MARCH 18TH 2025

Experiences with Scaling AI Workloads on Aurora

VÄINÖ HATANPÄÄ
Assistant Computer Scientist
vhatanpaeae@anl.gov

U.S. DEPARTMENT OF ENERGY
Argonne National Laboratory is a U.S. Department of Energy laboratory managed by UChicago Argonne, LLC.

Argonne NATIONAL LABORATORY
Argonne Leadership Computing Facility

Experiences with Scaling AI Workloads on Aurora

Presentation contents

1. Argonne Leadership Computing Facility
2. Gordon Bell 2024 finalist: MProt-DPO
3. Efficiently launching and running PyTorch at scale
4. Ongoing work for AI/ML performance at ALCF

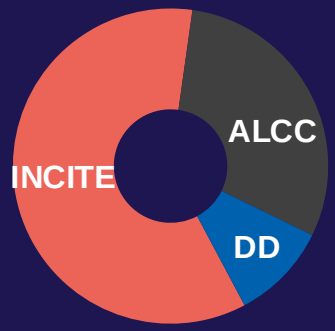
Argonne Leadership Computing Facility

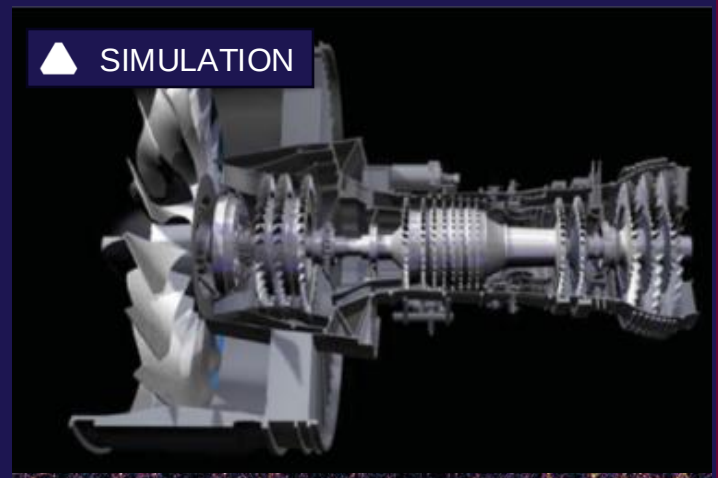
ALCF provides supercomputers to enable scientists to:

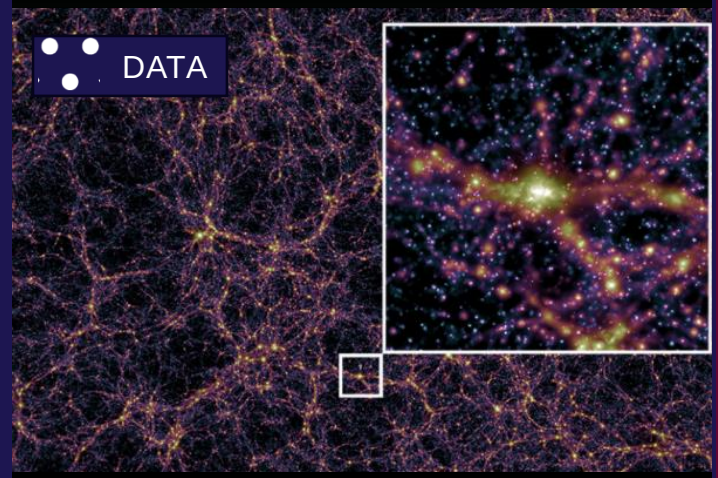
- To address grand challenges for the nation
- Perform research that is too complex and expensive to do in a laboratory setting
- Keep the nation safe and competitive

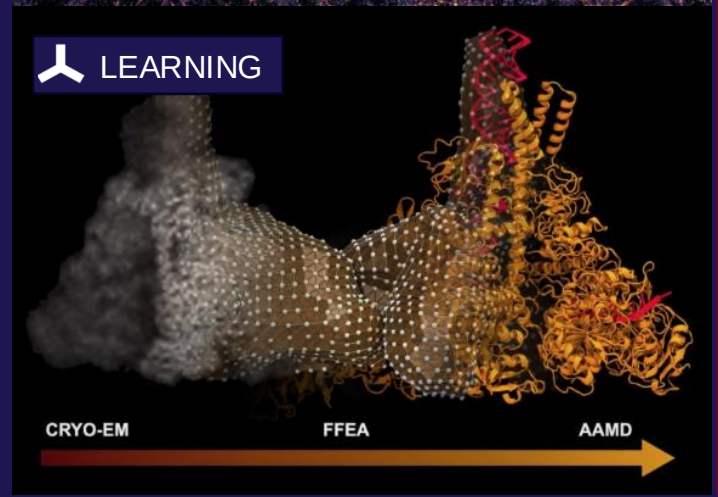
Researchers with a large-scale computing problem can apply to use ALCF resources.

- Multiple allocation award programs available to fit your needs









COMPUTING RESOURCES

#1 AI (HPL-MxP) supercomputer and > 1 exaFLOPS.

System supports 3 types of computing:

