Enabling Low-Overhead Communication in Multi-threaded OpenSHMEM Applications using Contexts

Wenbin Lu
Tony Curtis
Barbara Chapman

PAW-ATM SC19, Denver, CO

November 17th, 2019

iACS INSTITUTE FOR ADVANCED COMPUTATIONAL SCIENCE

Outline

- Motivation
- The OpenSHMEM programming model
- Implementing contexts in OSSS-UCX
- Performance evaluation of multi-threaded benchmarks
- Conclusions

Trends in HPC Hardware

- Massive on-node parallelism
  - SMP & SMT
  - Accelerators

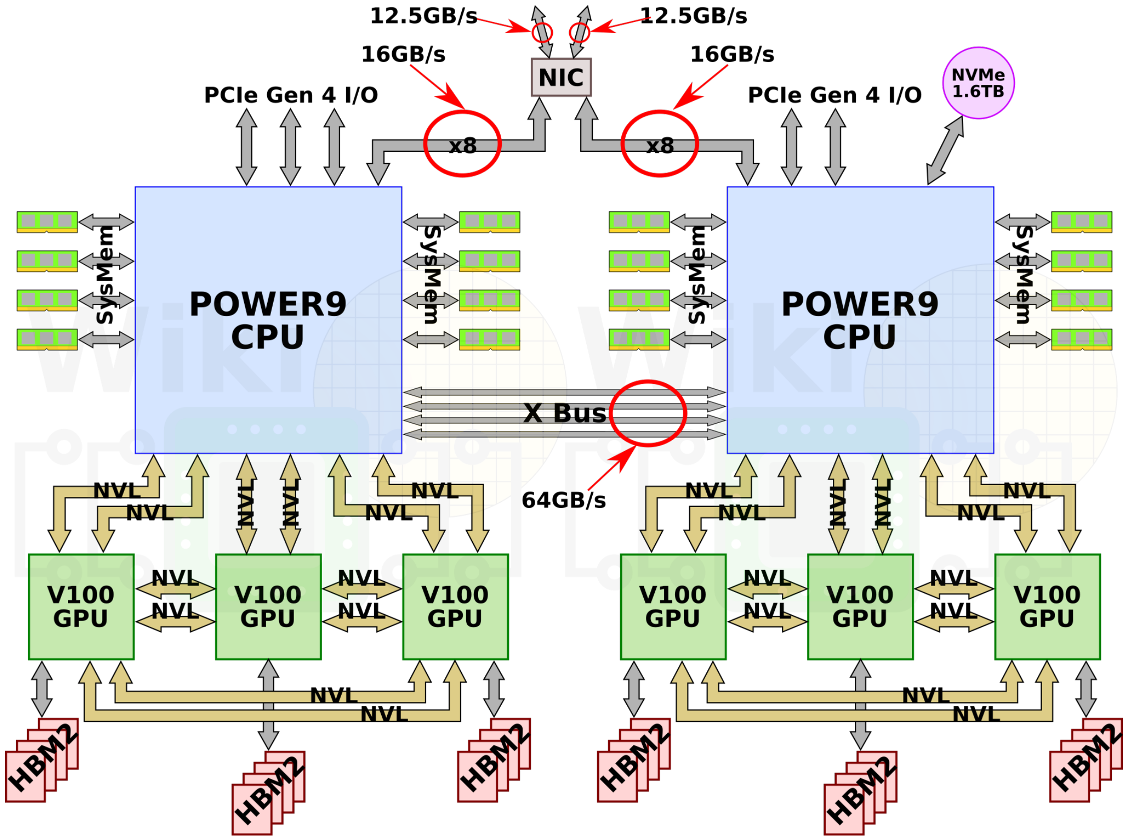


Image source: https://en.wikichip.org/wiki/supercomputers/summit

Trends in HPC Hardware

- Massive on-node parallelism
  - SMP & SMT
  - Accelerators
- Limited amount of memory per core
  - Less than 690MiB per core for A64FX
- Hardware support for efficient multi-process network access

