

Compiler & MPI Recommendations for CLAIX-2025

Lessons learnt from acceptance benchmarking

CLAIX-2025 Porting & Tuning Workshop
October 1 – 2, 2026

Felix Towski

Outline

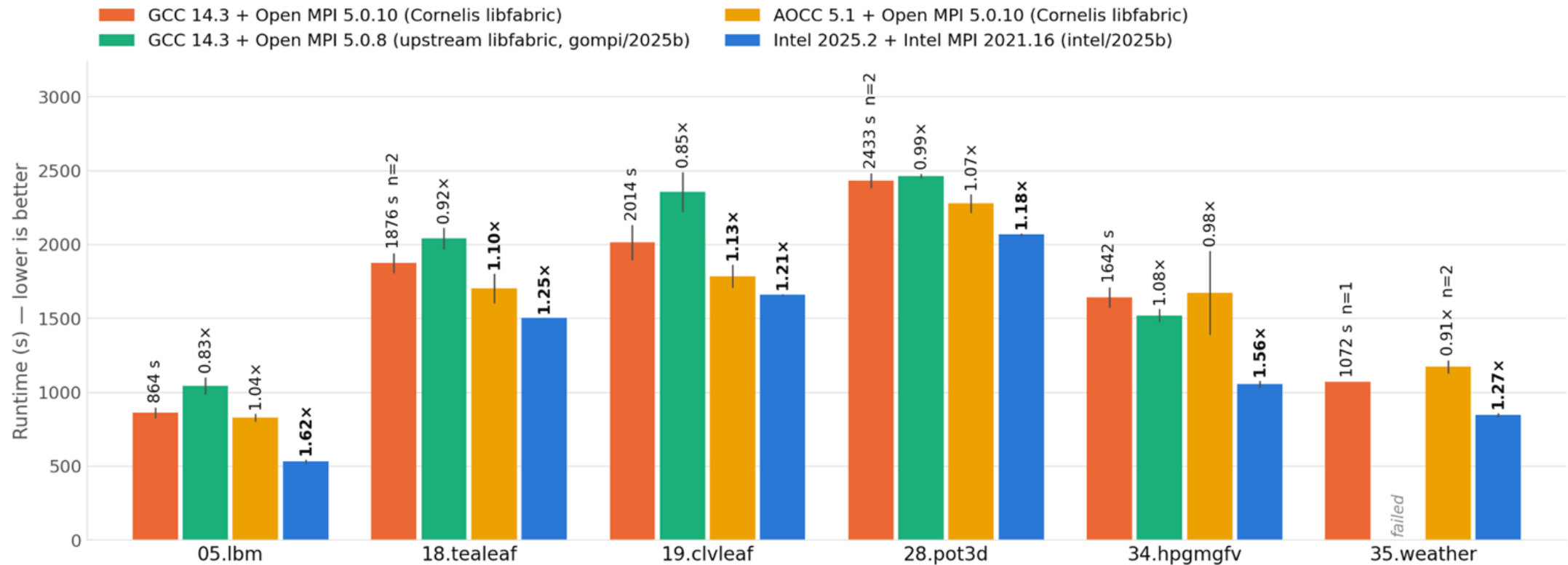
- CPU Workloads
 - MPI Performance
- GPU Workloads
 - CUDA-aware MPI
 - Multi-node ML Workloads (PyTorch, Tensorflow)
- I/O Performance

CPU Workloads

SPEChpc 2021 CPU Comparison (Compiler & MPI)

SPEChpc 2021 CPU: the Intel toolchain is fastest on every benchmark

12 nodes × 192 MPI ranks (2304 ranks), medium workload · mean of 3 runs, whiskers = std · labels: GCC + Open MPI 5.0.10 runtime, speedup vs. it

