How I Learned to Stop Worrying About User-Visible Endpoints and Love MPI

Rohit Zambre,*
Aparna Chandramowlishwaran,*
Pavan Balaji^

*University of California, Irvine
^Argonne National Laboratory
MPI everywhere

Node  Core  Process
MPI everywhere

- **Model artifact**: high memory requirements that worsen with increase domain-dimensionality and number of ranks.

- **Hardware usage**: resource wastage with static split of limited resources on processor.
HOW I LEARNED TO STOP WORRYING ABOUT USER-VISIBLE ENDPOINTS AND LOVE MPI

MPI+threads

- **Model artifact**: reduces duplicated data by a factor of number of threads.

- **Hardware usage**: able to use the many cores while sharing all of processor’s resources.

Node □  Core □  Process □  Thread □

Increasing number of cores → Decreasing memory per core

![42 Years of Microprocessor Trend Data](image)
MPI everywhere

OOM!

Corresponding

MPI+threads

runs

HOW I LEARNED TO STOP WORRYING ABOUT USER-VISIBLE ENDPOINTS AND LOVE MPI

MPI everywhere

OOM!

Corresponding MPI+threads runs


Communication performance of MPI+threads is dismal
Node

*Outdated view:* Network is a single device

*Modern reality:* Network features parallelism

Network Interface Card

Network hardware context
HOW I LEARNED TO STOP WORRYING ABOUT USER-VISIBLE ENDPOINTS AND LOVE MPI

Network hardware context

Software communication channel

Network hardware context

MPI everywhere

MPI+threads
HOW I LEARNED TO STOP WORRYING ABOUT USER-VISIBLE ENDPOINTS AND LOVE MPI

![Diagram](image-url)

**MPI everywhere**
- Application
- MPI library
- Network Interface Card

**MPI+threads**
- Global critical section + 1 communication channel per process

- Green: Software communication channel
- Orange: Network hardware context
HOW I LEARNED TO STOP WORRYING ABOUT USER-VISIBLE ENDPOINTS AND LOVE MPI

**MPI everywhere**

- Application
- MPI library
- Network Interface Card

**MPI+threads**

- Application
- MPI library
- Network Interface Card

No logical parallelism expressed

Global critical section + 1 communication channel per process

- Software communication channel
- Network hardware context
MPI_Comm_create_endpoints(...,num_ep,...,comm_eps[]);

MPI_Isend/Irecv(...,comm_eps[tid],ep_rank,...);
MPI_Comm_create_endpoints(..., num_ep, ..., comm_eps[]);

MPI_Isend/Irecv(..., comm_eps[tid], ep_rank, ...);

**Pros**
- Explicit control over network contexts

**Cons**
- Intrusive extension of the MPI standard
- Onus of managing network contexts on user
HOW I LEARNED TO STOP WORRYING ABOUT USER-VISIBLE ENDPOINTS AND LOVE MPI

MPI everywhere

Application

P0

P1

P2

P3

MPI library

Network Interface Card

MPI+threads

Logical parallelism expressed

P0

C0

C1

C2

C3

Network hardware context

Software communication channel

Network hardware context

Fine-grained critical sections + Multiple communication channel per process

MPI Communicator
Do we need user-visible endpoints?

MPI everywhere

Application

MPI library

Network Interface Card

MPI+threads

Logical parallelism expressed

Fine-grained critical sections +
Multiple communication channel per process

Network hardware context

Software communication channel

Network hardware context

MPI Communicator
CONTRIBUTIONS AS DEVIL’S ADVOCATE

- In-depth comparison between MPI-3.1 and user-visible endpoints
- A fast MPI+threads library that adheres to MPI-3.1’s constraints
  - Optimized parallel communication streams applicable to all MPI libraries
- Recommendations for the MPI user to express logical parallelism with MPI-3.1

<table>
<thead>
<tr>
<th>Evaluation platforms</th>
<th>MPI library</th>
<th>Interconnects</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Based on MPICH:CH4</td>
<td>Intel Omni-Path (OPA) with OFI:PSM2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mellanox InfiniBand (IB) with UCX:Verbs</td>
</tr>
</tbody>
</table>
OUTLINE