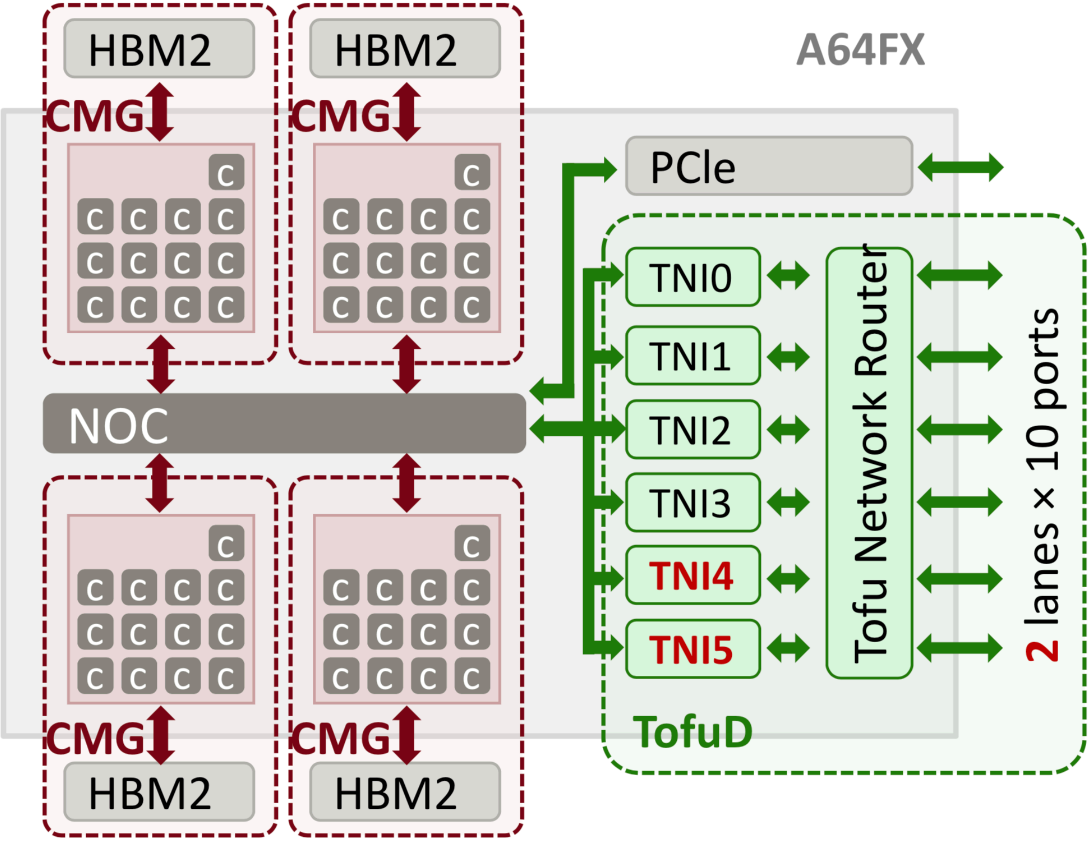
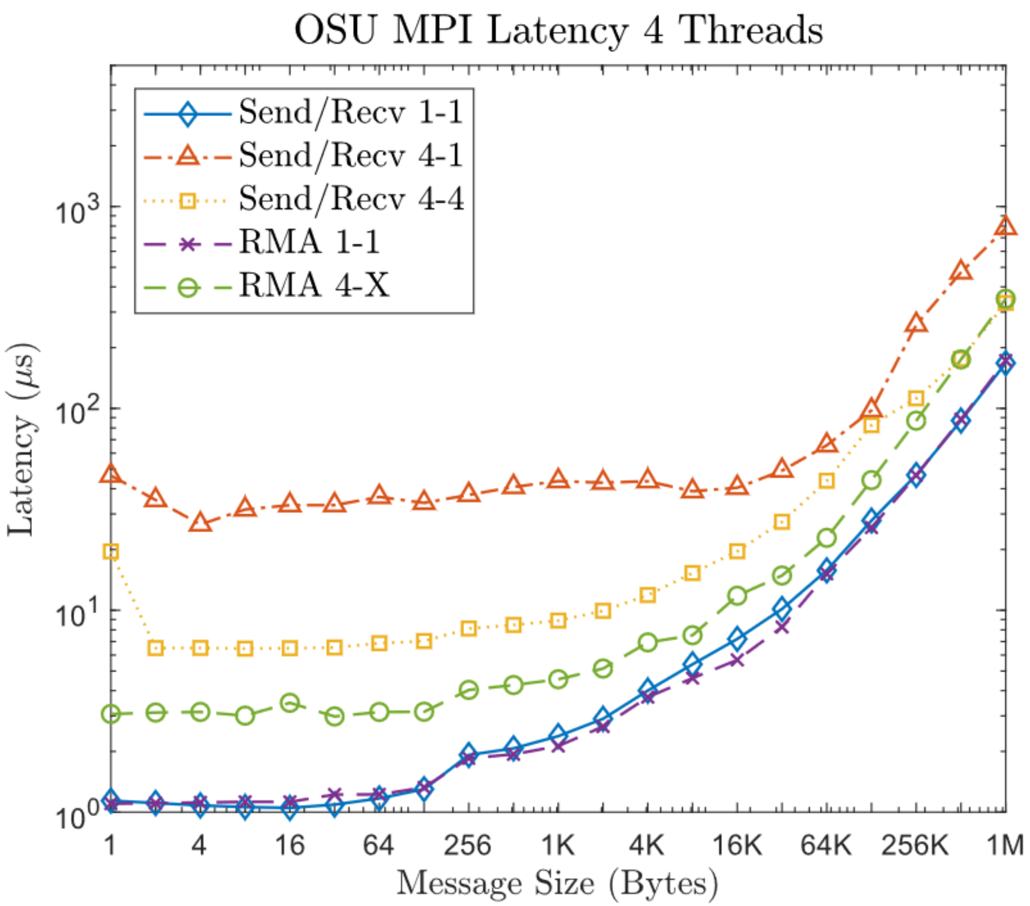


