ALCF Hands-on HPC workshop
October 7th, 2025

Argonne NATIONAL LABORATORY

AI and HPC Applications on Leadership Computing Platforms: Performance and Scalability Studies

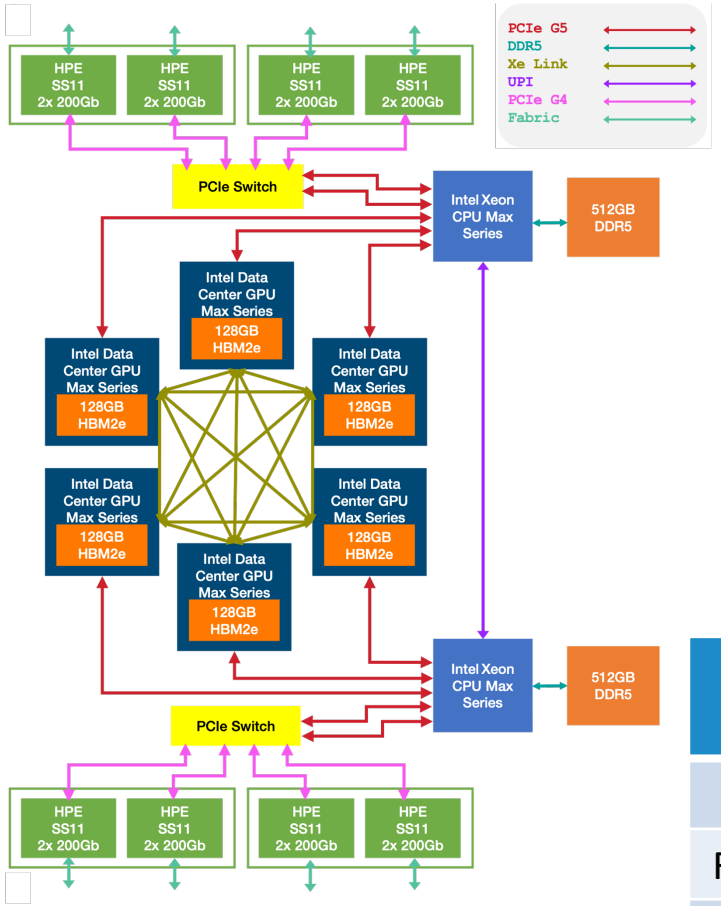
JaeHyuk Kwack*, Colleen Bertoni, Umesh Unnikrishnan, Riccardo Balin, Khalid Hossain, Yasaman Ghadar, Timothy J. Williams, Abhishek Bagusetty, Mathialakan Thavappiragasam, Väinö Hatanpää, Archit Vasan, John Tramm, Scott Parker

Leadership Computing Facility (LCF), Computational Science (CPS)
Argonne National Laboratory
Lemont, IL, USA

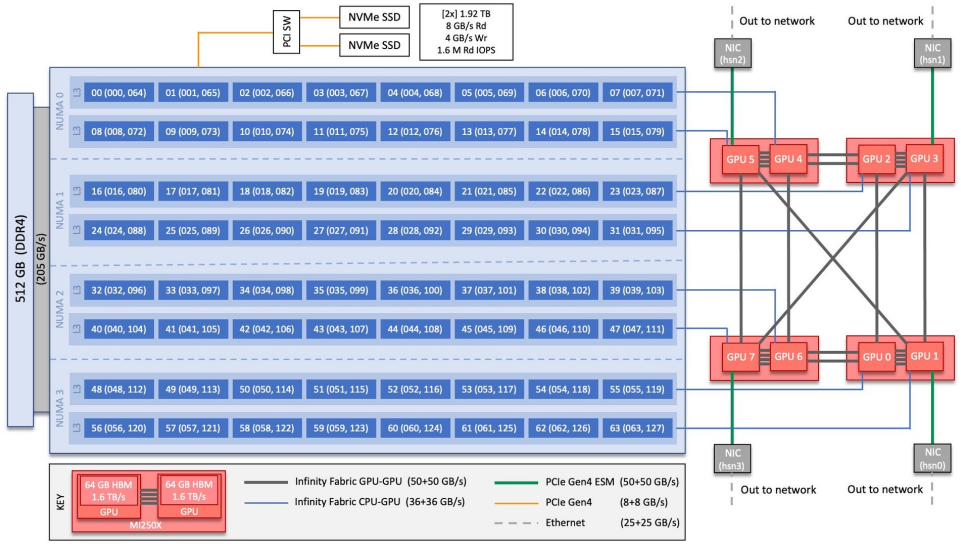
System/Application Overview

System overview

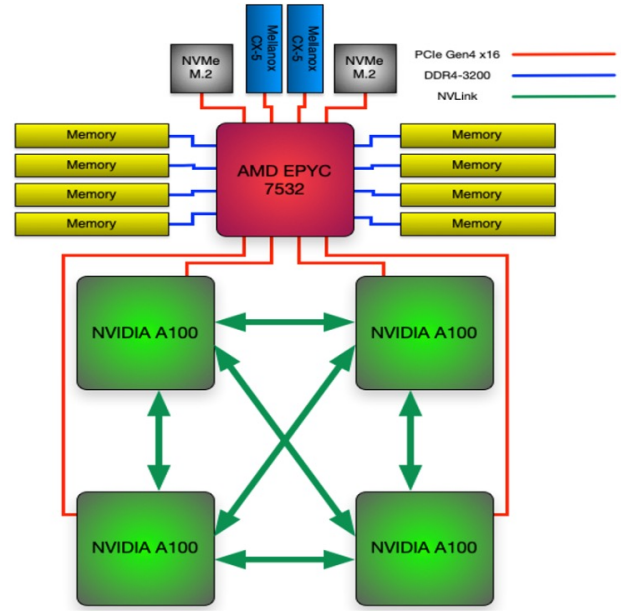
Aurora node w/ Intel Max 1550 GPUs



Frontier node w/ AMD MI-250X GPUs



Polaris node w/ NVIDIA A100 GPUs



	# nodes	CPU per node	GPU per node	DRAM per node (GB)	NIC BW per node (GB/s)
Aurora	10,624	2 x Intel SPR	6 x Intel GPU Max 1550	1024 GB	200
Frontier	9,856	1 x AMD Milan	4 x AMD MI250x	512 GB	100
Polaris	560	1 x AMD Milan	4 x NVIDIA A100	512 GB	50

