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

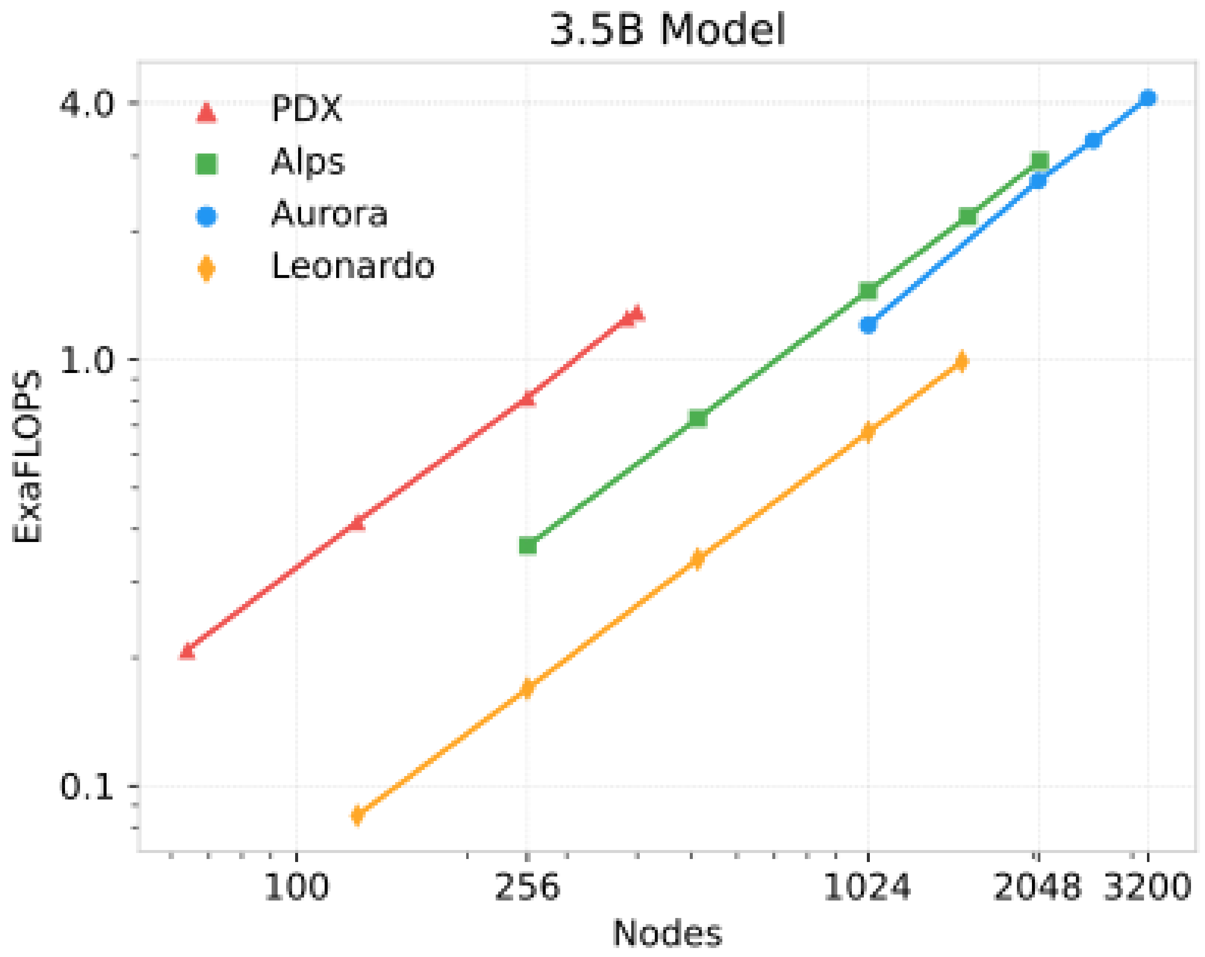
†Joint first authors, ∗Contact authors: **venkat@anl.gov, anima@caltech.edu, ramanathana@anl.gov**

SC24 Atlanta, GA | hpc creates.