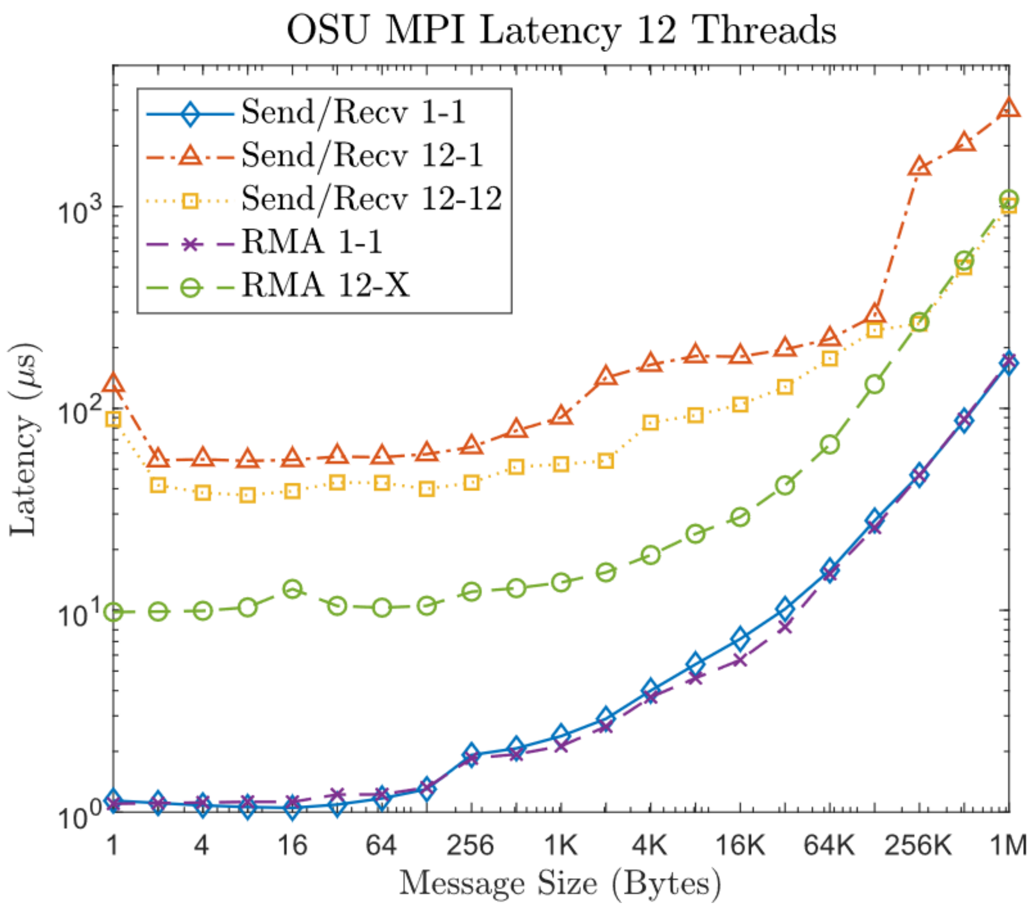
Image source: FUJITSU

Why Thread-Hybrid Programming Models

- Their advantages are more obvious on modern HPC machines
  - Faster collective operations & synchronizations
  - Simple intra-node load balancing
  - Reduced memory footprint
- MPI+threads is the most widely-adopted model
  - ***MPI_THREAD_FUNNELED/SERIALIZED***: serialized access to the MPI runtime
  - ***MPI_THREAD_MULTIPLE***: concurrent access to the MPI runtime
  - Supported by all major implementations

Performance of ***MPI_THREAD_MULTIPLE***





Lower is better

Performance Pitfalls of MPI+Threads

- MPI provides thread-safety, not thread-interoperability
  - Protecting its communication engine using coarse-grained locks
  - Threads compete for network resources, increases overhead
  - Does not utilize multi-core support in the NICs
- Existing solutions
  - Explicit: MPI Endpoints, MPI Finepoints
  - Implicit: IBM PAMI, Inria PIOMan

The OpenSHMEM Programming Model

- Library-based, SPMD
- Partitioned Global Address Space (PGAS)
- C API for RMA and collective operations
- Thread safety support modeled after MPI
- Communication contexts
  - Default context + user-created contexts
  - Represent independent streams of communication
  - Context-variant of all the point-to-point communication/synchronization APIs

Open SHMEM

OpenSHMEM without Contexts

```
// Do some computation, buffer_ptr points to the buffer that stores the result
foo1(buffer_ptr, buffer_len);

// Send the result to the master PE using a non-blocking RMA put
shmem_putmem_nbi(recv_buffer_ptr, buffer_ptr, buffer_len, master_PE_ID);

// Update counter on the master PE, and fetch the old counter value
old_tot_work = shmem_atomic_fetch_add(&work_counter, buffer_len, master_PE_ID);

// Do more work while data transfer is performed in the background
foo2(...);

// Ensure local & remote completion of the RMA put
shmem_quiet();
```

OpenSHMEM with Contexts

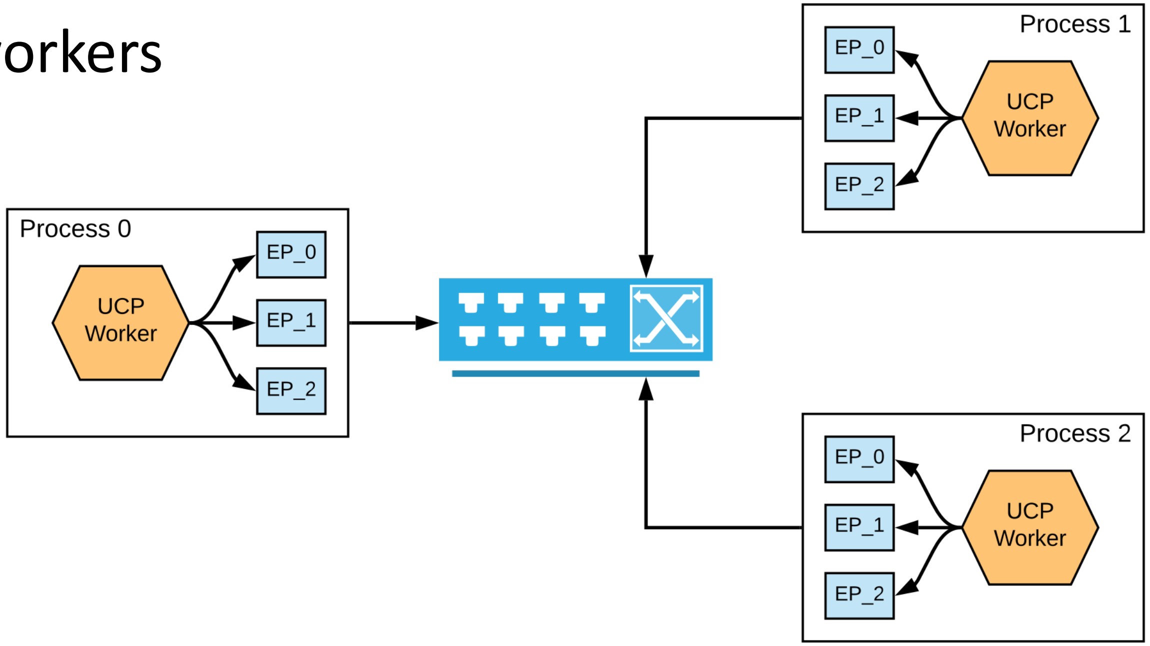
```
shmem_ctx_t ctx_1, ctx_2;
shmem_ctx_create(SHMEM_CTX_PRIVATE, &ctx_1);
shmem_ctx_create(SHMEM_CTX_PRIVATE, &ctx_2);

// Do some computation, buffer_ptr points to the buffer that stores the result
foo1(buffer_ptr, buffer_len);

// Send the result to the master PE using a non-blocking RMA put
shmem_ctx_putmem_nbi(ctx_1, recv_buffer_ptr, buffer_ptr, buffer_len, master_PE_ID);

// Update counter on the master PE, and fetch the old counter value
old_tot_work = shmem_ctx_atomic_fetch_add(ctx_2, &work_counter, buffer_len, master_PE_ID);

// Do more work while data transfer is performed in the background
foo2(...);

// Ensure local & remote completion of the RMA put
shmem_ctx_quiet(ctx_1);
```