- Introduction
- For MPI users: Parallelism in the MPI standard
- For MPI developers: Fast MPI+threads
  - Fine-grained critical sections for thread safety
  - Virtual Communication Interfaces (VCIs) for parallel communication streams
- Microbenchmark and Application analysis
POINT-TO-POINT COMMUNICATION

- `<comm, rank, tag>` decides matching
- Non-overtaking order
- Receive wildcards

<table>
<thead>
<tr>
<th>Two or more operations on a process with</th>
<th>Can be issued on parallel communication streams?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comm</td>
<td>Rank</td>
</tr>
</tbody>
</table>

How I learned to stop worrying about user-visible endpoints and love MPI
POINT-TO-POINT COMMUNICATION

- `<comm,rank,tag>` decides matching
- Non-overtaking order
- Receive wildcards

<table>
<thead>
<tr>
<th>Two or more operations on a process with</th>
<th>Can be issued on parallel communication streams?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comm</td>
<td>Rank</td>
</tr>
<tr>
<td>Different</td>
<td>Different or Same</td>
</tr>
<tr>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rank 0 (sender)</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;CA,R1,T1&gt;</td>
</tr>
<tr>
<td>&lt;CB,R1,T1&gt;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rank 1 (receiver)</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;CA,R0,T1&gt;</td>
</tr>
<tr>
<td>&lt;CB,R0,T1&gt;</td>
</tr>
</tbody>
</table>

HOW I LEARNED TO STOP WORRYING ABOUT USER-VISIBLE ENDPOINTS AND LOVE MPI
POINT-TO-POINT COMMUNICATION

- `<comm, rank, tag>` decides matching
- Non-overtaking order
- Receive wildcards

<table>
<thead>
<tr>
<th>Comm</th>
<th>Rank</th>
<th>Tag</th>
<th>Send</th>
<th>Recv</th>
</tr>
</thead>
<tbody>
<tr>
<td>Different</td>
<td>Different or Same</td>
<td>Same</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Same</td>
<td>Different</td>
<td>Different or Same</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

Can be issued on parallel communication streams?

- Rank 0 (sender): `<CA,R1,T1>`, `<CA,R2,T1>`
- Rank 1 (receiver): `<CA,R0,T1>`, `<CA,ANY,T1>`
**POINT-TO-POINT COMMUNICATION**

- `<comm,rank,tag>` decides matching
- Non-overtaking order
- Receive wildcards

<table>
<thead>
<tr>
<th>Two or more operations on a process with</th>
<th>Can be issued on parallel communication streams?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comm</td>
<td>Rank</td>
</tr>
<tr>
<td>Different</td>
<td>Different or Same</td>
</tr>
<tr>
<td>Same</td>
<td>Different</td>
</tr>
<tr>
<td>Same</td>
<td>Same</td>
</tr>
</tbody>
</table>

Non-overtaking order

Wildcards

Rank 0 (sender)

<CA,R1,T1> <CA,R0,T3>

Rank 1 (receiver)

<CA,R0,ANY>
Two or more operations on a process with

<table>
<thead>
<tr>
<th>Window</th>
<th>Rank</th>
<th>Can be issued on parallel communication streams?</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Put</td>
<td>Get</td>
</tr>
<tr>
<td>Different</td>
<td>Different or Same</td>
<td>Yes</td>
</tr>
<tr>
<td>Same</td>
<td>Different</td>
<td>Yes</td>
</tr>
<tr>
<td>Same</td>
<td>Same</td>
<td>Yes</td>
</tr>
</tbody>
</table>
### RMA Communication

<table>
<thead>
<tr>
<th>Two or more operations on a process with</th>
<th>Can be issued on parallel communication streams?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Window</td>
<td>Put</td>
</tr>
<tr>
<td>Different</td>
<td>Yes</td>
</tr>
<tr>
<td>Same</td>
<td>Yes</td>
</tr>
<tr>
<td>Same</td>
<td>Yes</td>
</tr>
</tbody>
</table>

