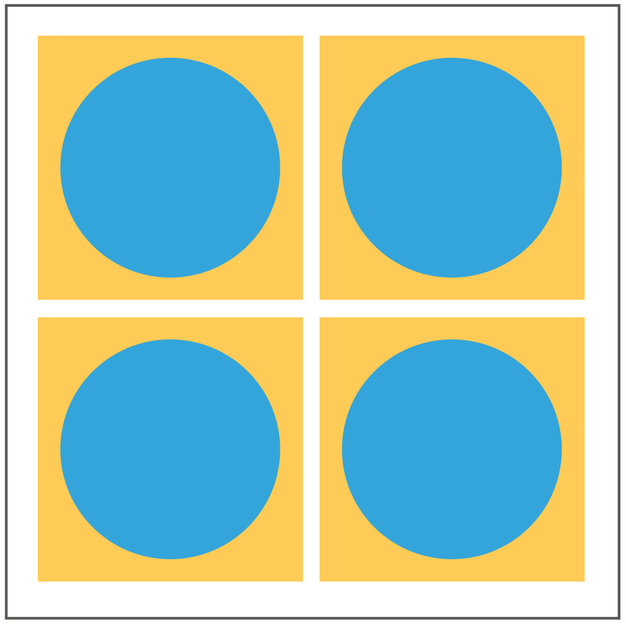
How I Learned to Stop Worrying About User-Visible Endpoints and Love MPI

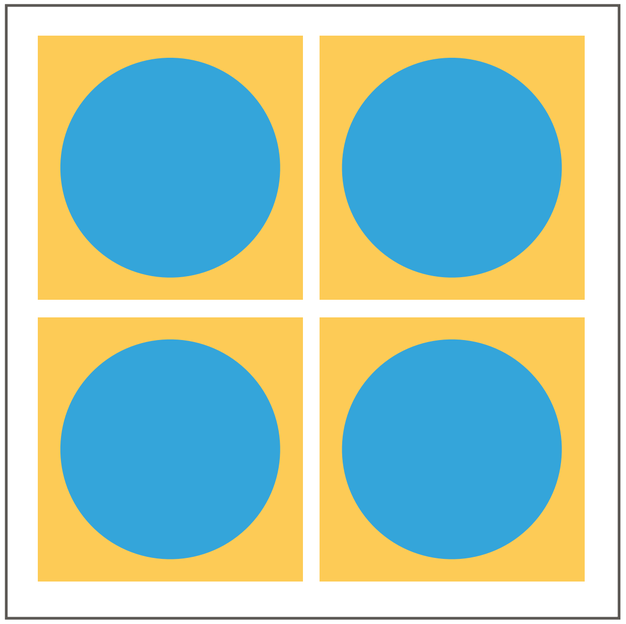
Rohit Zambre,*
Aparna Chandramowlishwaran,*
Pavan Balaji^

*University of California, Irvine
^Argonne National Laboratory

MPI everywhere

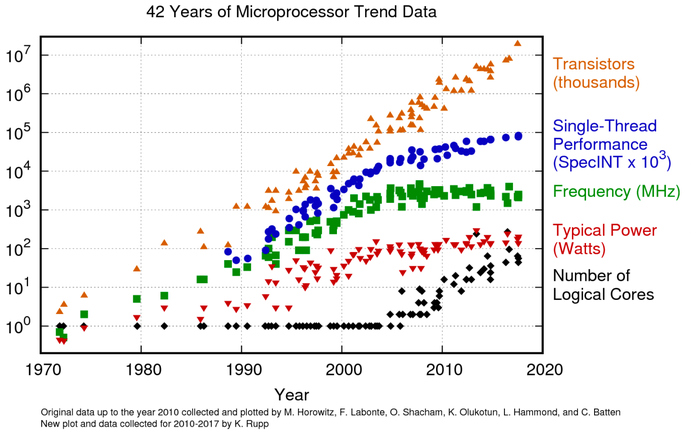


MPI everywhere

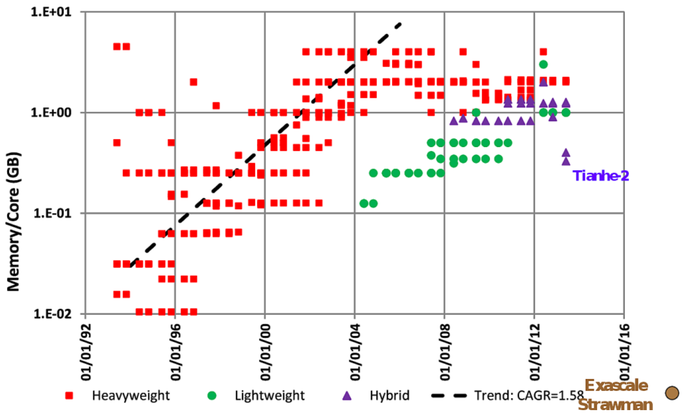


Node Core Process

- *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

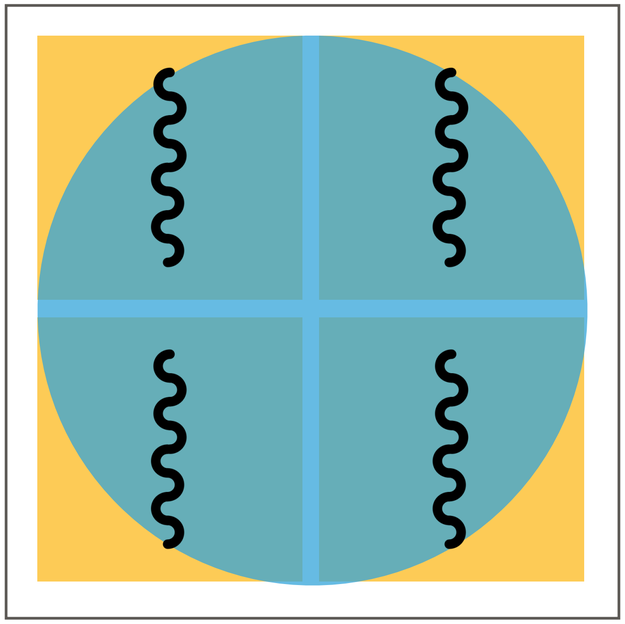


Increasing number of cores

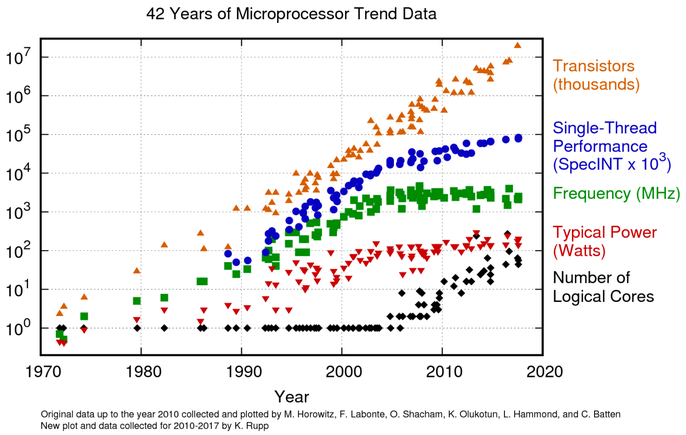


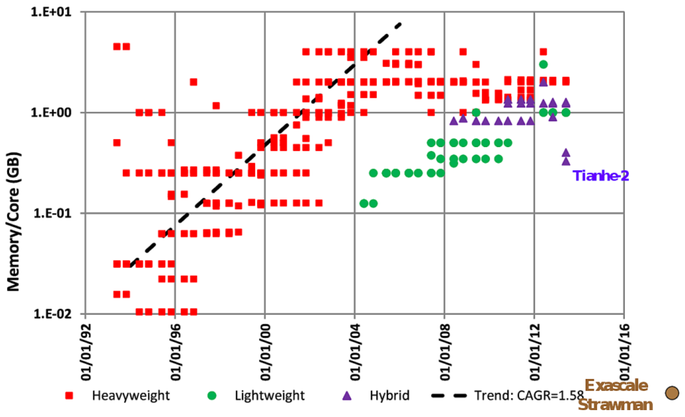
Decreasing memory per core

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.





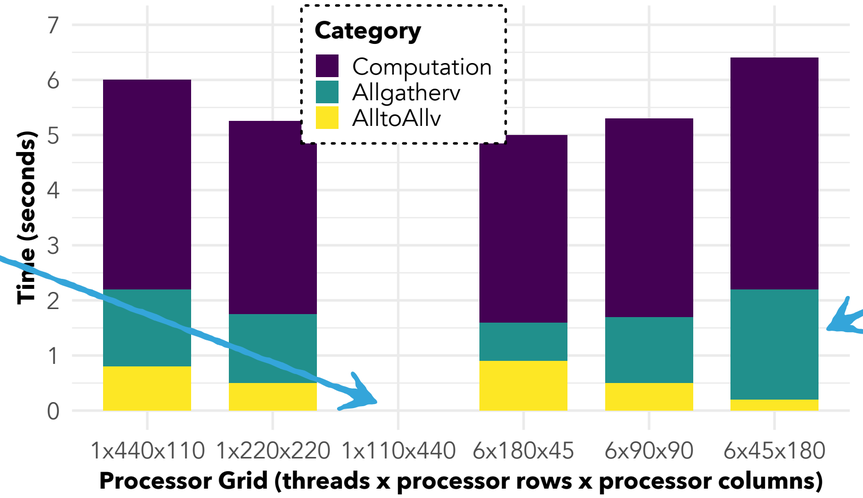
Node Core Process Thread

Increasing number of cores

Decreasing memory per core

MPI everywhere

OOM!

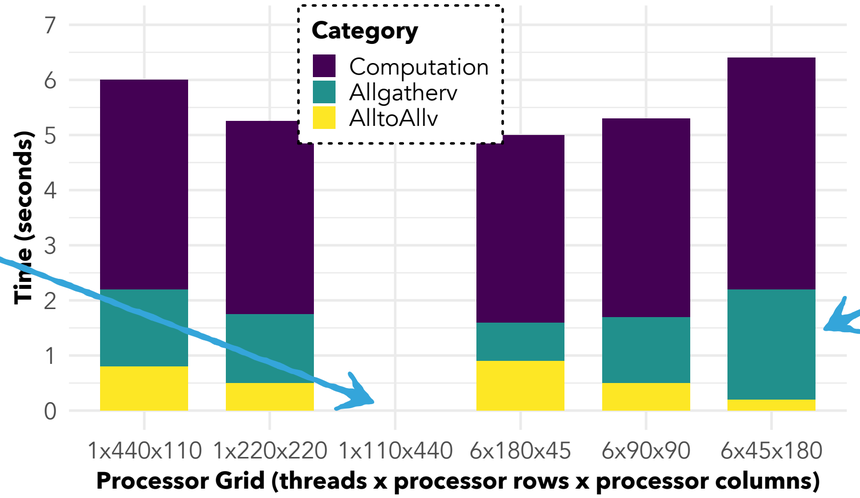


Corresponding MPI+threads runs

Buluc et al. Distributed BFS (https://arxiv.org/abs/1705.04590)

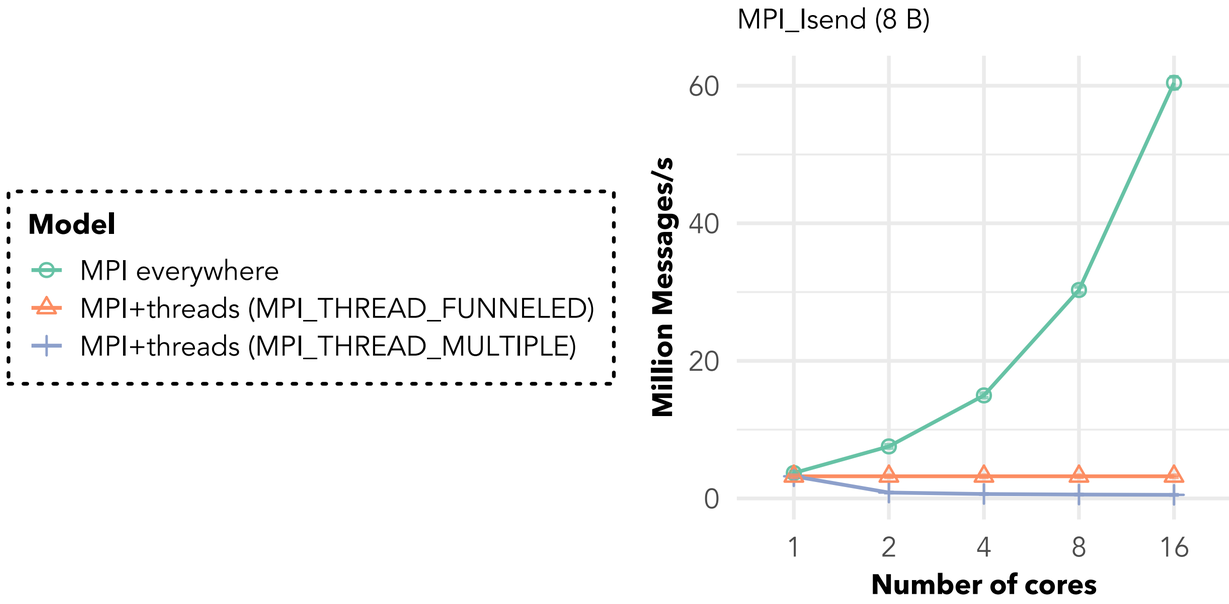
MPI everywhere

OOM!