Scaling on Aurora

1. Performance results
2. Software frameworks
3. LLM training, finetuning, and inference on Aurora
4. Handling Python packages at scale
5. Initializing PyTorch at scale
6. Collective communication at scale

Sustained Scaling of the 3.5B Model



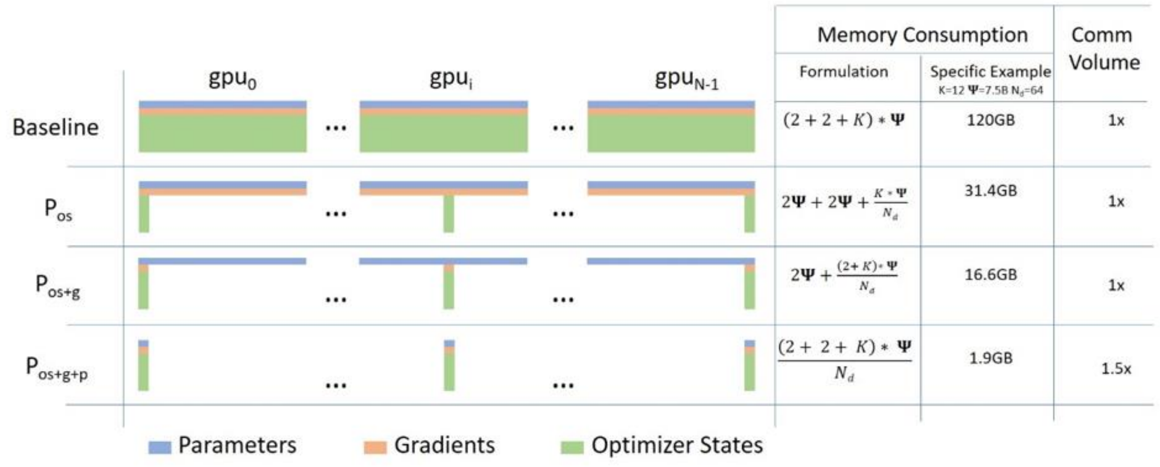
- We achieve 4.11 EF on Aurora sustained performance
- Linear scaling in throughput across all systems

- Sustained performance is for the entire training iterations, including the various communication intensive phases as well as I/O.
- Performance measured with an augmented DeepSpeed profiler to correctly account for model parameters

Megatron-DeepSpeed

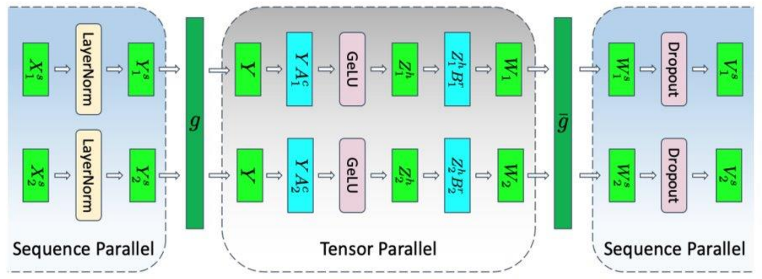
- Megatron-DeepSpeed is one of the most performant frameworks for training language models at trillion parameter scale
- Combines the 3D parallelism and fused CUDA kernels from NVIDIA's Megatron-LM with the ZeRO offloading of DeepSpeed
  - Kernels ported to XPUs to run on Aurora
- FlashAttention-2 was enabled to provide improved throughput
- We employed one rank per GPU-tile and tune the micro batch size (MBS) for performance at scale
- We used sequence lengths of 512 and 1024 as the target protein families are well captured within this range
- ALCF fork of Megatron-DeepSpeed at: https://github.com/argonne-lcf/Megatron-DeepSpeed

ZeRo optimizer



	Memory Consumption		Comm Volume
	Formulation	Specific Example K=12 Ψ=7.5B N_d=64	
Baseline	$(2+2+K)*\Psi$	120GB	1x
P_{os}	$2\Psi + 2\Psi + \frac{K*\Psi}{N_d}$	31.4GB	1x
P_{os+g}	$2\Psi + \frac{(2+K)*\Psi}{N_d}$	16.6GB	1x
P_{os+g+p}	$\frac{(2+2+K)*\Psi}{N_d}$	1.9GB	1.5x