- **Explicitly expressing parallelism**
  - Window: Different
  - Rank: Different or Same
    - Put: Yes
    - Get: Yes
    - Accumulate: Yes
  - Rank: Same
    - Put: Yes
    - Get: Yes
    - Accumulate: Yes
  - Rank: Same
    - Put: Yes
    - Get: Yes
    - Accumulate: No
- **Implicit parallelism**
  - Window: Same
  - Rank: Different
    - Put: No order between multiple Gets and Puts
    - Get: No order between multiple Gets and Puts
    - Accumulate: No
### RMA Communication

<table>
<thead>
<tr>
<th>Two or more operations on a process with</th>
<th>Can be issued on parallel communication streams?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Window</td>
<td>Rank</td>
</tr>
<tr>
<td>Different</td>
<td>Different or Same</td>
</tr>
<tr>
<td>Same</td>
<td>Different</td>
</tr>
<tr>
<td>Same</td>
<td>Same</td>
</tr>
</tbody>
</table>

- **Explicitly expressing parallelism**
  - No order between multiple Gets and Puts
- **Implicit parallelism**
  - Ordering of accumulate operations to the same memory location
OUTLINE

- Introduction
- For MPI users: Parallelism in the MPI standard
- For MPI developers: Fast MPI+threads
  - Fine-grained critical sections for thread safety
  - Virtual Communication Interfaces (VCIs) for parallel communication streams
- Microbenchmark and Application analysis
DESERIALIZING ACCESS TO THE MPI LIBRARY

- State of the art: global critical section
- Adopt fine-grained critical sections (Balaji et al., Amer et al.)
  - Higher parallelism
  - More lock acquisitions
  - Atomics for counters

Overheads of FG in the single thread case

FG outperforms Global at higher thread count
PARALLEL COMMUNICATION STREAMS

- Virtual Communication Interfaces (VCIs)
  - Independent set of communication resources with FIFO order
  - Each VCI protected by its own lock
  - Maps to a network hardware context

- VCI pool
  - Allocate a VCI to a communicator/window
  - Fallback VCI
Fine-grained critical sections + multiple VCIs alone give practically no benefit
Per-VCI progress

- **Global progress**: progress all VCIs
  - High contention on VCIs’ locks

- **Pure per-VCI progress**: progress only VCI of operation
  - Deadlock when shared progress required

- **Hybrid per-VCI progress**

---

![Graph showing performance with different optimizations](graph.png)

<table>
<thead>
<tr>
<th>Optimizations</th>
<th>Messages/s (x10^6)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original (Global + 1 VCI)</td>
<td>1</td>
</tr>
<tr>
<td>All</td>
<td>10</td>
</tr>
<tr>
<td>All w/o per-VCI progress</td>
<td>30</td>
</tr>
</tbody>
</table>

8B MPI_Isend; MPICH/OFI/OPA
Per-VCI progress

- *Global progress*: progress all VCIs
  - High contention on VCIs’ locks

- *Pure per-VCI progress*: progress only VCI of operation
  - Deadlock when shared progress required

- *Hybrid per-VCI progress*

Per-VCI Request management

- *Request class lock*: high contention
  - *Per-VCI request cache*

- *Global lightweight request*: contended atomics for refcounting
  - *Per-VCI lightweight request*

---

Optimizations

- Original (Global + 1 VCI)
- All
- All w/o per-VCI progress

**8B MPI_Isend; MPICH/OFI/OPA**

---

Optimizations

- Original (Global + 1 VCI)
- All
- All w/o per-VCI req-mgmt

**8B MPI_Isend; MPICH/OFI/OPA**
HOW I LEARNED TO STOP WORRYING ABOUT USER-VISIBLE ENDPOINTS AND LOVE MPI

Per-VCI progress

- **Global progress**: progress all VCIs
  - High contention on VCIs’ locks
- **Pure per-VCI progress**: progress only VCI of operation
  - Deadlock when shared progress required
- **Hybrid per-VCI progress**

8B MPI_Isend; MPICH/OFI/OPA

- **Optimizations**
  - Original (Global + 1 VCI)
  - All
  - All w/o per-VCI progress

Messages/s (x10^6) vs Number of cores

Per-VCI Request management

- **Request class lock**: high contention
  - **Per-VCI request cache**
- **Global lightweight request**: contended atomics for refcounting
  - **Per-VCI lightweight request**

