Migrating A Scientific Application from MPI to Coarrays

John Ashby and John Reid
HPCx Consortium
Rutherford Appleton Laboratory
STFC
UK

CUG 2008
Crossing the Boundaries

Why and Why Not?

+ MPI programming is arcane
+ New emerging paradigms for parallelism
+ Coarrays part of the next Fortran Standard
+ Gain experience, make informed recommendations
- Established MPI expertise
- MPI widely available – coarrays only available on (some) Crays.

Coarray Fortran in a nutshell

- SPMD paradigm, instances of the program are called images, have their own local data and run asynchronously.
- Data can be directly addressed across images: `A(j,k)[i]. i` is image index.
- Subroutine calls to synchronize execution.

Coarray Fortran in a nutshell (2)

- Intrinsics for information: **`num_images()`**, **`this_image()`** and **`image_index()`**.
- Coarrays have the same cobounds on all images, but can have allocatable components:

```
type co_double_2
  double precision, allocatable:: array(:,:)
end type co_double_2
integer nx,ny
type(co_double_2) vel[*]
allocate(vel%array(nx,ny))
```

The Application

- SBLI: a three-dimensional time-dependent finite difference Navier-Stokes solver
- Grid transformation for complex geometries
- Parallelisation by domain decomposition and halo exchange.

Parallel sections in SBLI

- Initial data read in by “master” process, broadcast to all others.
- Grid read in by “master” process, distributed to others.
- Exchange of halo data.
- Solution gathered onto master process for output or written in parallel (MPI-IO).

Parameter Broadcast

- SBLI reads in data such as number of grid points, Reynolds number, which turbulence model to use. Only one process reads the data.
- MPI: these are packed into real, integer and logical arrays, sent to the other processes using MPI_BCAST.
- The receiving processes unpack the arrays:

Parameter Broadcast (2)

```
if (ioproc) then
   r(1) = reynolds
   ...
   r(18) = viscosity
   call mpi_bcast(r, 18, real_mp_type, ioid, &
   MPI_comm_world, ierr)
else
   call mpi_bcast(r, 18, real_mp_type, ioid, &
   MPI_comm_world, ierr)
   reynolds = r(1)
   ...
   viscosity = r(18)
endif
```

Parameter broadcast (3)

- Each CAF version fetches the data from the I/O image.

```
call sync_all()
if (.not. ioproc) then
    reynolds = reynolds[ioid]
    ...
    viscosity = viscosity[ioid]
end if
```

Mesh Distribution

- Here the i/o processor has the global data and needs to send different portions of it to each image.
- Added complication that the local bounds of the data may be different on different images.
- In current version mesh is 2-d, projected

Science & Technology Facilities Council
Rutherford Appleton Laboratory

CUG 2008

Mesh distribution (2)

- MPI version:

if (ioproc)

Find start and end indices of mesh for process j

Pack global mesh(start:end) into buffer

Send buffer to process j

else

Receive buffer

Unpack to local mesh

endif

Mesh Distribution (3)

- CAF version:

```
do j=1, num_images()
   find start and end for image j
   local(:)[j] = global(start:end)
end do
```

Mesh distribution(4)

- Better:

find start and end for this_image()

local(:) = global[ioid](start:end)

- Advantage
  - Possible parallelism if multiple access to global is supported

Halo Exchange

- The MPI version again packs the data to exchange into a buffer, sends it to the appropriate neighbour which unpacks it.
- The coarray version uses simple co-addressing:
- Example: sending data to the image one x-step lower (procmx):

Halo Exchange (2)

```
type(co_double_3) a[nxim, nyim,*]
integer nx[nxim, nyim,*], d(3), nxp
d = this_image(a)
if (d(1) .gt. 1) then
   nxp = array(nx[d(1)-1,d(2),d(3)]
   a[d(1)-1,d(2),d(3)]%array(nxp+1:nxp+xhalo,:,:)&
        = a%array(1:xhalo,:,:)
end if
```

- Separate routines cover x, y and z exchanges, each does both directions.

Caveat

- Synchronization is important
- MPI often implies synchronization
- Coarrays need it to be made explicit (though for some algorithms it can be left out or reduced)

Code Comparison Summary

- \+ Simple assignment statements replace MPI calls
- \+ No need to pack and unpack data (scope for programming errors)
- \+ Simpler, shorter, more maintainable code
- \- Added indirection through allocatable components