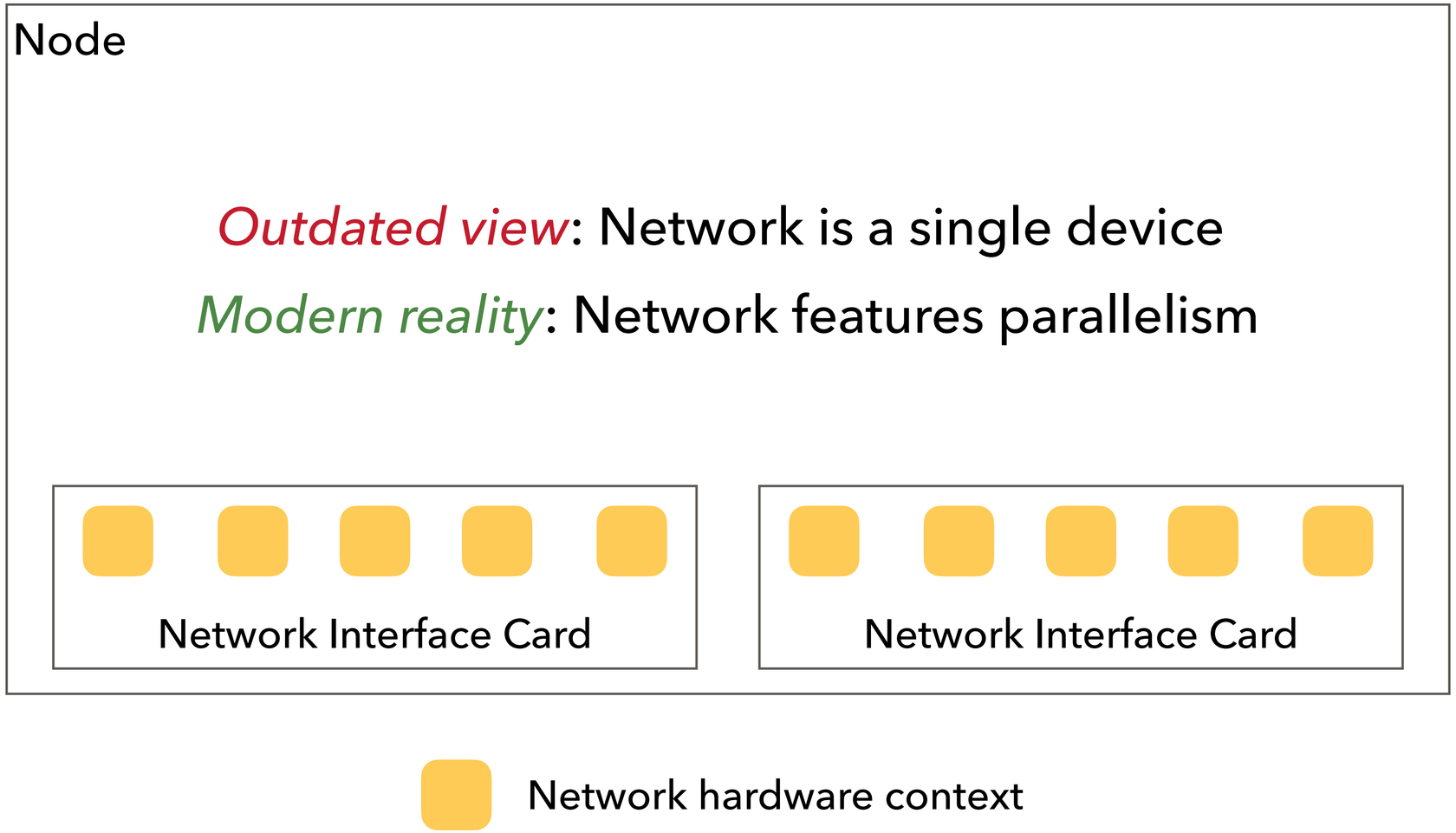


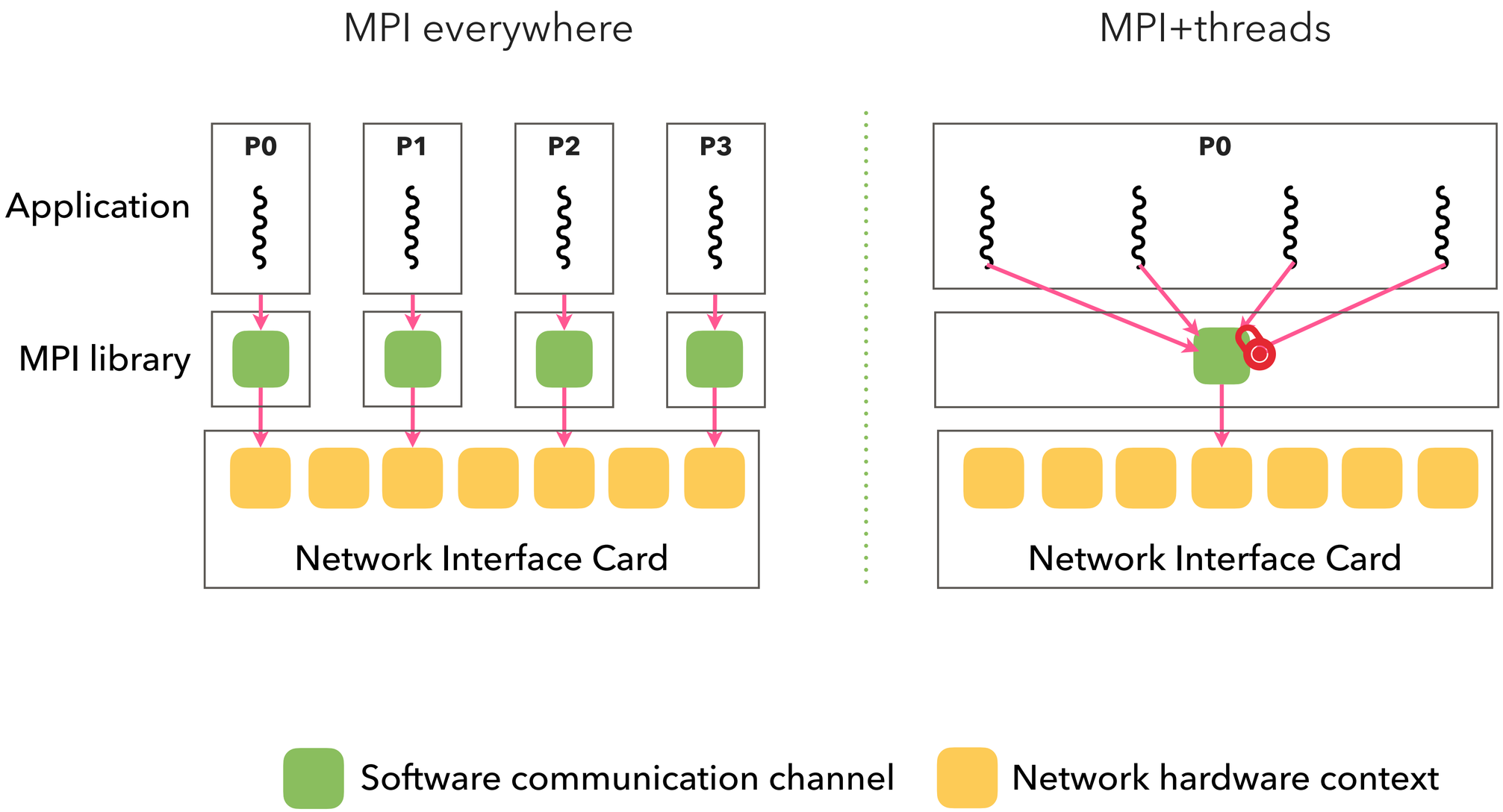
Corresponding MPI+threads runs

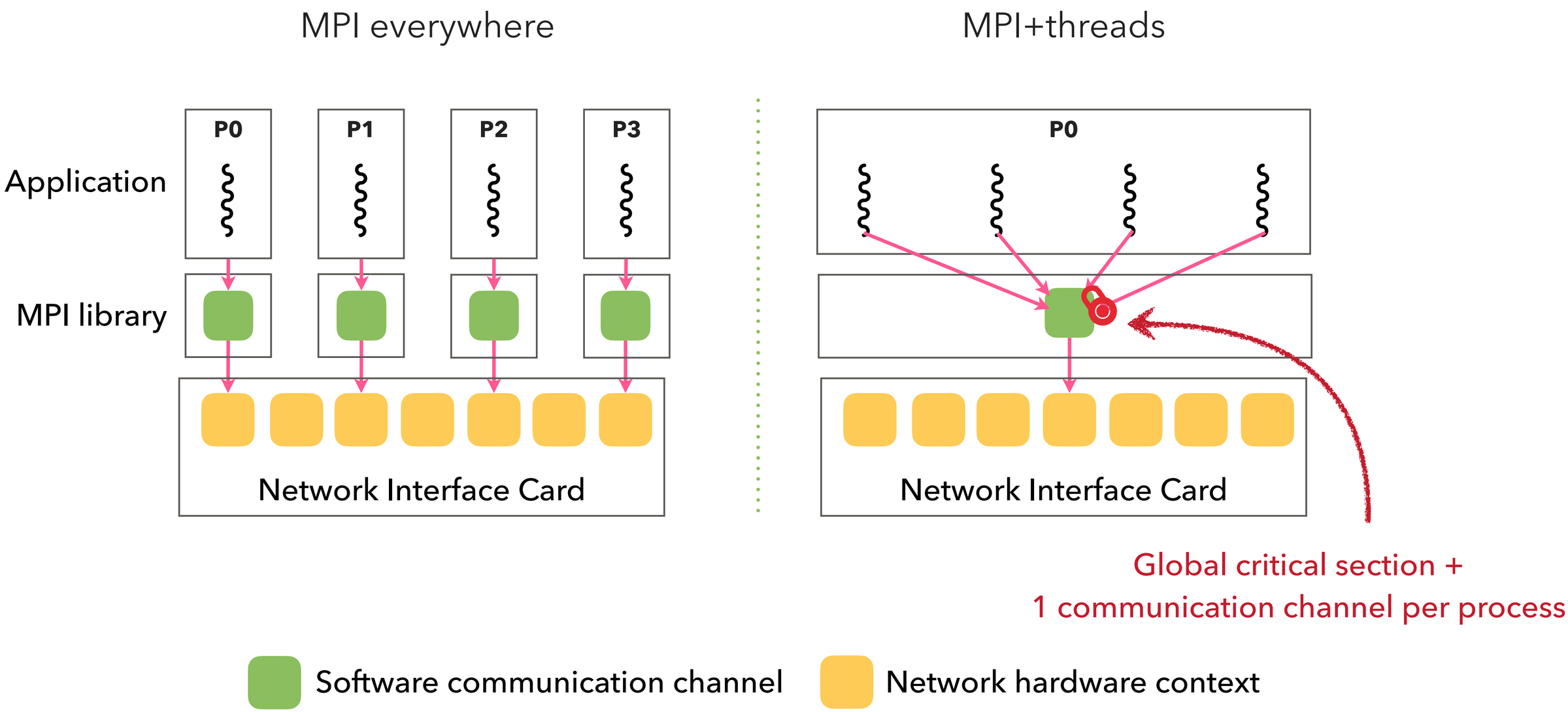
Buluc et al. Distributed BFS (https://arxiv.org/abs/1705.04590)

