Bringing a scientific application to the distributed world using PGAS

Performance, Portability and Usability of Fortran Coarrays

Jeffrey Salmond
August 15, 2017

Research Software Engineering
University of Cambridge
Summary

We aim to investigate the

• performance,
• portability and
• usability

of Fortran 2008 coarrays for porting large, complex scientific applications to distributed memory.

Outline

• Coarrays & implementations of coarrays
• Synthetic benchmarks
• Porting a scientific code: TROVE
What are Coarrays

Coarrays are:

- a PGAS extension of Fortran,
- Fortran 2008 adds remote access to variables,
- Fortran 2015 adds collectives, atomics and teams.

```fortran
real :: x(10)[*]

x(:) = x(:)[1]
!call mpi_get(x, 10, MPI_REAL, 1, disp, 10, MPI_REAL, mywin, ierr)

sync all
!mpi_barrier(MPI_COMM_WORLD, ierr)

call co_sum(x, result_image=1)
!mpi_reduce(x, x, 10, MPI_REAL, MPI_SUM, 0, MPI_COMM_WORLD, ierr)
```
### Coarray implementations

<table>
<thead>
<tr>
<th></th>
<th>F2008</th>
<th>F2015</th>
<th>OpenMP</th>
<th>MPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>gfortran + OpenCoarrays</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Intel Parallel Studio</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>OpenUH + GASNet</td>
<td>✓</td>
<td>?</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Cray</td>
<td>✓</td>
<td>✓</td>
<td>?</td>
<td>✓</td>
</tr>
</tbody>
</table>
Coarray implementations: gfortran + OpenCoarrays

- gfortran frontend generates calls to libcaf
  
  \_gfortran\_caf\_get (...)

- OpenCoarrays supplies libcaf
- OpenCoarrays libcaf calls into a standard MPI library

✓ gfortran + mpi are very widely supported
  ⇒ can run (almost) anywhere

✓ gfortran can compile most things

× MPI not the ideal target for implementing coarray support
× gasnet backend exists but is ‘unsupported’ (and doesn’t compile)

- Intel implementation uses a similar structure
Coarray implementations: OpenUH + GASNet

- OpenUH compiler frontend
- communication backends provided by GASNet

✓ potentially higher performance
× OpenUH as compliant with complex (nasty) science code
× OpenUH not simple to deploy
Coarray implementations: using MVAPICH2-X

Use MVPAICH2 MPI

1. gfortran
2. Open-Coarrays
3. MVAPICH2

Use MVPAICH2-X GASNet conduit

1. OpenUH
2. MVAPICH2-X
Synthetic Benchmarks

• EPCC Fortran Coarray micro-benchmark suite
• OSU microbenchmarks.

All measurements performed on with

• Intel Broadwell (E5-2650)
• Mellanox EDR

**OpenCoarrays**

gfortran 7.1.0 + OpenCoarrays 1.8.10 + MVAPICH2 2.2

**Intel**

Intel Parallel Studio (Intel compiler + Intel MPI) 17.4

**MVPAICH2-X**

OpenUH 3.1.0 + MVPIACH2-X 2.2

**MPI put/get** (using MPI-3 put and get)

gfortran 7.1.0 + MVAPICH2-X 2.2

**MPI (p2p)** (using MPI send to fake puts and gets)

gfortran 7.1.0 + MVAPICH2-X 2.2
Synthetic Benchmarks: Latency

![Graph showing latency for put and get operations across different methods.](image_url)

- **Latency [microseconds]**

- **OpenCoarrays**
- **Intel**
- **MVAPICH2-X**
- **MPI**
- **MPI (pt2pt)**
Synthetic Benchmarks: Bandwidth

[Graph showing bandwidth in GB/s for put and get operations for different technologies: OpenCoarrays, Intel, MVAPICH2-X, MPI, and MPI (pt2pt).]
• part of the ExoMol (exoplanet molecular line search) project
• looking at the composition of atmospheres on exoplanets
• ultimately searching for aliens!

TROVE has been developed by Sergey Yurchenko (currently at UCL)

• under active development with many contributors
• written in modern Fortran
• developed targeting shared memory :-(
• ~ 160k lines of code
TROVE: Why distributed memory

- TROVE has long run-times (>1M CPU Hours)
- Construction & diagonalisation of 1000s of matrices
- Matrices of size 1M x 1M
- Scientists want to compute bigger problems

Why Coarrays?

- Translating a large code base to distributed memory is daunting
- Maintainers prefer not to use MPI
- PGAS approach allows an ‘incremental’ approach
TROVE: code before

\[
g = 0
\]

\[
!\text{omp parallel do private(h,phi,D) reduce(+::g)}
\]

\[
do \text{ iterm} = 1, N
\]

\[
\text{phi} = !\text{construct Hamiltonian}
\]

\[
D = !\text{construct basis set}
\]

\[
h = \text{matmul(transpose(D), matmul(phi, D))}
\]

\[
g = g + h
\]

\[
\text{enddo}
\]

\[
!\text{omp end parallel do}
\]

\[
call \text{diagonalize}(g)
\]

- Each loop iteration takes >10 seconds
  \[\Rightarrow\] almost embarrassingly parallel
g = 0
do ierm = 1, N
    if (mod(imerm, num_images()) /= this_image()) cycle
    phi = !construct Hamiltonian
    D = !construct basis set
    h = matmul(transpose(D), matmul(phi, D))
    g = g + h
endo

call co_sum(g)
call diagonalize(g)

• very few changes required from OpenMP
• scheduling is basic
TROVE: Results

Coarrays implementation works!

Before
Ideal scaling on 4 socket node

After
Ideal scaling on 4 nodes with 1 socket each

Problems

• TROVE not compatible with OpenUH
  ⇒ can’t use MVAPICH2-X :-(

Future Work

• Extracting more parallelism
• MPI interoperability
Wishlist

Building a coarray implementation with gfortran and an MVAPICH2-X based libcaf.

- gfortran frontend generates calls to libcaf
- a new implementation of libcaf
- this new libcaf calls MVAPICH2-X

- combines the portable and friendly gfortran
- with the high-performance of MVAPICH2-X
Thank you!