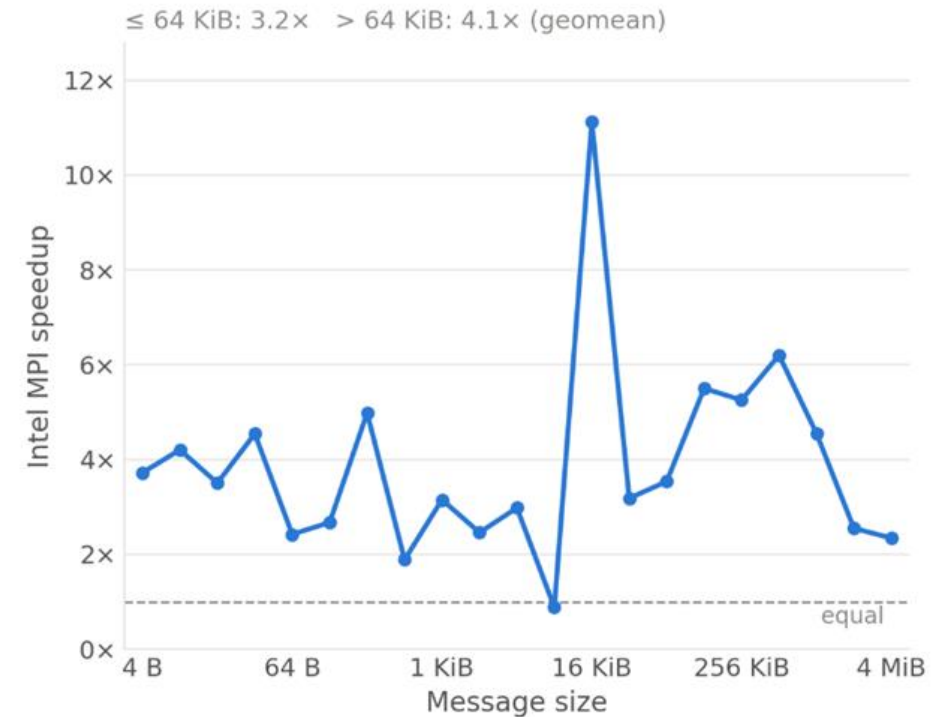
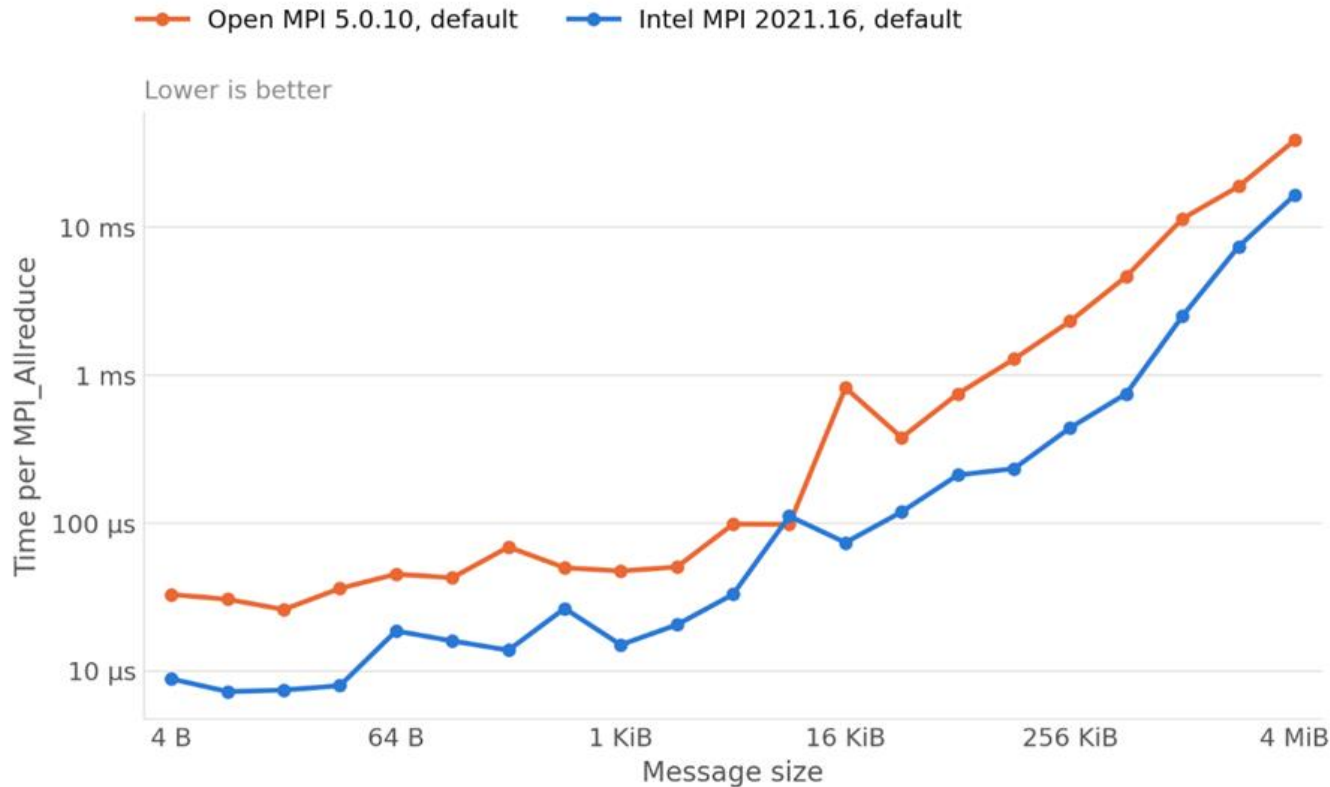


- Recommendation: use the default intel toolchain whenever possible
- Most of the speedup of the intel toolchain can be attributed to MPI performance

MPI Collective Performance: Intel MPI vs Open MPI

MPI_Allreduce: Intel MPI is 3.4× faster than Open MPI out of the box

12 nodes × 192 ranks/node (2304 ranks) · IMB-MPI1, t_{avg} · both over libfabric/OPX, no tuning