Tensor Parallelism



NVIDIA/**Megatron-LM**

Ongoing research training transformer models at scale

NVIDIA

Performance details

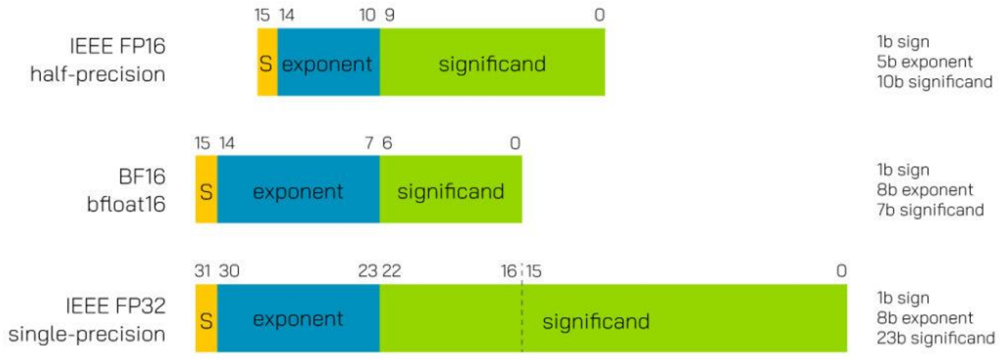
Most of the compute in lower precisions:

- FP16/BF16 depending on the system
- 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
- We did not use model-parallelism for the final results but multiple features are tested to work.

OneCCL communication library used for optimized collectives for AI workloads on Intel GPUs, similar to NCCL for Nvidia



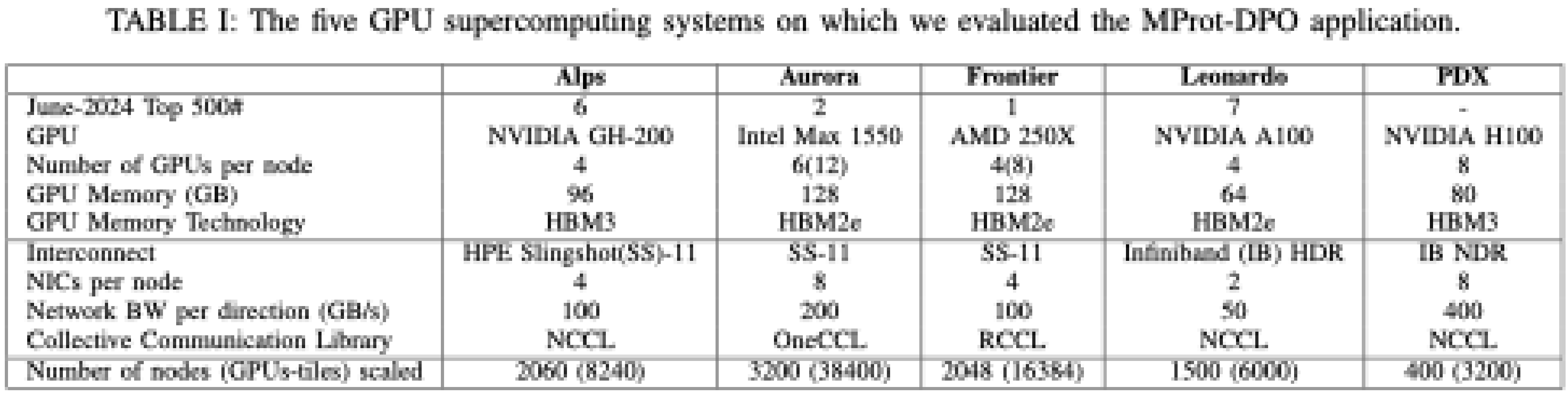


TABLE I: The five GPU supercomputing systems on which we evaluated the MProt-DPO application.