Overview of Aurora, Frontier, and Polaris

GPU/Node specification for Aurora, Frontier, & Polaris

		Aurora	Frontier	Polaris
GPU	GPU Vendor	Intel	AMD	NVIDIA
	GPU model	GPU Max 1550 (PVC)	MI250x	A100-SMX4
	FP32 non-tensor peak (TF/s)	45.9	47.9	19.5
	FP64 non-tensor peak /tensor peak (TF/s)	45.9	47.9 / 95.7	9.7 / 19.5
	Stream triad (TB/s)	2	2.6	1.4
	HBM capacity (GB)	128	128	40
	TDP (W)	500	500	400
Node	GPU cards per node	6	4	4
	FP32 non-tensor peak (TF/s)	275.4	191.6	78
	FP64 non-tensor peak/tensor peak (TF/s)	275.4	191.6 / 383.2	38.8 / 78
	Stream triad (TB/s)	12	10.4	5.6
	HBM capacity (GB)	768	512	160
	NIC Bandwidth (GB/s)	200	100	50
	Network fabric	Slingshot-11	Slingshot-11	Slingshot-11

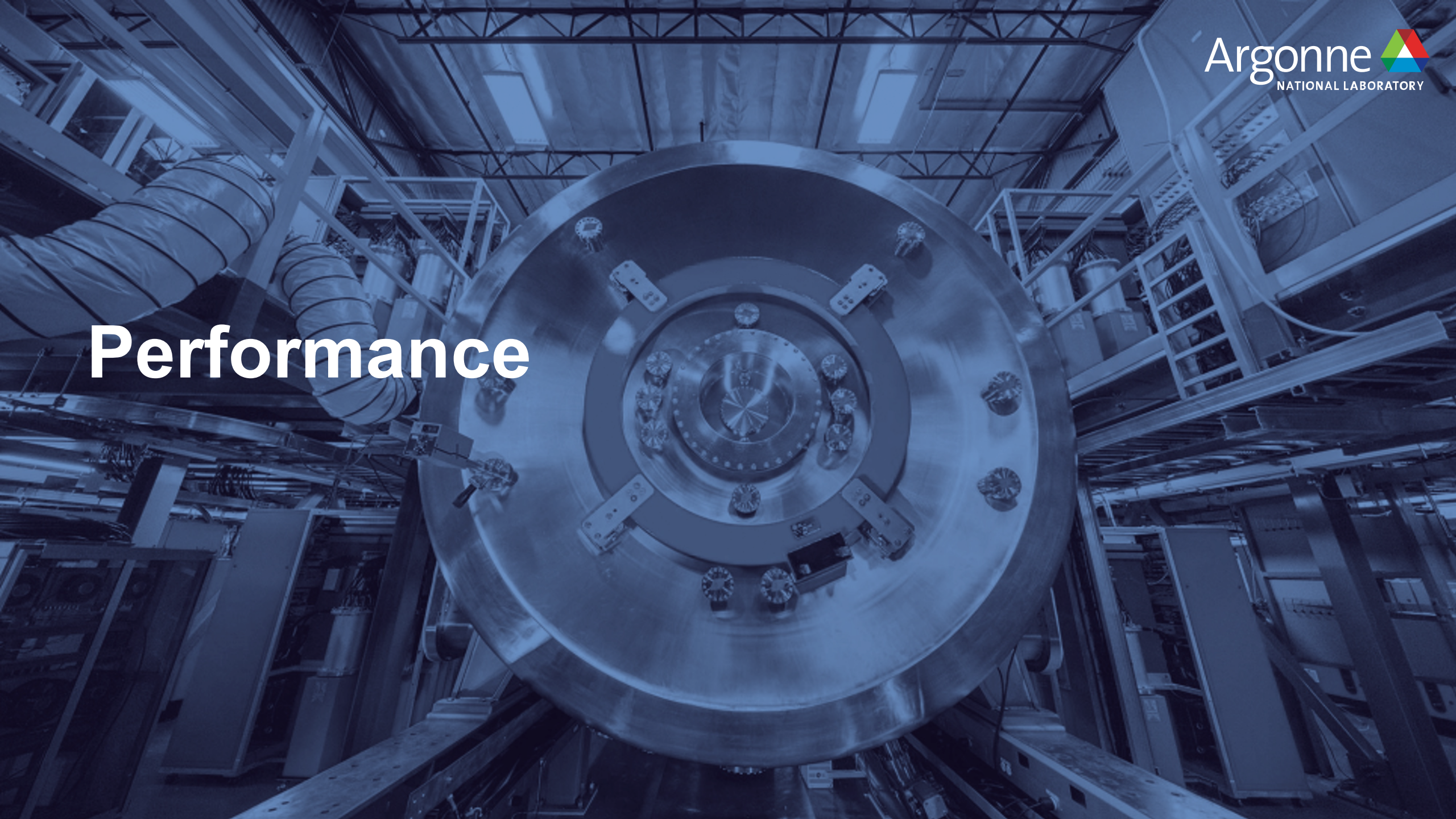
Application overview

- Twelve HPC/ML applications to represent various science domains
  - CFD, electronic structure, material science, plasma kinetics, stellar explosion, particle transport
  - AI/ML GNN/ML, LLM/ML, Chemical ML
- Base languages
  - C++, Fortran, Python
- Programming model, or framework
  - OpenMP offload / OpenACC
  - Kokkos
  - OCCA
  - TAMM
  - AMReX
  - PyTorch
  - TensorFlow

Application	Science Domain	Base language	Programming Model or Portability Layer	Scaling
AMR-Wind	CFD	C++	AMReX	Weak
GAMESS	Electronic Structure	Fortran	OpenMP Offload	Strong
CNS-libParanumal	CFD	C++	OCCA	Weak
SimAI-Bench	GNN/ML	Python	PyTorch	Weak
CosmicTagger	AI/ML	Python	PyTorch	Weak
LAMMPS	Chemistry/Material Science	C++	Kokkos	Weak
XGC	Tokamak Plasma Kinetics	C++	Kokkos	Weak
NWChemEX	Electronic Structure	C++	TAMM(CUDA,HIP,SYCL)	Strong
FlashX	Stellar Explosion	Fortran	OpenMP/OpenACC	W/S
Magatron-DeepSpeed	LLM/ML	Python	PyTorch	Weak
Simple SMILE Transformer	Chemical ML	Python	TensorFlow	Weak
OpenMC	Particle Transport	C++	OpenMP Offload	Weak