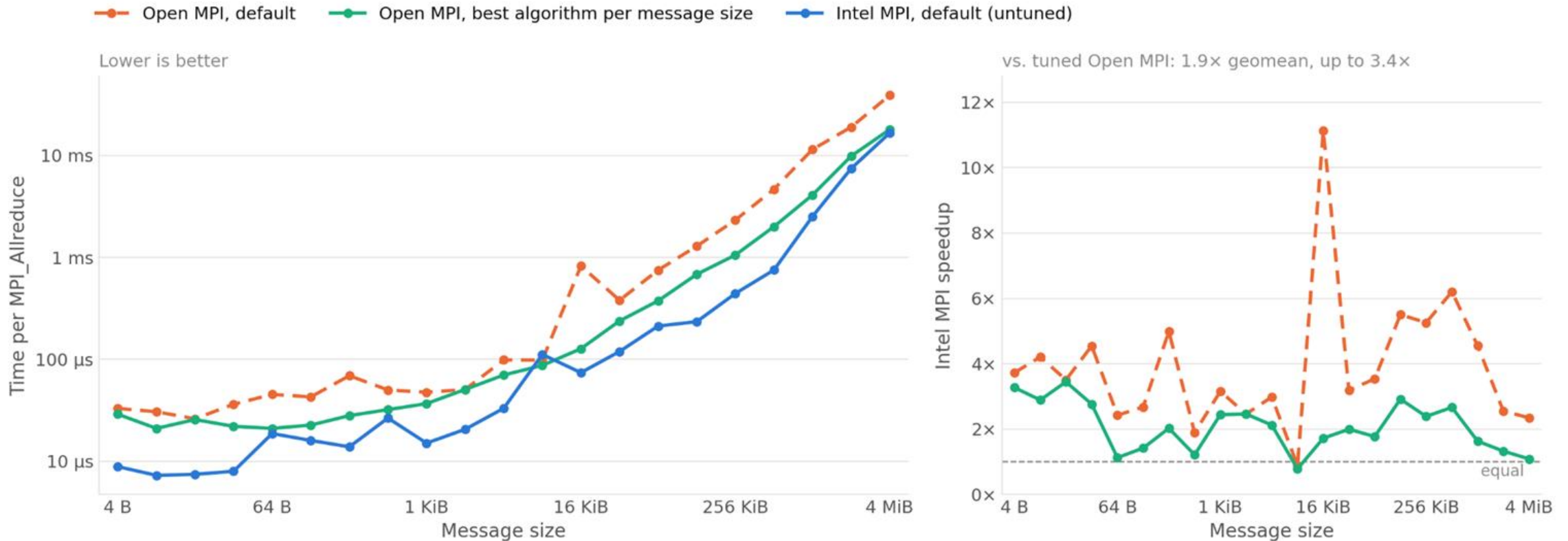


- Difference most prominent for Allreduce
- Other collectives are on-par or Intel MPI is only slightly better

MPI Collective Performance: Tuning

Even hand-tuned, Open MPI stays behind: Intel MPI still 1.9× faster

12 nodes × 192 ranks/node (2304 ranks) · IMB-MPI1, t_{avg} · Open MPI: best of 8 forced coll_tuned Allreduce algorithms and the default, per message size

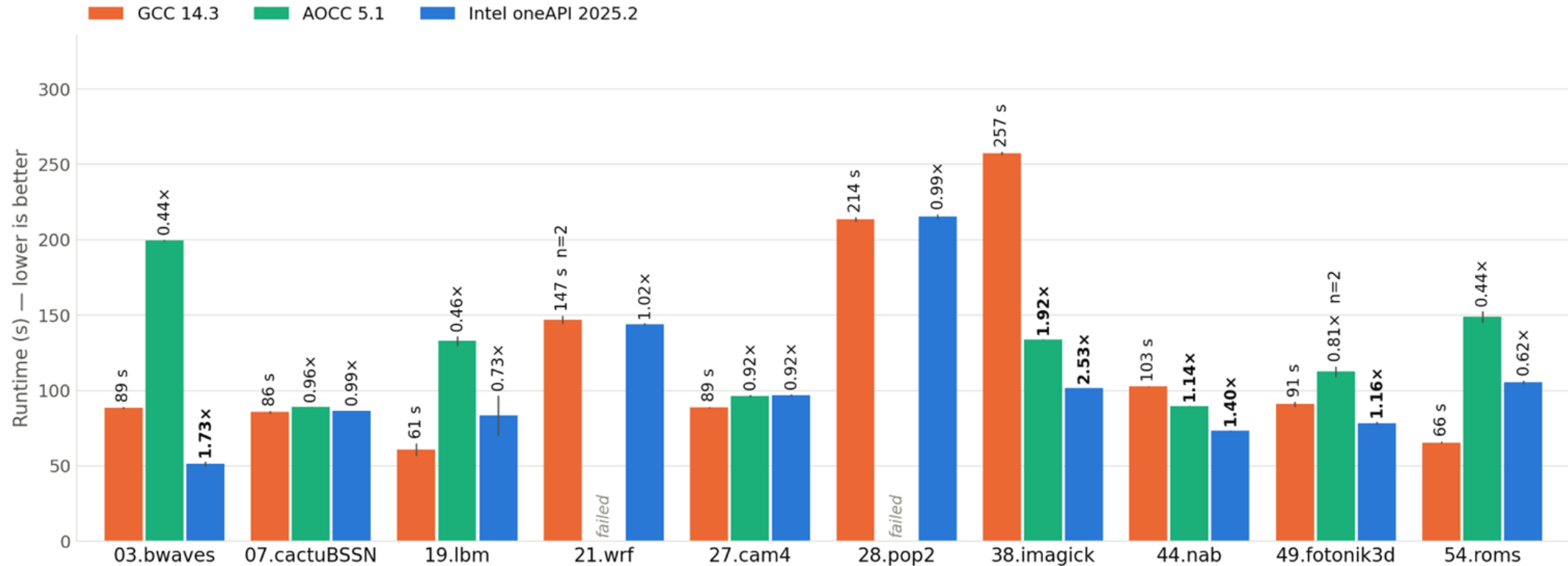


- Recommendations
 - Use Intel MPI when using MPI collectives
 - If restricted to Open MPI, tuning is worth it (we will configure it in the future)

SPEC CPU 2017 Compiler Comparison

SPEC CPU 2017: no single fastest compiler (24 threads)

OpenMP speed suite, 1 process x 24 threads on one AMD CPU node · mean of 3 runs, whiskers = std · labels: GCC runtime, speedup vs. GCC