Implementation of Contexts in OSSS-UCX

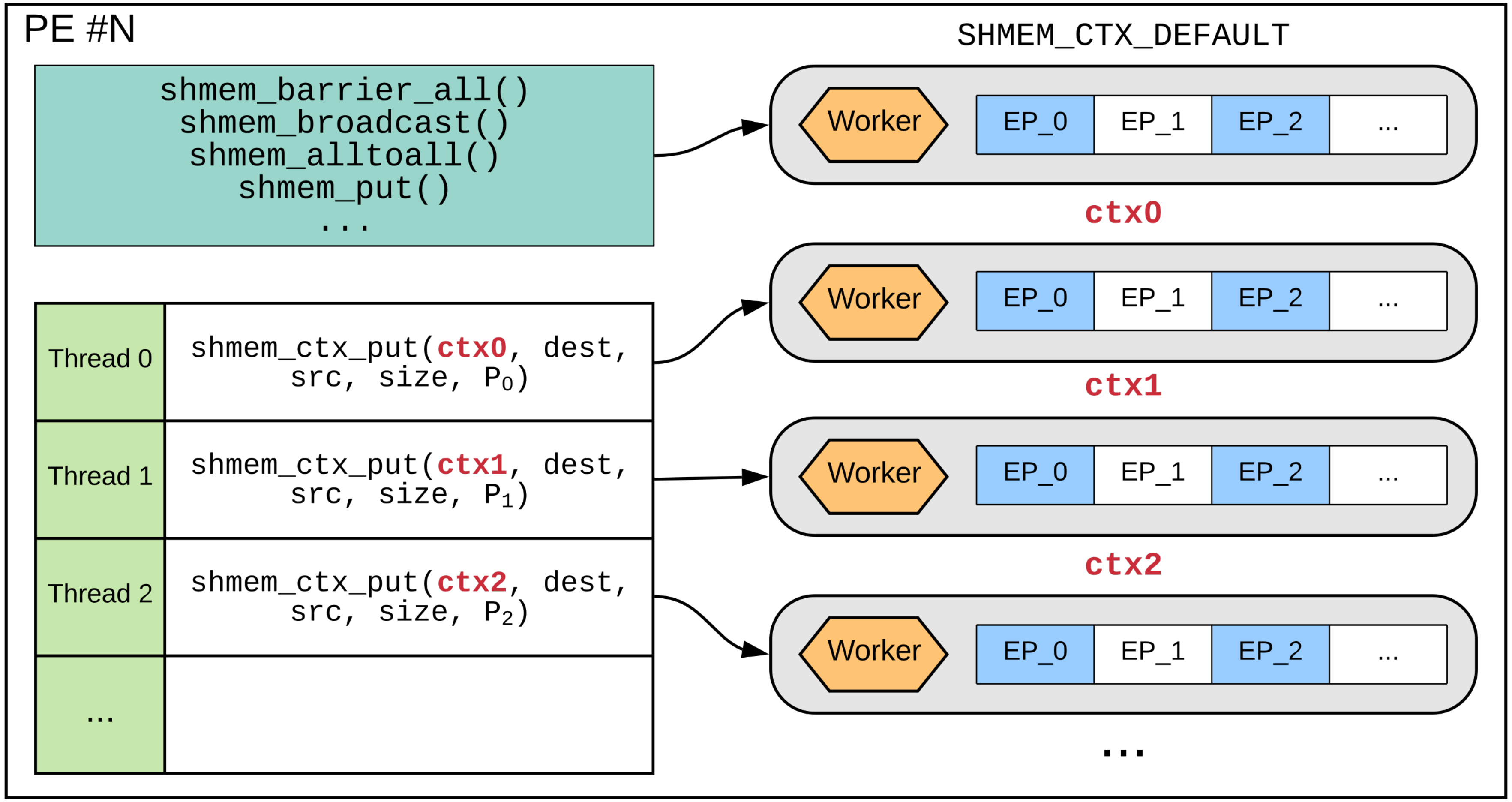
- OpenUCX: Unified Communication X
  - HPC-oriented communication library
  - Provides high-level APIs that abstracts vendor-specific network interfaces
  - https://www.openucx.org
- OSSS-UCX: OpenSHMEM reference implementation
  - Developed by our research group at the Stony Brook University
  - Uses OpenUCX as its communication backend
  - https://github.com/openshmem-org/osss-ucx

UCP Workers

- Independent actors in UCX's communication model
  - Worker = local network resource + progress engine
  - Allows concurrent access from multiple threads
  - One process can create multiple workers

