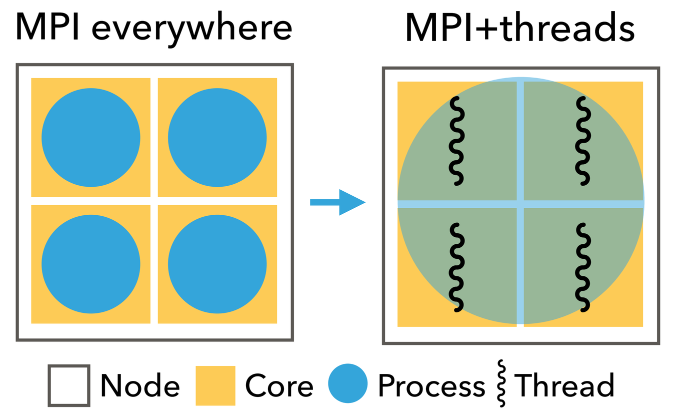
Scalable Communication Endpoints for MPI+Threads Applications

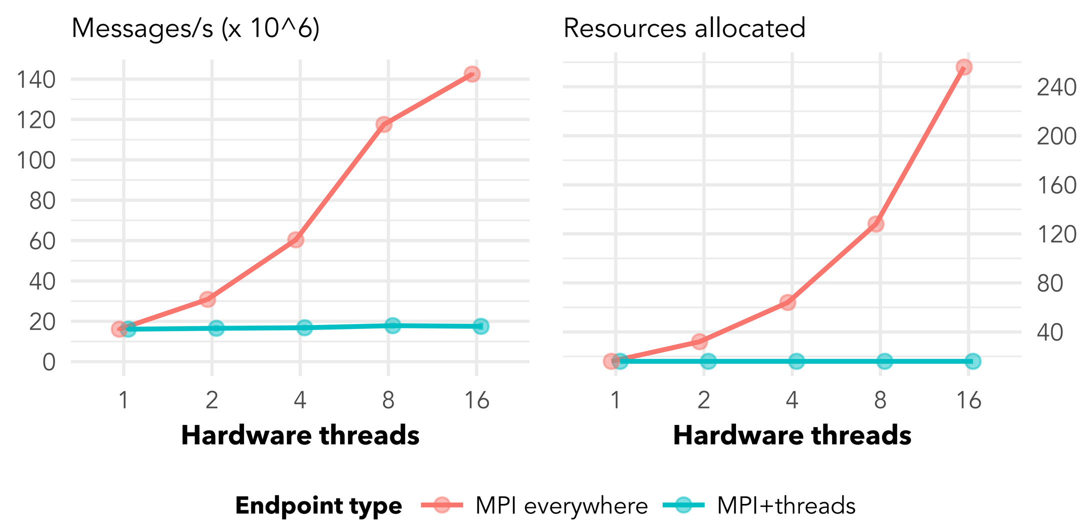
Rohit Zambre,* Aparna Chandramowlishwaran,* Pavan Balaji^
*University of California, Irvine | ^Argonne National Laboratory

