Exploring Interprocess Work Stealing for Balanced MPI Communication

Kaiming Ouyang*, Min Si†, Zizhong Chen*
*University of California, Riverside
†Argonne National Laboratory

Introduction

❑ What is wrong with existing HPC applications?
- Due to complex network interconnection and inherent irregularity and dynamics of HPC applications, workload balance is very hard to be truly achieved among processes under large-scale parallel environment.

❑ Is it possible to balance workload dynamically?
- Apply inter-process job stealing to balance workload in MPI layer.

❑ What are possible job stealing implementation?
- POSIX-based: not flexible, impossible
- XPMEM-based: applicable, but high system call and page fault overhead

❑ **What is our solution?**
- Process-in-Process based inter-process job stealing

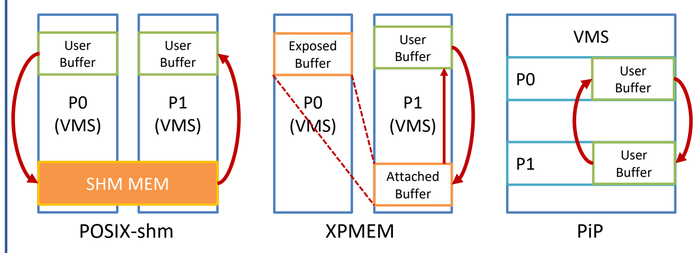
Background

❑ POSIX-shm
- Pros: portable and supported by POSIX-based operating systems
- Cons: data exchange can only be done through pre-allocated shared memory buffer making private memory access hard.

❑ XPMEM
- Pros: allow any process to access private memory of other processes; data access happens in userspace.
- Cons: incredibly high system call and page fault overhead.

❑ **Process-in-Process ☺**
- Pros: flexible data access, no system call and page fault overhead.
- Cons: page table contention during memory allocation (challenge).

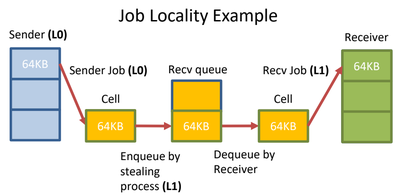
Data Access and Exchange Pattern of POSIX-shm, XPMEM and PiP

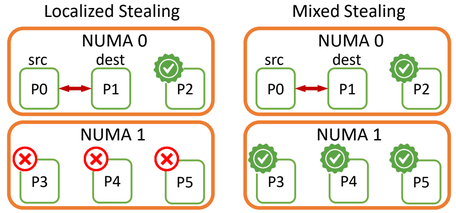


Job Stealing

❑ Which MPI reference implementation do we use?
- MPICH-3.3b3 version

❑ What is job in MPI layer?
- Job can be defined as any intra-node data movement or computation in MPI.
- For example, in intra-node P2P, sender cuts source data into 64KB chunks and copies each chunk into intermediate "cell" (64KB), and receiver copies data out of "cell" into destination buffer. **Each copy is defined as a job.**





❑ Job Locality Definition
- Sender's job locality is defined by the sender locality, and receiver's job locality is defined by enqueuing stealing process's locality.

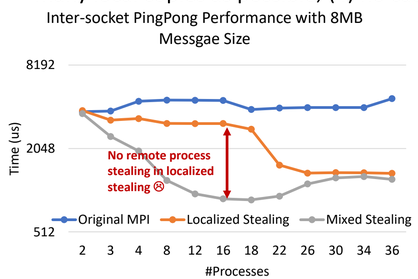
❑ Localized Job Stealing
- Job locality must match stealing processes' locality

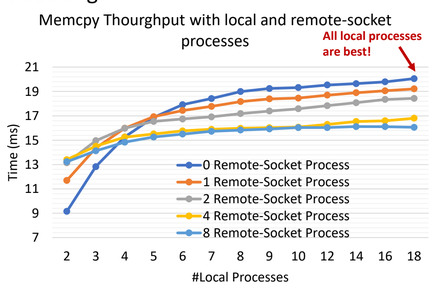
❑ Mixed Job Stealing
- Receiver's job locality and stealing processes' locality must match
- Matching for sender's job locality does not matter

Hardware Info	
CPU	2 Intel Xeon E5-2695 v4 (18 cores)
Memory	128GB

	Pros	Cons
Localized Stealing	• Every copy is local (except remote receiver) • Achieve highest memory throughput • No cache thrashing	• Cannot utilize remote idle processes for stealing
Mixed Stealing	• Able to utilize all idle processes	• Can result in lower throughput • Cache thrashing

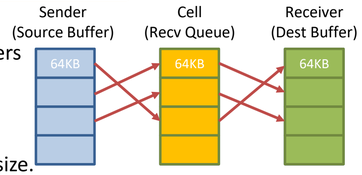
❑ Which algorithm to choose?
- **Neither of them is perfect.**
- Expected algorithm features:
  (1) utilize all idle processes; (2) achieve highest memory throughput; (3) perform as many local copies as possible; (4) no cache thrashing.





Optimizations

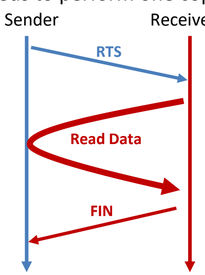
❑ Out-of-order Enqueue
- Sequentially enqueue cell into receive queue hinders the parallelism of multi-process stealing.
- We should not care about enqueue sequence.

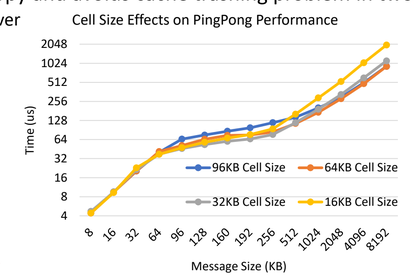


❑ Dynamic Cell Size
- In MPICH, cell size is fixed to 64KB for all message size.
- Our results show 32KB cell size is the fastest when $Msg_{size} \in [96KB, 512KB]$; 64KB cell size is the fastest when $Msg_{size} > 512KB$. Dynamically switching cell size based on message size can bring maximal performance benefits.

❑ PIP-based P2P Implementation
- MPICH provides two-copy P2P mechanism to exchange data
- PIP-based P2P only needs to perform one copy and avoids cache trashing problem in two-copy pattern

Single copy
RTS: Contains source data address
FIN: Finish packet to tell sender copy is done





Results

❑ Looks like OOO enqueue works fine?
- Not as much as we expected. After multiple iterations execution, "cell" bounces between the NUMA-node cache, which increases L3 cache misses.
- This cache thrashing will happen when message size is large (≥ 4MB) or communication is executed in many iterations sequentially.

❑ Localized Stealing works well for intra-socket PingPong but not inter-socket one.

❑ Dynamical cell optimization brings maximal 19% performance boost (at 128KB message size).