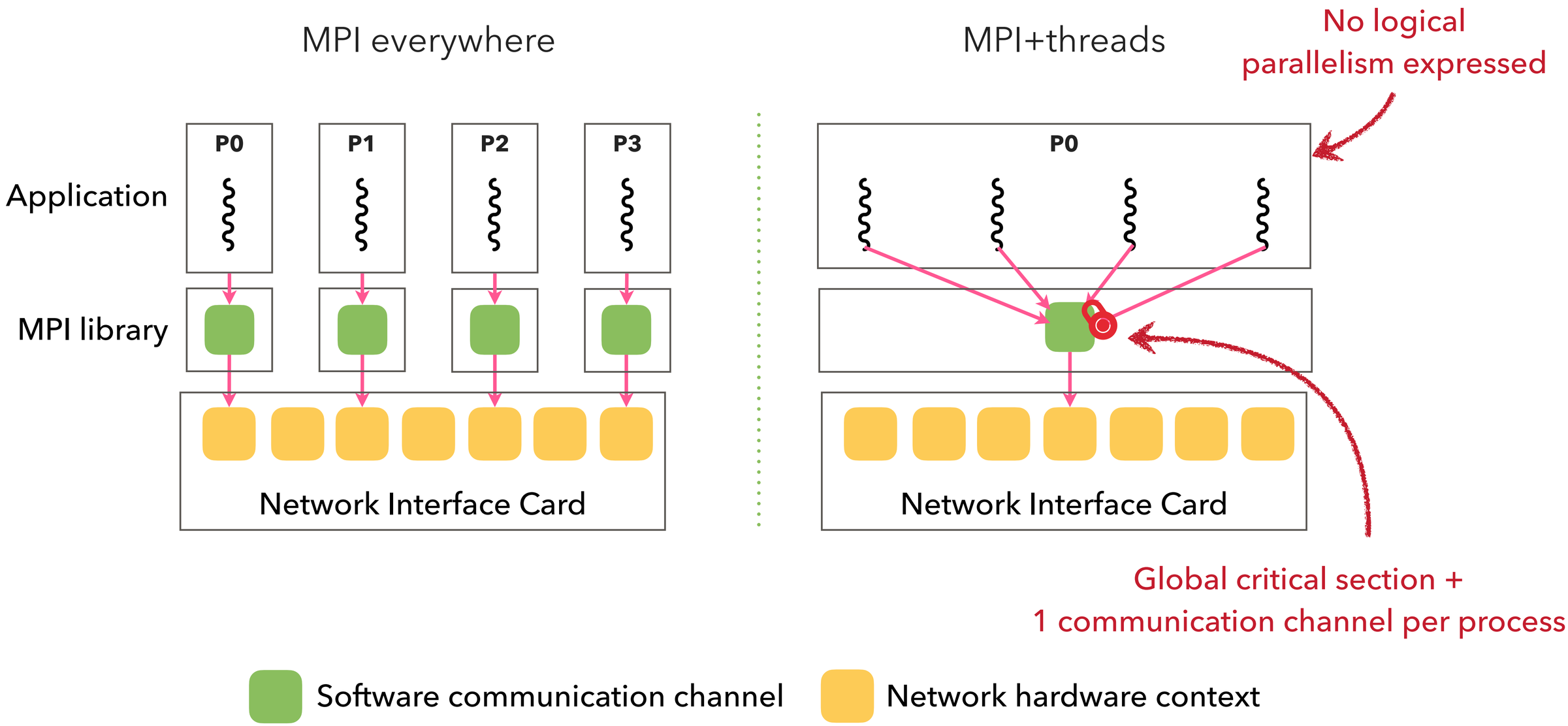


Communication performance of MPI+threads is dismal


Node
Outdated view: Network is a single device
Modern reality: Network features parallelism
Network Interface Card
Network Interface Card
Network hardware context

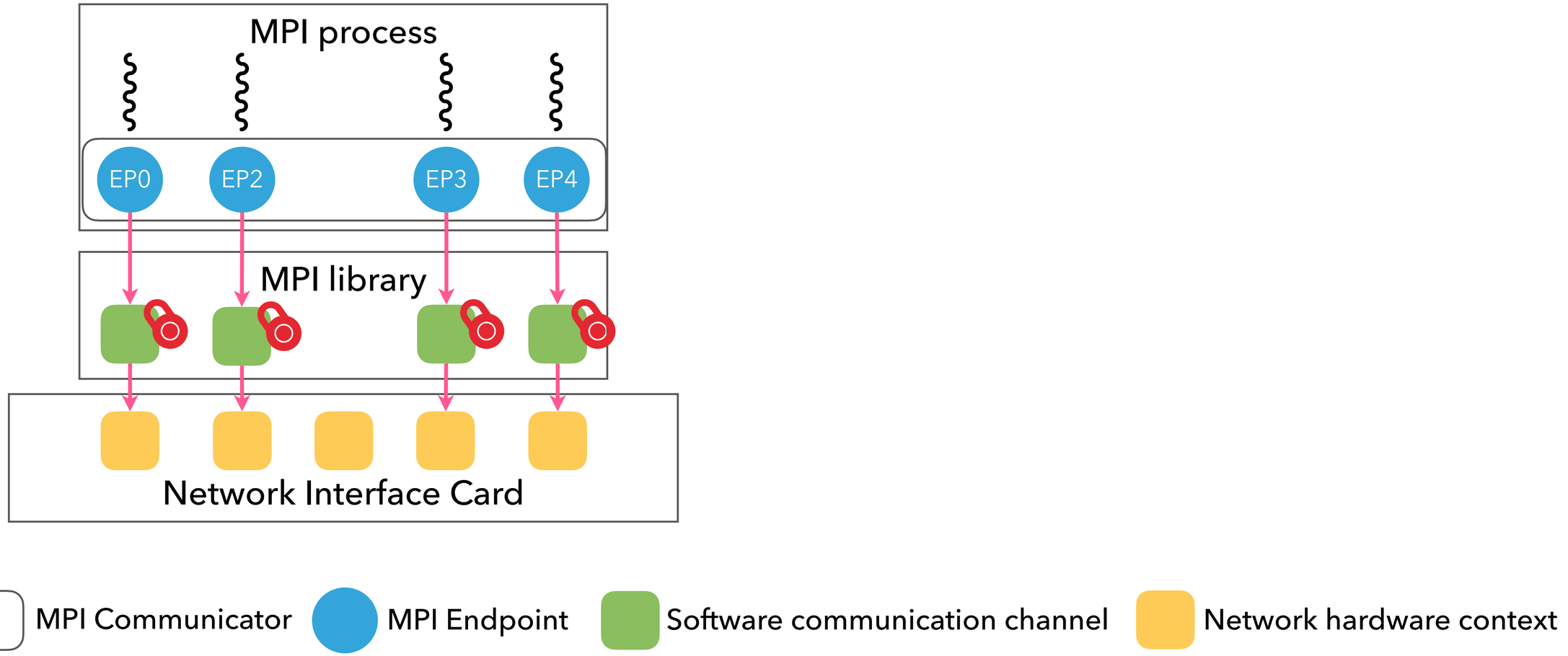

MPI everywhere
MPI+threads
Application
MPI library
P0
P1
P2
P3
Network Interface Card
P0
Network Interface Card
Software communication channel
Network hardware context


MPI everywhere
MPI+threads
Application
MPI library
P0
P1
P2
P3
Network Interface Card
P0
Network Interface Card
Global critical section +
1 communication channel per process
Software communication channel
Network hardware context

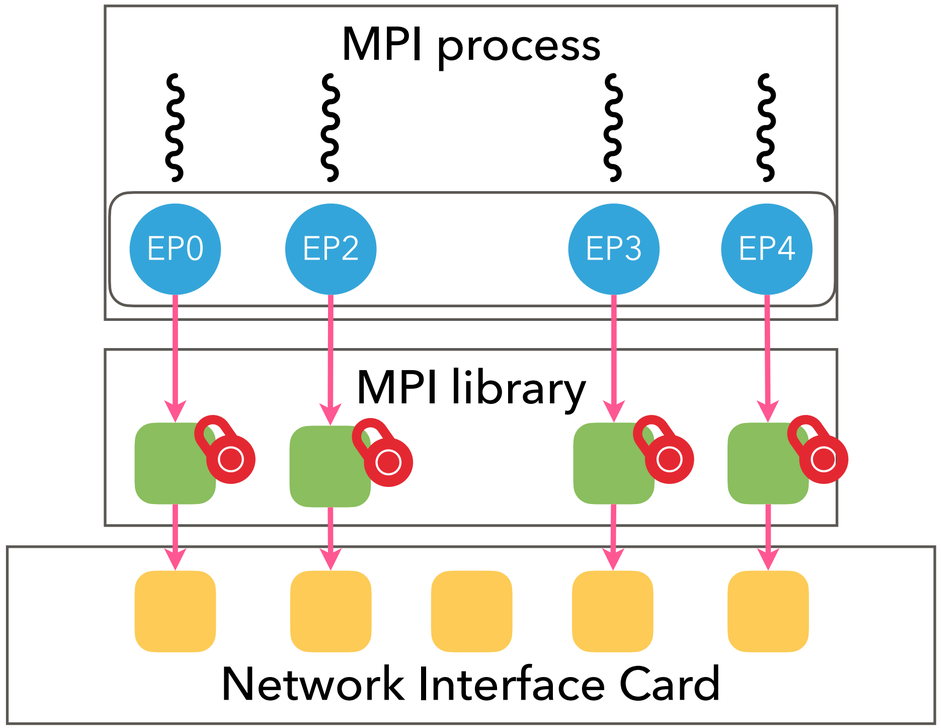

MPI everywhere
MPI+threads
No logical parallelism expressed
Application
MPI library
P0
P1
P2
P3
P0
Network Interface Card
Network Interface Card
Global critical section +
1 communication channel per process
Software communication channel
Network hardware context

```
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,…);
```



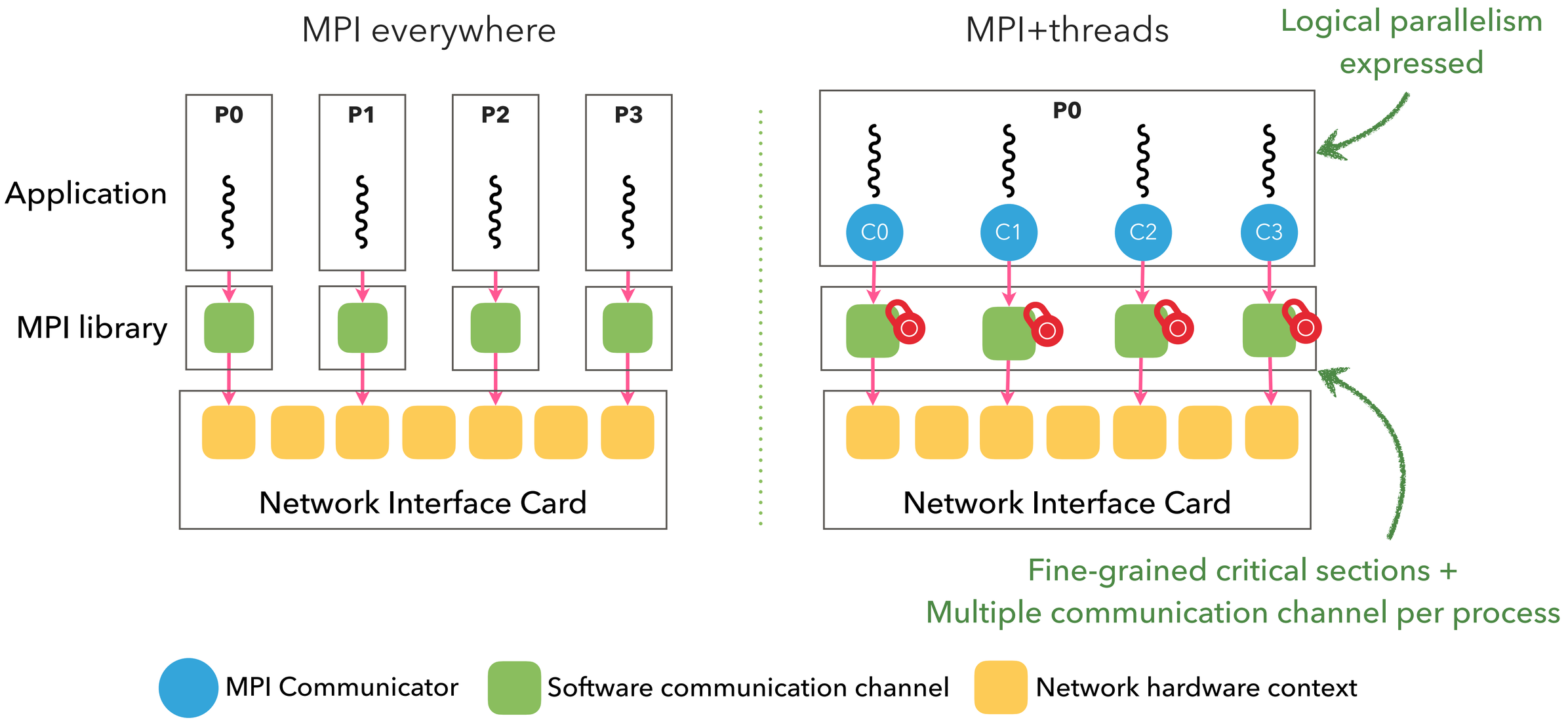
Pros

- Explicit control over network contexts

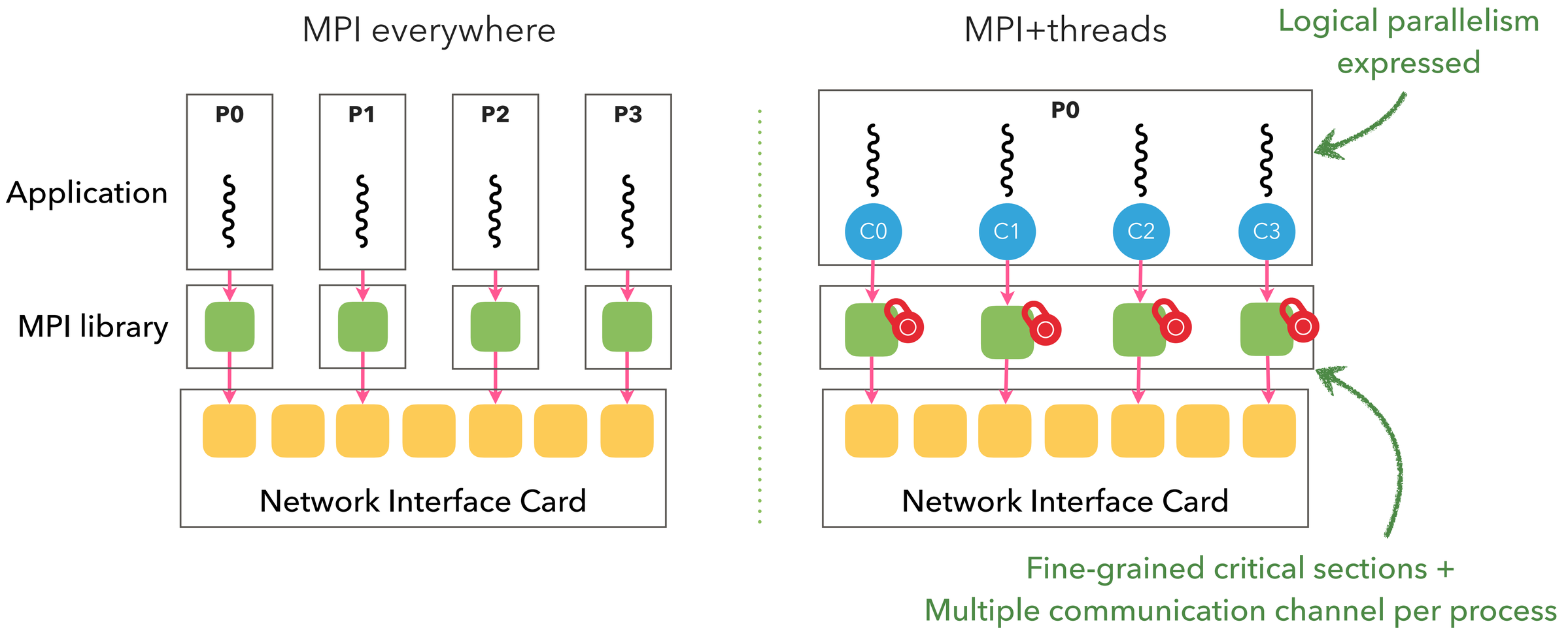
Cons

- Intrusive extension of the MPI standard
- Onus of managing network contexts on user