Introduction

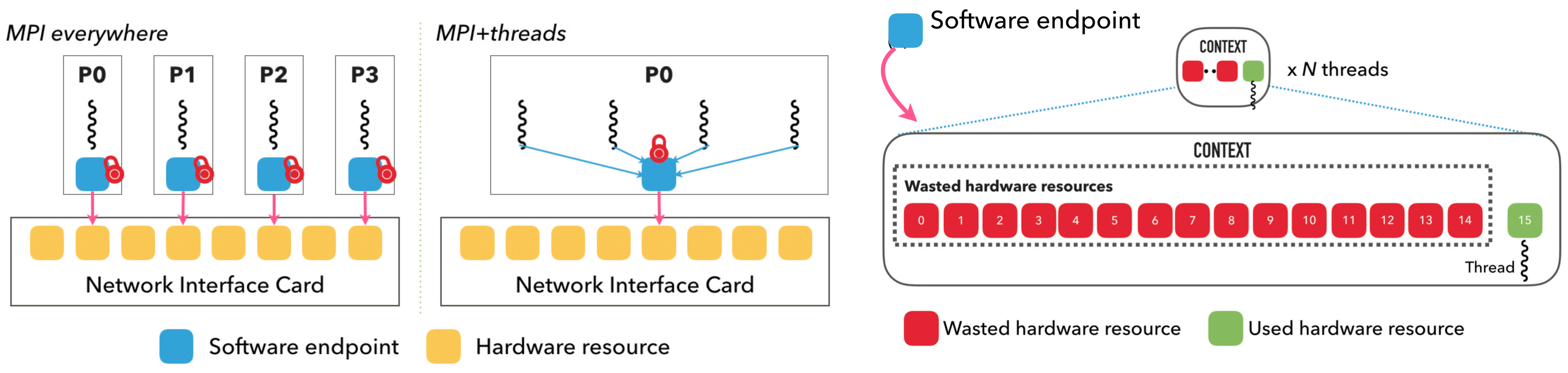


- *MPI everywhere not scalable on modern systems*
  - Disproportionate increase in number of cores compared to other on-node resources
  - Dwindling share of resources per process
  - MPI+Threads model addresses scalability issue

- *The tradeoff*
  - Communication performance of MPI+Threads is 9x worse
  - MPI everywhere uses 16x more communication resources

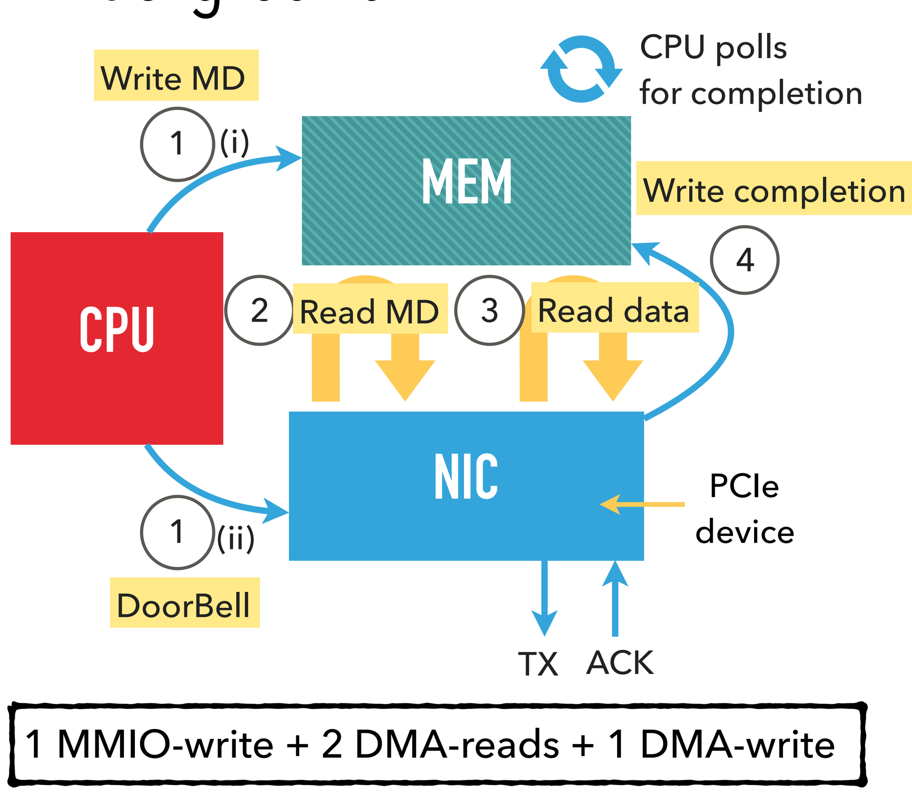


- *Why this tradeoff?*
  - Endpoint configuration in state-of-the-art MPI libraries:



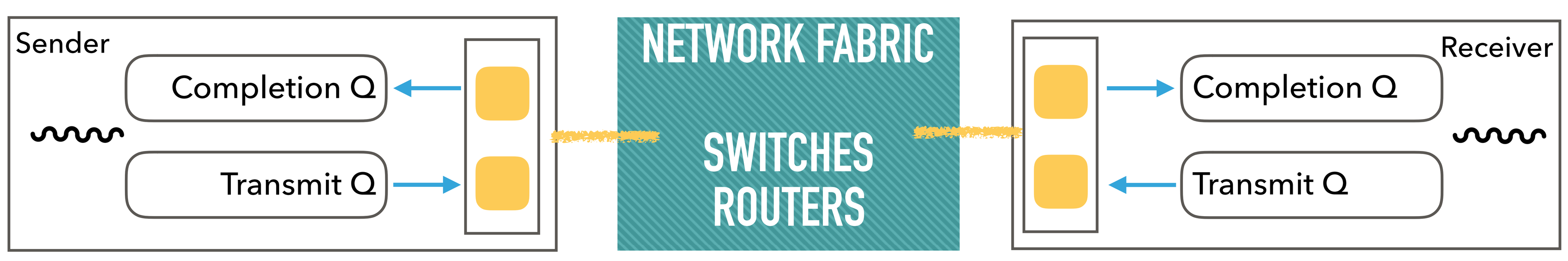
- *Naive solution for MPI+Threads: emulate MPI everywhere endpoints*
  - Leads to 93.75% wastage of limited hardware resources
  - Need a second NIC after using only 6.25% of the resources on the first
- *MPI+Threads allows for arbitrary level of sharing: what level of sharing is ideal?*
  - Depends on performance requirements and availability of resources
  - A tradeoff space between performance and sharing resources exists
- Scalable Communication Endpoints
  - A resource sharing model that concretely categorizes the tradeoff space ranging from fully independent paths to fully shared paths