Application Characteristics

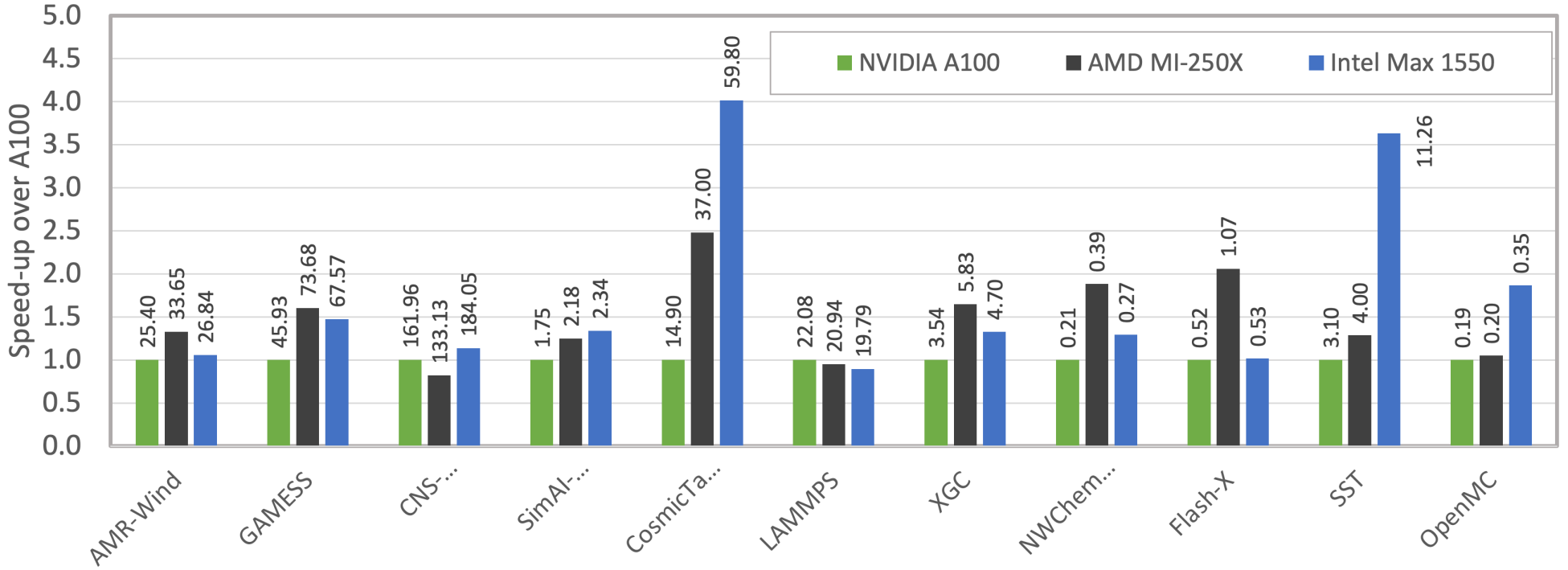
Application	Main precisions	Algorithmic Motifs	Performance Characteristics of Kernels
AMR-Wind	FP64	Element-wise kernels, sparse iterative solvers	Mem BW, network latency, GPU occupancy
GAMESS	FP64	Eigen solve, linear algebra, tensor contractions	Mem BW, compute, GPU occupancy
CNS-libParanumal	FP64	Discontinuous Galerkin (DG) method, linear algebra, tensor contractions	Mem BW (mostly), compute
SimAI-Bench	FP32/TF32	SGEMM, layer normalization, ReLU activation	Compute, mem BW, reduction
ComsmicTagger	FP32/TF32	Batch normalization, ReLU activation, implicit GEMM convolution particle method	Mem BW (mostly), compute
LAMMPS	FP64	Particle method	Compute (30%), mem BW (15%)
XGC	FP64	Particle-mesh gather/scatter operation with unstructured mesh	Comm, mem BW
NWChemEx	FP64	Eigen solve, linear algebra, tensor contractions	Mem BW, compute, GPU occupancy, somm
FlashX	FP64	Finite volume and DG methods, linear algebra	Mem BW (mostly)
Megatron-DeepSpeed	BF16	GEMM, Softmax/Swiglu activations, RMSNorm normalization	Compute, mem BW, comm
SST	FP64	Bidirectional self-attention, layer normalization, SeLU activation	Mem BW, compute
OpenMC	FP64	Particle method	Mem latency, mem BW



Performance

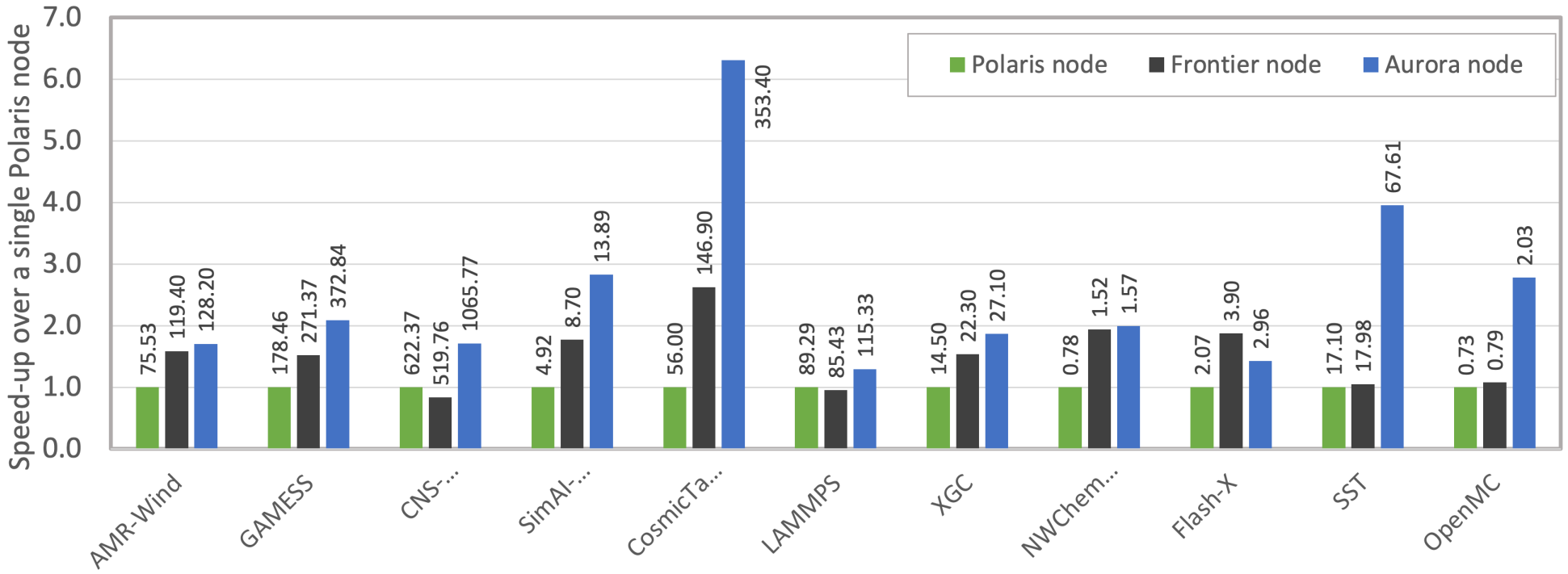
Argonne NATIONAL LABORATORY

Single GPU Performance



- Performance on 1 A100 GPU, 2 GCDs of MI-250X GPU, and 2 stacks of Intel Max 1550 GPU
- MI-250X performs best with 5 applications (AMR-Wind, GAMESS, XGC, NWChemEX, FlashX)
- Intel Max 1550 GPU operates best with other five applications (CNS, SimAI, CosmicTagger, SST, and OpenMC)

