Performance of Fortran Coarrays on the Cray XE6
Contents

• Overview of PGAS
• MPI vs CAF vs UPC
• Fortran coarray syntax
• Benchmark design
• Platforms
• Results
• Conclusions
• Further work
• Acknowledgements

Performance of Fortran Coarrays on the Cray XE6

May 2012
PGAS languages and exascale

• Issues foreseen for MPI at large scale
  – manycore memory footprint
  – excessive synchronisation
  – high latency
  – message matching overheads
  – ...

• Partitioned Global Address Space models
  – private and shared data
  – direct read/write from/to remote memory
  – compiler support
  – no explicit buffering

• Two main implementations
  – Fortran Coarrays (part of Fortran 2008 standard)
  – Unified Parallel C
Message-Passing

- Memory
- CPU

May 2012
Performance of Fortran Coarrays on the Cray XE6
MPI_Send(a,...,1,...)  MPI_Recv(a,...,0,...)
Fortran coarrays

May 2012
Performance of Fortran Coarrays on the Cray XE6
Minimal syntax extension to Fortran

- SPMD execution model on multiple *images*

- Declare data to be remotely accessible
  
  ```fortran
  real, dimension(10), codimension[*] :: x
  real x(10)[*]
  ```

- Access remote data directly
  
  ```fortran
  x(2)  = x(3)[7]  ! remote read from image 7
  x(6)[4] = x(1)   ! remote write to image 4
  ```

- Explicit synchronisation: global and point-to-point
  
  ```fortran
  sync all; sync images(imagelist)
  ```

- Supports multiple dimensions and codimensions
What to benchmark?

- **Minimal syntax**
  - plus a few synchronisation intrinsics
  - no collectives (yet …)
  - different to, e.g., MPI and UPC

- **Need to decide on particular data access and communications patterns**

- **Several ways to describe same pattern**
  - array syntax, explicit loops, inline or subroutine, …
  - important to understand compiler capabilities
Initial benchmark

- Single contiguous point-to-point read and write
- Multiple contiguous point-to-point read and write
- Strided point-to-point read and write
- All basic synchronisation operations
- Various representative synchronisation patterns
- Halo-swapping in a multi-dimensional regular domain decomposition

- All communications include synchronisation cost
  - use double precision (8-byte) values as the basic type
  - use Fortran array syntax (mostly)

- Synchronisations overhead measured separately as
  \[ \text{overhead} = (\text{delay} + \text{sync}) - \text{delay} \]
Platforms

- **HECToR**
  - UK National Supercomputer
  - 90,000 cores, 2x16-core Interlagos nodes, GEMINI interconnect
  - plus internal Cray XE6 development system
Ping-Pong

Performance of Fortran Coarrays on the Cray XE6
Synchronisation

• All coarray operations require explicit synchronisation calls for correctness
  – typically two!

• Ping-pong benchmark includes synchronisation
  – dominates the transfer time
  – actual transfers take about 1.3 μs per double (see later)

• Measure synchronisation explicitly in benchmark
  – global (sync all): 2.3 μs
  – pt-to-pt (sync images): 3.6 μs

• Not completely consistent, but implies latency worse than MPI

• However…
  – MPI synchronises per message pair
  – coarray synchronisation is between images

• Potential to issue many coarray ops between single sync pair
Multi Ping-Pong

May 2012

Performance of Fortran Coarrays on the Cray XE6

![Graph showing bandwidth (GB/s) vs message size (doubles)]
Global Synchronisation

![Graph showing the performance of Fortran Coarrays on the Cray XE6. The graph plots the time (in microseconds) against the number of images. Two lines are shown: one for MPI Barrier and one for Sync all. The time increases as the number of images increases.](image-url)
## Strided Ping-Pong

### Performance of Fortran Coarrays on the Cray XE6

<table>
<thead>
<tr>
<th>Mode</th>
<th>time per double (μs)</th>
<th>total bandwidth (MB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Single ping-pong</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Put (DO loop)</td>
<td>1.3</td>
<td>6.1</td>
</tr>
<tr>
<td>Put (array syntax)</td>
<td>0.3</td>
<td>25.3</td>
</tr>
<tr>
<td>Get (DO loop)</td>
<td>1.2</td>
<td>6.5</td>
</tr>
<tr>
<td>Get (array syntax)</td>
<td>1.2</td>
<td>6.6</td>
</tr>
<tr>
<td><strong>Multi ping-pong</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Put (DO loop)</td>
<td>0.06</td>
<td>129</td>
</tr>
<tr>
<td>Put (array syntax)</td>
<td>0.02</td>
<td>364</td>
</tr>
<tr>
<td>Get (DO loop)</td>
<td>0.06</td>
<td>142</td>
</tr>
<tr>
<td>Get (array syntax)</td>
<td>0.06</td>
<td>141</td>
</tr>
</tbody>
</table>
Point-to-point synchronisation patterns

- Should be advantageous when running on many images
  - images must mutually synchronise
    
    ```fortran
c sync images(imagelist)
    ```
  - user must ensure they all match up

- LX
  - each image pairs with \( X \) images in a (periodic) line
  - e.g. L4 pairs with image-2, image-1, image+1 and image+2

- RX
  - same as LX but images are randomly permuted
  - ensures many off-node messages

- 3D
  - six neighbours in a 3D cartesian grid
Point-to-point Synchronisation Performance

May 2012

Performance of Fortran Coarrays on the Cray XE6
3D halo swap

- Fixed volume, weak scaling

- 3D domain decomposition as suggested by MPI_Cart_dims
  - reversed for Fortran ordering

- Measure time for complete halo swap operation
  - remote read (get) and write (put)
  - sync images and sync all
3D Halo Swaps (10x10x10)

![Graph showing bandwidth (MB/s) vs. number of images for different operations on the Cray XE6.](image_url)
3D Halo Swaps (100x100x100)

Performance of Fortran Coarrays on the Cray XE6
Conclusions

• Simple benchmarks give good insight into performance characteristics of Fortran coarrays

• Cray compiler quite mature
  – good ability to pattern match (i.e. “vectorise” remote operations)
  – performance generally good on XE6
  – significant advantages from point-to-point synchronisation
  – but explicit buffering would still be faster than strided access

• Essential to have verification tests
  – benchmark has a “do verify” option which impacts performance
  – previous Cray compilers have sometimes had bugs
  – e.g. wrong data transferred in 3D halo swaps
  – earliest Intel compiler implemented sync images as a no-op!
  – Cray bugs fixed quickly but you do need to detect them …
Further work

• Increase scope of benchmark
  – overlap of communications and calculations (via compiler directive)
  – more complicated data transfer patterns (e.g. for locks)
  – include proposed coarray collectives

• Tidy up and make available for general release
Acknowledgements

• People
  – Harvey Richardson (Cray Inc)

• Funding
  – Edinburgh University Exascale Technology Centre
  – CRESTA project: www.cresta-project.eu
  – “This work has been supported by the CRESTA project that has received funding from the European Community’s Seventh Framework Programme (ICT-2011.9.13) under Grant Agreement no. 287703”

• Facilities
  – HECToR, the UK’s national high-performance computing service, which is provided by UoE HPCx Ltd at the University of Edinburgh, Cray Inc and NAG Ltd, and funded by the Office of Science and Technology through EPSRC’s High End Computing Programme
  – Cray Inc
Backup: Lock and Critical

![Graph showing the performance of Fortran Coarrays on the Cray XE6](image_url)

- Time (microseconds) vs. Number of Images
- Lines represent Lock and Critical operations
- Performance metrics for different numbers of images

May 2012

Performance of Fortran Coarrays on the Cray XE6