Portable, MPI-Interoperable Coarray Fortran

Chaoran Yang,¹ Wesley Bland,²
John Mellor-Crummey,¹ Pavan Balaji²

¹Department of Computer Science
Rice University
Houston, TX

²Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, IL
Partitioned Global Address Space Languages

• Why PGAS Languages today?
  • need for shared data outstrips node-level memory
  • access data with shared-memory abstractions within and across nodes

• Example PGAS Languages
  • Unified Parallel C (C)
    • http://upc.wikinet.org
  • Titanium (Java)
    • http://titanium.cs.berkeley.edu
  • Coarray Fortran (Fortran)
    • http://caf.rice.edu

• Related efforts:
  • X10 (IBM)
    • http://x10-lang.org
  • Chapel (Cray)
    • http://chapel.cray.com
  • Fortress (Oracle)
    • http://projectfortress.java.net
Partitioned Global Address Space Languages

vs. MPI

• Why PGAS Languages today?
  • need for shared data outstrips node-level memory
  • access data with shared-memory abstractions within and across nodes

• Example PGAS Languages
  • Unified Parallel C (C)
    • http://upc.wikinet.org
  • Titanium (Java)
    • http://titanium.cs.berkeley.edu
  • Coarray Fortran (Fortran)
    • http://caf.rice.edu

• Related efforts:
  • X10 (IBM)
    • http://x10-lang.org
  • Chapel (Cray)
    • http://chapel.cray.com
  • Fortress (Oracle)
    • http://projectfortress.java.net
MPI-interoperability

• Hard to adopt new programming models in existing applications **incrementally**

• Interoperable problems in new programming models (examples later)
  • Error-prone
  • Duplicate runtime resources

• Benefits of interoperable programming models
  • Leverage high-level libraries that are built with MPI
  • Hybrid programming models combine the strength of different models
Using multiple runtimes is error-prone

PROGRAM MAY_DEADLOCK

USE MPI

CALL MPI_INIT(IERR)

CALL MPI_COMM_RANK(MPI_COMM_WORLD, MY_RANK, IERR)

IF (MYRANK .EQ. 0) A(:)[1] = A(:)

CALL MPI_BARRIER(MPI_COMM_WORLD, IERR)

CALL MPI_FINALIZE(IERR)

END PROGRAM
Using multiple runtimes is error-prone

```fortran
PROGRAM MAY_DEADLOCK
  USE MPI
  CALL MPI_INIT(IERR)
  CALL MPI_COMM_RANK(MPI_COMM_WORLD, MY_RANK, IERR)
  IF (MY_RANK .EQ. 0) A(:)[1] = A(:)
  CALL MPI_BARRIER(MPI_COMM_WORLD, IERR)
  CALL MPI_FINALIZE(IERR)
END PROGRAM
```

diagram:
- P0
- P1
- blocking PUT
Using multiple runtimes is error-prone

```fortran
PROGRAM MAY_DEADLOCK
    USE MPI
    CALL MPI_INIT(IERR)
    CALL MPI_COMM_RANK(MPI_COMM_WORLD, MY_RANK, IERR)
    IF (MYRANK .EQ. 0) A(:)[1] = A(:)
    CALL MPI BARRIER(MPI_COMM_WORLD, IERR)
    CALL MPI_FINALIZE(IERR)
END PROGRAM
```

Diagram:
- P0: blocking PUT
- P1: MPI_BARRIER
Using multiple runtimes is error-prone

```
PROGRAM MAY_DEADLOCK
  USE MPI
  CALL MPI_INIT(IERR)
  CALL MPI_COMM_RANK(MPI_COMM_WORLD, MY_RANK, IERR)
  IF (MYRANK .EQ. 0) A(:)[1] = A(:)
  CALL MPI_BARRIER(MPI_COMM_WORLD, IERR)
  CALL MPI_FINALIZE(IERR)
END PROGRAM
```
Using multiple runtimes duplicates resources

- Memory usage is measured right after initialization
- Memory usage per process increases as the number of processes increases
- At larger scale, excessive memory use of duplicate runtimes will hurt scalability
How do we solve the problem?