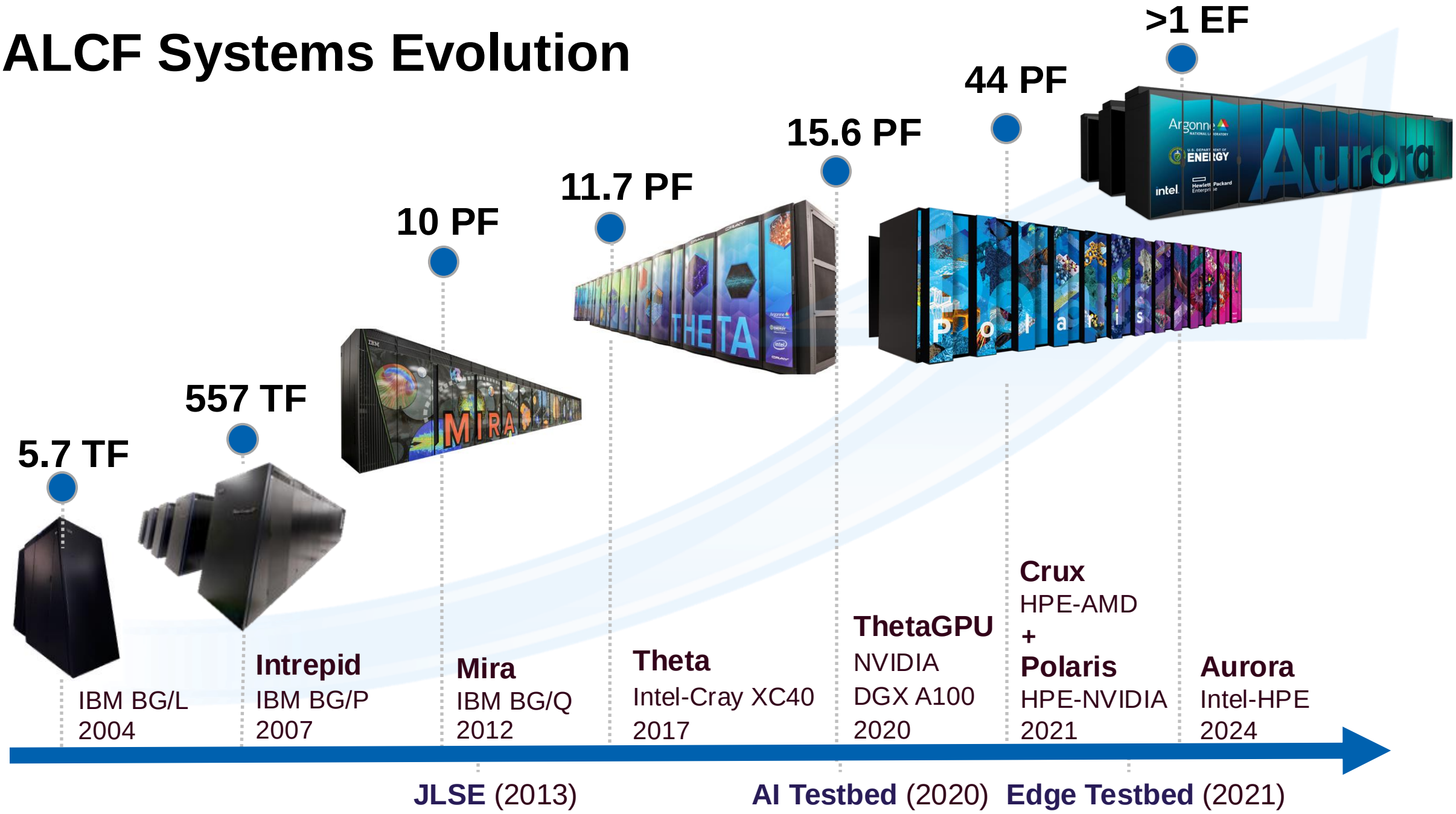
- Large-scale Simulations
- Data Intensive Applications
- AI for Science

ALCF AI TESTBED

Next-generation AI platforms to rapidly deploy and accelerate state-of-the-art AI for science

U.S. DEPARTMENT OF ENERGY
Argonne National Laboratory is a U.S. Department of Energy laboratory Managed by Argonne UChicago, LLC

ALCF Systems Evolution



Gordon Bell 2024 Finalist

MProt-DPO: Breaking the ExaFLOP Barrier for Multimodal Protein Design Workflows with Direct Preference Optimization

U.S. DEPARTMENT OF ENERGY
Argonne National Laboratory is a U.S. Department of Energy laboratory managed by UChicago Argonne, LLC.

Argonne NATIONAL LABORATORY
Argonne Leadership Computing Facility

MProt-DPO: Breaking the ExaFLOP Barrier for Multimodal Protein Design Workflows with Direct Preference Optimization

Gautham Dharuman[1†], Kyle Hippe[1,2†], Alexander Brace[1,2†], Sam Foreman[1†], Vaino Hatanpaa[1], Varuni K. Sastry[1], Huihuo Zheng[1], Logan Ward[1], Servesh Muralidharan[1], Archit Vasan[1], Bharat Kale[1], Carla M. Mann[1,2], Heng Ma[1], Yun-Hsuan Cheng[3], Yuliana Zamora[3], Shengchao Liu[5], Chaowei Xiao[6], Murali Emani[1], Tom Gibbs[3], Mahidhar Tatineni[7], Deepak Canchi[8], Jerome Mitchell[8], Koichi Yamada[8], Maria Garzaran[8], Michael E. Papka[1,9], Ian Foster[1,2], Rick Stevens[1,2], Anima Anandkumar[10*], Venkatram Vishwanath[1,9*], Arvind Ramanathan[1,2*]

[1]Argonne National Laboratory, [2]University of Chicago, [3]NVIDIA Inc., [4]Swiss National Supercomputing Center, [5]University of California, Berkeley, [6]University of Wisconsin-Madison, Madison, [7]San Diego Supercomputing Center, [8]Intel Corporation, [9]University of Illinois Chicago, [10]California Institute of Technology

†Joint first authors, *Contact authors: **venkat@anl.gov, anima@caltech.edu, ramanathana@anl.gov**

Hypothesis: Multimodal language models can incorporate experimental observables to constrain the generation of protein sequences