MPI Communicator MPI Endpoint Software communication channel Network hardware context


MPI everywhere
MPI+threads
Logical parallelism expressed
Application
MPI library
P0
P1
P2
P3
C0
C1
C2
C3
Network Interface Card
Network Interface Card
Fine-grained critical sections +
Multiple communication channel per process
MPI Communicator
Software communication channel
Network hardware context

Do we need user-visible endpoints?



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

Evaluation platforms	MPI library	Interconnects
	Based on MPICH:CH4	Intel Omni-Path (OPA) with OFI:PSM2 Mellanox InfiniBand (IB) with UCX:Verbs

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

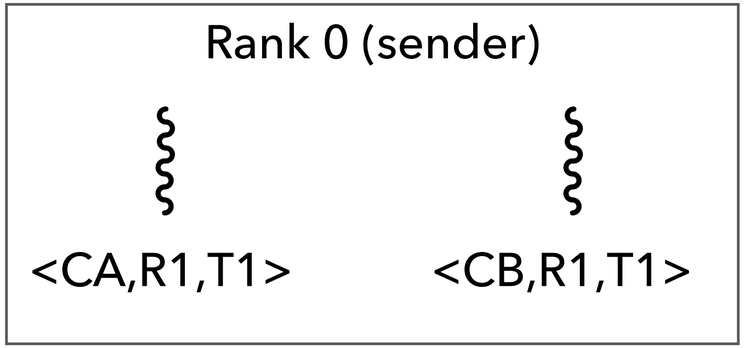
- <comm,rank,tag> decides matching
- Non-overtaking order
- Receive wildcards

Two or more operations on a process with			Can be issued on parallel communication streams?	
Comm	Rank	Tag	Send	Recv

POINT-TO-POINT COMMUNICATION

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

Two or more operations on a process with			Can be issued on parallel communication streams?	
Comm	Rank	Tag	Send	Recv
Different	Different or Same	Different or Same	Yes	Yes

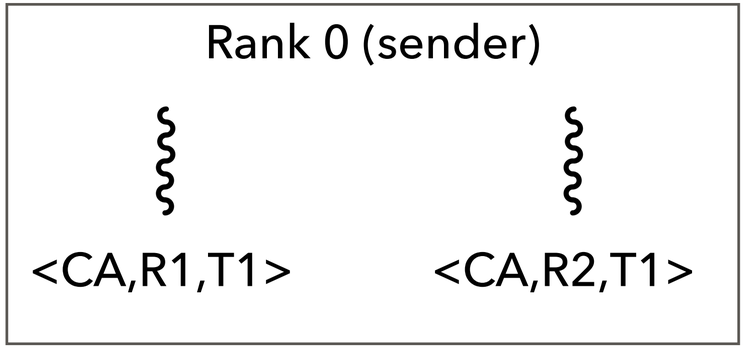


POINT-TO-POINT COMMUNICATION

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

Two or more operations on a process with			Can be issued on parallel communication streams?	
Comm	Rank	Tag	Send	Recv
Different	Different or Same	Different or Same	Yes	Yes
Same	Different	Different or Same	Yes	No

Wildcards



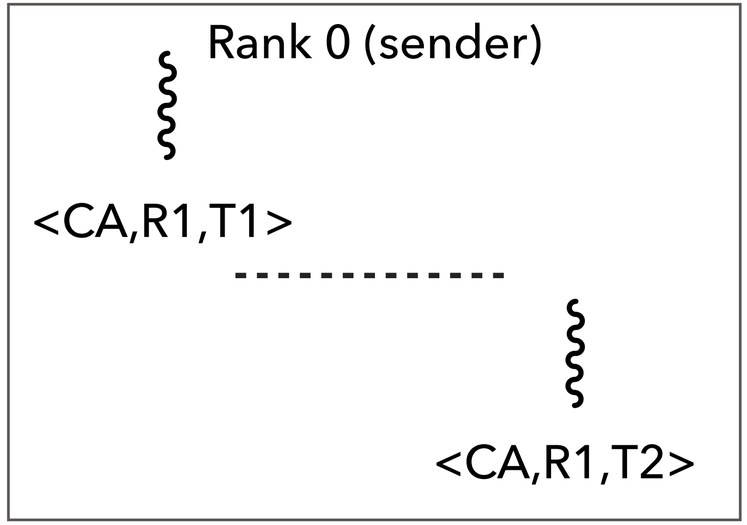
POINT-TO-POINT COMMUNICATION

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

Two or more operations on a process with			Can be issued on parallel communication streams?	
Comm	Rank	Tag	Send	Recv
Different	Different or Same	Different or Same	Yes	Yes
Same	Different	Different or Same	Yes	No
Same	Same	Different or Same	No	No

Wildcards

Non-overtaking order



RMA COMMUNICATION

Two or more operations on a process with		Can be issued on parallel communication streams?		
Window	Rank	Put	Get	Accumulate
Different	Different or Same	Yes	Yes	Yes
Same	Different	Yes	Yes	Yes
Same	Same	Yes	Yes	No

RMA COMMUNICATION

Explicitly expressing parallelism

Implicit parallelism

Two or more operations on a process with		Can be issued on parallel communication streams?		
Window	Rank	Put	Get	Accumulate
Different	Different or Same	Yes	Yes	Yes
Same	Different	Yes	Yes	Yes
Same	Same	Yes	Yes	No

No order between multiple Gets and Puts

RMA COMMUNICATION

Explicitly expressing parallelism

Implicit parallelism

Two or more operations on a process with		Can be issued on parallel communication streams?		
Window	Rank	Put	Get	Accumulate
Different	Different or Same	Yes	Yes	Yes
Same	Different	Yes	Yes	Yes
Same	Same	Yes	Yes	No

No order between multiple Gets and Puts

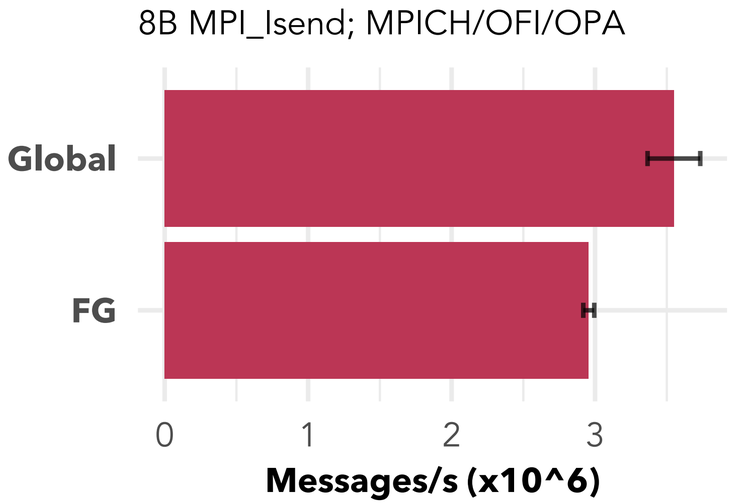
Ordering of accumulate operations to the same memory location

OUTLINE

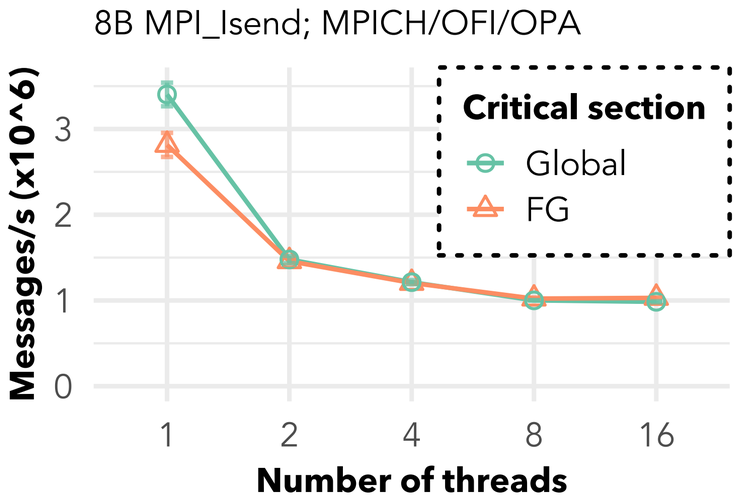
- Introduction
- For MPI users: Parallelism in the MPI standard
- For MPI developers: Fast MPI+threads
  - Fine-grained critical sections for thread safety
  - Virtual Communication Interfaces (VCIs) for parallel communication streams
- Microbenchmark and Application analysis

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



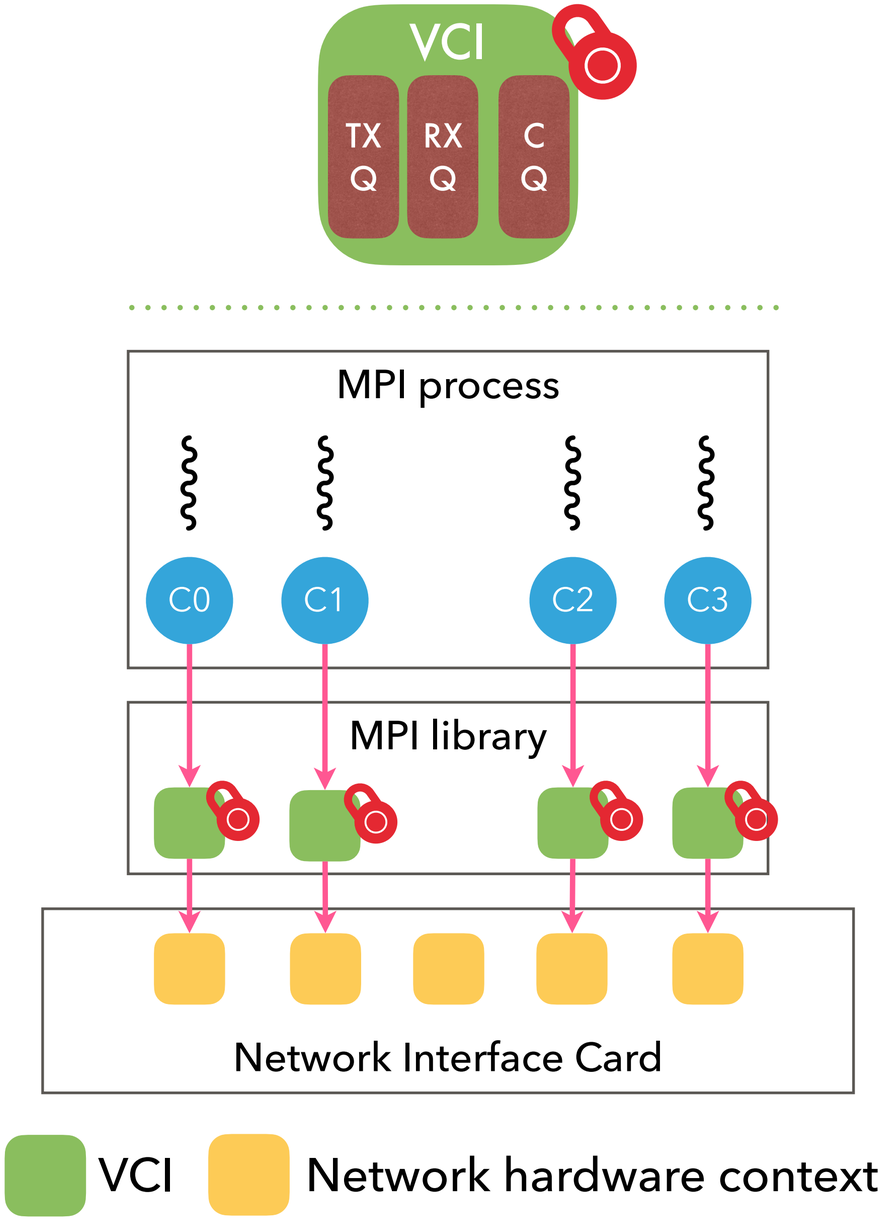
Overheads of FG in the single thread case