Build PGAS runtime systems with MPI

- Previously MPI was considered insufficient for this goal
- MPI-2 RMA is portable but too strict
- MPI-3 Remote Memory Access (RMA)
Build PGAS runtimes with MPI

- Does it provide full interoperability?
- Does it degrade performance?
Coarray Fortran (CAF)

- What is Coarray Fortran?
  - added to the Fortran 2008 Standard
  - a PGAS Language, SPMD Model
- What is a coarray?
  - extends array syntax with **codimensions**, e.g. `REAL :: X(10,10)[*]`
- How to access a coarray?
  - Reference with [] mean data on specified image, e.g. `X(1,:) = X(1,:)[p]`
  - May be allocatable, structure components, dummy or actual arguments
Coarray Fortran 2.0 (CAF 2.0)

“A rich extension to Coarray Fortran developed at Rice University”

• Teams (like MPI communicator) and collectives
• Asynchronous operations
  • asynchronous copy, asynchronous collectives, and function shipping
• Synchronization constructs
  • events, cofence, and finish

More details on CAF 2.0: http://caf.rice.edu and http://chaoran.me
Coarray and MPI-3 RMA

“standard CAF features”

- Initialization
  - \texttt{MPI\_WIN\_ALLOCATE}, then \texttt{MPI\_WIN\_LOCK\_ALL}
- Remote Read & Write
  - \texttt{MPI\_RPUT} & \texttt{MPI\_RGET}
- Synchronization
  - \texttt{MPI\_WIN\_SYNC} & \texttt{MPI\_WIN\_FLUSH} (_ALL)

\textbf{Blue} routine names are MPI-3 additions
Active Messages

“High performance low-level asynchronous remote procedure calls”

- Many CAF 2.0 features are built on top of AM
- Build AM on top of MPI’s send and receive routines
  - hurt performance - cannot overlap communication with AM handlers
  - hurt interoperability - could cause deadlock
Active Messages

“High performance low-level asynchronous remote procedure calls”

- Many CAF 2.0 features are built on top of AM
- Build AM on top of MPI’s send and receive routines
  - hurt performance - cannot overlap communication with AM handlers
  - hurt interoperability - could cause deadlock
Active Messages

“High performance low-level asynchronous remote procedure calls”

- Many CAF 2.0 features are built on top of AM
- Build AM on top of MPI’s send and receive routines
  - hurt performance - cannot overlap communication with AM handlers
  - hurt interoperability - could cause deadlock
CAF 2.0 Asynchronous Operations

- `copy_async(dest, src, dest_ev, src_ev, pred_ev)`
CAF 2.0 Asynchronous Operations

- `copy_async(dest, src, dest_ev, src_ev, pred_ev)`
CAF 2.0 Asynchronous Operations

- `copy_async(dest, src, dest_ev, src_ev, pred_ev)`
CAF 2.0 Asynchronous Operations

- `copy_async(dest, src, dest_ev, src_ev, pred_ev)`
CAF 2.0 Asynchronous Operations

- `copy_async(dest, src, dest_ev, src_ev, pred_ev)`
CAF 2.0 Asynchronous Operations

- copy_async(dest, src, dest_ev, src_ev, pred_ev)
CAF 2.0 Asynchronous Operations

- \texttt{copy\_async(dest, src, dest\_ev, src\_ev, pred\_ev)}
CAF 2.0 Asynchronous Operations

• `copy_async(dest, src, dest_ev, src_ev, pred_ev)`
CAF 2.0 Asynchronous Operations

- copy_async(dest, src, dest_ev, src_ev, pred_ev)