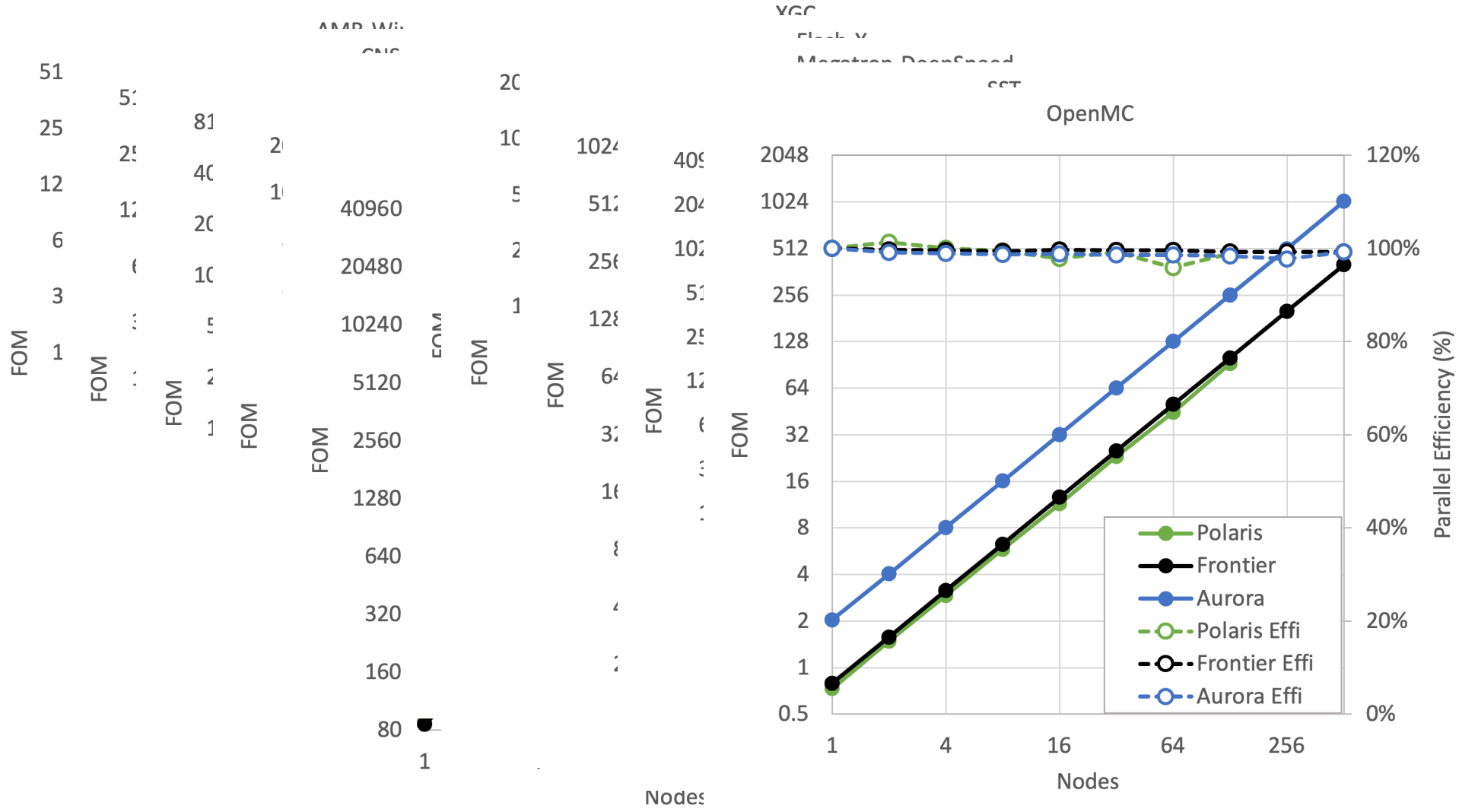
Single Node Performance



- Performance on 4 A100 GPUs on Polaris, 4 MI-250X GPUs on Frontier, 6 Intel Max 1550 GPUs on Aurora
- Since Aurora has 50% more GPU cards than others, an Aurora node performs best with ten applications (AMR-Wind, GAMESS, CNS, SimAI, CosmicTagger, LAMMPS, XGC, NWChemEX, SST, and OpenMC)
- A Frontier node operates best with one application (i.e., Flash-X)

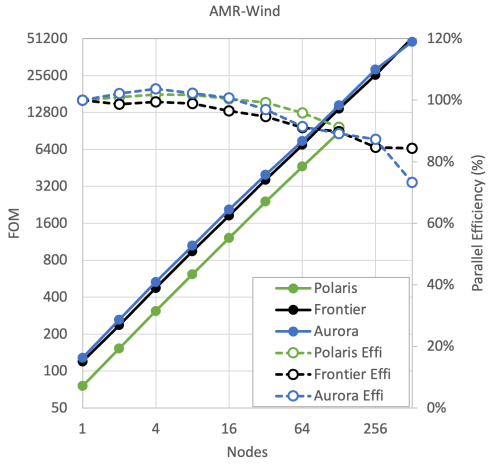
Weak Scaling Performance and Parallel Efficiency

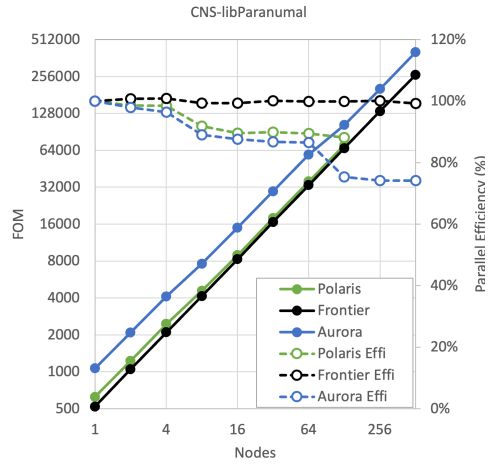
Scale-out with up to 5% of Aurora and Frontier (512 nodes) & 25% of Polaris (128 nodes)