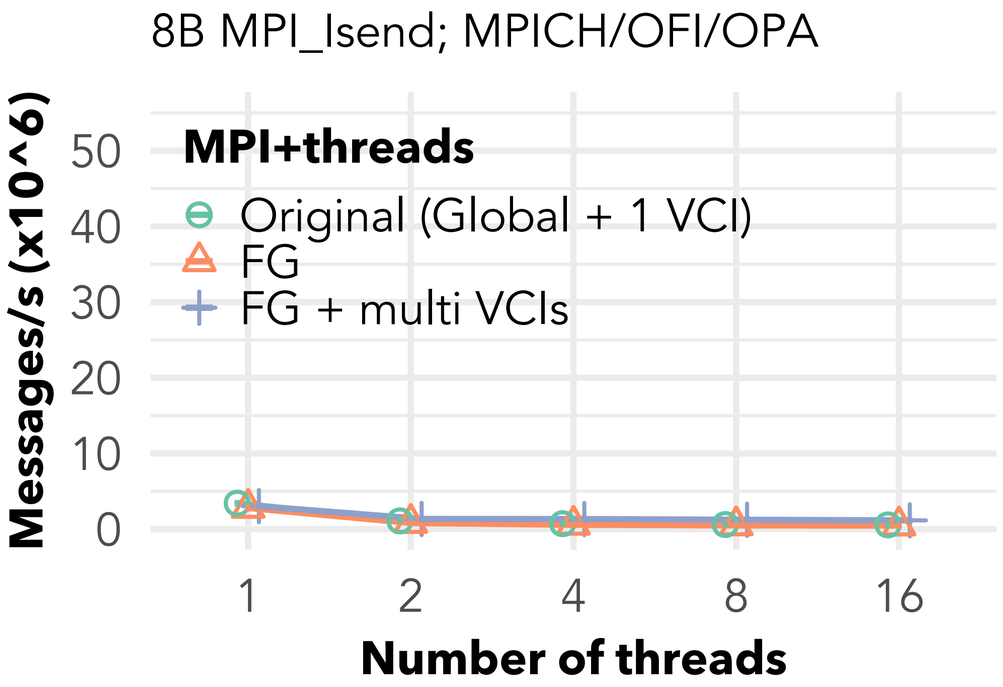


FG outperforms Global at higher thread count

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

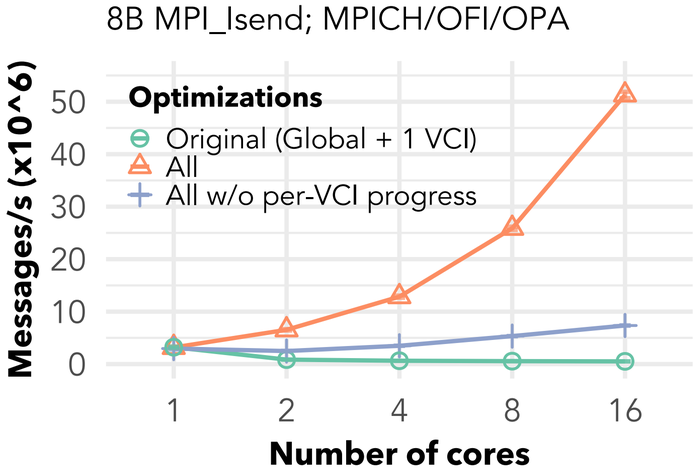




Fine-grained critical sections + multiple VCIs alone give practically no benefit

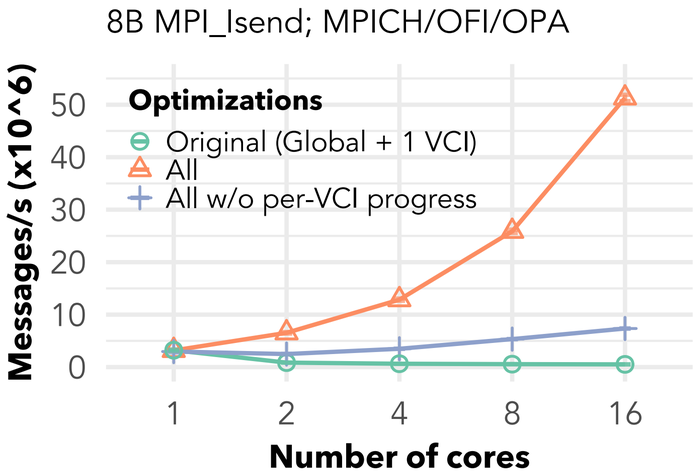
Per-VCI progress

- *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*



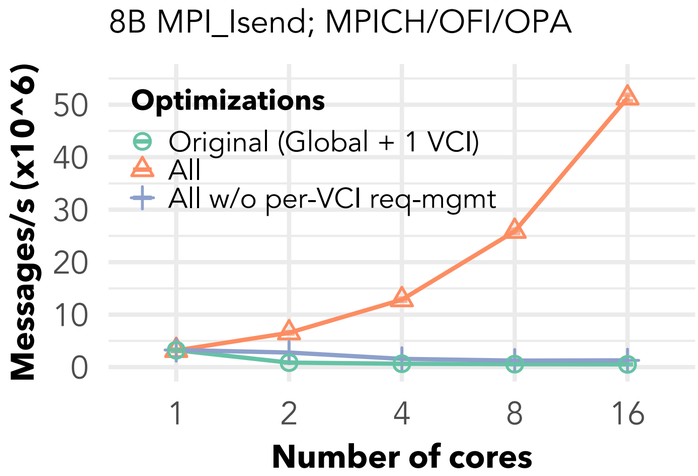
Per-VCI progress

- *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*



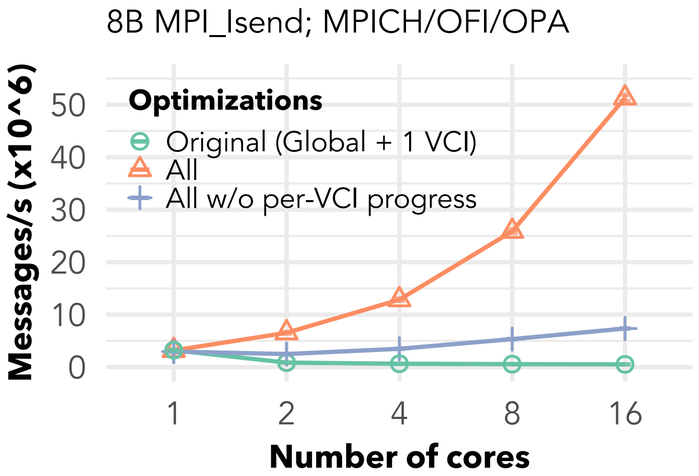
Per-VCI Request management

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



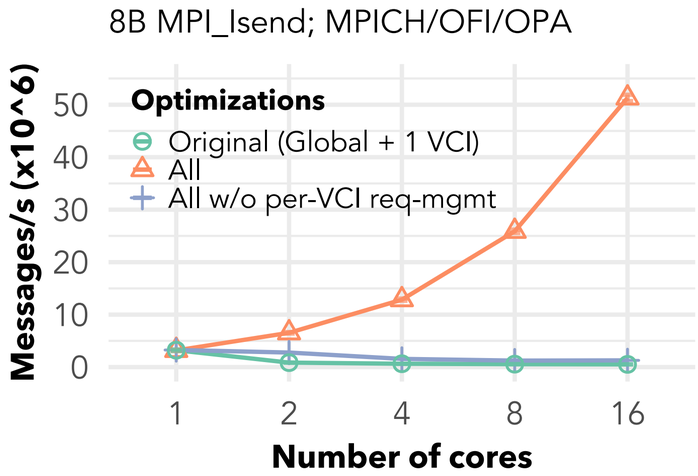
Per-VCI progress

- *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*



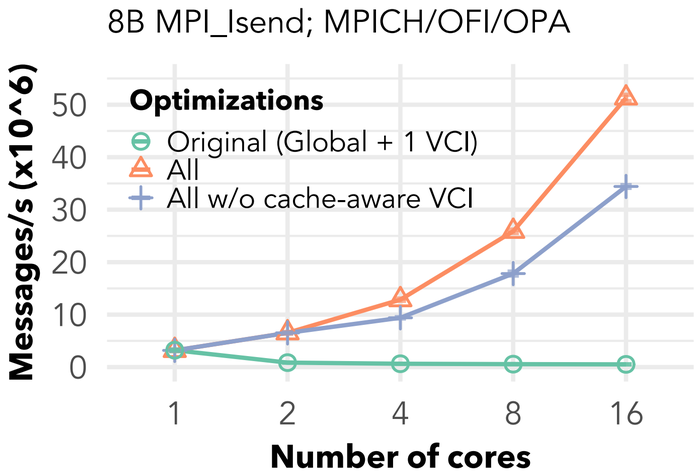
Per-VCI Request management

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



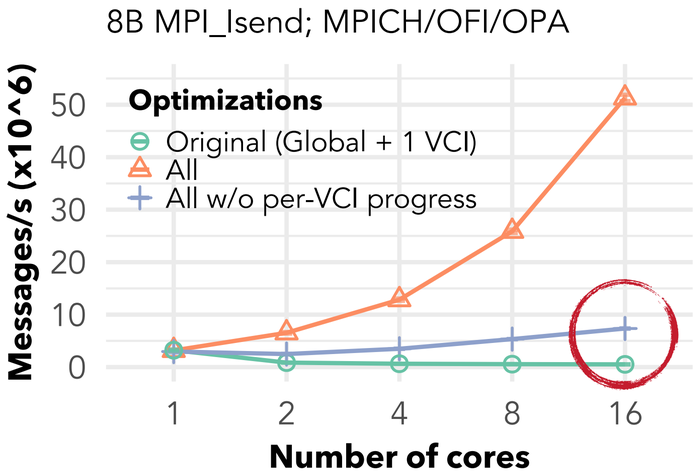
Per-VCI cache-line awareness

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



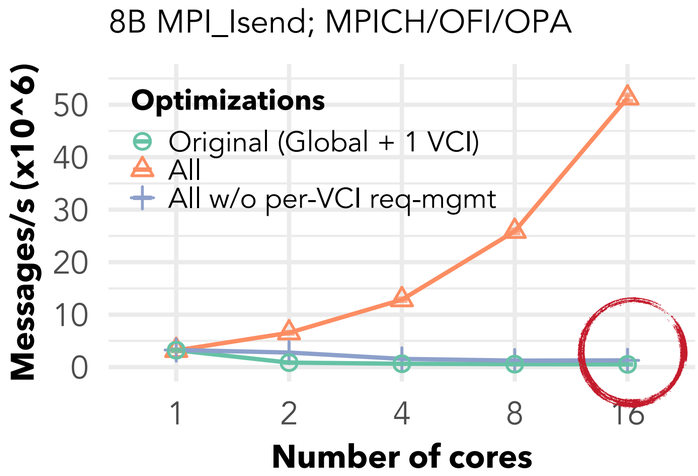
Per-VCI progress

- *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*



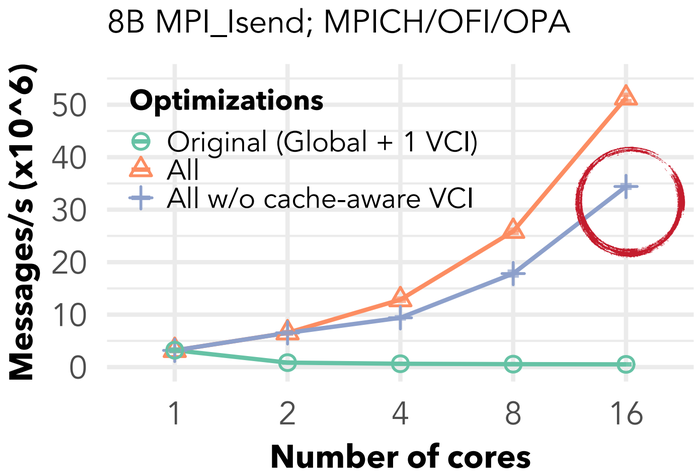
Per-VCI Request management

- *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*



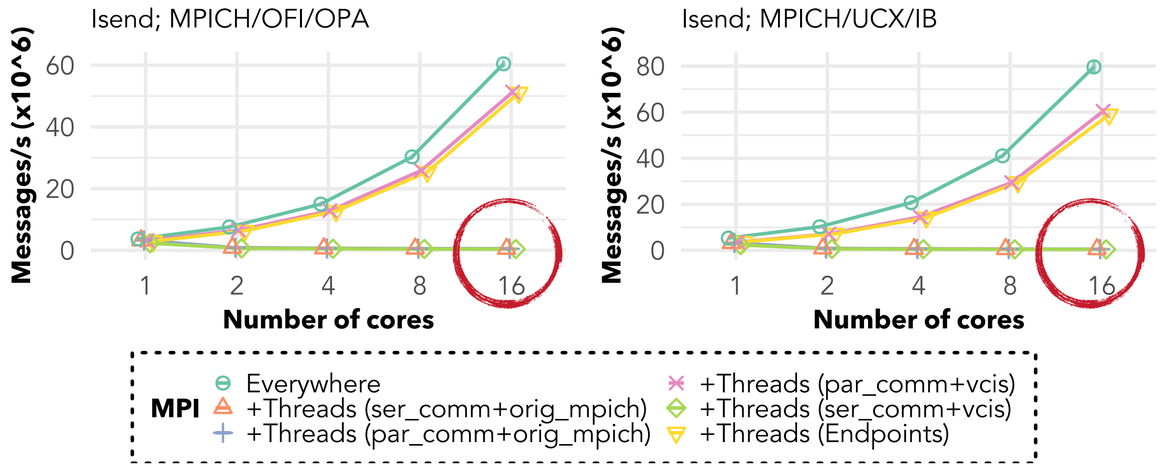
OUTLINE