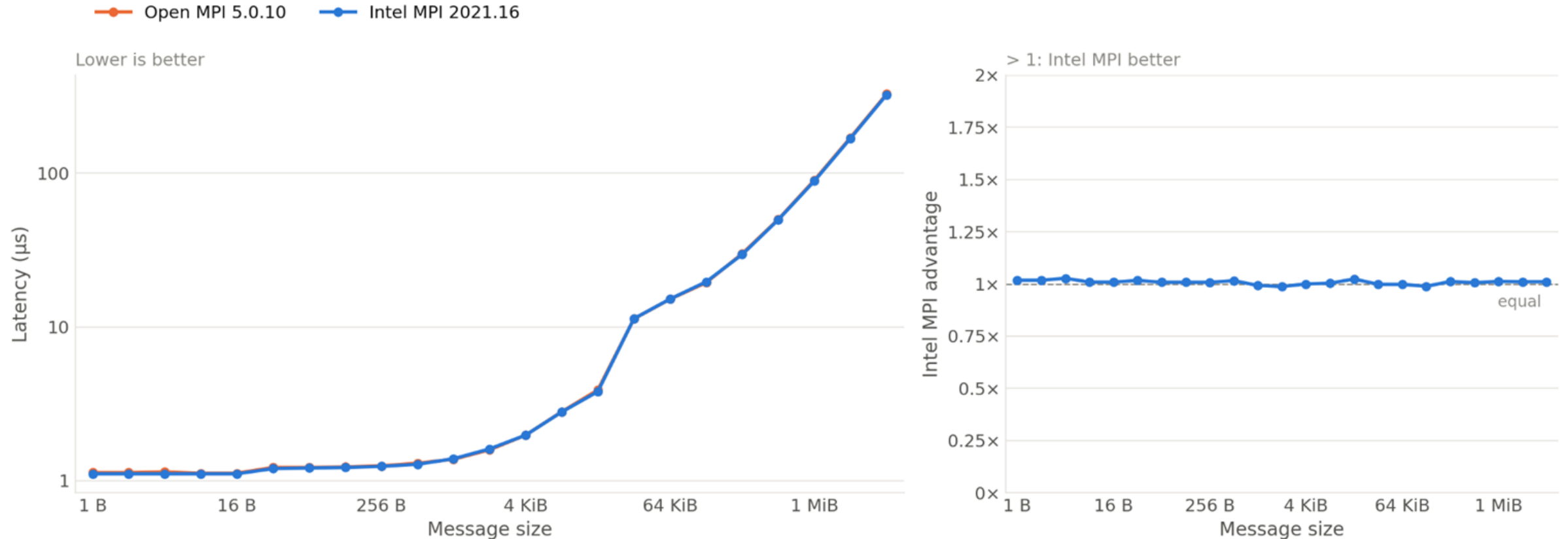


- Recommendations
 - Intel compilers are safe default, also works with Intel MPI out of the box (incl. Fortran)
 - For your own code: manually test which compiler delivers best performance

MPI Point-to-Point Latency: Intel MPI vs Open MPI

Point-to-point latency is the same with both MPIs: 1.11 μ s Intel MPI, 1.13 μ s Open MPI

osu_latency · 2 nodes, 1 rank per node pinned to the HFI's NUMA domain (cores 72-95) · both over libfabric/OPX

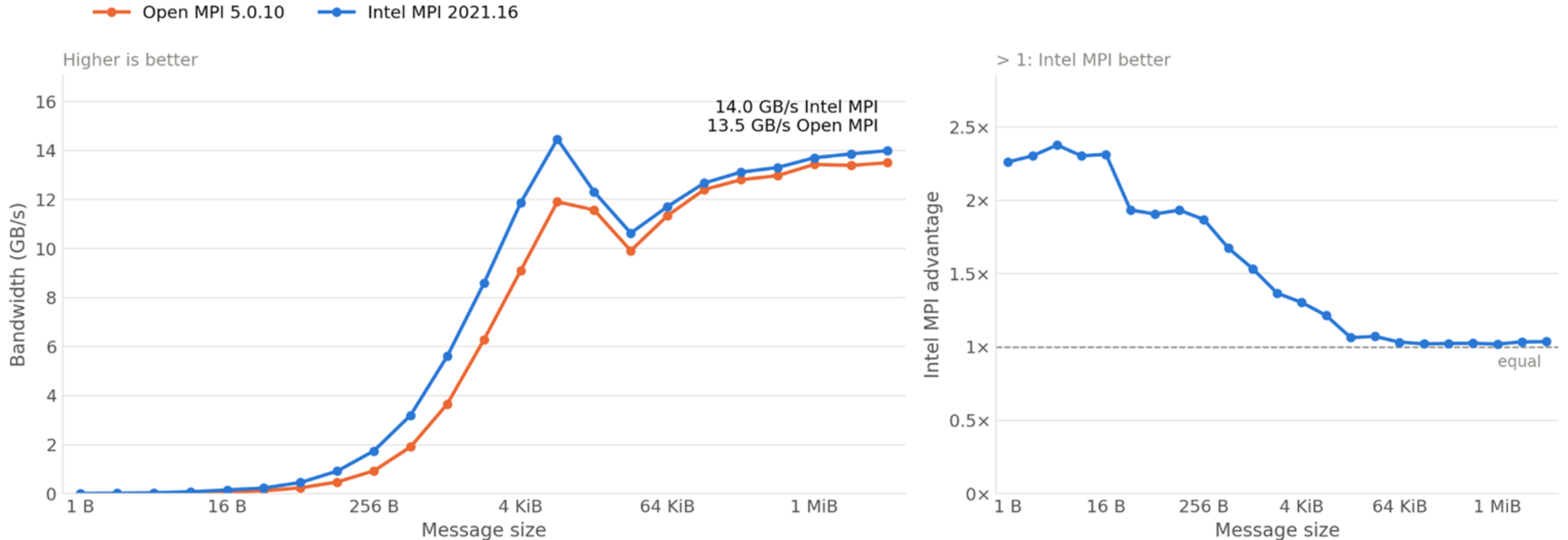


- For P2P communication latency, the MPI choice is irrelevant

MPI Point-to-Point Bandwidth: Intel MPI vs Open MPI

Point-to-point bandwidth: equal for large messages, Intel MPI 1.9x higher up to 4 KiB

osu_bw · 2 nodes, 1 rank per node pinned to the HFI's NUMA domain (cores 72-95) · both over libfabric/OPX

