OPTIMIZATION STRATEGIES FOR MPI-INTEROPERABLE ACTIVE MESSAGES

Xin Zhao, Pavan Balaji, William Gropp, Rajeev Thakur

University of Illinois at Urbana-Champaign
Argonne National Laboratory

ScalCom’13

December 21, 2013
Data-Intensive Applications

• Examples: graph algorithm (BFS), sequence assembly

• Common characteristics
  • Organized around sparse structures
  • Communication-to-computation ratio is high
  • Irregular communication pattern
  • Remote complex computation
Message Passing Models

- MPI: industry standard communication runtime for high performance computing

  - Process 0
    - Send (data)
    - Receive (data)

  - Process 1
    - Receive (data)
    - Send (data)

  two-sided communication (explicit sends and receives)

  - Process 0
    - Put (data)
    - Get (data)
    - Acc (data)

  one-sided (RMA) communication (explicit sends, implicit receives, simple remote operations)

- Active Messages
  - Sender explicitly sends message
  - Upon message’s arrival, message handler is triggered, receiver is not explicitly involved
  - User-defined operations on remote process
Past Work: MPI-Interoperable Generalized AM

**MPI-AM**: an MPI-interoperable framework that can dynamically manage data movement and user-defined remote computation.

- **Streaming AMs**
  - define “segment”—minimum number of elements for AM execution
  - achieve pipeline effect and reduce buffer requirement

- **Explicit and implicit buffer management**
  - **user buffers**: rendezvous protocol, guarantee correct execution
  - **system buffers**: eager protocol, not always enough

- **Correctness semantics**
  - **Memory consistency**
    - MPI runtime must ensure consistency of window

- **Three different type of ordering**
  - **Concurrency**: by default, MPI runtime behaves “as if” AMs are executed in sequential order. User can release concurrency by setting MPI assert.

How MPI-Interoperable AMs work

- Leveraging MPI RMA interface

<table>
<thead>
<tr>
<th>Passive Target Mode</th>
<th>Active Target Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>(EXCLUSIVE or SHARED lock)</td>
<td>(Fence or Post-Start-Complete-Wait)</td>
</tr>
</tbody>
</table>
Performance Shortcomings with MPI-AM

- Synchronization stalls in data
- Inefficiency in data transmission buffering

Effective strategies are needed to improve performance
- System level optimization
- User level optimization

Diagram:
- ORIGIN
  - segment 1
  - request for user buffer
  - stall
  - AM
  - segment 2
  - return output data in system buffer
  - segment 3
- TARGET
  - receive in system buffer
  - reserve user buffer
  - receive in user buffer
  - receive in system buffer

Consensus DNA sequences:
- ACGCGATTCAG
- GCGATTCAGTA
- ACGCGATTCAGTA

Remote search:
- remote node 1
  - ACGCGATTCAG
  - remote search
- remote node 2
Opt #1: Auto-Detected Exclusive User Buffer

- MPI internally detects EXCLUSIVE passive mode

![Diagram showing exclusive and shared modes with hand-shakes](image)

- Only one hand-shake is required for entire epoch
- Hand-shake is required whenever AM uses user buffer

- Optimization is transparent to user
Opt #2: User-Defined Exclusive User Buffer

- User can define amount of user buffer guaranteed available for certain processes
  - Beneficial for SHARED passive mode and active mode
  - No hand-shake operation is required!
Opt #3: Improving Data Transmission

- Two different models

  - Contiguous output data layout
    - Really straightforward and convenient?
    - Out-of-order AMs require buffering or reordering

  - Non-contiguous output data layout
    - Require new API — not a big deal
    - Must transfer back count array
    - Packing and unpacking
    - No buffering or reordering is needed

- Data packing vs. data transmission
  - Controlled by system-specific threshold
Experimental Settings

- BLUES cluster at ANL: 310 nodes, with each consisting 16 cores, connected with QLogic QDR InfiniBand
- Based on MPICH-3.1b1
- Micro-benchmarks: two common operations
  - Remote search of string sequences (20 characters per sequence)
  - Remote summation of absolute values in two arrays (100 integers per array)
  - Result data is returned
Effect of Exclusive User Buffer

scalability: 25% improvement
Comparison between MPIX_AM and MPIX_AMV

MPIX_AMV(0.8) transmits more data than MPIX_AM due to additional counts array.