- 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

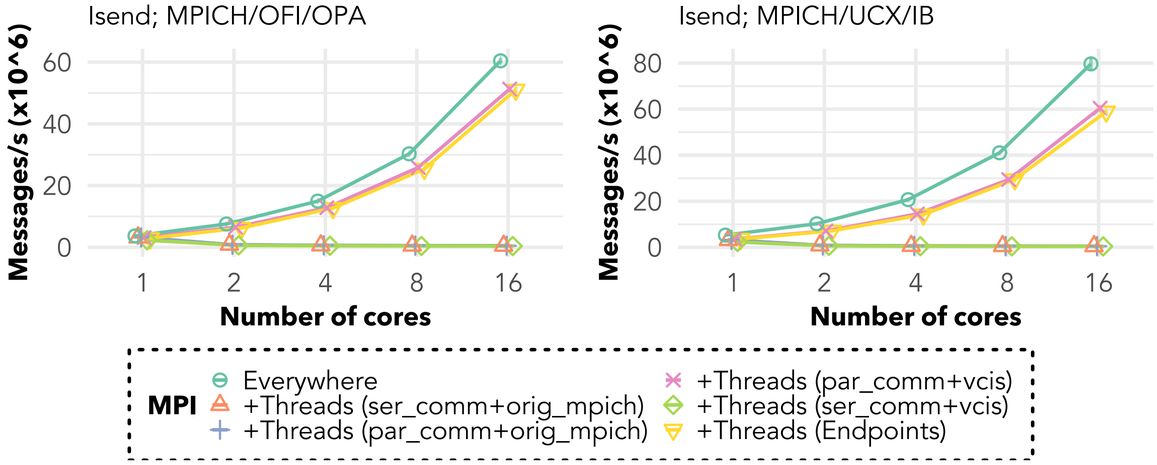
- Category 1
  - Direct use of parallel communication streams
  - VCIs as good as user-visible endpoints and MPI everywhere
- Category 2
  - Require shared progress
  - Both VCIs and user-visible endpoints perform poorly
- Category 3
  - Abstraction through MPI-3.1 prevents user from expressing parallelism
  - User-visible endpoints perform better than VCIs

CATEGORY 1: POINT-TO-POINT MICROBENCHMARK

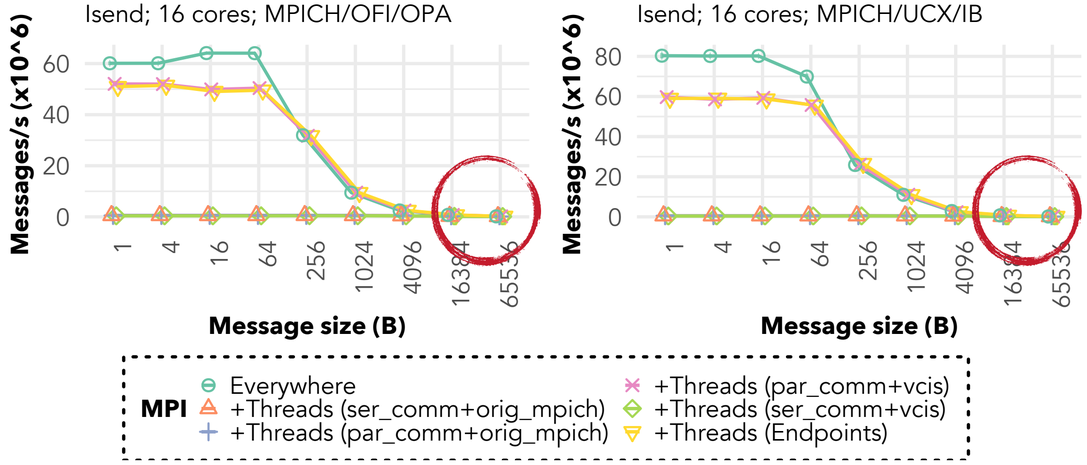


No scaling without user-expressed parallelism (ser_comm) or without VCIs (orig_mpich)

CATEGORY 1: POINT-TO-POINT MICROBENCHMARK

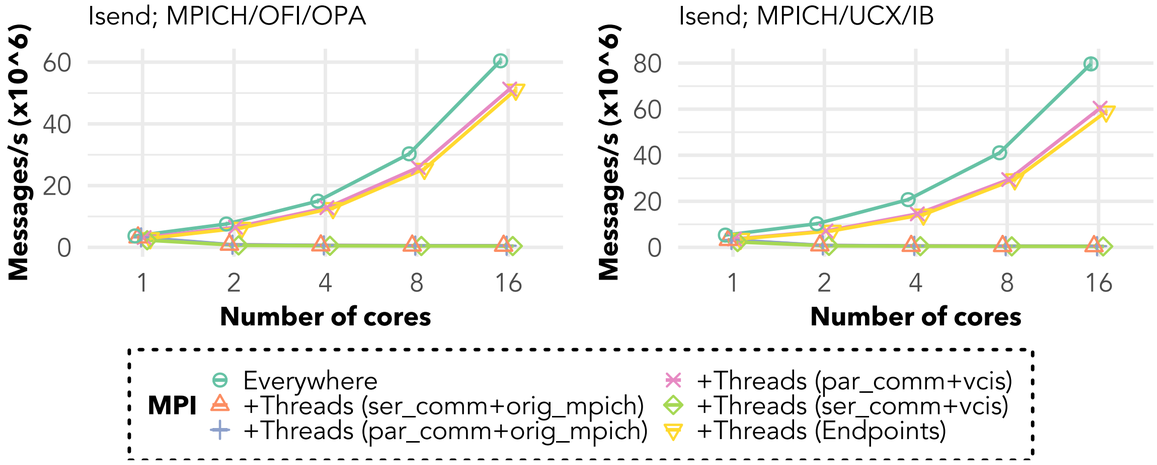


No scaling without user-expressed parallelism (ser_comm) or without VCIs (orig_mpich)

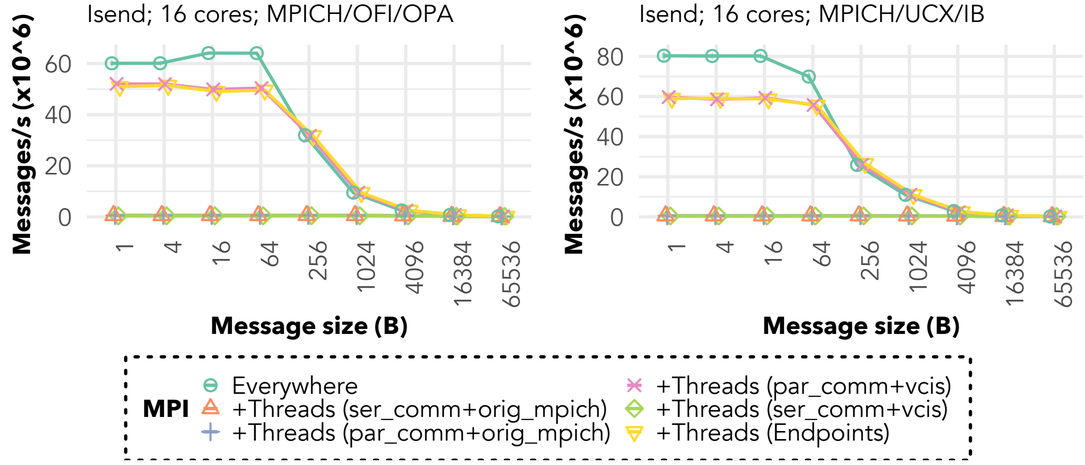


Parallel communication streams effective only when bound by the rate of issue of operations

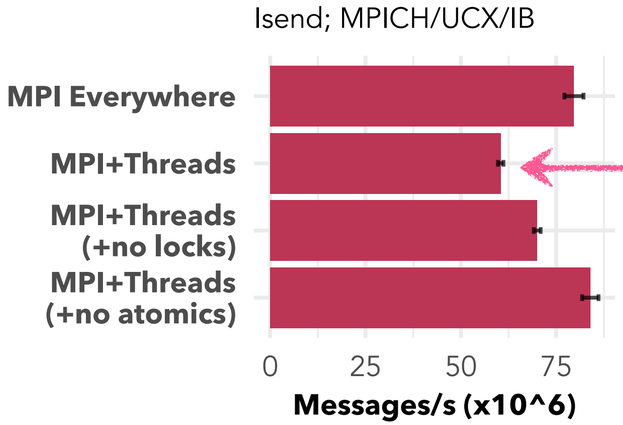
CATEGORY 1: POINT-TO-POINT MICROBENCHMARK



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



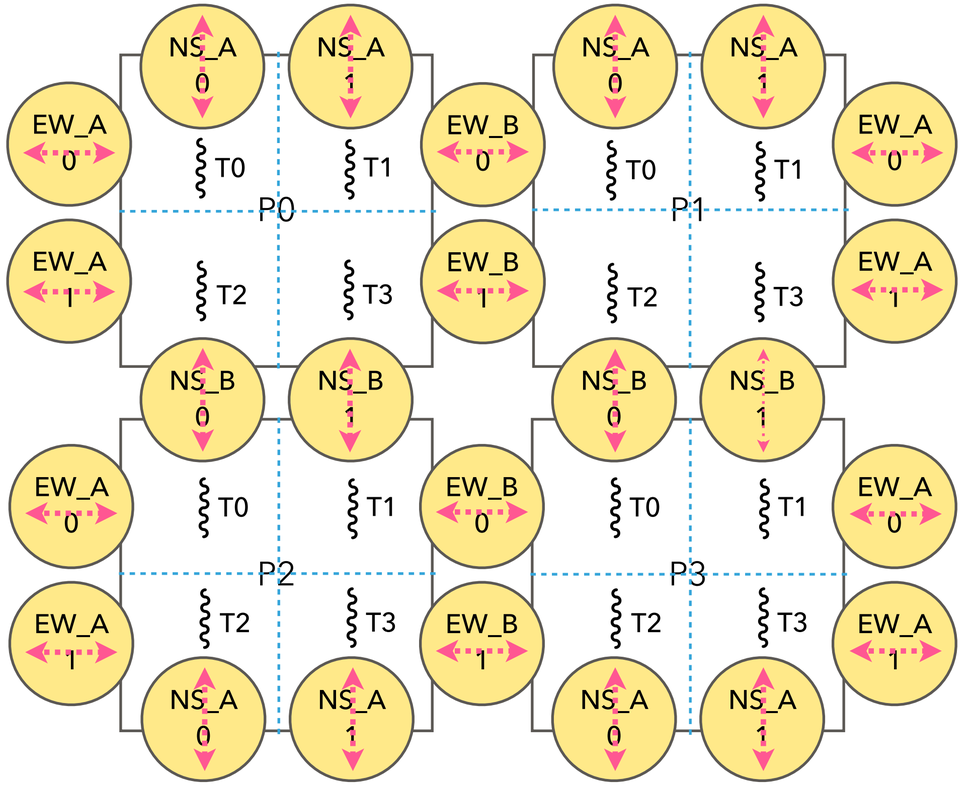
Parallel communication streams effective only when bound by the rate of issue of operations

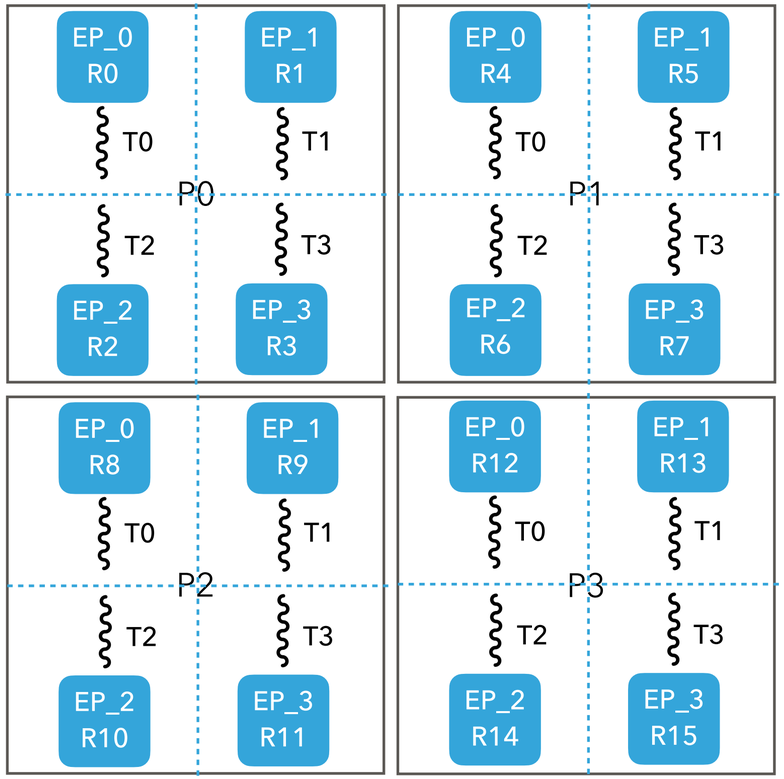


VCIs and user-visible endpoints short of MPI everywhere due to thread-safety costs

