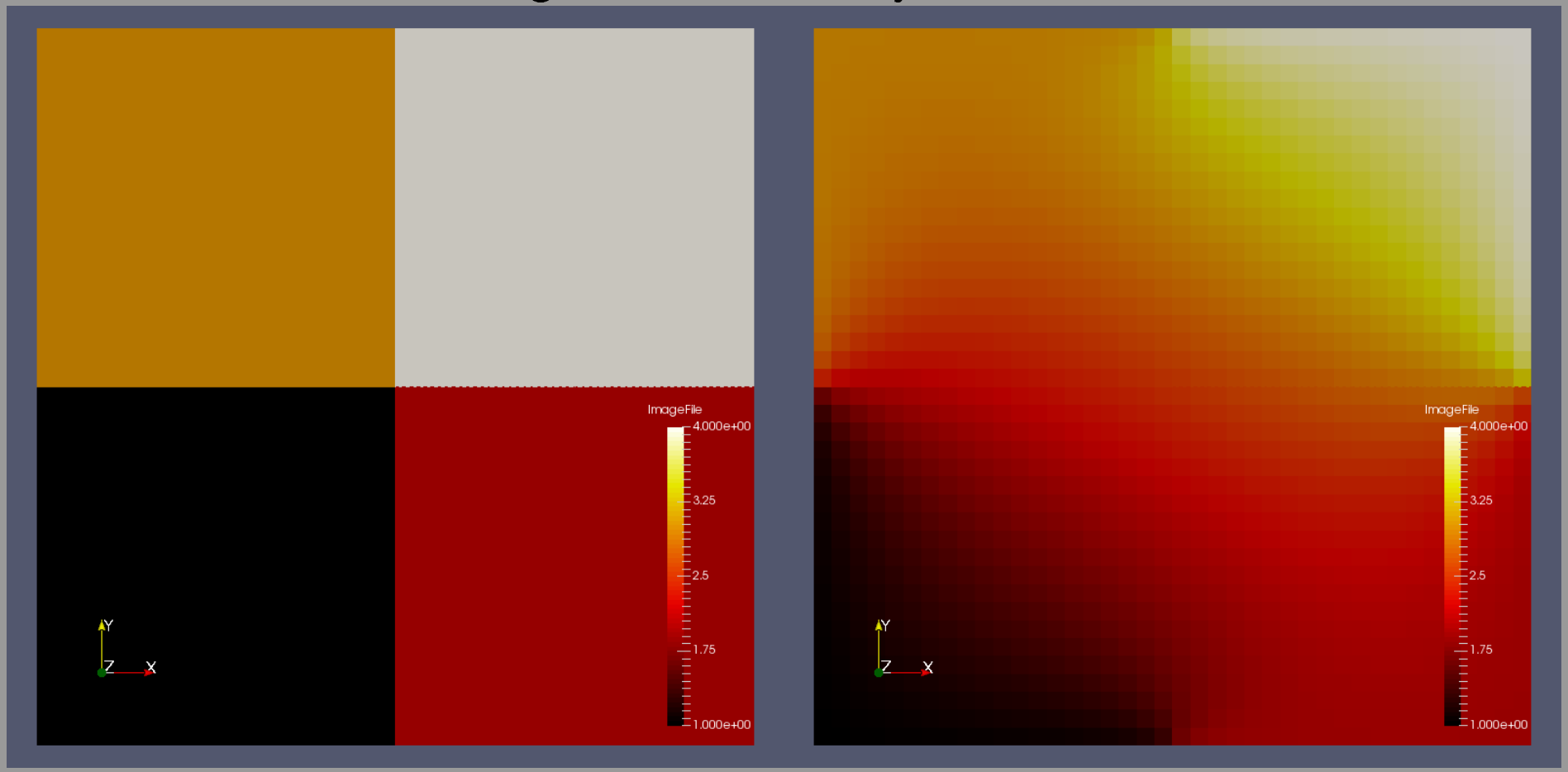
Sync still needed *outside* OMP parallel regions, → limits async execution opportunities → no performance gain?

Coarrays/OMP multiple - no image control statements!

A relaxation kernel example.

4 images, 20×20 coarray array per image $\rightarrow 40 \times 40$ model.

Values are set to the image number initially.



Start

After 1000 iterations

Coarrays/OMP multiple - Atomics?

For any two executions in unordered segments of atomic subroutines whose ATOM argument is the same object, the effect is as if one of the executions is performed completely before the other execution begins.

- But... only Integer and Logical types
- Seems no solution for other data types...

Further considerations - DO CONCURRENT

- A DO loop where the order of iterations is immaterial.
- Programmer guarantees to the compiler that such loop is parallelisable.
- Can be implemented via OpenMP.
- Severe restrictions, e.g:

  > *An image control statement shall not appear within a DO CONCURRENT construct.*

Possible solution → "split" HX (as in the relaxation example above):

- Coarray definitions *outside* OMP parallel regions
- Coarray references *inside* OMP parallel regions

Conclusions - Challenges for ARB and Fortran committee

1. OpenMP/Coarrays specs are overdue...
2. Serialised coarray comms from OMP parallel regions - ok
3. Multiple coarray comms from OMP parallel regions - unlikely to work

Challenges for...
ARB:

- Coarrays/OMP $\neq$ MPI/OMP! New rules needed.
- OMP requirements - '*the base language must ... be thread-safe*' - need rethinking for F2008, F2018 - *parallelism is built into the language*!

Fortran committee:

- Richer atomics for unordered segments? For all intrinsic types? Derived types?

Acknowledgements

The author would like to thank Bill Long (Cray) for helpful suggestions, and to 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`
- Advanced Computing Research Centre, University of Bristol, `https://www.acrc.bris.ac.uk`