Multi-modal Datasets

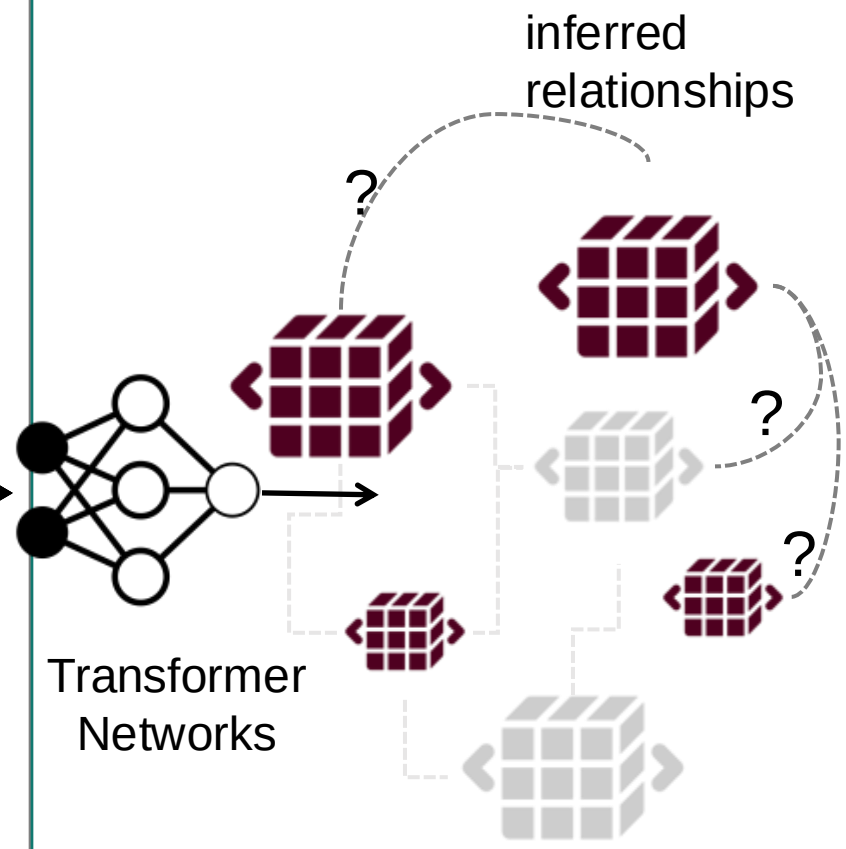
Scientific literature

Databases

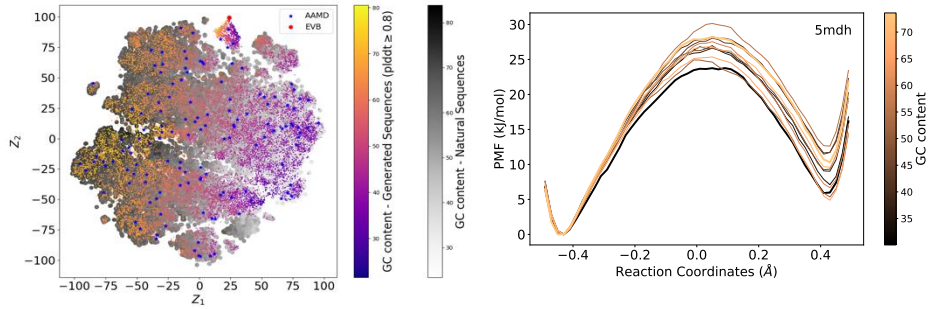
Experimental data

Simulations

Multi-modal Models



Simulation/ Experiment data

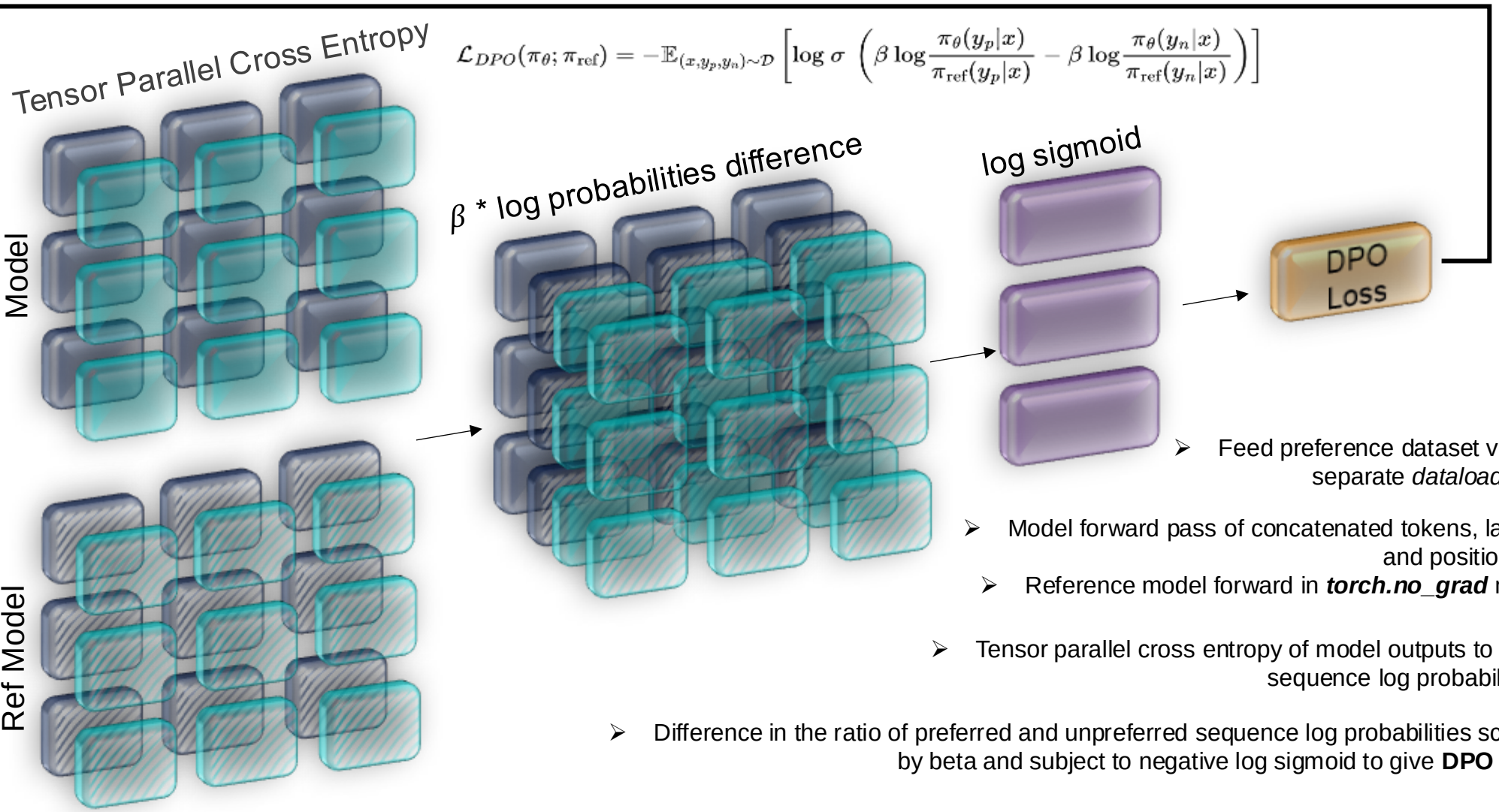


- Preference Optimization as a strategy to balance novelty of sequences vs. design constraints (e.g., GC content, catalytic activity, binding affinities)

Applications

- Enzyme redesign to improve catalytic activity with LLMs
- Antibody design with RF-Diffusion + feedback

DPO implementation in Megatron-DeepSpeed



- Feed preference dataset via 2 separate *dataloaders*
- Model forward pass of concatenated tokens, labels, and position IDs
- Reference model forward in ***torch.no_grad*** mode
- Tensor parallel cross entropy of model outputs to give sequence log probabilities
- Difference in the ratio of preferred and unpreferred sequence log probabilities scaled by beta and subject to negative log sigmoid to give **DPO loss**

Scaling across diverse supercomputing platforms for LLM training campaigns

System	Accelerator Specifications	Network Specifications

Aurora 	• Intel Data Center Max 1550 GPUs
	  • 128GB of memory
	  • Peak performance of 481 TFLOPS in FP16 and BF16
		• HPE Slingshot 11 using Dragonfly Topology
		  • 10624 nodes
		  • 6 GPUs per node