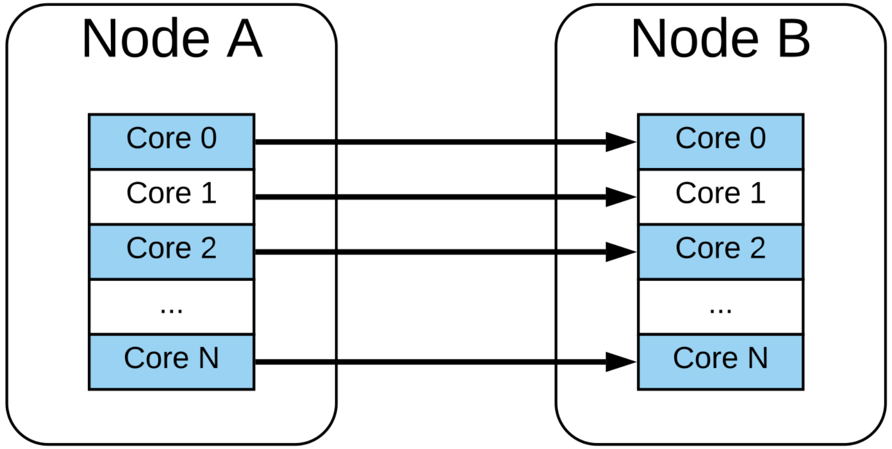


OSSS-UCX: Workers and Contexts

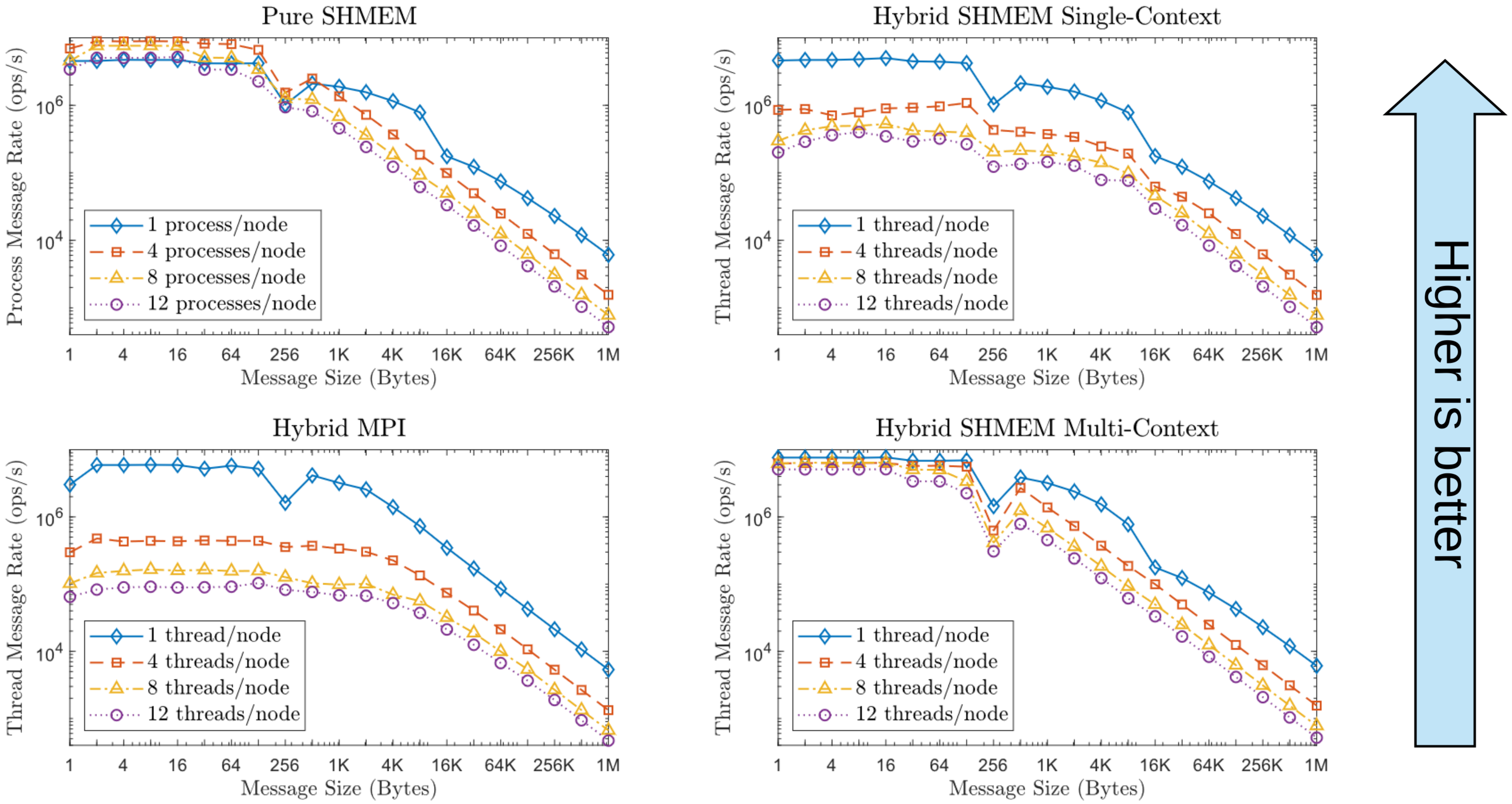


Performance Evaluation: Setup

- Hardware & Software
  - Node: dual 12-core Xeon E5-2690v3 CPUs + one Mellanox Connect-IB FDR (54.54Gb/s)
  - UCX 1.6.0
  - OSSS-UCX commit `690f54d`
  - OpenMPI 4.0.1 (with UCX backend)
- Point-to-Point micro-benchmarks
  - Each pair of processors works independently
  - Pure SHMEM vs Single-Context vs Multi-Context
- Mini-application benchmarks
  - Realistic communication patterns
  - Threads work as if they are individual PEs