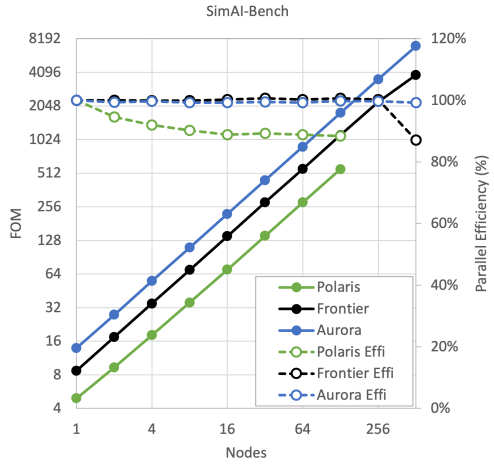


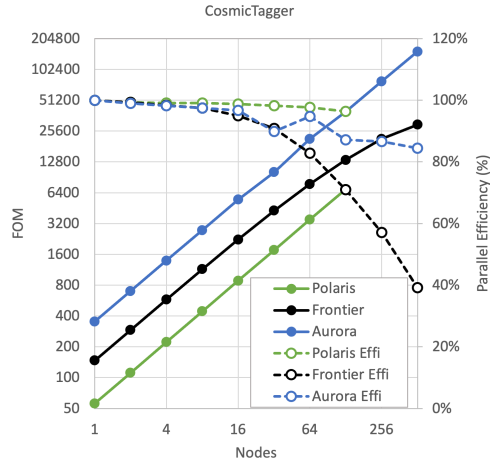
Weak Scaling Performance and Parallel Efficiency

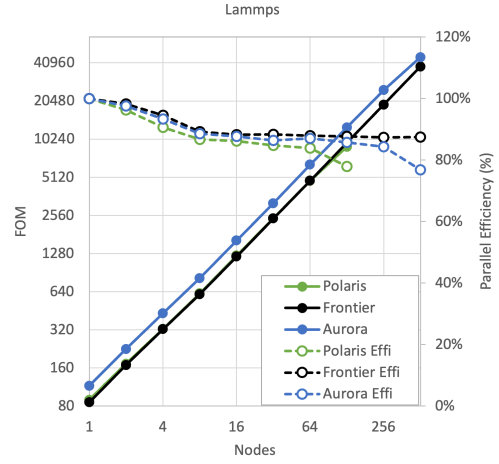
Scale-out with up to 5% of Aurora and Frontier (512 nodes) & 25% of Polaris (128 nodes)