ALPS 	• NVIDIA Grace-Hopper (GH-200) processors
	  • Grace and Hopper with 128GB and 96GB of memory
	  • Peak performance of 900 TFLOPS in FP16 and BF16
		• HPE Slingshot 11 using Dragonfly Topology
		  • 2048 nodes
		  • 4 GPUs per node

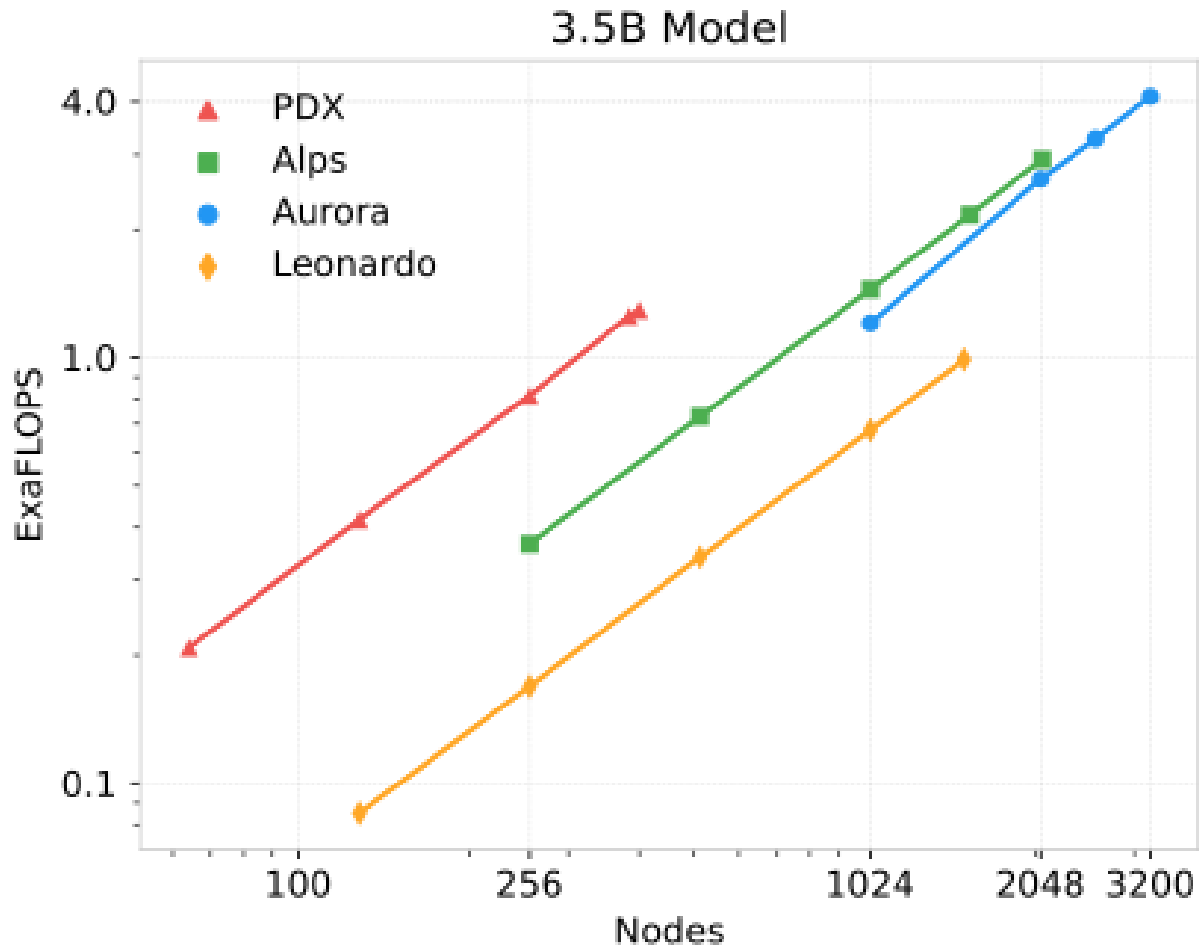
Frontier 	• AMD MI250X GPUs
	  • 128GB of memory
	  • Peak performance of 383 TFLOPS in FP16 and BF16
		• HPE Slingshot 11 using Dragonfly Topology
		  • 9408 nodes
		  • 4 GPUs per node

Leanardo 	• NVIDIA A100 SXM4 processors
	  • 64GB of memory
	  • Peak performance of 312 TFLOPS in FP16 and BF16
		• NVIDIA Mellanox Infiniband HDR with Dragonfly+
		  • 3456 nodes
		  • 4 GPUs per node

PDX 	• NVIDIA H100 processors
	  • 80GB of memory
	  • Peak performance of 835 TFLOPS in FP16 and BF16
		• Infiniband NDR
		  • 400 nodes
		  • 8 GPUs per node

Sustained Scaling of the 3.5B Model

- We achieve **4.11 EF (BF16) on Aurora, sustained performance**
- Linear scaling in throughput across all systems



- Sustained performance is for the entire training iterations, including the various communication intensive phases as well as I/O.

Efficiently launching and running PyTorch at scale

Collective operations at scale

Allreduce of 30k+ ranks with large buffers is nontrivial

- Ring-based reduce-scatter/all-gather is able to utilize the bandwidth well until a certain point, but does not scale indefinitely due to the latency component
- **We utilized [Rabenseifner's algorithm] to get better performance at large scale**