	Alps	Aurora	Frontier	Leonardo	PDX
June-2024 Top 500#	6	2	1	7	-
GPU	NVIDIA GH-200	Intel Max 1550	AMD 250X	NVIDIA A100	NVIDIA H100
Number of GPUs per node	4	6(12)	4(8)	4	8
GPU Memory (GB)	96	128	128	64	80
GPU Memory Technology	HBM3	HBM2e	HBM2e	HBM2e	HBM3
Interconnect	HPE Slingshot(SS)-11	SS-11	SS-11	Infiniband (IB) HDR	IB NDR
NICs per node	4	8	4	2	8
Network BW per direction (GB/s)	100	200	100	50	400
Collective Communication Library	NCCL	OneCCL	RCCL	NCCL	NCCL
Number of nodes (GPUs-tiles) scaled	2060 (8240)	3200 (38400)	2048 (16384)	1500 (6000)	400 (3200)

Other options for LLM training, finetuning, and inference on Aurora:

- This work used a fork of Megatron-DeepSpeed for best performance. Original Microsoft repository can also be used on Aurora.
- PyTorch FSDP-based solutions also tested to work on Aurora, recommended for small to medium scale training and finetuning
- Multiple other finetuning and inference libraries tested and work is ongoing to support more

Detailed instructions for AI workloads at ALCF docs:

- https://docs.alcf.anl.gov/aurora/data-science/frameworks/pytorch/
- https://docs.alcf.anl.gov/aurora/data-science/frameworks/megatron-deepspeed/
- https://docs.alcf.anl.gov/aurora/data-science/inference/libtorch/

Loading Python environments at scale

Package management tools such as conda can have tens of thousands of small files and Python imports can iterate over many of them.

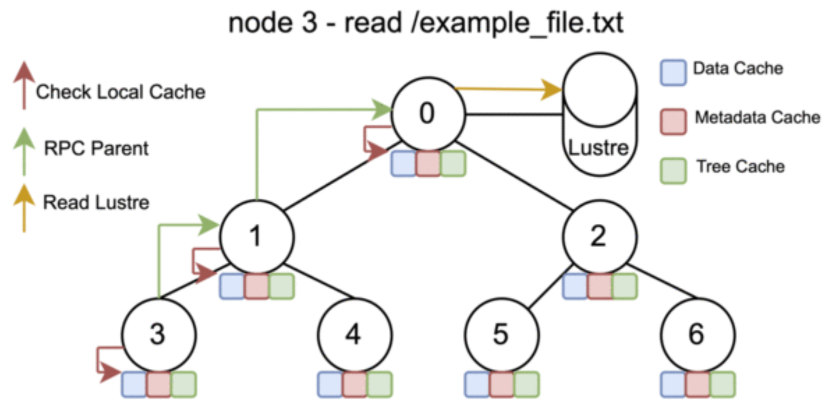
- Over 30000 files opened on 38400 ranks ->
- >900 million metadata operations can cause Filesystem stalls for all users

Many custom changes required an editable installation:

- As a quick solution we packaged the environment and broadcasted it with mpi4py

Should not be an issue for normal usage:

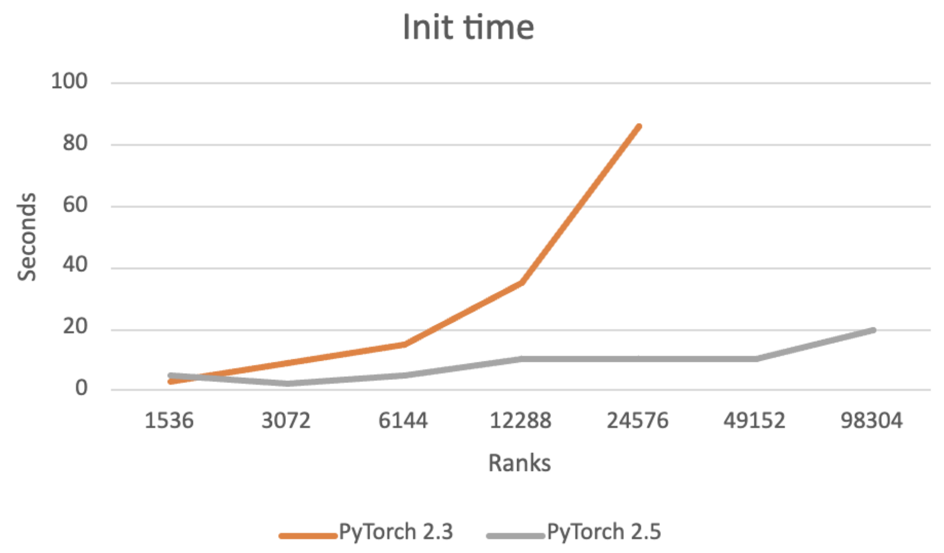
- Frameworks module default environment is in the node image, does not need to load from the Lustre filesystem
- Copper (co-operative caching layer for scalable parallel data movement in Exascale Supercomputers) recommended for speeding up loading Python environments