8B MPI_Isend; MPICH/OFI/OPA

- **Optimizations**
  - Original (Global + 1 VCI)
  - All
  - All w/o per-VCI req-mgmt

Messages/s (x10^6) vs Number of cores

Per-VCI cache-line awareness

- **False-sharing**: locks of consecutive VCIs
- **Per-VCI cache alignment**

8B MPI_Isend; MPICH/OFI/OPA

- **Optimizations**
  - Original (Global + 1 VCI)
  - All
  - All w/o cache-aware VCI

Messages/s (x10^6) vs Number of cores
HOW I LEARNED TO STOP WORRYING ABOUT USER-VISIBLE ENDPOINTS AND LOVE MPI

Per-VCI progress

- **Global progress**: progress all VCIs
  - High contention on VCIs’ locks

- **Pure per-VCI progress**: progress only VCI of operation
  - Deadlock when shared progress required

- **Hybrid per-VCI progress**

Per-VCI Request management

- **Request class lock**: high contention
  - **Per-VCI request cache**

- **Global lightweight request**: contended atomics for refcounting
  - **Per-VCI lightweight request**

Per-VCI cache-line awareness

- **False-sharing**: locks of consecutive VCIs

- **Per-VCI cache alignment**

---

<table>
<thead>
<tr>
<th>Optimizations</th>
<th>8B MPI_Isend; MPICH/OFI/OPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original (Global + 1 VCI)</td>
<td><img src="#" alt="Graph 1" /></td>
</tr>
<tr>
<td>All</td>
<td><img src="#" alt="Graph 2" /></td>
</tr>
<tr>
<td>All w/o per-VCI progress</td>
<td><img src="#" alt="Graph 3" /></td>
</tr>
</tbody>
</table>

![Graph 1](#)  
![Graph 2](#)  
![Graph 3](#)
OUTLINE

- Introduction
- For MPI users: Parallelism in the MPI standard
- For MPI developers: Fast MPI+threads
  - Fine-grained critical sections for thread safety
  - Virtual Communication Interfaces (VCIs) for parallel communication streams
- Microbenchmark and Application analysis
APPLICATION CATEGORIES

- Category 1
  - Direct use of parallel communication streams
  - VCIs as good as user-visible endpoints and MPI everywhere

- Category 2
  - Require shared progress
  - Both VCIs and user-visible endpoints perform poorly

- Category 3
  - Abstraction through MPI-3.1 prevents user from expressing parallelism
  - User-visible endpoints perform better than VCIs
 CATEGORY 1: POINT-TO-POINT MICROBENCHMARK

No scaling without user-expressed parallelism (ser_comm) or without VCIs (orig_mpich)
CATEGORI 1: POINT-TO-POINT MICROBENCHMARK

No scaling without user-expressed parallelism (ser_comm) or without VCIs (orig_mpich)

Parallel communication streams effective only when bound by the rate of issue of operations
CATEGORY 1: POINT-TO-POINT MICROBENCHMARK

No scaling without user-expressed parallelism (ser_comm) or without VCIs (orig_mpich)

Parallel communication streams effective only when bound by the rate of issue of operations

VCIs and user-visible endpoints short of MPI everywhere due to thread-safety costs

Takeaway: For basic communication, VCIs and endpoints perform similarly and nearly as well as MPI everywhere
CATEGORY 1: STENCIL APPLICATIONS

MPI Communicator

MPI Endpoints
CATEGORIE 1: STENCIL APPLICATIONS

**Recommendation:** Maximize independence between threads for point-to-point communication with communicators

**Warning:** Independent communication with ranks or tags is not sufficient because of receive wildcards

**Warning:** Expressing parallelism with MPI-3.1 can be clumsier compared with user-visible endpoints due to matching requirements
APPLICATION CATEGORIES

- Category 1
  - Direct use of parallel communication streams
  - VCIs as good as user-visible endpoints and MPI everywhere

- Category 2
  - Require shared progress
  - Both VCIs and user-visible endpoints perform poorly

- Category 3
  - Abstraction through MPI-3.1 prevents user from expressing parallelism
  - User-visible endpoints perform better than VCIs
CATEGORY 2: RMA MICROBENCHMARK