Background



- *Sending 1 message*
  - (1)(i) Write a message descriptor (MD)
  - (1)(ii) CPU MMIO-writes to NIC
  - (2) NIC DMA-reads MD
  - (3) NIC DMA-reads payload
  - (4) NIC DMA-writes completion after receiving ACK from target
- *Features that help small messages*
  - Postlist: Reduces (1)(ii)
  - Unsignaled Completions: Reduces (4)
  - Inlining: Removes (3)
  - Programmed I/O: Removes (2)

Communication Resources



- *Transmit Queue*: Queue Pair (QP) in Verbs (consumes memory)
- *Completion Queue*: Completion Queue (CQ) in Verbs (consumes memory)
- *Hardware resource*: micro User Access Region (uUAR) within UAR pages on Mellanox InfiniBand (consumes hardware resources)
- *Naive solution impacts memory and hardware resource usage*
  - Memory: Creating 16 naive endpoints will occupy 5.15 MB
  - Not of immediate concern; memory on supercomputers in the order of GB
  - Hardware resources: much smaller limit than that of memory in general
  - Max of 16K uUARs on ConnectX-4 (1021 naive endpoints); max of 160 HW contexts on Omni-Path

Bytes used by Verbs resources

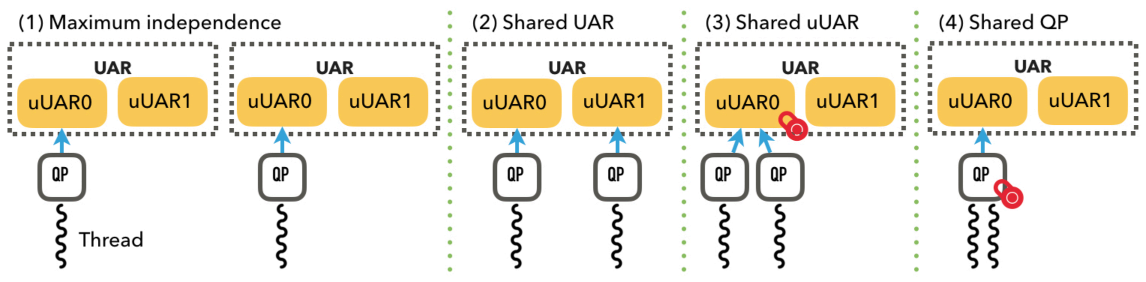
CTXs	PDs	MRs	QPs	CQs	Total
256K	144	144	80K	9K	345K

Evaluation Setup

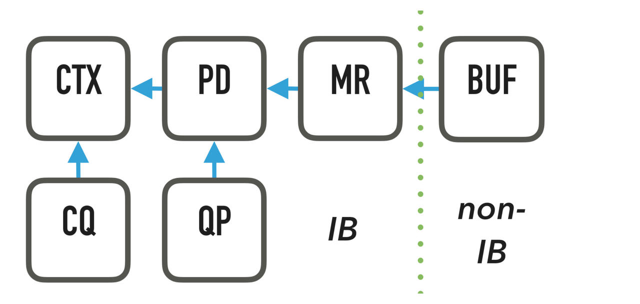
- 2 nodes with Intel Haswell (16 cores per socket) @ 2.5 GHz + Mellanox ConnectX-4 adapter on each node
- To study effect of feature *f* on multithreaded RDMA-write message rate: "All w/o *f*"
- OFED stack; QP-depth: 64; Postlist: 32; Unsignaled Completions: 64