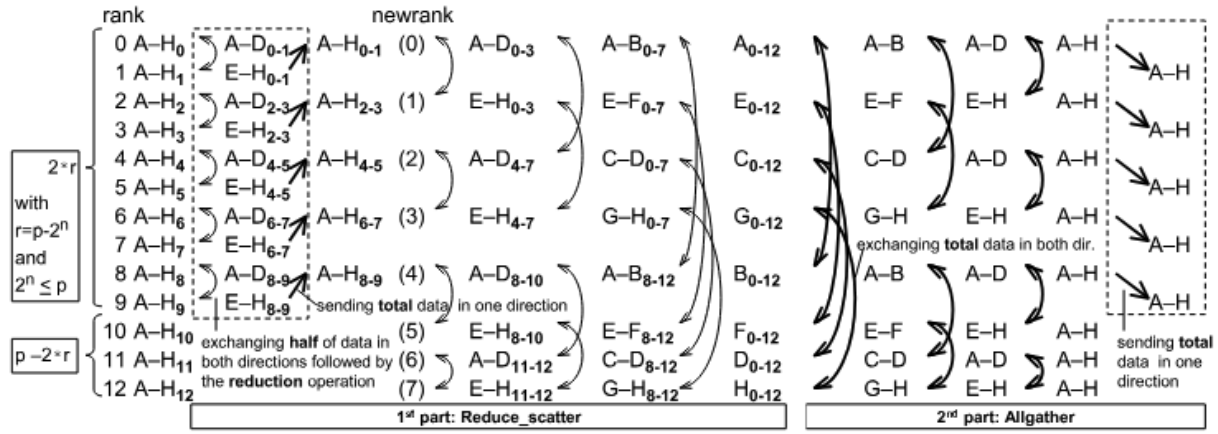
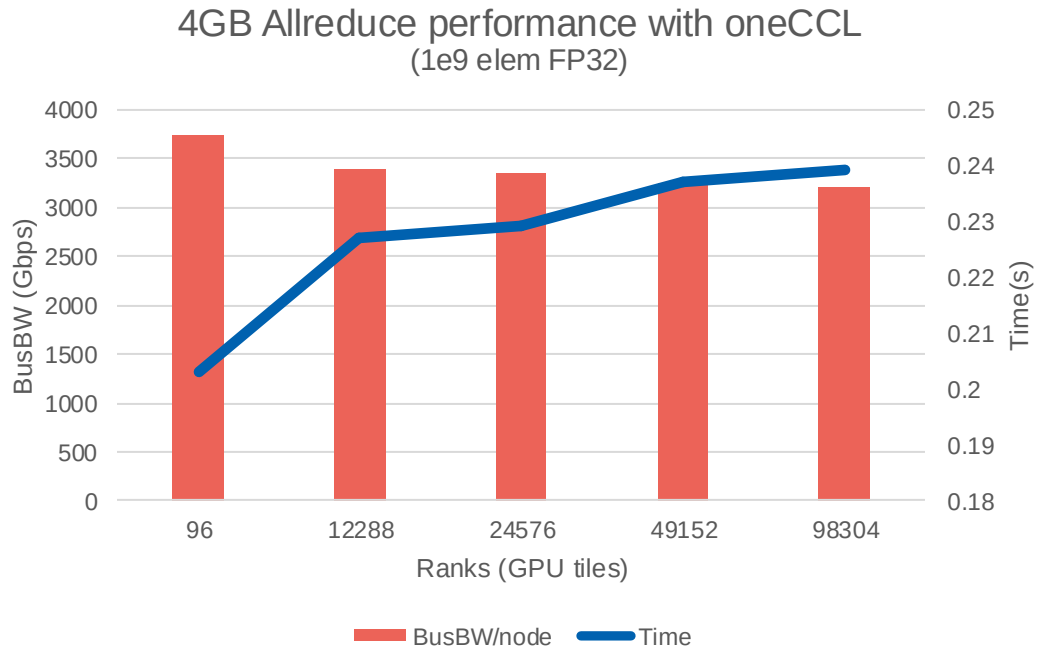


Fig. 1. Recursive Halving and Doubling. The figure shows the intermediate results after each buffer exchange (followed by a reduction operation in the 1st part). The dotted frames show the overhead caused by a non-power-of-two number of processes.

Collective operations at scale

Performance improvements since SC24:

- Discovery and evaluation of multiple environment variables for OneCCL and Libfabric to tune the low-level communication stack
  - Documented in ALCF docs
- Improvements in the network stack:
  - **PyTorch allreduce Tested up to 8192 nodes * 12 ranks**. We are able to complete an allreduce with good bandwidth utilization



85% scaling from 8 nodes to 8192

Computational details

Achieved 44.5% sustained mean FLOPS utilization (MFU) on Aurora

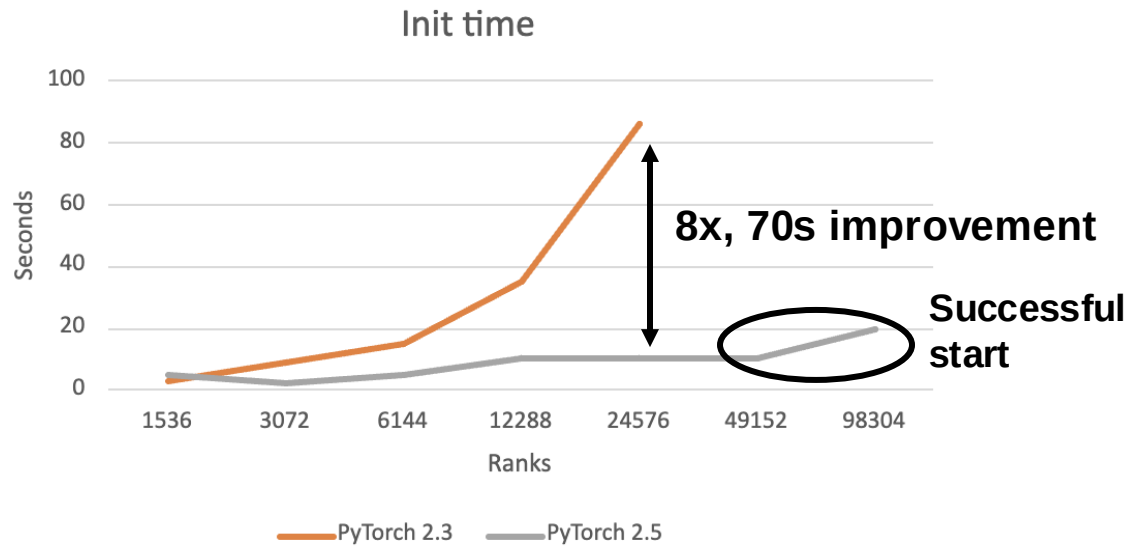
- Mixed precision compute.
  - FP16/BF16 compute
  - Gradient accumulation and sync in FP32 for numerical stability
- Aurora GPUs contain two compute tiles each
  - We used “tile as a device” configuration, recommended on Aurora for Deep Learning applications
  - 6 GPU cards per Node, 2 tiles per card -> 12 ranks per node
  - Achieved 107 TFLOPS per tile, **1280 TFLOPS per node**
- OneCCL communication library used for optimized collectives for AI workloads on Intel GPUs, similar to NCCL for Nvidia and RCCL for AMD

TABLE III: Peak and sustained performance of the best performing model on each system.