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

CATEGORY 1: STENCIL APPLICATIONS

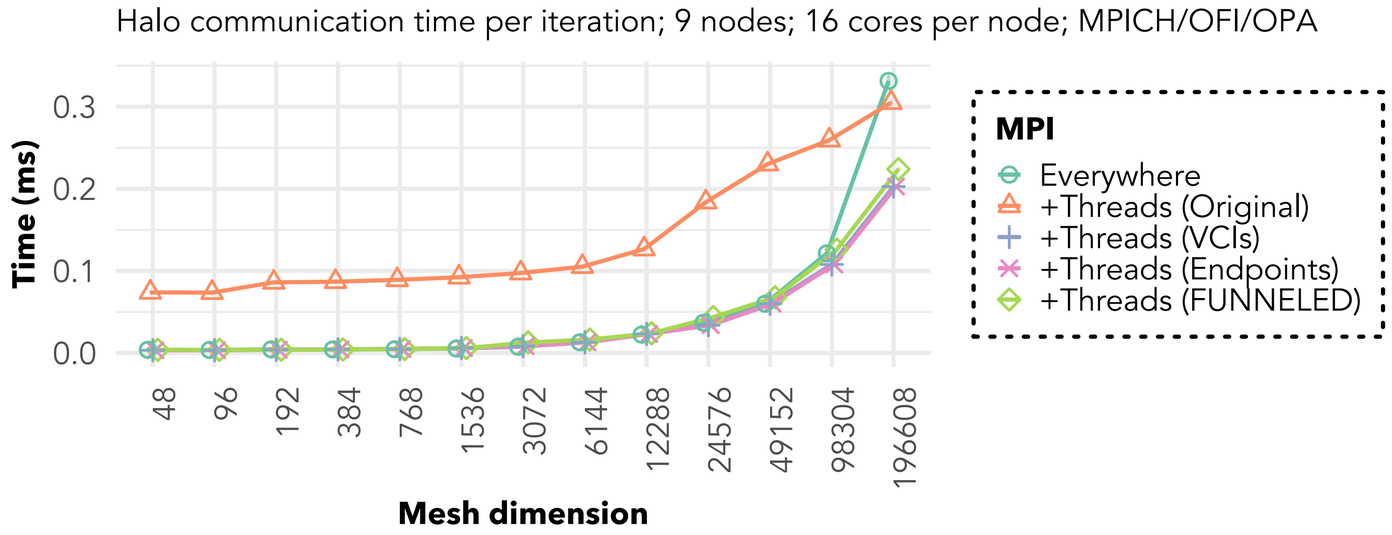




MPI Communicator

MPI Endpoints

CATEGORY 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

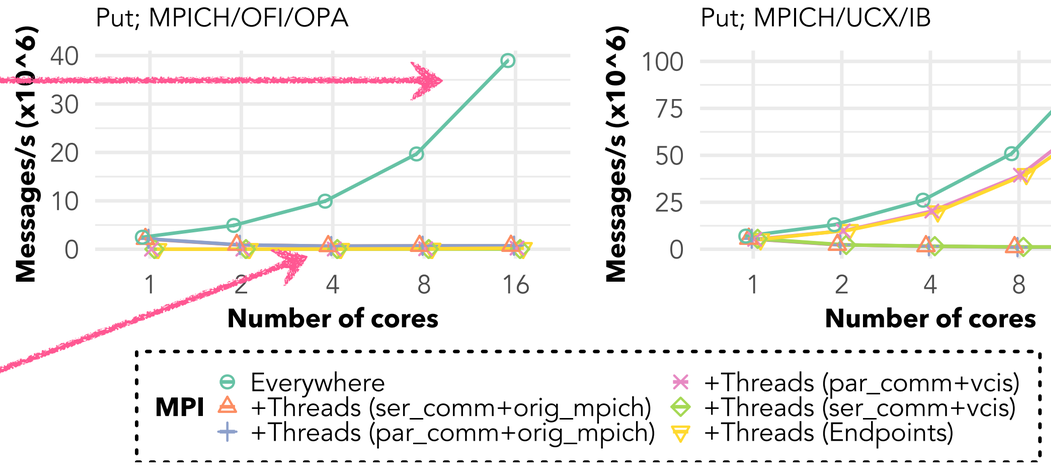
APPLICATION CATEGORIES

- 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

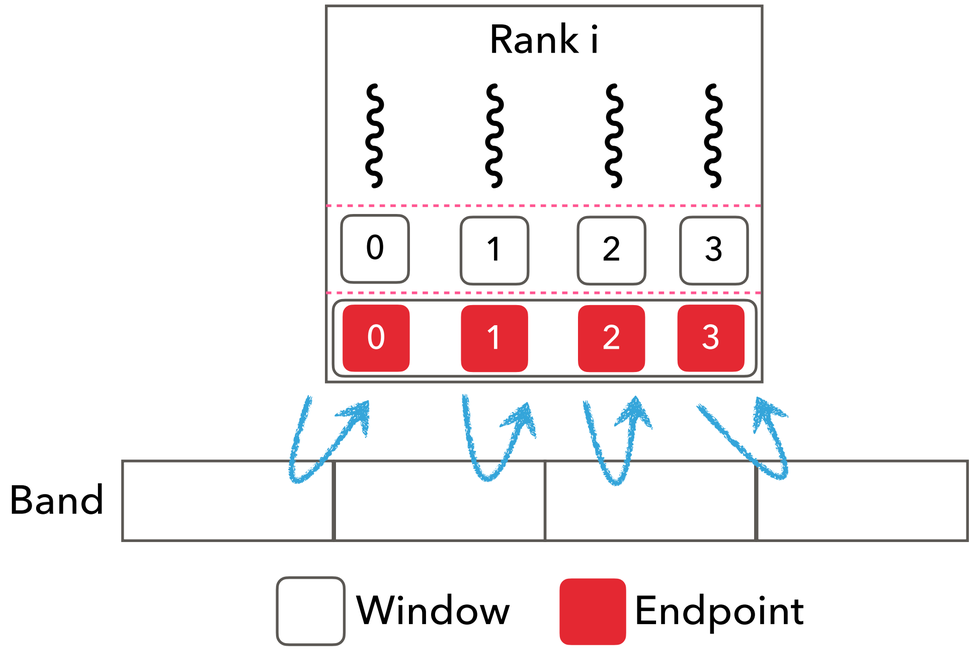


Mellanox IB implements Puts completely in hardware

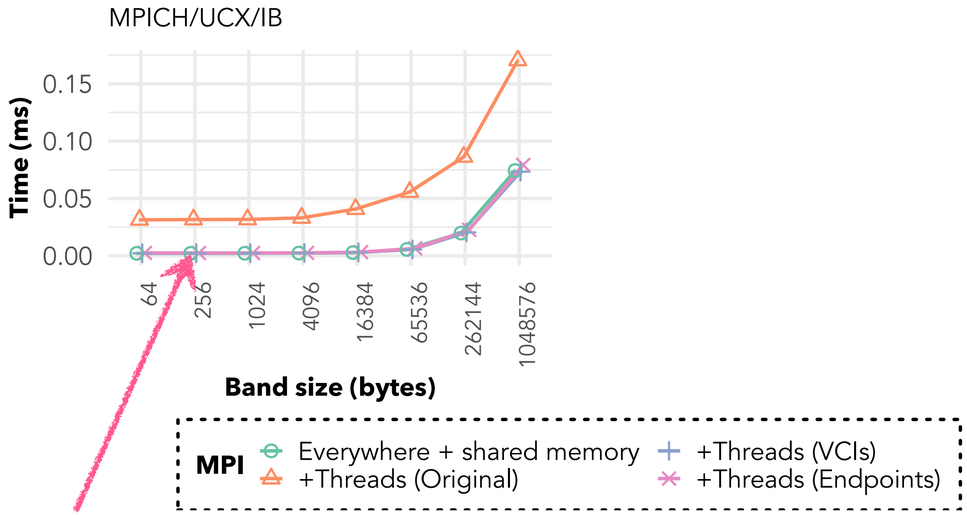
Takeaway: When shared progress is required, neither VCIs nor endpoints perform well

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



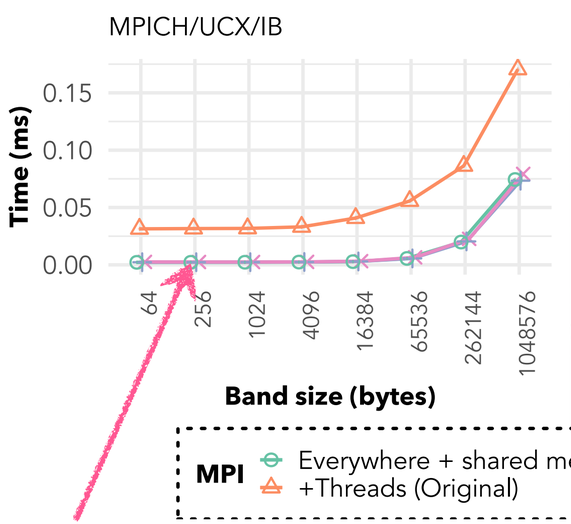
CATEGORY 2: OPENMC

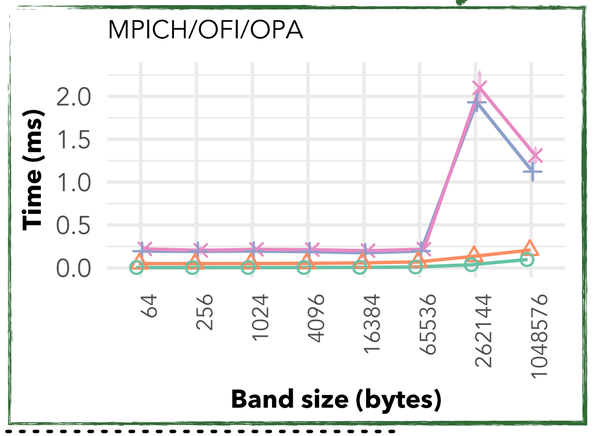


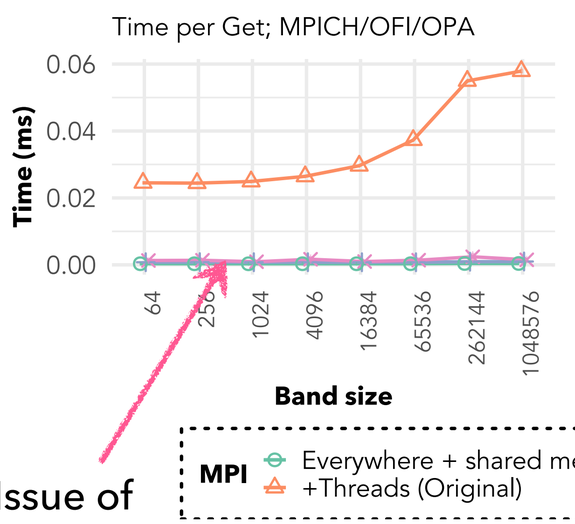
VCIs as good as endpoints and MPI everywhere when shared progress not required

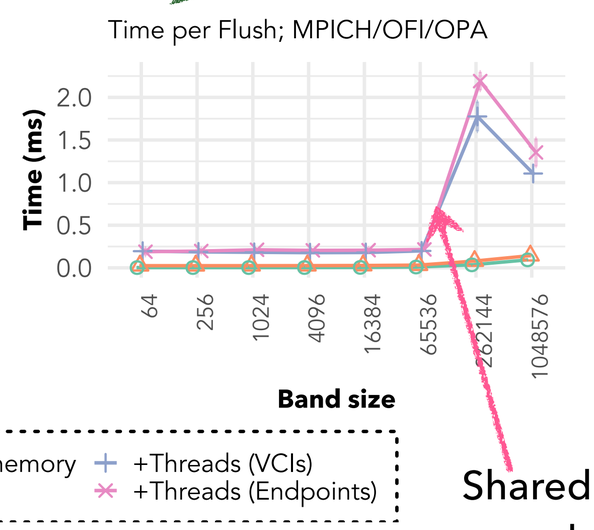
CATEGORY 2: OPENMC

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









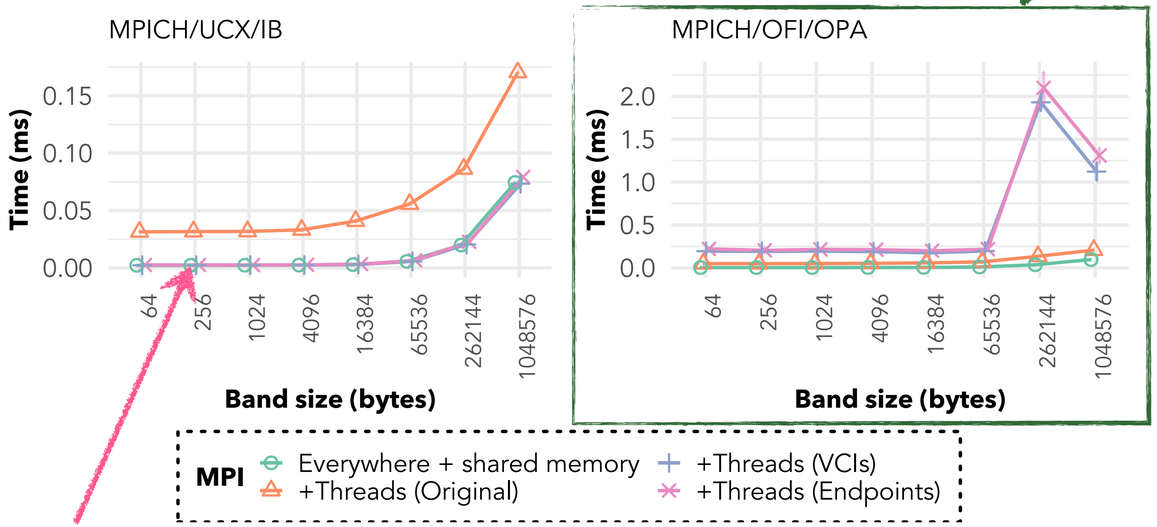
VCIs as good as endpoints and MPI everywhere when shared progress not required