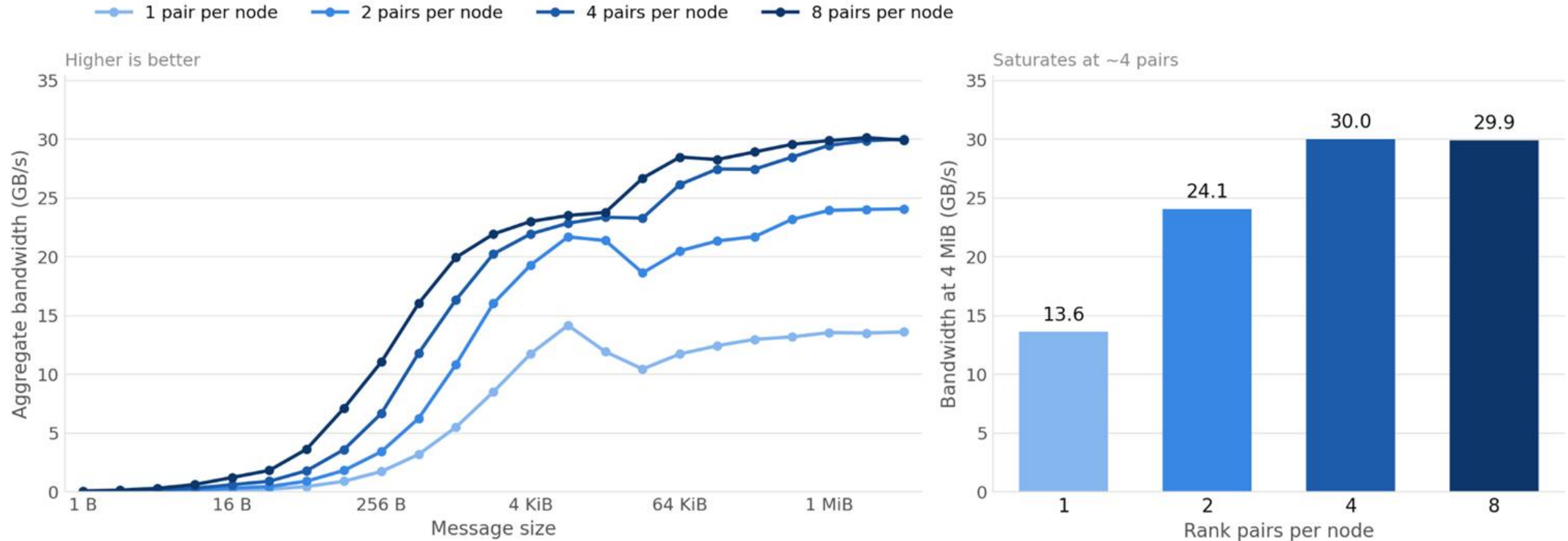


- For P2P communication bandwidth, Intel MPI outperforms Open MPI for smaller messages

MPI Point-to-Point Bandwidth Saturation

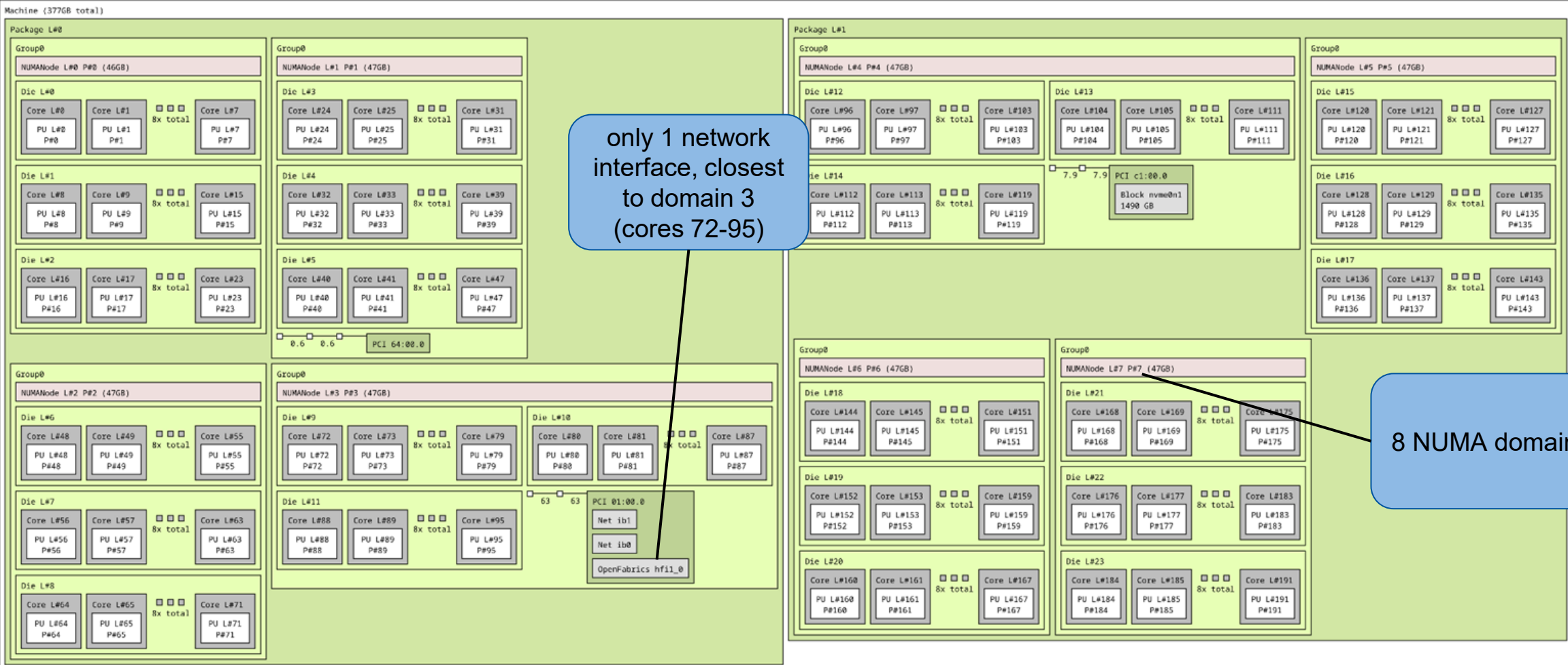
One rank cannot saturate the HFI: 30 GB/s needs ≥ 4 ranks per node ($2.2\times$ one rank)

