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
Parallel Programming Models

- **Shared memory**
  - Pthread, OpenMP, …
  - “easy” to program
  - hard to scale

- **Message passing**
  - MPI, …
  - “scalable”
  - hard to program
Partitioned Global Address Space Languages

- Combine the strength
  - shared memory models
    - "Easy to program"
  - message passing models
    - "Scalable"

- Example PGAS Languages
  - Unified Parallel C (C)
  - Titanium (Java)
  - Coarray Fortran (Fortran)

- Related efforts
  - X10 (IBM)
  - Chapel (Cray)
  - Fortress (Oracle)
Current status

• The dominance of Message Passing Interface (MPI)
  • Most applications on clusters are written using MPI
  • Many high-level libraries on clusters are built with MPI

• Why people do not adopt PGAS languages?
  • Most PGAS languages are built with a different runtime system e.g. GASNet
  • Hard to adopt new programming models in existing applications incrementally
Problem 1: deadlock

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

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

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

PROGRAM MAY_DEADLOCK

USE MPI

CALL MPI_INIT(E)

CALL MPI_COMM_RANK(MPI_COMM_WORLD, MY_RANK, E)

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

CALL MPI_BARRIER(MPI_COMM_WORLD, IERR)

CALL MPI_FINALIZE(E)

END PROGRAM
Problem 2: duplicates resources

- Memory usage per process increases as the number of processes increases
- At larger scale, excessive memory use of duplicate runtimes will hurt scalability
The problem

- PGAS languages DO NOT play well with MPI
  - program may deadlock using MPI and other runtimes
  - program unnecessarily uses duplicated resources
The problem

- PGAS languages DO NOT play well with MPI
  - program may deadlock using MPI and other runtimes
  - program unnecessarily uses duplicated resources

The solution

Build PGAS language runtimes with MPI
Why people haven’t done it?

- Previously MPI was considered insufficient for this goal*
- MPI-3 adds extensive support for Remote Memory Access (RMA)

---

Question to investigate

- Build PGAS runtimes with MPI-3
  - Does it provide full interoperability?
  - Does it degrade performance?
Question to investigate

- Build PGAS runtimes with MPI-3
  - Does it provide full interoperability?
  - Does it degrade performance?

Approach

- Build a PGAS language (Coarray Fortran) with MPI-3 and evaluate its interoperability and performance
Coarray in Fortran 2008 (CAF)

- Fortran 2008 Standard contains features for parallel programming using a SPMD (Single Program Multiple Data) 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

Features in **Blue** has been adopted by Fortran standard committee
Coarray and MPI-3 RMA

Mapping coarray features to MPI-3 APIs

• Initialization
  • INTEGER :: A(100,100)[*]
  • MPI_WIN_ALLOCATE, then MPI_WIN_LOCK_ALL

• Coarray Read & Write
  • A(:)[1] = A(:); A(:) = A(:)[2]
  • MPI_RPUT; MPI_RGET

• Synchronization
  • MPI_WIN_SYNC (local) & MPI_WIN_FLUSH (_ALL) (global)
Active Messages

“Lightweight, low-level, asynchronous, remote procedure calls”

- Many CAF 2.0 features are built on top of Active Messages
- MPI does not provide an implementation of Active Messages
- Emulate Active Messages with `MPI_Send` and `MPI_Recv` routines
  - hurt performance - cannot overlap communication with AM handlers
  - hurt interoperability - could cause deadlock
Active Messages

“Lightweight, low-level, asynchronous, remote procedure calls”

- Many CAF 2.0 features are built on top of Active Messages
- MPI does not provide an implementation of Active Messages
- Emulate Active Messages with `MPI_Send` and `MPI_Recv` routines
  - hurt performance - cannot overlap communication with AM handlers
  - hurt interoperability - could cause deadlock
Active Messages

“Lightweight, low-level, asynchronous, remote procedure calls”

- Many CAF 2.0 features are built on top of Active Messages
- MPI does not provide an implementation of Active Messages
- Emulate Active Messages with `MPI_Send` and `MPI_Recv` routines
  - hurt performance - cannot overlap communication with AM handlers
  - hurt interoperability - could cause deadlock

`MPI_Reduce` DEADLOCK!
CAF 2.0 Events

“similar to counting semaphores”

Maps to Active Messages

\[ \text{CALL event\_notify(event, n)} \]

\[ \begin{align*}
\bullet & \text{Need to ensure all previous asynchronous operations have completed before the notification} \\
& \text{for each window} \\
& \quad \text{MPI\_Win\_sync(win)} \\
& \text{for each dirty window} \\
& \quad \text{MPI\_Win\_flush\_all(win)} \\
& \text{AM\_Request(…)} // \text{MPI\_Isend}
\end{align*} \]

\[ \text{CALL event\_wait(event, n)} \]

\[ \begin{align*}
\bullet & \text{Also serves as a compiler barrier (prevent compiler from reorder operations upward)} \\
& \text{while (count < n)} \\
& \quad \text{for each window} \\
& \quad \quad \text{MPI\_Win\_sync(win)} \\
& \quad \quad \text{AM\_Poll(…)} // \text{MPI\_Iprobe}
\end{align*} \]
CAF 2.0 Asynchronous Operations