System-specific threshold helps eliminate packing and unpacking overhead.
Conclusion

- Data-intensive applications are increasingly important, MPI is not a well-suited model
- We proposed MPI-AM framework in our previous work, to make data-intensive applications more efficient and require less programming effort
- There are performance shortcomings in current MPI-AM framework
- Our optimization strategies, including auto-detected and user-defined methods, can effectively reduce synchronization overhead and improve efficiency of data transmission
Our previous work on MPI-AM:


They can be found at [http://web.engr.illinois.edu/~xinzhou3/](http://web.engr.illinois.edu/~xinzhou3/)

More about MPI-3 RMA interface can be found in MPI-3 standard ([http://www.mpi-forum.org/docs/](http://www.mpi-forum.org/docs/))

Thanks for your attention! 😊
Data-Intensive Applications

• “Traditional” applications
  • Organized around dense vectors or matrices
  • Regular communication, use MPI SEND/RECV or collectives
  • Communication-to-computation ratio is low
  • Example: stencil computation, matrix multiplication, FFT

• Data-intensive applications
  • Organized around graphs, sparse vectors
  • Communication pattern is irregular and data-dependent
  • Communication-to-computation ratio is high
  • Example: bioinformatics, social network analysis
## Vector Version of AM API

### MPIX_AMV

<table>
<thead>
<tr>
<th>Type</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>origin_input_addr</td>
<td></td>
</tr>
<tr>
<td>IN</td>
<td>origin_input_segment_count</td>
<td></td>
</tr>
<tr>
<td>IN</td>
<td>origin_input_datatype</td>
<td></td>
</tr>
<tr>
<td>OUT</td>
<td>origin_output_addr</td>
<td></td>
</tr>
<tr>
<td>IN</td>
<td>origin_output_segment_count</td>
<td></td>
</tr>
<tr>
<td>IN</td>
<td>origin_output_datatype</td>
<td></td>
</tr>
<tr>
<td>IN</td>
<td>num_segments</td>
<td></td>
</tr>
<tr>
<td>IN</td>
<td>target_rank</td>
<td></td>
</tr>
<tr>
<td>IN</td>
<td>target_input_datatype</td>
<td></td>
</tr>
<tr>
<td>IN</td>
<td>target_persistent_disp</td>
<td></td>
</tr>
<tr>
<td>IN</td>
<td>target_persistent_count</td>
<td></td>
</tr>
<tr>
<td>IN</td>
<td>target_persistent_datatype</td>
<td></td>
</tr>
<tr>
<td>IN</td>
<td>target_output_datatype</td>
<td></td>
</tr>
<tr>
<td>IN</td>
<td>am_op</td>
<td></td>
</tr>
<tr>
<td>IN</td>
<td>win</td>
<td></td>
</tr>
<tr>
<td>OUT</td>
<td>output_segment_counts</td>
<td></td>
</tr>
</tbody>
</table>

### MPIX_AMV_USER_FUNCTION

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>input_addr</td>
</tr>
<tr>
<td>IN</td>
<td>input_segment_count</td>
</tr>
<tr>
<td>IN</td>
<td>input_datatype</td>
</tr>
<tr>
<td>INOUT</td>
<td>persistent_addr</td>
</tr>
<tr>
<td>INOUT</td>
<td>persistent_count</td>
</tr>
<tr>
<td>INOUT</td>
<td>persistent_datatype</td>
</tr>
<tr>
<td>OUT</td>
<td>output_addr</td>
</tr>
<tr>
<td>OUT</td>
<td>output_segment_count</td>
</tr>
<tr>
<td>OUT</td>
<td>output_datatype</td>
</tr>
<tr>
<td>IN</td>
<td>num_segments</td>
</tr>
<tr>
<td>IN</td>
<td>segment_offset</td>
</tr>
<tr>
<td>OUT</td>
<td>output_segment_counts</td>
</tr>
</tbody>
</table>
Effect of Exclusive User Buffer

scalability: 25% improvement

providing more exclusive buffer greatly reduces contention
Auto-detected Exclusive User Buffer

- Handle detaching user buffers
  - One more hand-shake is required after MPI_WIN_FLUSH, because user buffer may be detached on target
  - User can pass a hint to tell MPI that there will be no buffer detachment, in such case, MPI will eliminate hand-shake after MPI_WIN_FLUSH
Background —MPI One-sided Synchronization Modes

- Two synchronization modes

Active target mode (post-start-complete-wait/fence)

Passive target mode (SHARED or EXCLUSIVE lock/lock_all)