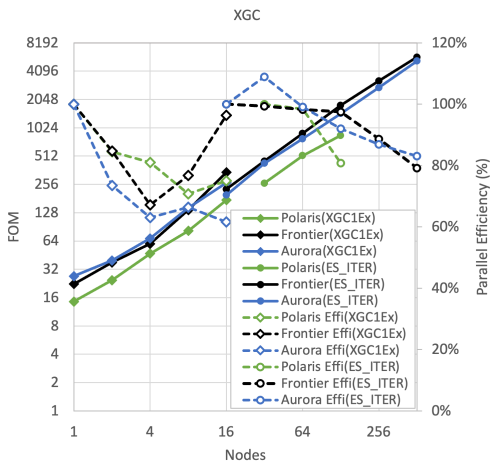


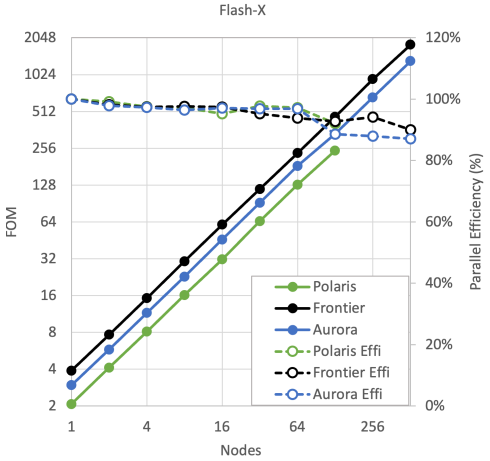


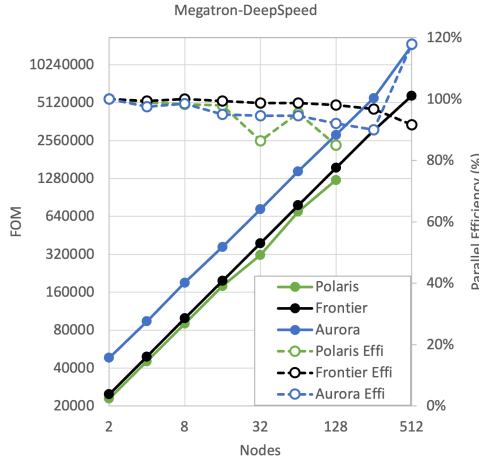


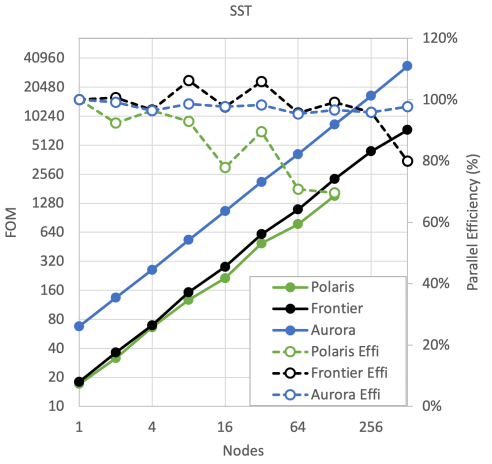


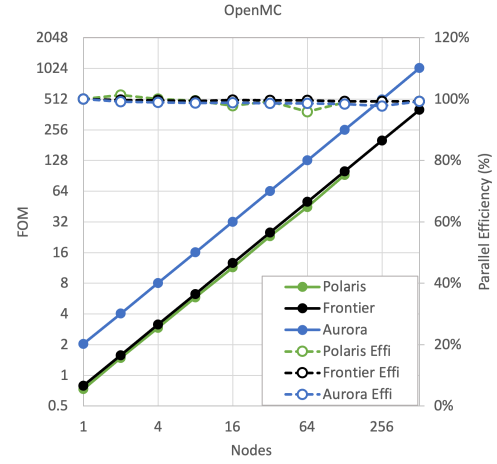






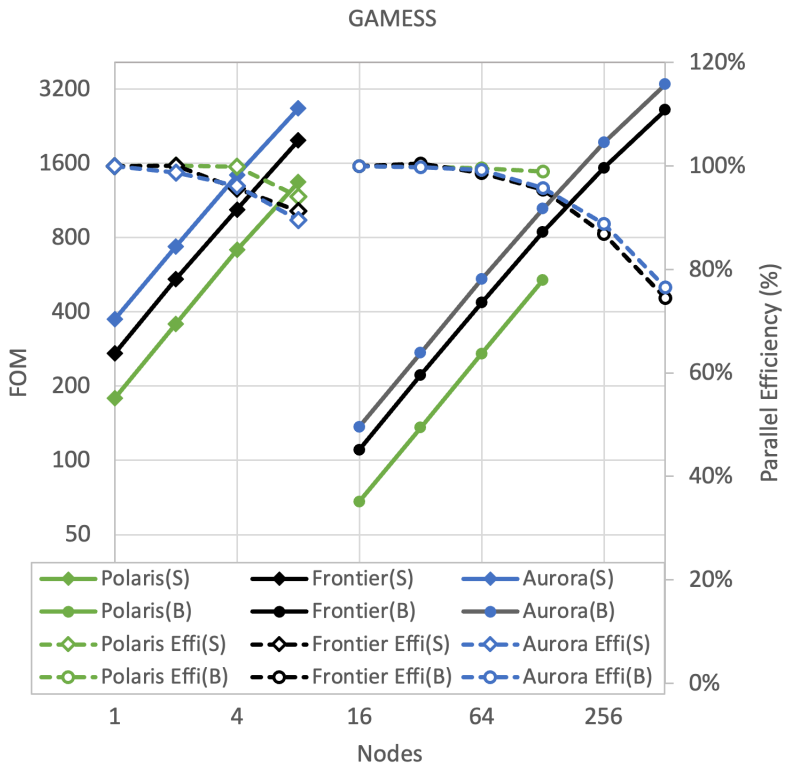


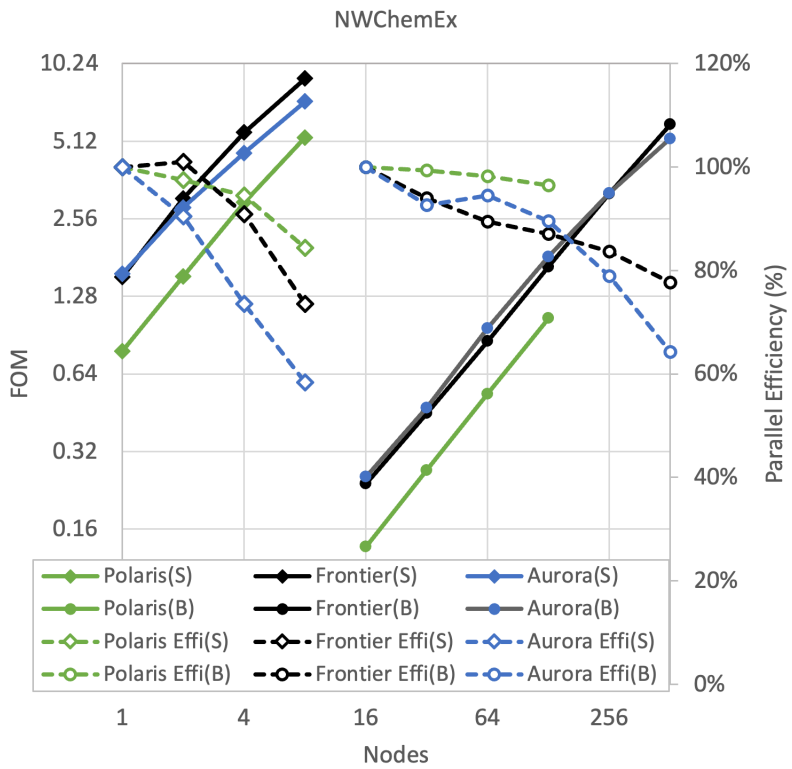


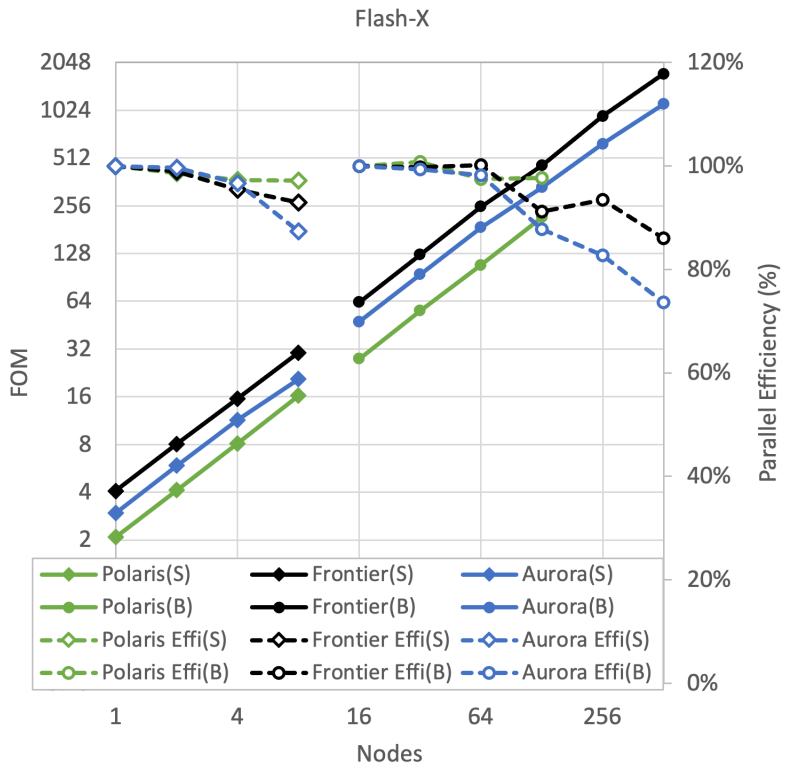


Strong Scaling Performance and Parallel Efficiency

- Scale-out up to 5% of Aurora and Frontier (512 nodes) & 25% of Polaris (128 nodes)
- Two inputs: a small input for up to 8 nodes, and a large input for more nodes