- Map copy_async to MPI_RPUT (or MPI_RGET)
  - when dest_ev should be notified? MPI_WIN_FLUSH is not useful

- Map copy_async to Active Message
  - MPI does not have AM support
Evaluation

- 2 machines
  - Cluster (InfiniBand) and Cray XC30
- 3 benchmarks and 1 mini-app
  - RandomAccess, FFT, HPL, and CGPOP
- 2 implementations
  - CAF-MPI and CAF-GASNet

<table>
<thead>
<tr>
<th>System</th>
<th>Nodes</th>
<th>Cores / Node</th>
<th>Memory / Node</th>
<th>Interconnect</th>
<th>MPI Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster (Fusion)</td>
<td>320</td>
<td>2x4</td>
<td>32GB</td>
<td>InfiniBand QDR</td>
<td>MVAPICH2-1.9</td>
</tr>
<tr>
<td>Cray XC30 (Edison)</td>
<td>5,200</td>
<td>2x12</td>
<td>64GB</td>
<td>Cray Aries</td>
<td>CRAY MPI-6.0.2</td>
</tr>
</tbody>
</table>
RandomAccess

“Measures worst case system throughput”

RandomAccess on Edison (Cray XC30)

RandomAccess on Fusion (InfiniBand)
Performance Analysis of RandomAccess

- The time spent in communication are about the same
- `event_notify` is slower in CAF-MPI because of `MPI_WIN_FLUSH_ALL`
FFT

FFT on Edison (Cray XC30)

FFT on Fusion (InfiniBand)

Number of processes

10000
1000
100
10
1

GFlops

16 32 64 128 256 512 1024 2048 4096

16

8 16 32 64 128 256 512 1024 2048

GFlops

CAF-MPI
CAF-GASNet
IDEAL-SCALING
Performance Analysis of FFT

- The CAF 2.0 version of FFT solely uses ALLtoALL for communication.
- CAF-MPI performs better because of fast all-to-all implementation.

**Time breakdown of FFT**

<table>
<thead>
<tr>
<th></th>
<th>CAF-GASNet</th>
<th>CAF-MPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computation</td>
<td>7.94</td>
<td>8.31</td>
</tr>
<tr>
<td>All-to-all</td>
<td>17.92</td>
<td>6.06</td>
</tr>
</tbody>
</table>

![Bar chart showing time breakdown of FFT](chart.png)
High Performance Linpack

“computation intensive”

HPL on Edison (Cray XC30)

HPL on Fusion (InfiniBand)

Number of processes

TFlops

CAF-MPI  CAF-GASNet  IDEAL-SCALING

Number of processes

TFlops

CAF-MPI  CAF-GASNet  IDEAL-SCALING
CGPOP

“A CAF+MPI hybrid application”

- The conjugate gradient solver from LANL Parallel Ocean Program 2.0
- performance bottleneck of the full POP 2.0 application
- Performs linear algebra computation interspersed with two comm. steps:
  - **GlobalSum**: a 3-word vector sum (MPI_Reduce)
  - **UpdateHalo**: boundary exchange between neighboring subdomains (CAF)

Andrew I Stone, John M. Dennis, Michelle Mills Strout, “Evaluating Coarray Fortran with the CGPOP Miniapp”
CGPOP

CGPOP on Edison (Cray XC30)

Execution time (in seconds)

Number of processes

CAF-MPI
CAF-GASNet

CGPOP on Fusion (InfiniBand)

Execution time (in seconds)

Number of processes

CAF-MPI
CAF-GASNet
Conclusions

use MPI to build PGAS runtimes, good or bad?

• The benefits of building runtime systems on top of MPI
  • Interoperability with numerous MPI based libraries (Standard CAF)
  • Deliver performance comparable to runtimes built with GASNet
  • MPI’s rich interface is time-saving

• What current MPI RMA lacks
  • **MPI_WIN_RFLUSH** - overlap synchronization with computation
  • **Active Messages** - full interoperability