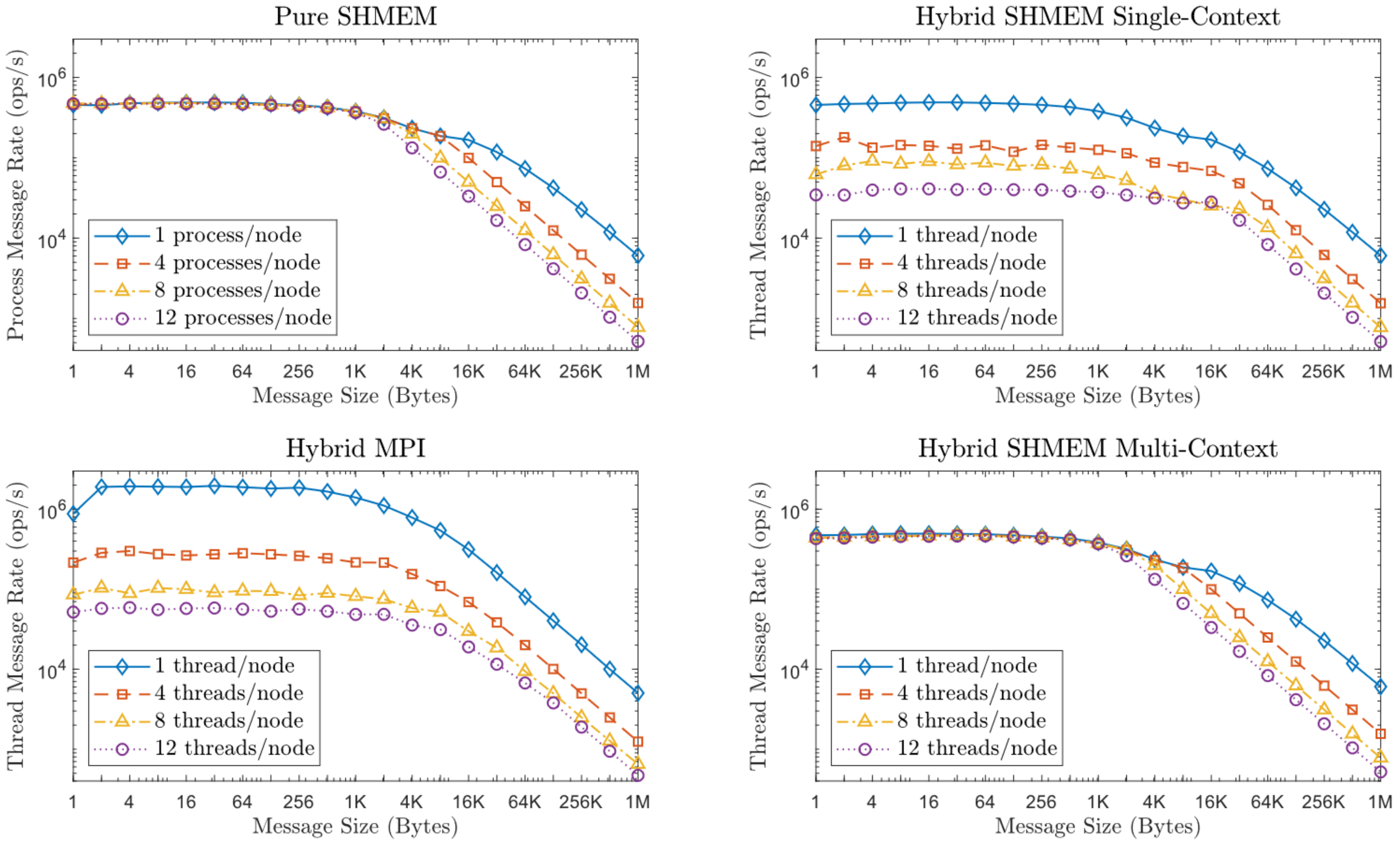


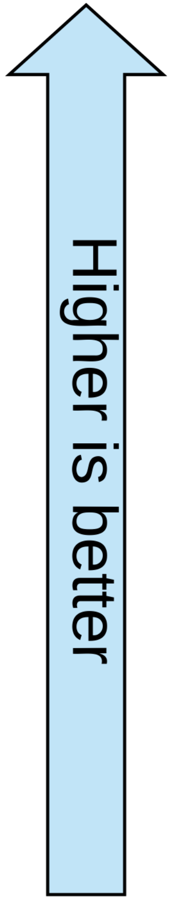
Performance Evaluation: PUT Message Rate



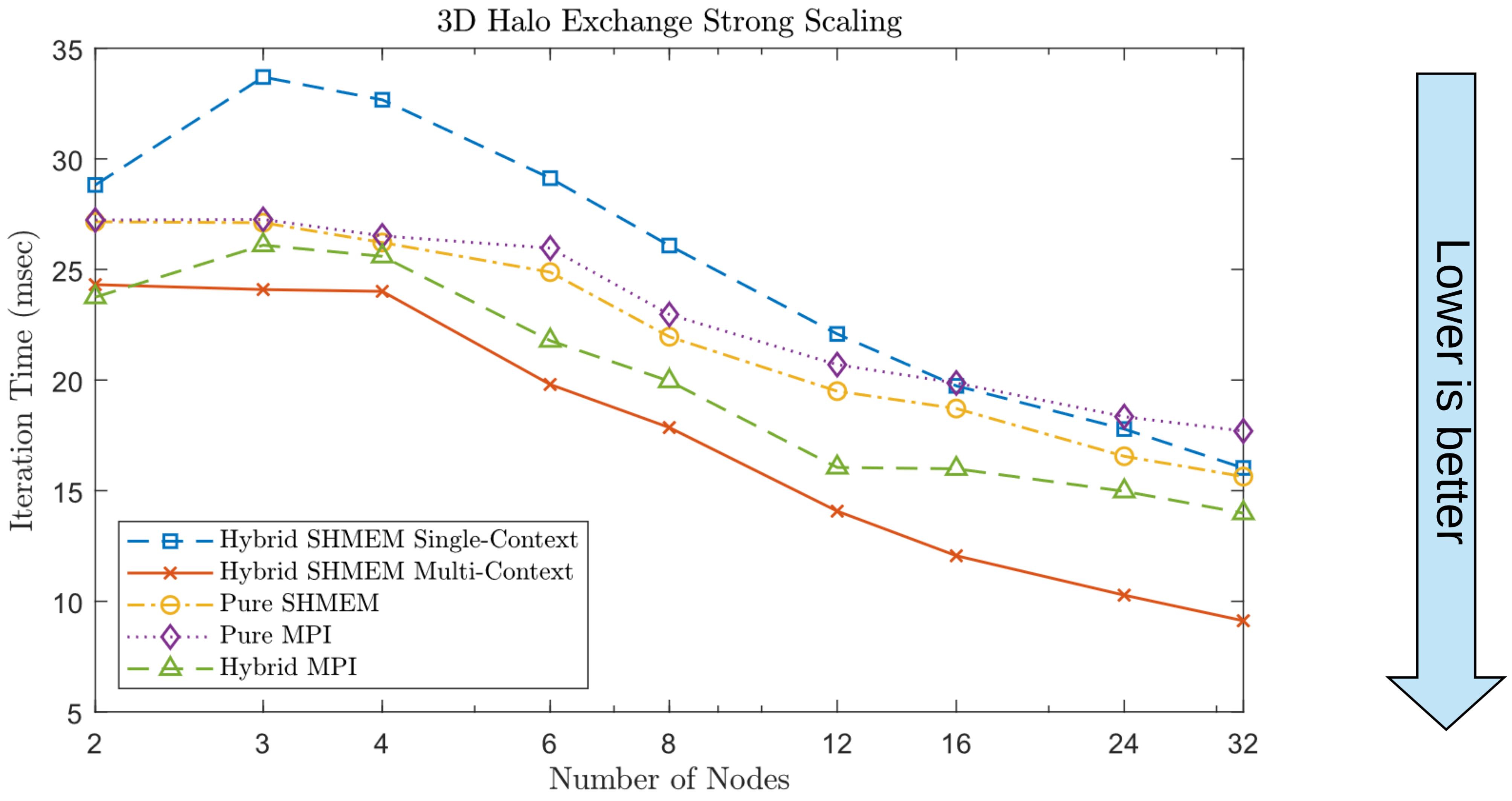
iACS INSTITUTE FOR ADVANCED COMPUTATIONAL SCIENCE