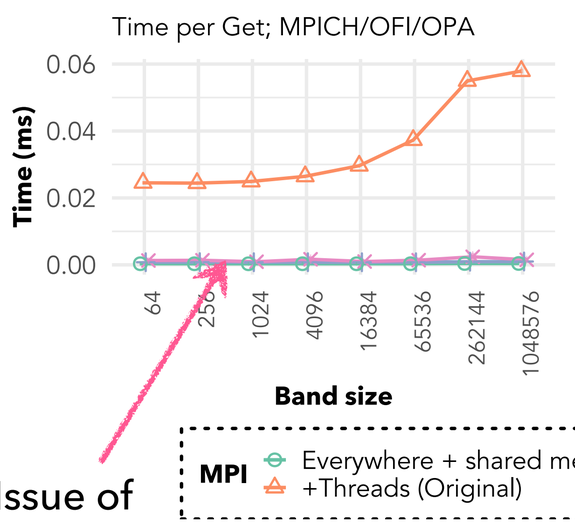
Issue of operations is fast

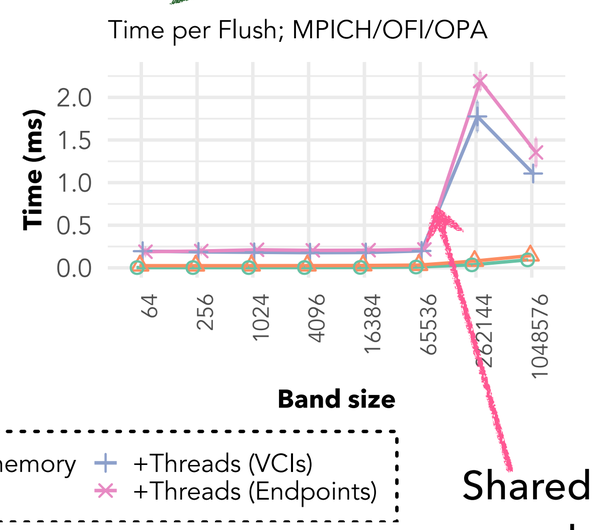
Shared progress hurts completion of operations

CATEGORY 2: OPENMC

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







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

Issue of operations is fast

Shared progress hurts completion of operations

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

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

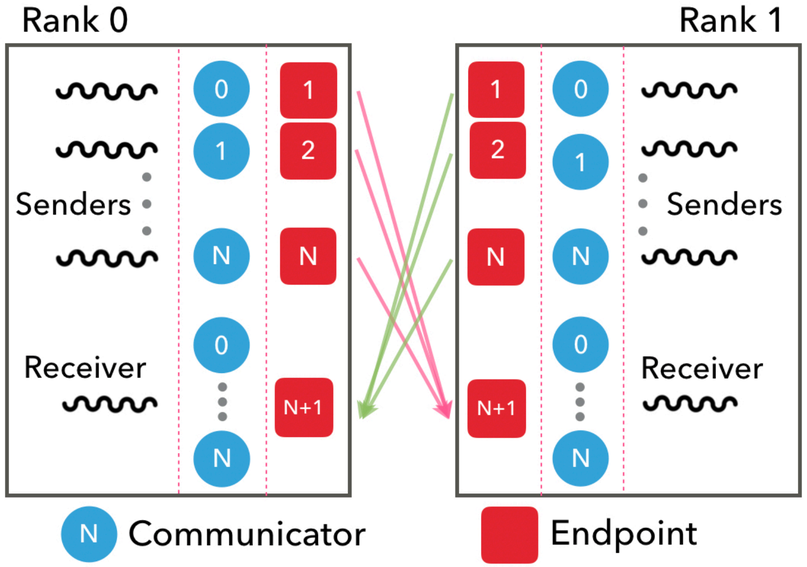
APPLICATION CATEGORIES

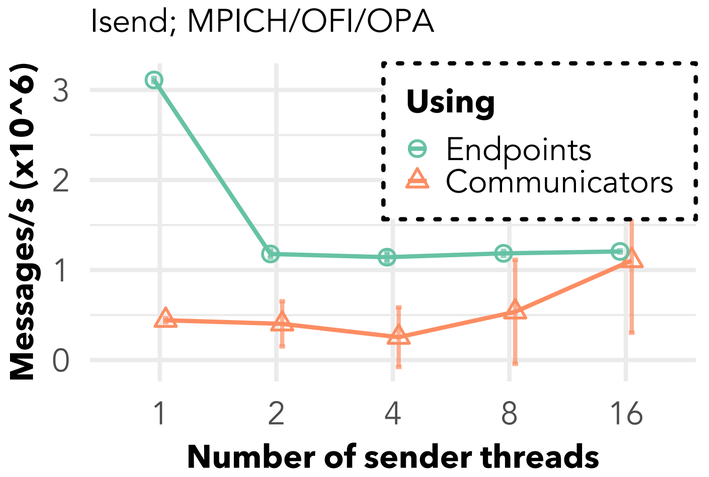
- Category 1
  - Direct use of parallel communication streams
  - VCIs as good as user-visible endpoints and MPI everywhere
- Category 2
  - Require shared progress
  - Both VCIs and user-visible endpoints perform poorly
- Category 3
  - Abstraction through MPI-3.1 prevents user from expressing parallelism
  - 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 then VCIs when MPI's semantics prevent user from expressing parallelism, especially in irregular communication patterns



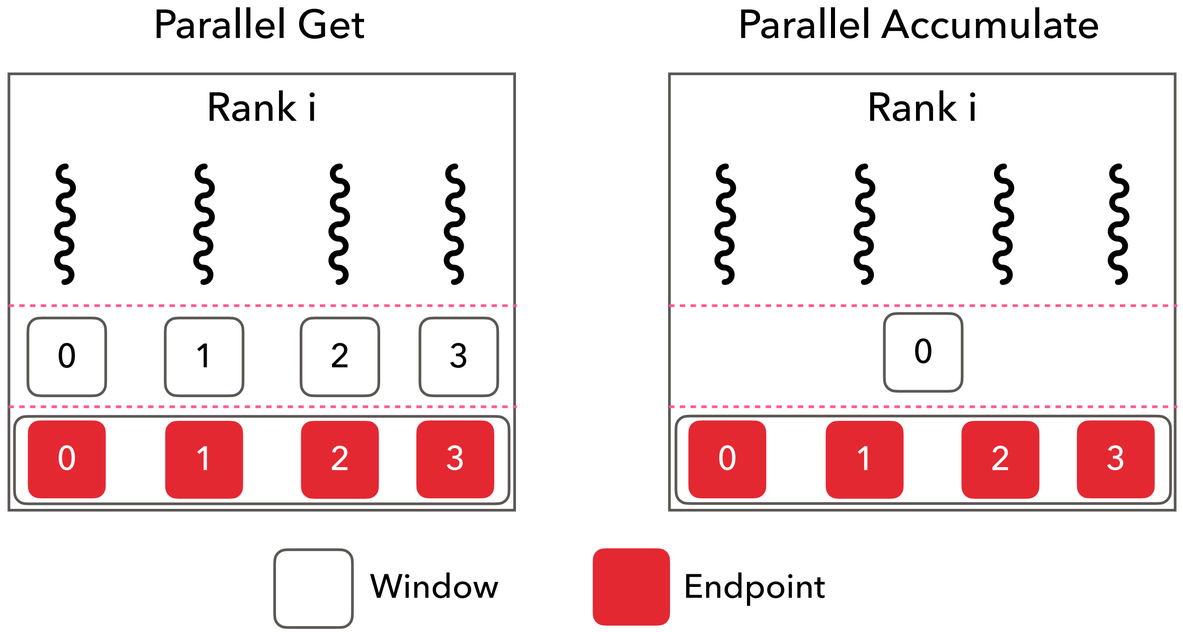


CATEGORY 3: NWCHEM

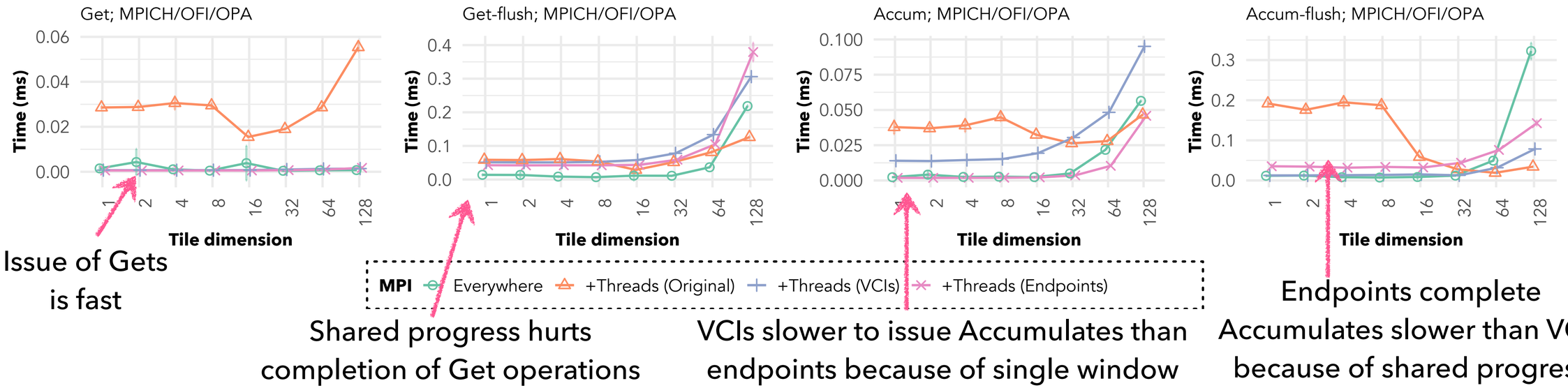
- NWChem: quantum chemistry application suite
  - Dominant cost is that of block-sparse matrix multiplication (BSPMM)
  - A x B += C get-compute-update pattern (MPI_Get + MPI_Accumulate)
  - Each worker on a node (thread or process) participates in BSPMM independently

CATEGORY 3: NWCHEM

- NWChem: quantum chemistry application suite
  - Dominant cost is that of block-sparse matrix multiplication (BSPMM)
  - A x 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



Issue of Gets is fast

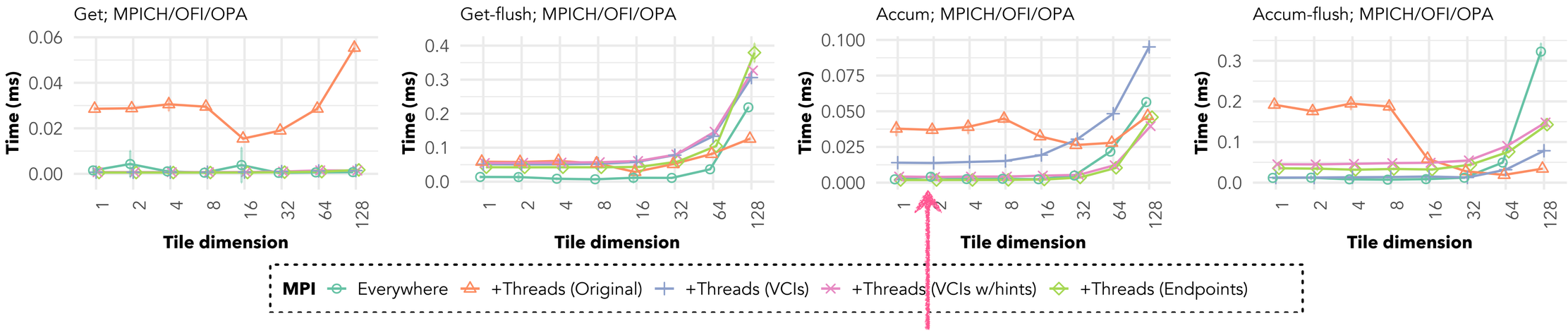
Shared progress hurts completion of Get operations

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.

CATEGORY 3: NWCHEM



VCIs with hints perform as well as Endpoints

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