Resource Sharing Analysis

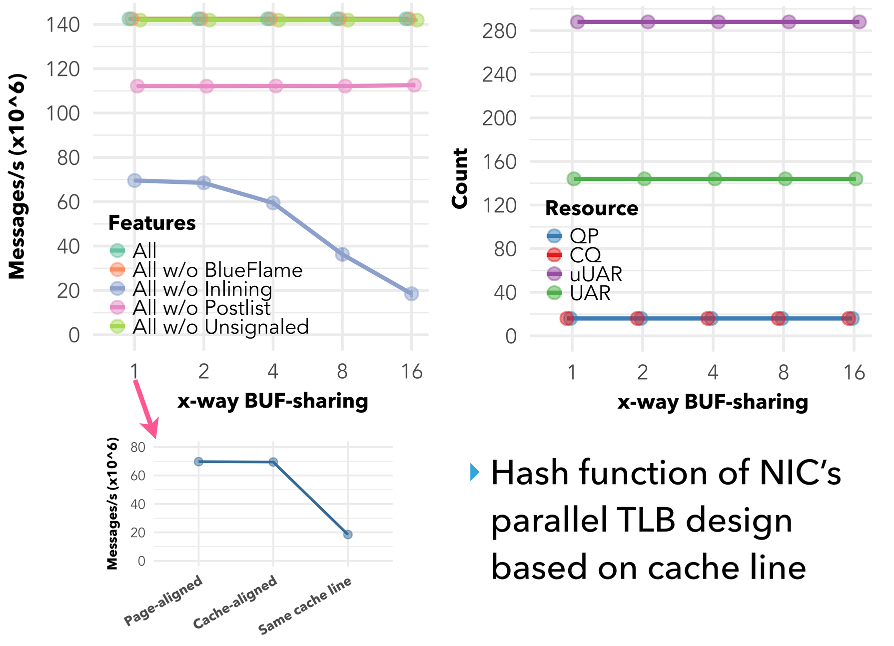
- Analytically, four levels of sharing



- Hierarchy of Verbs resources

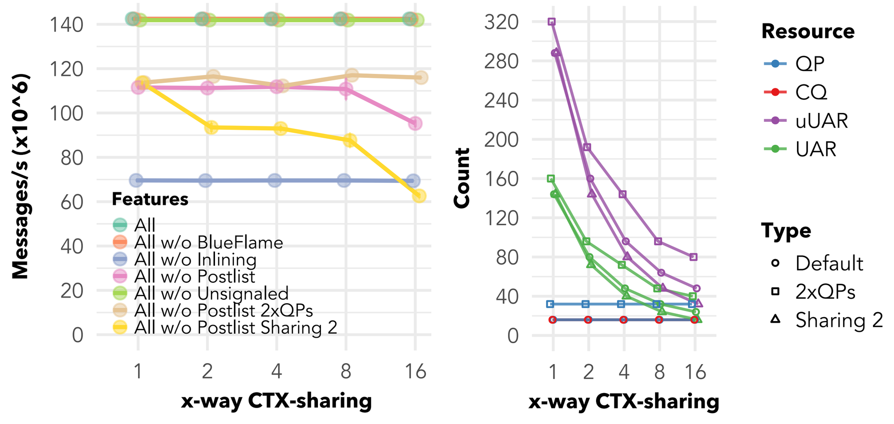


- Buffer sharing



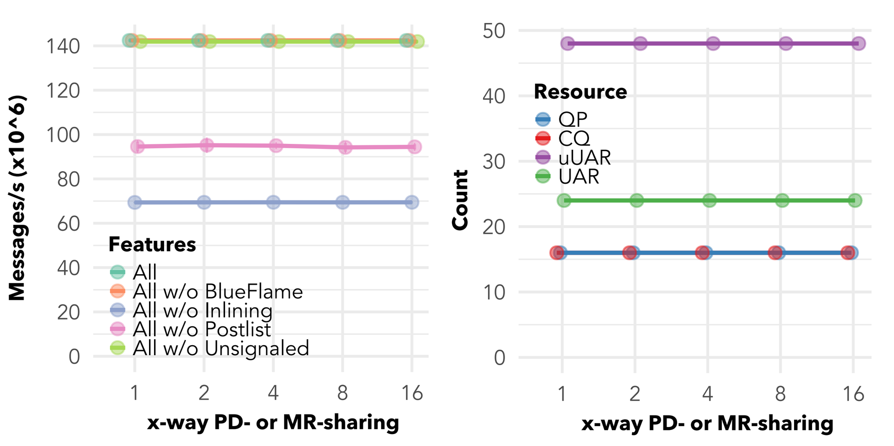
- Hash function of NIC's parallel TLB design based on cache line