- `copy_async(dest, src, dest_event, src_event, pred_event)`
CAF 2.0 Asynchronous Operations

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

- copy_async(dest, src, dest_event, src_event, pred_event)
CAF 2.0 Asynchronous Operations

- `copy_async(dest, src, dest_event, src_event, pred_event)`
CAF 2.0 Asynchronous Operations

- copy_async(dest, src, dest_event, src_event, pred_event)
CAF 2.0 Asynchronous Operations

- `copy_async(dest, src, dest_event, src_event, pred_event)`
CAF 2.0 Asynchronous Operations

- `copy_async(dest, src, dest_event, src_event, pred_event)`
CAF 2.0 Asynchronous Operations

- `copy_async(dest, src, dest_event, src_event, pred_event)`
CAF 2.0 Asynchronous Operations

- \text{copy\_async}(\text{dest}, \text{src}, \text{dest\_event}, \text{src\_event}, \text{pred\_event})

- Map \text{copy\_async} to \textbf{Active Message}
CAF 2.0 Asynchronous Operations

- `copy_async(dest, src, dest_event, src_event, pred_event)`

  - When posted, send AM request

- Map `copy_async` to **Active Message**
CAF 2.0 Asynchronous Operations

- `copy_async(dest, src, dest_event, src_event, pred_event)`

  * When posted, send AM request
  * After AM request sent

- Map `copy_async` to **Active Message**
CAF 2.0 Asynchronous Operations

- **copy_async**(dest, src, dest_event, src_event, pred_event)

  - When posted, send AM request
  - After AM request sent
  - After AM handler complete

- **Map** copy_async **to** Active Message
CAF 2.0 Asynchronous Operations

- `copy_async(dest, src, dest_event, src_event, pred_event)`

  - When posted, send AM request
  - After AM request sent
  - After AM handler complete

- Map `copy_async` to Active Message
  - MPI does not have AM support
CAF 2.0 Asynchronous Operations

- `copy_async(dest, src, dest_event, src_event, pred_event)`

- Map `copy_async` to **Active Message**
  - MPI does not have AM support
  - Map `copy_async` to `MPI_RPUT` (or `MPI_RGET`)
CAF 2.0 Asynchronous Operations

- copy_async(dest, src, dest_event, src_event, pred_event)

- Map copy_async to Active Message
  - MPI does not have AM support
- Map copy_async to MPI_RPUT (or MPI_RGET)
CAF 2.0 Asynchronous Operations

- `copy_async(dest, src, dest_event, src_event, pred_event)`

When posted, send `MPI_Rput`

- Map `copy_async` to **Active Message**
  - MPI does not have AM support
- **Map `copy_async` to `MPI_RPUT` (or `MPI_RGET`)**
CAF 2.0 Asynchronous Operations

- `copy_async(dest, src, dest_event, src_event, pred_event)`

  - When posted, send `MPI_Rput`
  - After `MPI_Rput` returns

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

- Map `copy_async` to `MPI_RPUT` (or `MPI_RGET`)
CAF 2.0 Asynchronous Operations

- `copy_async(dest, src, dest_event, src_event, pred_event)`

- Map `copy_async` to **Active Message**
  - MPI does not have AM support
  - Map `copy_async` to `MPI_RPUT` (or `MPI_RGET`)
CAF 2.0 Asynchronous Operations

- `copy_async(dest, src, dest_event, src_event, pred_event)`

  - Map `copy_async` to Active Message
    - MPI does not have AM support
    - Map `copy_async` to `MPI_RPUT` (or `MPI_RGET`)
      - No asynchronous synchronization operation in MPI

  - When posted, send `MPI_Rput`
  - After `MPI_Rput` returns
  - After `MPI_WIN_FLUSH`?
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`.
- `MPI_WIN_FLUSH_ALL` performs `MPI_WIN_FLUSH` one by one.
FFT

“Measures all-to-all communication”

FFT on Edison (Cray XC30)

FFT on Fusion (InfiniBand)
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>

![Graph showing time breakdown of FFT]
High Performance Linpack

“computation intensive”

HPL on Edison (Cray XC30)

HPL on Fusion (InfiniBand)
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)
CGPOP

CGPOP on Edison (Cray XC30)

Execution time (in seconds)

Number of processes

CGPOP on Fusion (InfiniBand)

Execution time (in seconds)

Number of processes

CAF-MPI
CAF-GASNet
Conclusions

- The benefits of building runtime systems on top of MPI
  - Interoperable with numerous MPI based libraries (Fortran 2008)
  - 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