osu_mbw_mr · 2 nodes, Intel MPI 2021.16, ranks pinned next to the HFI · bandwidth summed over all pairs



- Recommendation: for hybrid (MPI+OpenMP) jobs, use ≥ 4 MPI ranks per node
 - In general: 1 MPI rank per NUMA domain is a good starting point

CPU Hardware Topology (c25ms)



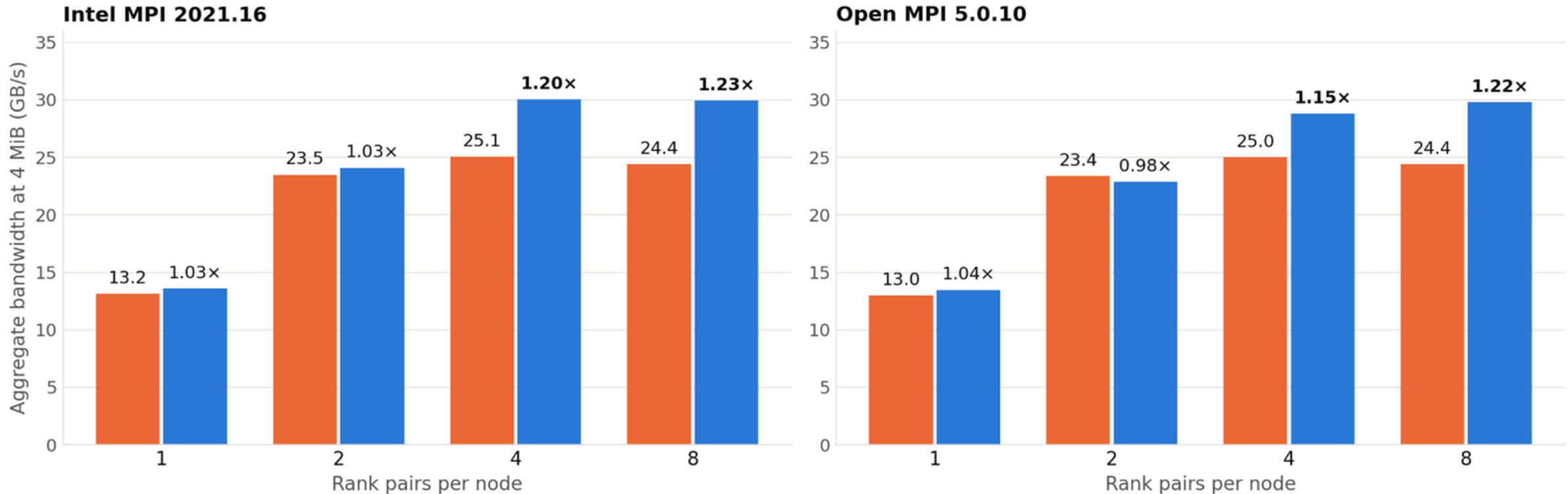
Host: n25s0163.hpc.itc.rwth-aachen.de
Date: Tue 29 Sep 2026 02:15:32 PM CEST

MPI Point-to-Point Binding Impact

Pin ranks next to the HFI: up to 23% more bandwidth with several ranks per node

osu_mbw_mr · 2 nodes, n ranks per node each paired with one on the other node · labels: default bandwidth in GB/s, gain from pinning

Default placement (from core 0 / spread over the node) Pinned to cores next to the HFI (NUMA-local)