Performance Evaluation: GET Message Rate

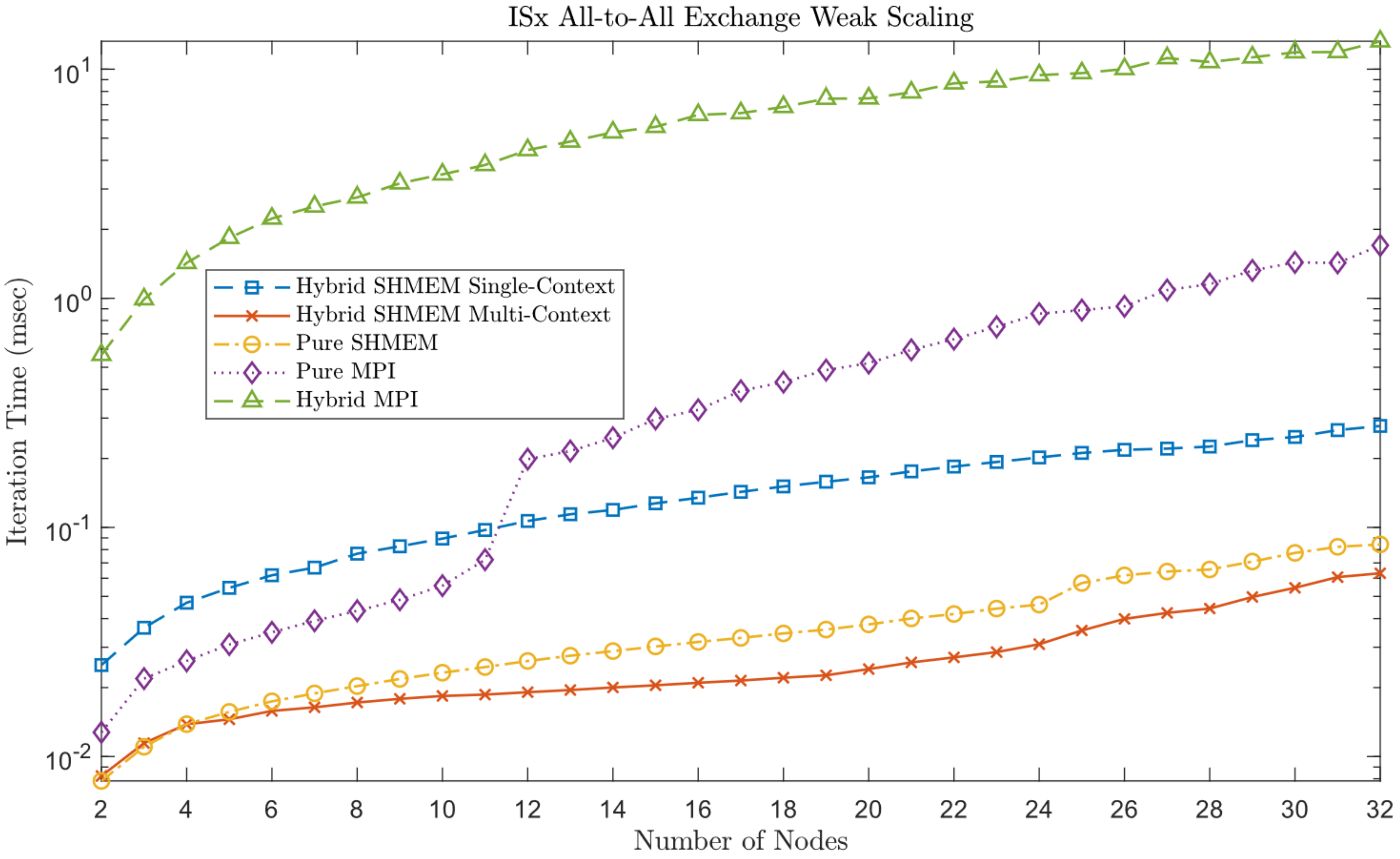


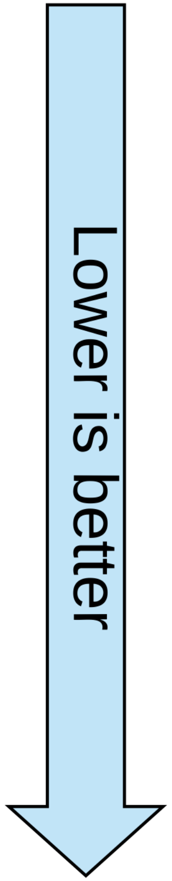


Performance Evaluation: 3D Halo Exchange



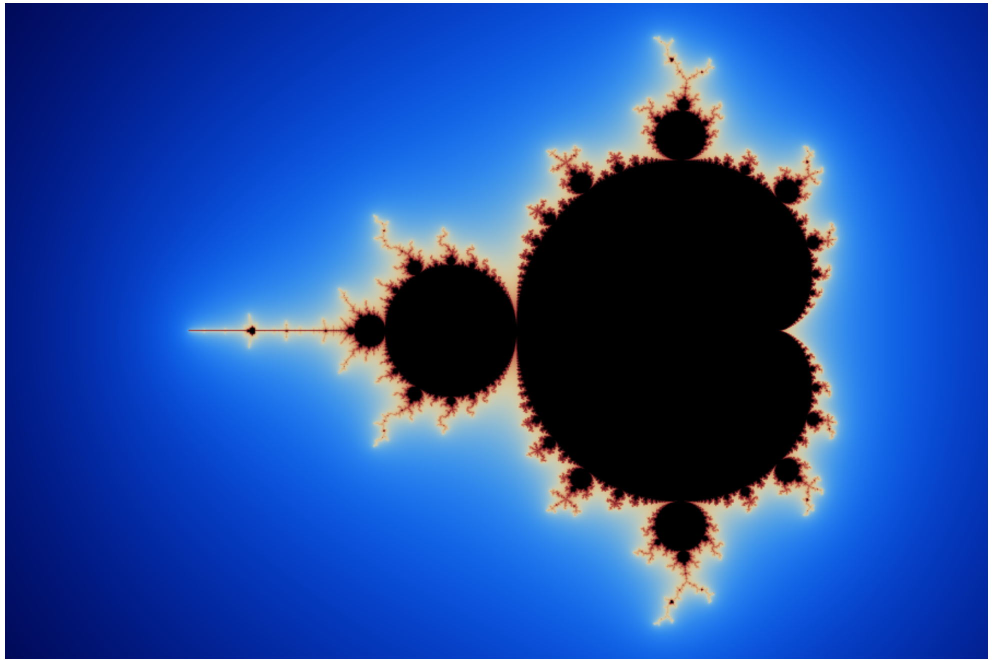
Performance Evaluation: AlltoAllv Exchange



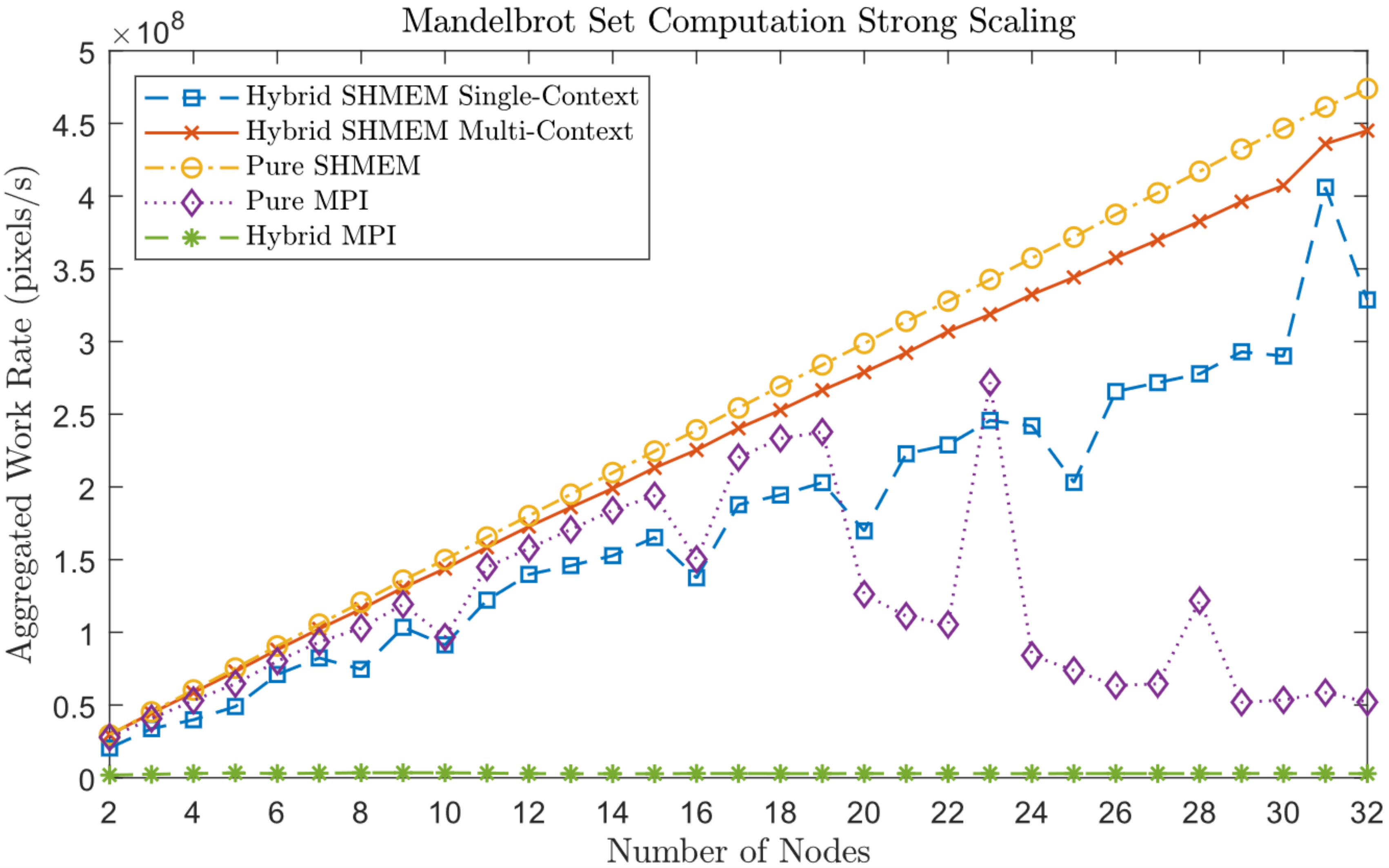


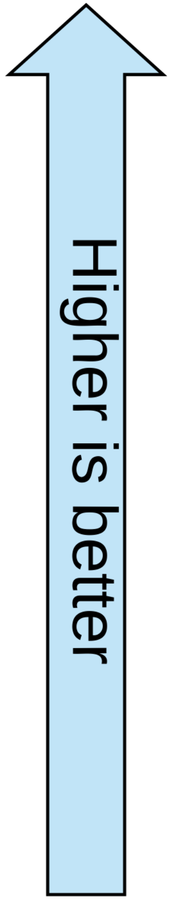
Performance Evaluation: Work-Stealing

- Compute a 3200 x 3200 image of the Mandelbrot set
- Embarrassingly parallel
- Between 0 ~ 4096 iterations for each pixel
- Round-Robin cross-PE work stealing



Performance Evaluation: Work-Stealing





Conclusions

- Interoperability leads to efficiency
- OpenSHMEM has the right abstraction for thread-hybrid applications
- An efficient implementation of contexts gives threads first-class access to the network

Thank you!