MPI everywhere performs best because target ranks progress their VCIs

Intel OPA emulates RMA in software, requiring target VCI involvement

Takeaway: When shared progress is required, neither VCIs nor endpoints perform well

Mellanox IB implements Puts completely in hardware
**CATEGORY 2: OPENMC**

- OpenMC: distributed Monte-Carlo neutron-transport code
  - Band data equally distributed between nodes
  - Particles distributed between nodes for simulation
  - Each node fetches (MPI_Get) a band of data, processes its particles, and iterates
VCIs as good as endpoints and MPI everywhere when shared progress not required
CATEGORY 2: OPENMC

VCIs as good as endpoints and MPI everywhere when shared progress not required.

Shared progress: thread A progresses VCI of thread B. Required for correctness.

Issue of operations is fast.

Shared progress hurts completion of operations.
HOW I LEARNED TO STOP WORRYING ABOUT USER-VISIBLE ENDPOINTS AND LOVE MPI

CATEGORY 2: OPENMC

VCIs as good as endpoints and MPI everywhere when shared progress not required

Recommendation: Maximize independence between threads for RMA communication with MPI windows

Warning: Independent communication with VCIs and user-visible endpoints fundamentally opposes shared progress

Shared progress: thread A progresses VCI of thread B. Required for correctness.

Issue of operations is fast

Shared progress hurts completion of operations
APPLICATION CATEGORIES

- Category 1
  - Direct use of parallel communication streams
  - VCIs as good as user-visible endpoints and MPI everywhere

- Category 2
  - Require shared progress
  - Both VCIs and user-visible endpoints perform poorly

- Category 3
  - Abstraction through MPI-3.1 prevents user from expressing parallelism
  - User-visible endpoints perform better than VCIs
CATEGORY 3: LIMITING MPI SEMANTICS

- Example: microbenchmark capturing communication pattern in Legion’s runtime

- Contention between receiver thread and sender threads with communicators. No contention with endpoints.

Takeaway: User-visible endpoints perform better than VCIs when MPI’s semantics prevent user from expressing parallelism, especially in irregular communication patterns.
NWChem: quantum chemistry application suite

- Dominant cost is that of block-sparse matrix multiplication (BSPMM)
- \( A \times B += C \) get-compute-update pattern (MPI_Get + MPI_Accumulate)
- Each worker on a node (thread or process) participates in BSPMM independently
**CATEGORY 3: NWChem**

- NWChem: quantum chemistry application suite
  - Dominant cost is that of block-sparse matrix multiplication (BSPMM)
  - $A \times B += C$ get-compute-update pattern ($\text{MPI\_Get} + \text{MPI\_Accumulate}$)
  - Each worker on a node (thread or process) participates in BSPMM independently

![Parallel Get and Parallel Accumulate Diagram]

- **Parallel Get**
  - Rank $i$
  - Window: 0, 1, 2, 3
- **Parallel Accumulate**
  - Rank $i$
  - Endpoint: 0
CATEGORY 3: NWChem

Issue of Gets is fast

Shared progress hurts completion of Get operations

VCIs slower to issue Accumulates than endpoints because of single window

Endpoints complete Accumulates slower than VCIs because of shared progress

Warning: Atomic operation semantics are not easy to achieve with multiple windows; using multiple VCIs may not help.
HOW I LEARNED TO STOP WORRYING ABOUT USER-VISIBLE ENDPOINTS AND LOVE MPI

CATEGORY 3: NWCHEM

Warning: Atomic operation semantics are not easy to achieve with multiple windows; using multiple VCIs may not help.

Tip: If the application allows it, hint accumulate_ordering=none. The MPI library can exploit implicit parallelism.
CONCLUDING REMARKS

- MPI+threads is critical for modern processors
  - Users must proactively express logical parallelism
- User-visible endpoints not critical to express logical parallelism
  - MPI-3.1 already features lots of parallelism
- VCIs perform as well as user-visible endpoints without burdening the user
- New info hints in MPI-4.0 give more options to express logical parallelism
  - Enabling exploration of advanced mapping policies in the MPI library
THANK YOU!

Email questions to rzambre@uci.edu