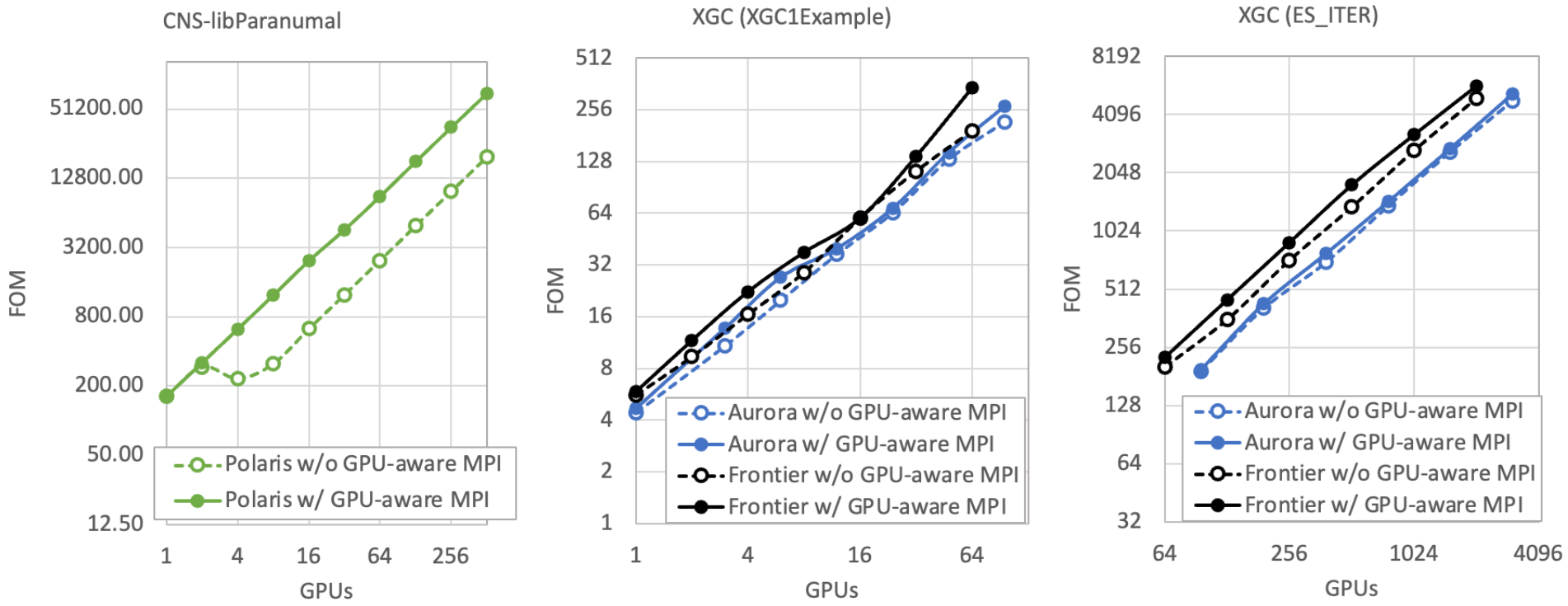
Key Concepts for Performance

CPU/GPU binding

- SLURM on Frontier: automatically binding cores, threads, and GPUs to nearest NUMA domain
- PBS-Pro on Aurora and Polaris: requiring manual binding of cores and threads with MPICH options, and manual GPU binding via ZE_AFFINITY_MASK (Aurora) / CUDA_VISIBLE_DEVICES (Polaris)
- Proper CPU to GPU binding on Aurora
  - GAMESS: 2.5x speed-up
    - Half of the MPI ranks performs computations on GPUs and other half handle communication across nodes
    - Proper pairing of computing ranks and communication ranks
  - NWChemEX: 2x speed-up
    - Mapping MPI processes exclusively to the cores of a single socket
  - AMR-Wind: 8% speed-up
    - Less sensitive to CPU binding
- Proper GPU binding on Aurora
  - AMR-Wind: 4.3x speed-up by assigning each MPI rank to GPU stack w/o oversubscription
- Proper core binding to oneCCL workers
  - CosmicTagger: 10-15% speed-up by leaving separate CPU cores for oneCCL workers

GPU-aware MPI

- Removing the cost of staging GPU buffers via host memory



- Challenges
  - Improper initialization with GPU-aware MPI at large node counts with CNS-libParanumal on Polaris
  - Memory issues with GPU-aware MPI (cray-mpich) with XGC on Polaris
  - Memory leaks in Aurora MPICH with GPU-aware MPI on Aurora

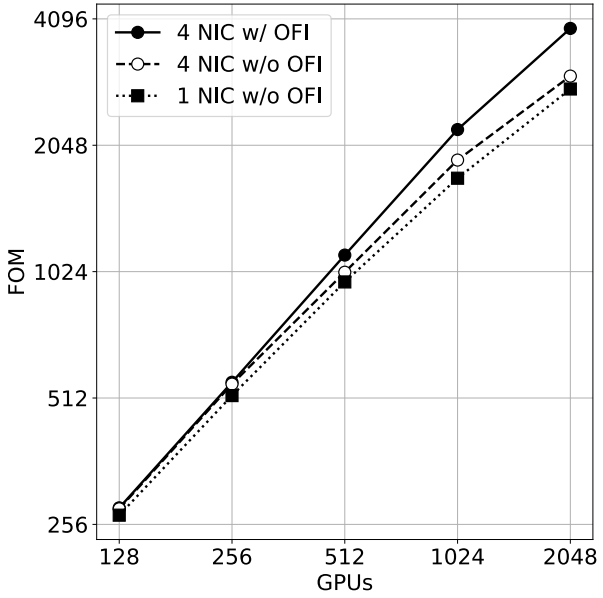
Configurations for ML applications

- Good representative of portable AI/ML applications using common framework libraries such as PyTorch and TensorFlow across a variety of hardware and accelerator platforms

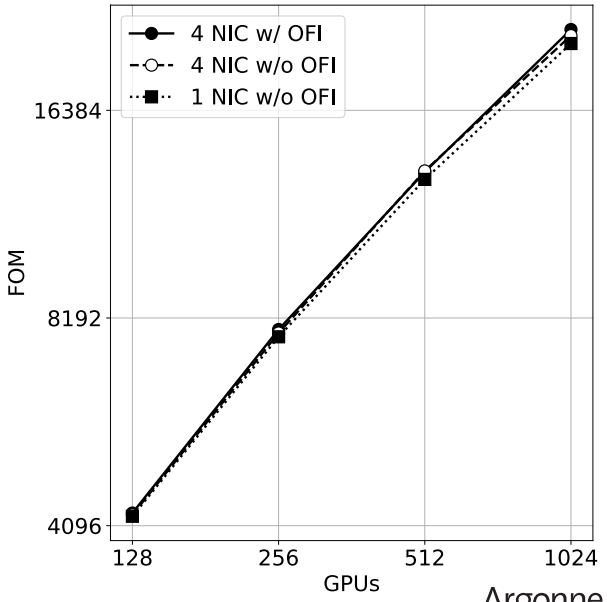
- NCCL_NET_GDR_LEVEL=PHB on Frontier (512 node case) with CosmicTagger
  - A long execution time for collective operations → exceeding timeout for PyTorch DDP for RCCL/NCCL backend
  - Other suggestion by OLCF, setting it to SYS, and PXB → resulting in a dramatic loss of efficiency
  - Still investigating the issue

- Explicit setting the number of NICs to RCCL collective library (NCCL_SOCKET_IFNAME) & enabling RCCL to leverage libfabric's transport layer w/ ROCm AWS-OFI-RCCL plugin
  - SimAI-Bench GNN: 39% performance gain at 2048 GPUs
  - Cosmic Tagger: less sensitive to these parameters