Machine	Nodes	# GPUs	Sustained EFLOPS	Peak EFLOPS	Sustained/Peak Ratio	% MFU (Sustained)
Aurora	3200	19200	4.11	5.57	0.73	44.5
Alps	2060	8240	2.92	3.16	0.92	41.7
Frontier	2048	8192	1.06	1.18	0.89	33.8
PDX	400	3200	1.29	1.39	0.93	48.4

Initializing PyTorch at scale

- torch.distributed init will start a TCP socket-based server on the first process that will communicate information about collectives and distributed setup
- **PyTorch 2.4 brings significant improvements to initialization times at scale**
- Our Gordon Bell software image was based on PyTorch 2.1
  - We made some hacks to be able to initialize the distributed environment for the scale runs
- Certain settings need to be tuned to allow thousands of socket connections



Need to configure:

- Number of return sockets
  - ulimit –n <somethingbig>
- Number of outstanding requests in the socket queue
  - Value set in /proc/sys/net/core/somaxconn
    We are investigating changing this globally on Aurora. Workarounds exist for now

Loading Python environments at scale

Package management tools such as conda can have tens of thousands of small files

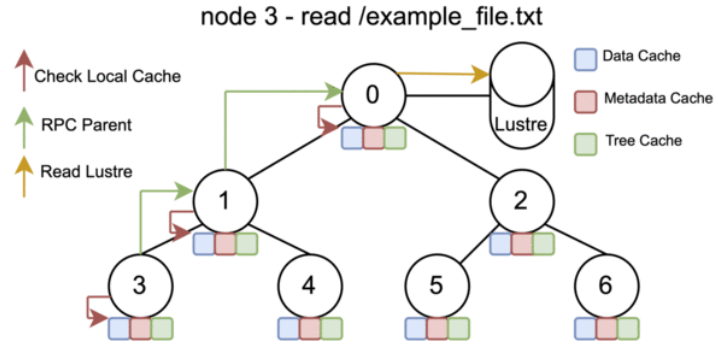
- Over 30000 files opened on 38400 ranks -> >900 million metadata operations can cause **filesystem stalls for all users** of the cluster

Many custom changes require an editable installation:

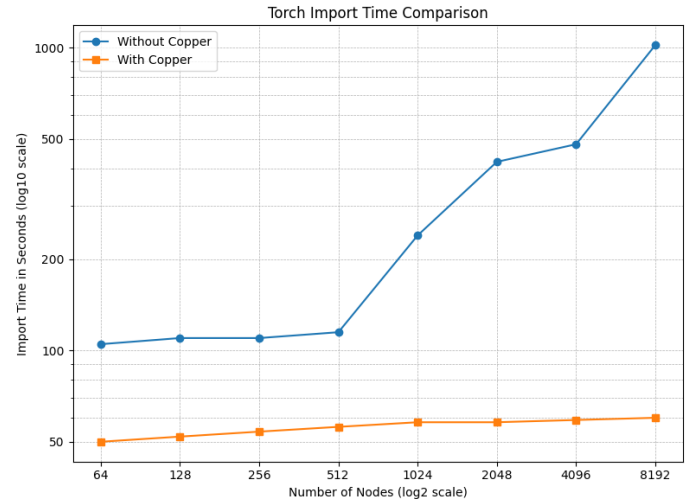
- As a quick solution for the Gordon Bell scaling runs, we packaged the environment and broadcasted it with mpi4py

Should not be an issue for normal usage:

- **Default environment is in the node image on Aurora**, does not need to load from the Lustre filesystem
- Copper (co-operative caching layer for scalable parallel data movement in Exascale Supercomputers) recommended for speeding up loading custom Python environments



Copper can be used at scale to reduce Lustre load



Credit: Kaushik Velusamy

Ongoing work

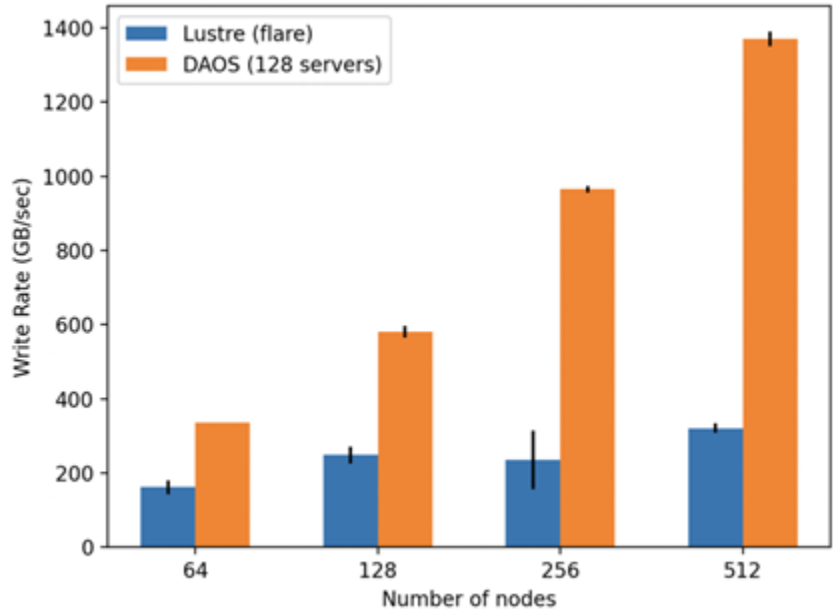
Fault tolerance and rapid saving or loading

Failures become more likely with increased scale

- LLM checkpointing with DAOS (Distributed Asynchronous Object Storage)
  - Total capacity: 230 PB
  - **Peak bandwidth: 30 TB/s** with 1024 DAOS server storage Nodes
- Universal checkpointing from DeepSpeed enables converting a Zero-enabled checkpoint from one scale to another

Upcoming work:

- How to integrate PyTorch fault tolerance with HPC
- How the communication libraries can handle fault tolerance and dynamic communicator groups



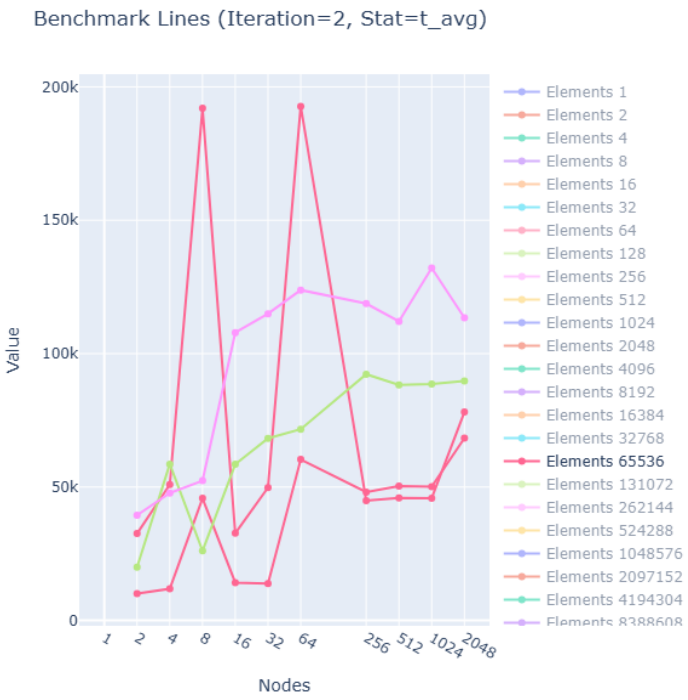
Checkpointing a 1T Model (17TB) Using DAOS: Achieved **1.4TB/s** throughput on 512 Aurora nodes with 128 DAOS servers. At 512 nodes, the checkpoint was written in ~13 seconds, significantly outperforming our 650 GB/s Lustre file system.