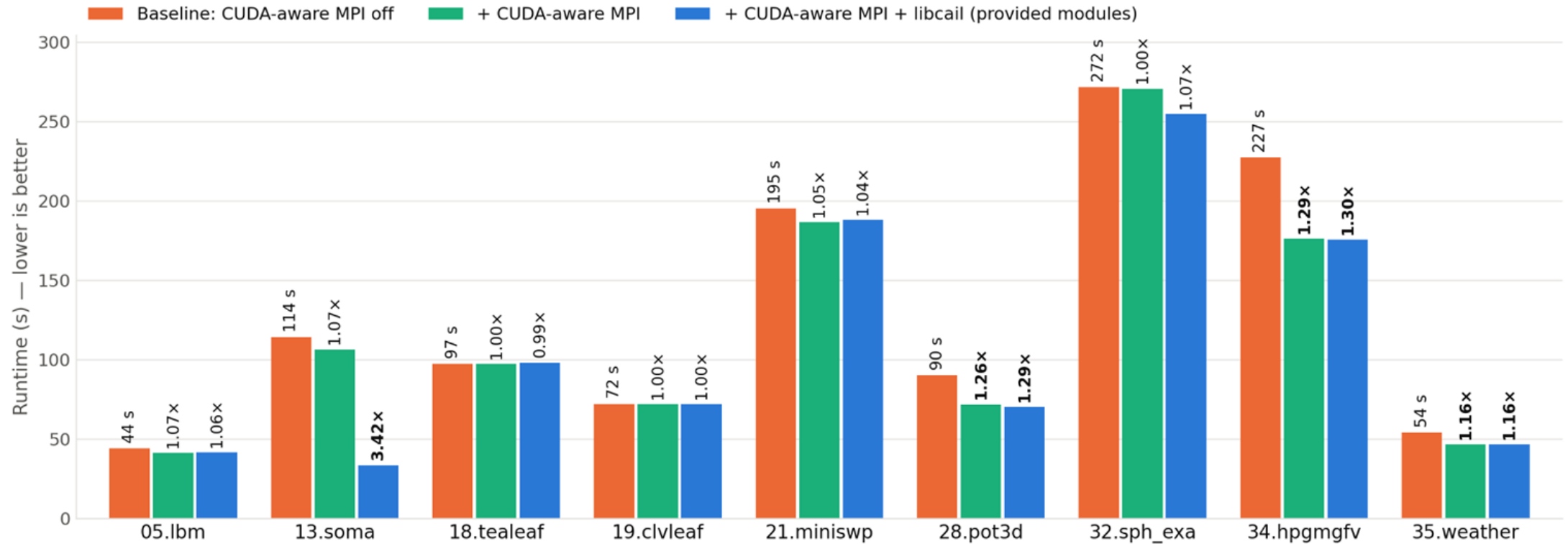
- Recommendation: employ proper pinning if applicable
 - Pin inter-node communication processes to NUMA domain 3 (e.g. Slurm's `--cpu-bind`)

GPU Workloads

CUDA-Aware MPI Performance Comparison

SPEChpc 2021 GPU: impact of CUDA-aware MPI and libcail

4 nodes × 4 GPUs (NVIDIA H100) · OpenMP target offload · labels: baseline runtime, speedup vs. baseline



- Deploying libcail is currently work in progress
- The MPI installation (hpcx) of the NVHPC module is not functional at the moment

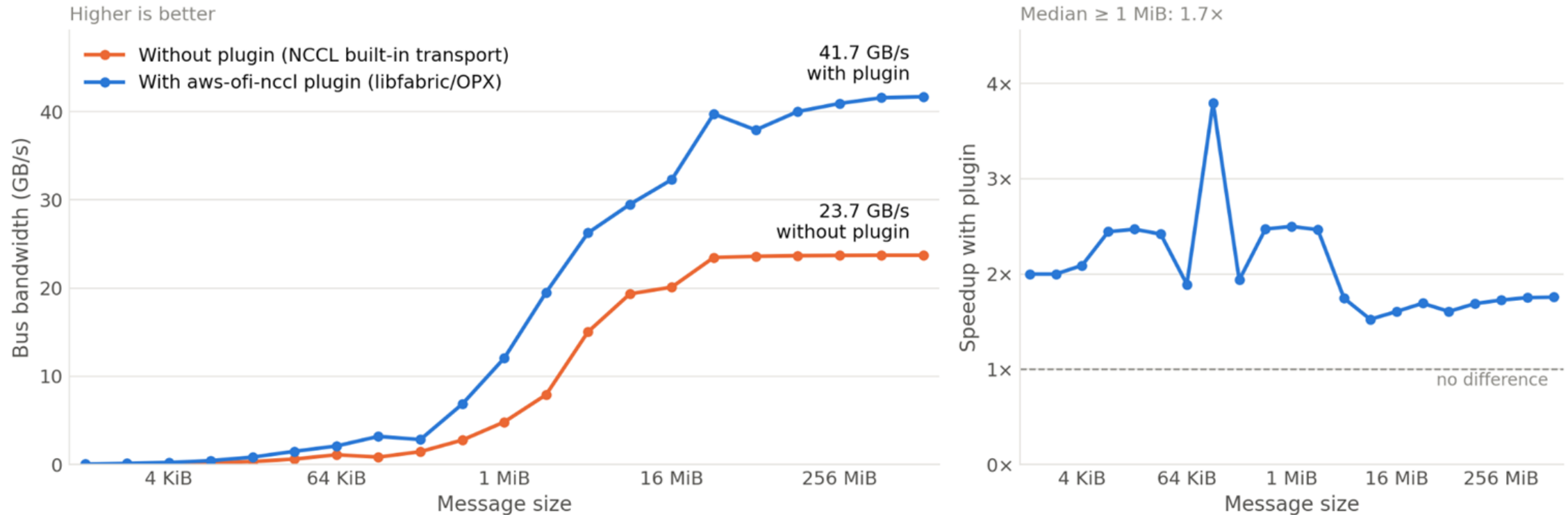
NCCL Setup on CLAIX-2025

- ML frameworks (e.g. PyTorch, TensorFlow) rely on NCCL for inter-node GPU-GPU communication
- NCCL: NVIDIA Collective Communication Library
 - Provides low-latency, high-bandwidth collectives
- On CLAIX-2025 NCCL requires a plugin for low-latency, high-bandwidth inter-node communication
- Currently not deployed at CLAIX-2025
- GPU partition limited to single-node jobs