- Context sharing



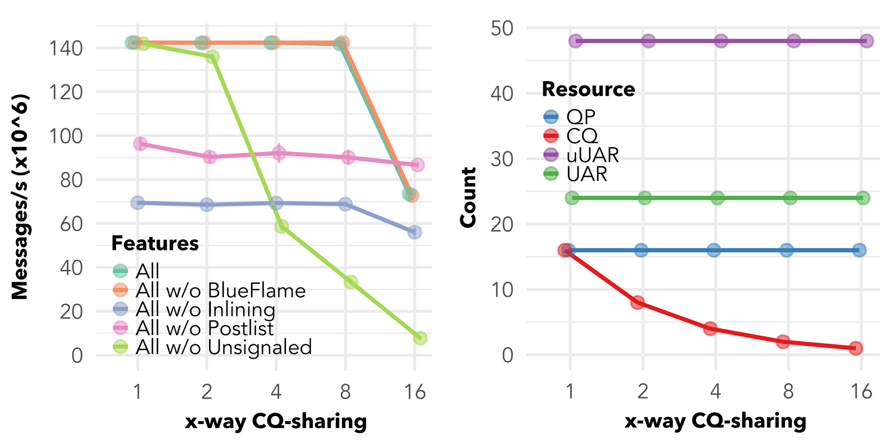
- Effects most visible with Programmed I/O
- Sharing the UAR hurts performance (yellow line)
- Sharing the Context reduces uUAR usage

- Protection Domain Memory Region sharing



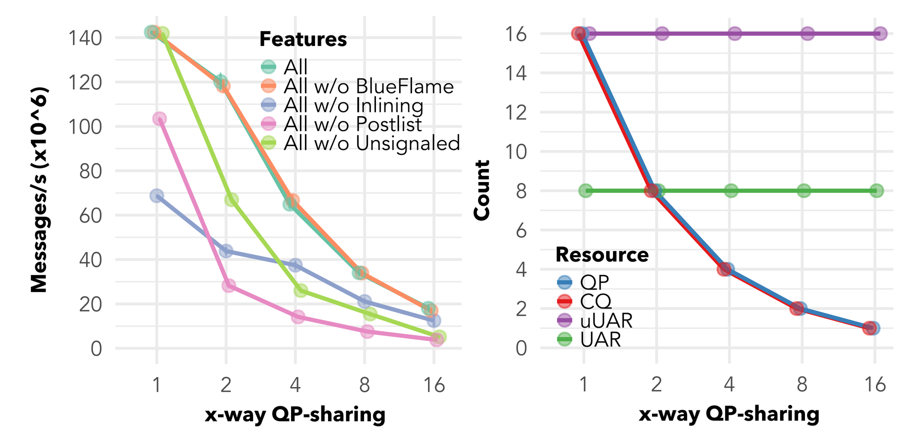
- No impact on performance or resource usage
- PD and MR are only software objects
- They are not accessed on the critical path

- Completion Queue sharing



- Lock on shared CQ affects performance
- Postlist/Unsignaled benefits VS. hurtful sharing
- Affects only memory usage

- Queue Pair sharing



- Affects performance with reduced network parallelism
- ~27x worse performance; 16x reduced memory usage

- *Each thread must have its own cache-aligned buffer*
- *Can use Protection Domain and Memory Region at will*
- *Sharing the Context most critical for hardware resource usage*
- *Only QP and CQ sharing impact memory usage*

Scalable Endpoints

- Based on analysis above, we define six categories of endpoints for N threads:

Category	Description	Performance	Hardware: UAR	Hardware: uUAR	Memory resources: QP	Memory resources: CQ
MPI everywhere	Separate Context per thread	Slightly lower than maximum	8N	16N	N	N
2xDynamic	Shared Context; 2N max. indep. Thread Domains	Maximum	8 + 2N	16 + 4N	2N	2N
Dynamic	N max. indep. Thread Domains	Lower than MPI everywhere	8 + N	16 + 2N	N	N
Shared Dynamic	N Thread Domains with Shared UAR	Lower than Dynamic	8 + ⌈N/2⌉	16 + N	N	N
Static	Statically allocated resources of Context	Depends on N	8	16	N	N
MPI+Threads	1 QP	Worst	8	16	1	1

- *Evaluation using 16 threads*
- Global array kernel
  - DGEMM
- Stencil kernel
  - 5-point stencil with 1-D partitioning

Performance decreases with increasing resource efficiency