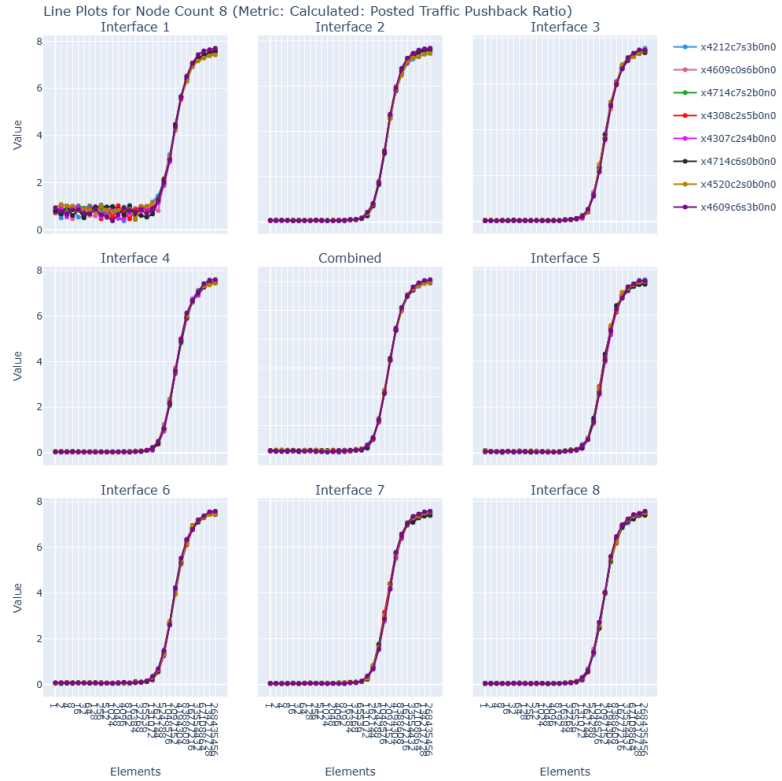
Credit: Huihuo Zheng

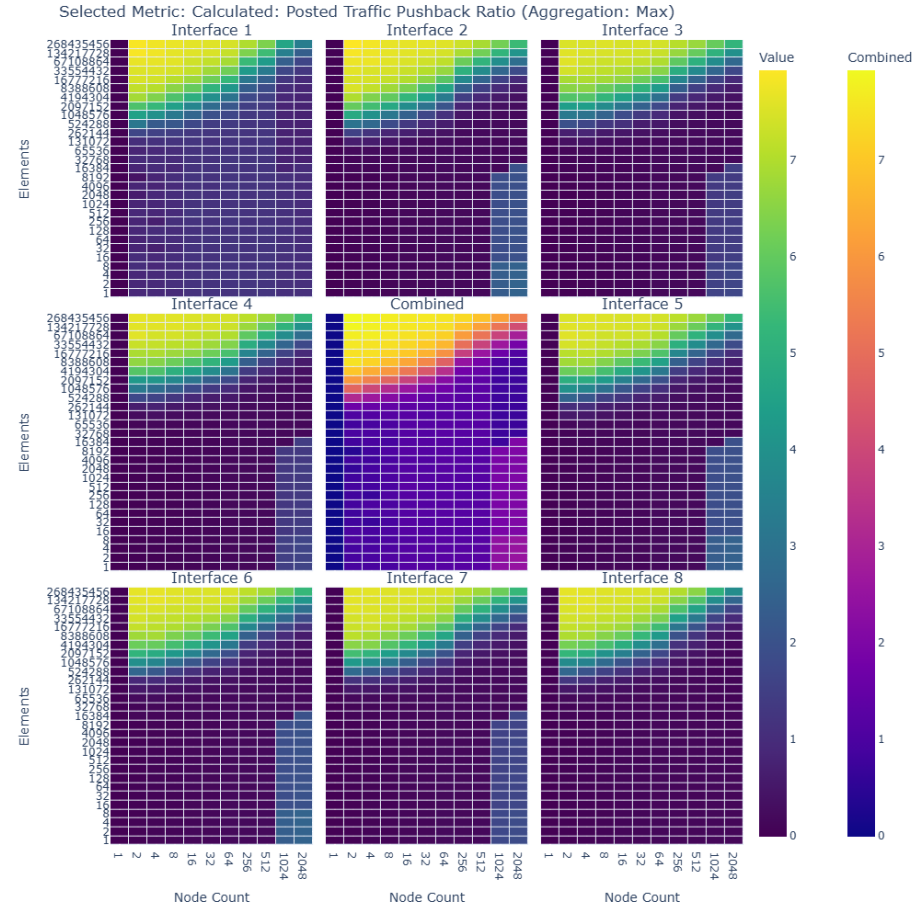
Investigating congestion and hangs with Slingshot Cassini NIC performance counters