NCCL Plugin Impact

NCCL all-gather: 1.8× higher peak bus bandwidth with the aws-ofi-nccl plugin

2 nodes × 4 GPUs · nccl-tests, out-of-place · NCCL 2.27.7



- For ML benchmark (PyTorch, LLM Foundry MPT-30B) speedup of x 1.19 (1600 s versus 1344 s)

Roadmap for Multi-Node ML Workloads

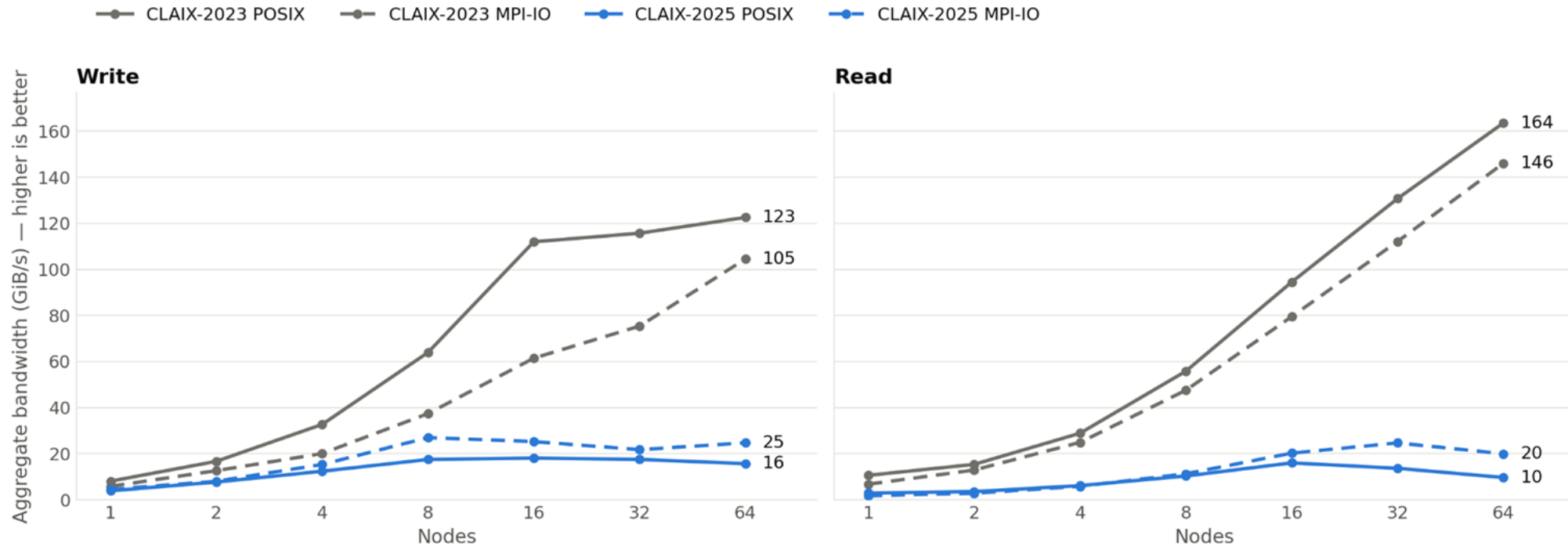
- Multi-Node jobs allowed on GPU partition in October 2026
- Plugin for NCCL/PyTorch modules will be available through modules
 - No additional setup necessary
- For your own PyTorch installations (e.g. via pip or containers), we will either
 - Make additional generic modules available
 - You have to install the plugin yourself and enable it
- For build instructions see <https://github.com/aws/aws-ofi-nccl>
 - Use the libfabric installation at /usr/lib
- To enable the plugin
 - export LD_LIBRARY_PATH=/path/to/plugin/installation/lib:\$LD_LIBRARY_PATH
 - export NCCL_NET=ofi
 - export NCCL_NET_PLUGIN=ofi
 - export FI_PROVIDER=opx
 - For containers: mount the plugin installation directory
- Once decided and deployed details will be available in the IT Center Help documentation

I/O

CLAIX-2025 Lustre (\$HPCWORK) I/O Performance & Recommendations

Lustre from CLAIX-2025 tops out at ~27 GiB/s, CLAIX-2023 keeps scaling

IOR aggregate bandwidth · 4 GiB block, 128 MiB transfers, 8 tasks/node, file per process · same Lustre file system, CLAIX-2025 via Omni-Path → InfiniBand gateways



- Bandwidth currently limited by gateway (between CLAIX-2025 and Lustre filesystem)
- Will (hopefully) improve in October 2026
- Recommendations
 - For temporal or incremental output stage data in \$BEEOND (\$TMP or /dev/shm on single node)
 - Use CLAIX-2023 for I/O-heavy workloads until performance has been improved (\$WORK is even worse)

Summary

- **Prefer provided MPI modules** over own installations
 - Module installations are properly configured
- **Stick to default toolchain: intel** (Intel compilers + Intel MPI)
 - Prefer Intel MPI over Open MPI, especially for collective communication
 - Compiler choice secondary, test manually for specific application
- Multi-node machine learning workloads require proper setup for efficient communication
 - Ready-to-use setup is still work in progress
- I/O performance currently much worse than on CLAIX-2023
 - Lustre expected to improve soon (Oct. 2026)
 - **Use CLAIX-2023 for IO-heavy workloads until further notice**

Thank you for your attention!
Questions?