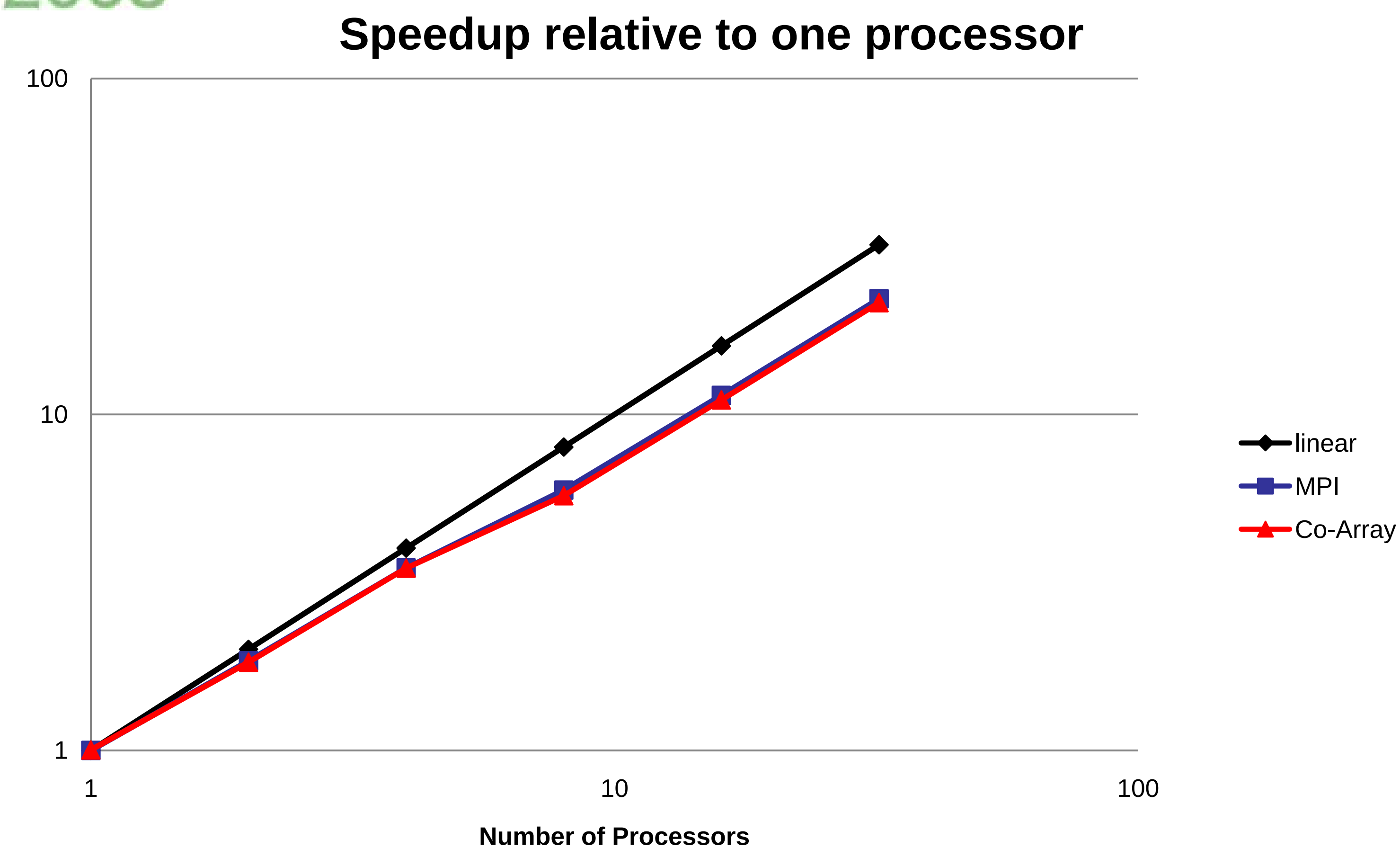
BUT…

- How does the code perform?
- Have we gained clarity and lost speed?
- SBLI is a mature code and a lot of work has gone into making its MPI work as efficiently as possible.

Experiments

- Small mesh (120 cubed)
- Small Cray X1E
- Run for 100 timesteps so overall time is dominated by the exchange time (realistic for how this code would work in production).

Speedup


Speedup relative to one processor
100
10
1
1
10
100
Number of Processors
linear
MPI
Co-Array

Performance

- Comparable with MPI (a few percent lower at most, but within the range of variability of individual runs)
- Scaling behaviour unaffected, but note this is a problem that scales strangely from 4 to 8 images, probably for memory reasons.

Optimization

- MPI is powerful and contains many ways of communicating which can be used by the programmer to optimize a code.
- Coarrays are simple and give plenty of scope for compiler optimization.
- But…

Optimization(2)

- There are a few things one can do:
  - Order of memory accesses, just as in serial Fortran, can have an impact
  - “push” vs “pull”
    - Which side of the assignment statement should one have the co-array reference?
    - Push: a[k] = a
    - Pull: a = a[k]

“push” vs “pull”

- Experiments: distribute 240 cubed mesh

Number of Processors	push	pull
8	2.289	1.492
16	2.154	1.406
32	1.427	0.593
64	1.018	0.644

- Pulling data is more efficient, especially at high processor counts

“push” vs “pull” (2)

- These experiments are indicative only
- Low impact on current code
- If your code does a lot of scatter/gather this is an area to optimize

Conclusions

- Coarray Fortran provides a language which:
  - Expresses parallelism in a “natural”, Fortran-like manner
  - Produces transparent, maintainable code
  - Is easy to learn by extending existing language skills
  - Provides comparable performance with mature MPI code in this case

CUG 2008

Science & Technology Facilities Council
Rutherford Appleton Laboratory

Acknowledgements

- Thanks to:
  - Bill Long of Cray for access to their X1 and X1E machines
  - Mike Ashworth and Roderick Johnstone of STFC Daresbury Laboratory for access to the SBLI code.