SimAI-Bench GNN

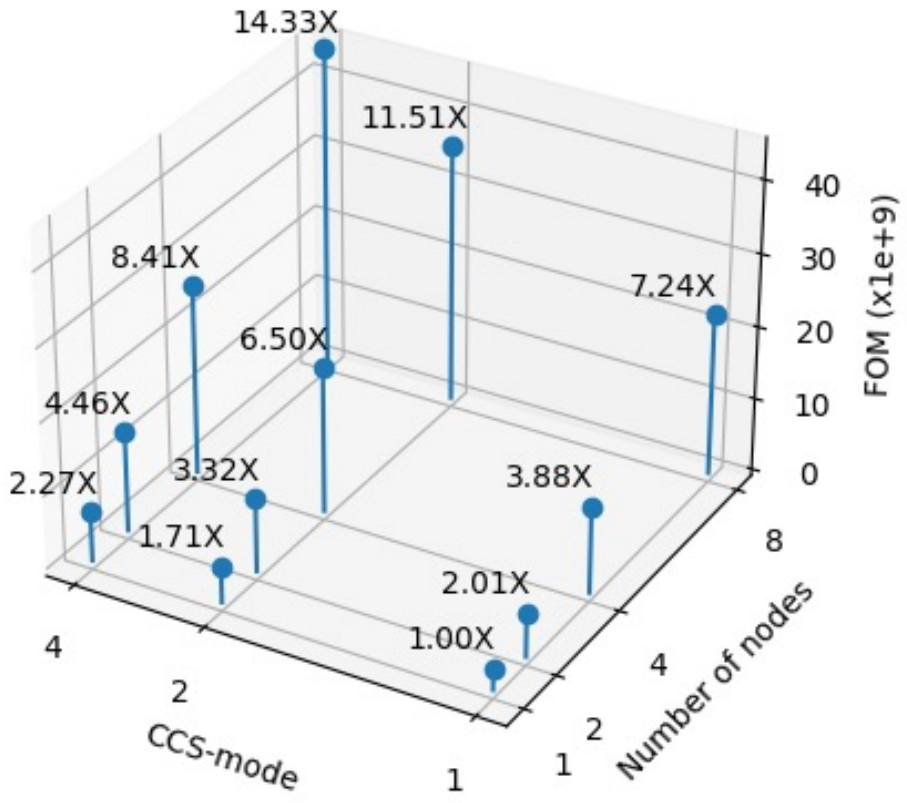


Cosmic Tagger



CCS mode on Aurora

- Intel's Compute-Command Streamers (CCSs) to cluster Execution Unit (EU) , similar to the MPS mode on NVIDIA GPUs
  - —4, 2, or 1 (default) per stack using an environmental variable ZEX_NUMBER_OF CCS
  - —8, 4, or 2 clusters per PVC
  - —EUs are evenly distributed among CCSs
- Flash-X SSW test cases: Assigning each MPI rank to EU cluster
  - —A single node
    - 2.27x with 4 CCSs compared to 1 CCS
  - —8 nodes
    - Around 2 x speed up with 4 CCSs over 1 CCS with 8 nodes



Discussion & Concluding Remarks

Performance Portability

- A quantitative assessment of the performance portability as proposed by Pennycook of these applications across the three systems is very challenging.
  - Requires an understanding of the efficiency of the implementation of each application on each of the systems
  - Requires a detailed assessment of how the application is interacting with the system hardware
  - Multiple kernels of each application with potentially different bounds, which makes analyzing each application's efficiency in practice a complex task.
  - Issues and limitations of the performance assessment tools (e.g., roofline analyses) on the three systems to quantify performance of applications

- Qualitative assessment of the performance portability
  - based on the double/single precision peaks, and stream bandwidth, and the observed performance of the applications
    - GPU DP peaks w/o tensor cores: 2x on Aurora and Frontier over Polaris
    - GPU HBM BWs: 1.3x on Frontier over Aurora & 2x on Frontier over Polaris
  - Amdahl's law: for example, if runtime of DP peak bound kernels of an application is around 40%, the expected speed-up over Polaris is around 1.25x.

Performance Portability (Cont'd)

- Qualitative assessment of the performance portability
  - Six applications (i.e., GAMESS, SimAI-Bench, Cosmic Tagger, XGC, NWChemEx, and SST) had FOMs on Aurora and Frontier which are both >= 1.25x that on Polaris
    - GAMESS, SimAI-Bench: Frontier and Aurora FOMs are within 10% of each other (*qualitatively performance portable*)
    - CosmicTagger, SST: higher FOMs on Aurora (38% and 65% difference from the Frontier FOMs)
    - NWChemEx, XGC: higher FOMS on Frontier (32% and 20% different respectively)
  - None of the applications were far out of bounds expected (highest 4x, lowest 0.8x)
  - Most of them have some level of performance portability
  - No programming models or algorithmic motifs achieving more performance portable than others.

Concluding Remarks

- All twelve applications evaluated in this study are all portable across the three systems and three different GPU architectures.
  - all complex scientific applications that simulate a wide range of different physics & employ a variety of different computational algorithms
  - Three different base languages (i.e., C++, Fortran, Python) and different portability layers (e.g., OpenMP, OCCA, Kokkos, AMReX, PyTorch, TensorFlow)
- Single GPU/Node Performance:
  - 0.9-4x on PVC (0.8-2.4x on MI-250x) over A100
  - 1.3-6.3x on an Aurora node (0.8-2.6x on a Frontier node) over a Polaris node
- Scaling Performance:
  - All application scaled reasonably well across three systems
  - Weak scaling parallel efficiencies were similar across systems
  - Strong scaling parallel efficiencies on Aurora & Frontier were slightly worse due to more compute resources per node there
  - Better chance to run larger problems efficiently on Aurora & Frontier than on Polaris

Concluding Remarks (Cont'd)

- Key issues and concepts encountered were discussed
- Qualitative performance portability was discussed.
- Future work
  - Better understanding of performance portability of applications across today's large scale GPU systems
  - Improving techniques and tools for measuring application performance bottleneck to quantify application efficiency
  - Impact of I/O on application performance
  - Investigating the impact of unique hardware such as the HBM memory on SPR CPUs on Aurora for mixed workloads utilizing CPU and GPU resources in a combined fashion
  - Investigating the power usage of applications on different systems in the future

Acknowledgment

- This work was supported by the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357, and by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of two U.S. Department of Energy organizations (Office of Science and the National Nuclear Security Administration).
- This work was done on a pre-production supercomputer with early versions of the Aurora software development kit.
- This research also used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE- AC05-00OR22725.
- We wish to extend our special thanks to PMA (Programming Models and Architectures) WG members (Thomas Applencourt, Longfei Gao, Kevin Harms, Brian Homerding, Chris Knight, Ye Luo, Vitali Morozov, Steve Rangel, Kris Rowe, and Brice Videau) who are not authors of this paper but have generously provided technical feedback on a regular basis.
- We also appreciate support from Austin Harris at ORNL.


Thank you