Ongoing work with Intel and HPE to understand and detect hangs from NIC counters

- 1426 metrics to track, 8 NIC interfaces per node





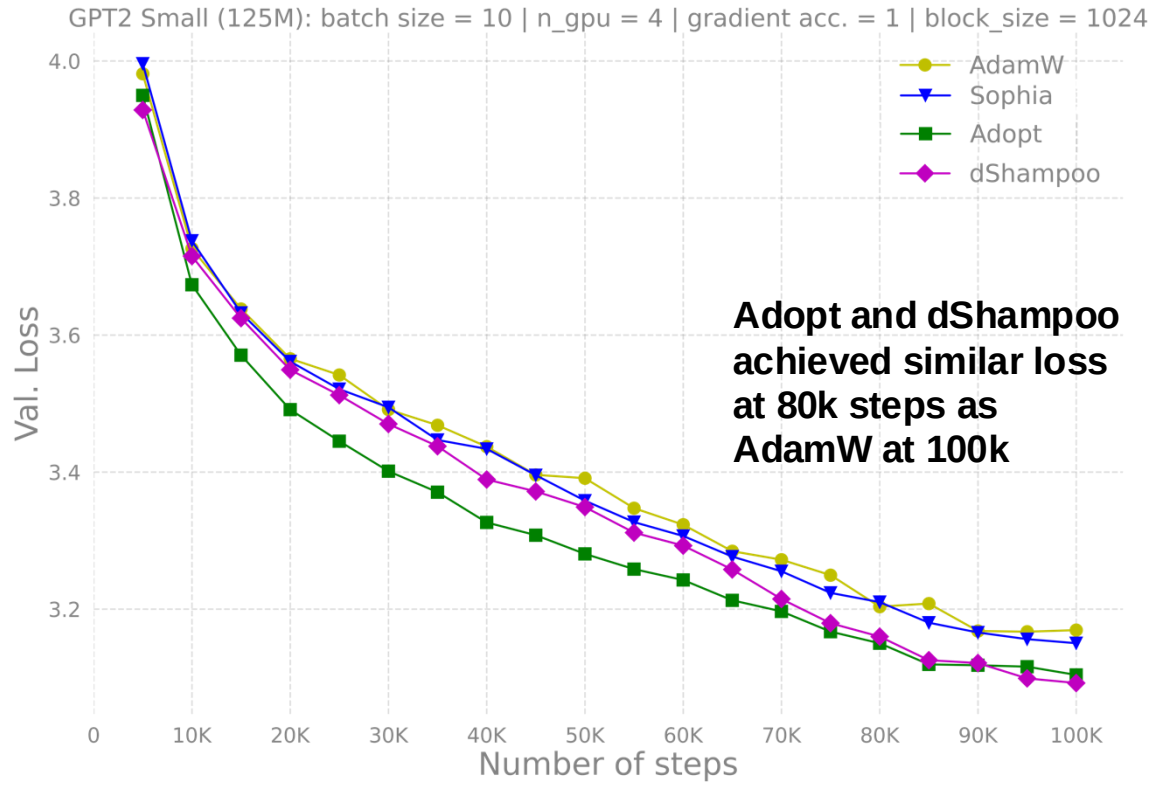


Credit: Nathan Nichols

Large batch training of LLMs

Techniques to scale the pre-training in production to large compute resources

- Larger batch size allows larger data parallel degree or reduce pipeline bubble by allowing increased gradient accumulation steps
- We are **evaluating multiple strategies to increase batch size**:
  - Learning rate scaling
  - Layer-wise adaptive learning rates (LARS,LAMB)
  - Increasing batch size as the training progresses
  - Second order optimizers

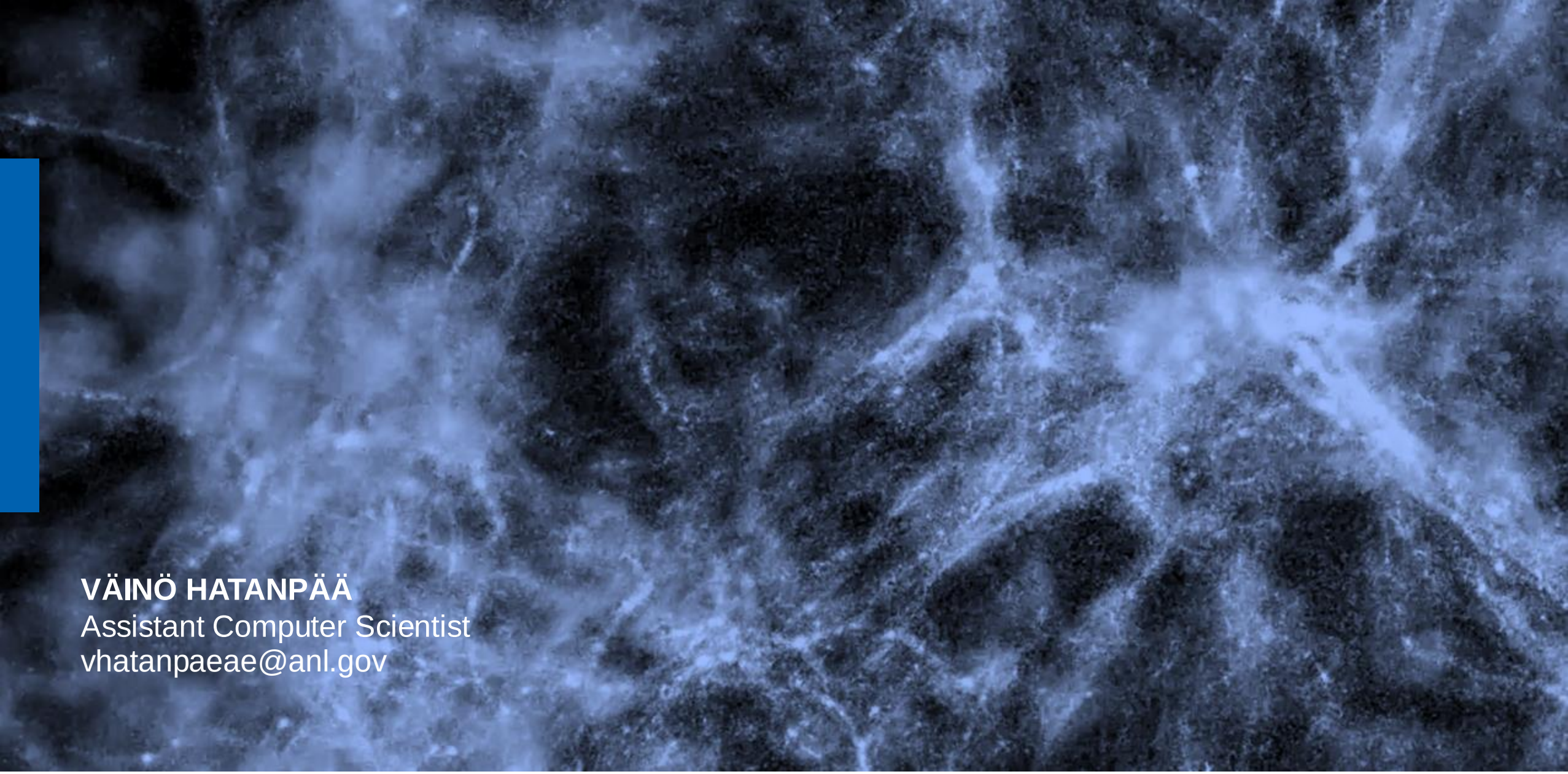


Credit: Marieme Ngom

Summary

- Gordon Bell 2024 finalist achieved 4.11exaFlops of BF16 throughput
- Increasing scale -> We have achieved progress in large scale initialization and collectives on Aurora
- Work ongoing for
  - fault tolerance
  - detecting network congestion and hangs
  - Large batch training

This research was funded in part and used resources of the Argonne Leadership Computing Facility (ALCF), a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357



VÄINÖ HATANPÄÄ
Assistant Computer Scientist
vhatanpaeae@anl.gov