Copper can be used at scale to reduce Lustre load

Initializing PyTorch at scale

- PyTorch will start a TCP socket-based server on the first process that will communicate information about collectives and distributed setup
- 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

Collective operations at scale

Allreduce of 30k+ ranks with large buffers is nontrivial

- Ring allreduce is able to utilize the bandwidth well until a certain point, but does not scale infinitely due to the latency component
- We utilized Rabenseifner's algorithm to get better performance at large scale

Performance improvements since:

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

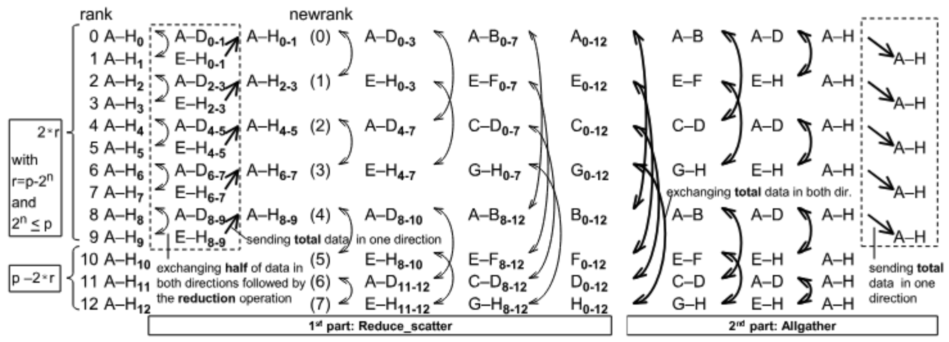

rank
newrank
exchanging half of data in both directions followed by the reduction operation
sending total data in one direction
exchanging total data in both dir.
sending total data in one direction
1st part: Reduce_scatter
2nd part: Allgather

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.

Optimization of Collective Reduction Operations, Rolf Rabenseifner

Summary

- We achieve 4.11 EF on Aurora sustained performance
- Linear scaling in throughput across all systems

Performance lessons in:

- Handling Python environments
  - Do not store conda environments on Lustre when launching at scale
- Initializing PyTorch distributed setup
  - PyTorch 2.5 brings significant improvements
  - Socket limits need to be configured
- Large scale collectives
  - Algorithms matter
  - Work is ongoing for defining optimal environmental variables in the modules we provide

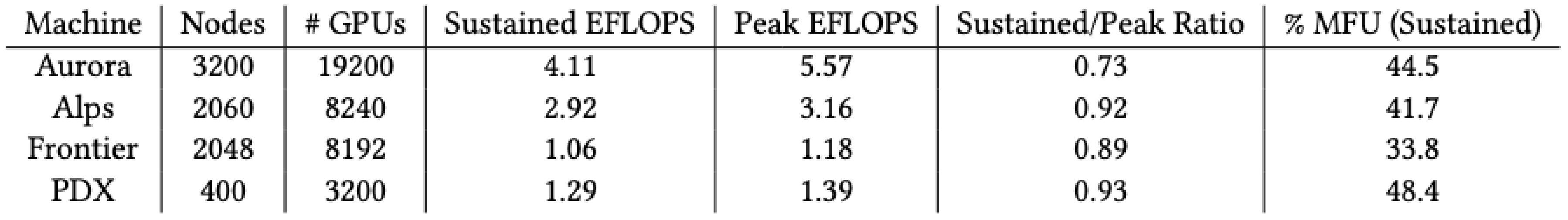


Table 3: Peak and sustained performance of the best performing model on each system.

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