

#### ALCF INCITE GPU Hackathon May 20-22, 2025

# Pytorch profiler for Al

Huihuo Zheng May 7, 2025

Argonne Leadership Computing Facility

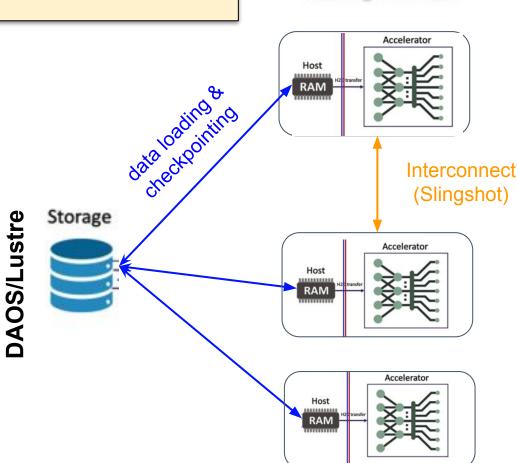
# What is profiling and why?

Profiling is the process of analyzing a program's execution to identify performance characteristics and bottlenecks.

- Data loading
- Computation
- Memory access
- Communication.

#### **Option of profilers on Aurora**

- TAU
- HPCToolKit
- Intel VTune/APS/Advisor
- mpitrace (communication)
- Darshan (I/O)
- Iprof/unitrace
- torch.profiler (Comp & comm)
- DFTracer (I/O)







# What information can PyTorch profiler provide

- Operator Execution Time (CPU time & XPU time)
- Kernel Execution Details (XPU): The specific compute kernels launched on the device, including their duration
- **Operator Input Shapes:** By setting record\_shapes=True
- Stack Traces and Module Hierarchy: Enabling with\_stack=True allows the profiler to record the Python source code location (file and line number) that invoked each operation.
- Estimated FLOPs: For certain common operators like matrix multiplication and 2D convolution, the profiler can estimate the number of floating-point operations (FLOPs) performed if with\_flops=True is set.<sup>9</sup> This can help in assessing the computational intensity of different parts of the model.
- **Execution Timeline (Trace View):** Perhaps the most powerful feature for detailed analysis is the ability to export a chronological trace of events.



# **Example of using PyTorch Profiler on Aurora**

# output timeline trace (important to have different files for different rank)
os.makedirs(args.trace\_dir, exist\_ok=True)
prof.export\_chrome\_trace(f"{args.trace\_dir}/torch-trace-{rank}-of-{world\_size}.jso
n")

CLASS torch.profiler\_KinetoProfile(\*, activities=None, record\_shapes=False, profile\_memory=False, with\_stack=False, with\_flops=False, with\_modules=False, experimental\_config=None, execution\_trace\_observer=None, acc\_events=False, custom\_trace\_id\_callback=None) [SOURCE]

Low-level profiler wrap the autograd profile

#### Parameters

- activities (*iterable*) list of activity groups (CPU, CUDA) to use in profiling, supported values: torch.profiler.ProfilerActivity.CPU, torch.profiler.ProfilerActivity.CUDA, torch.profiler.ProfilerActivity.XPU. Default value: ProfilerActivity.CPU and (when available)
   ProfilerActivity.CUDA or (when available) ProfilerActivity.XPU.
- record\_shapes (bool) save information about operator's input shapes.
- profile\_memory (bool) track tensor memory allocation/deallocation (see export\_memory\_timeline for more details).
- with\_stack (bool) record source information (file and line number) for the ops.
- with\_flops (bool) use formula to estimate the FLOPS of specific operators (matrix multiplication and 2D convolution).
- with\_modules (bool) record module hierarchy (including function names) corresponding to the
  callstack of the op. e.g. If module A's forward call's module B's forward which contains an aten::add op, then
  aten::add's module hierarchy is A.B Note that this support exist, at the moment, only for TorchScript
  models and not eager mode models.
- experimental\_config (\_ExperimentalConfig) A set of experimental options used by profiler libraries like Kineto. Note, backward compatibility is not guaranteed.
- execution\_trace\_observer (ExecutionTraceObserver) A PyTorch Execution Trace Observer object.
   PyTorch Execution Traces offer a graph based representation of Al/ML workloads and enable replay benchmarks, simulators, and emulators. When this argument is included the observer start() and stop() will be called for the same time window as PyTorch profiler.
- acc\_events (bool) Enable the accumulation of FunctionEvents across multiple profiling cycles

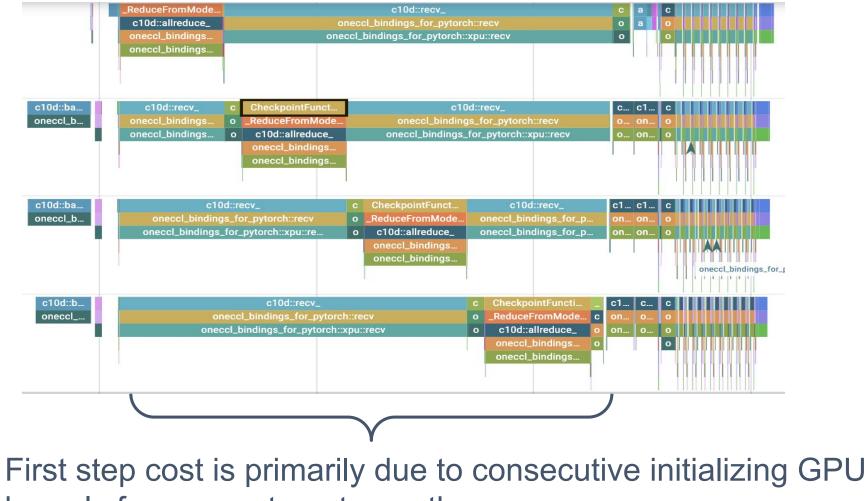


#### Timeline trace visualization: <u>https://ui.perfetto.dev/</u>

	.dev/#!/viewer?local_cache_key=0000	00000-0000-0000-aff7-284	ed573a8f2								* 🗋 🖻 🔮	요   초 후	0
🟠 Perfetto 🛛 😑							nds or ':' for SQL mode						
avigation	· · · · · · · · · · · · · · · · · · ·	1 1 1 1 1 1 1 1 1 1 2:10	T <sub>90.90.20</sub>	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	1 1 02:00:40	× · · ·   00:00:50	1 1 1 1 1 1 1 00:01.00	r 1 r 1 r 1 r 1 00:01:10	1 1 1 1 1 00.01.20	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	1 1 1 1 1 1 <sub>00</sub>		
<ul> <li>Open trace file</li> <li>Record new trace</li> </ul>	35d23-58:14+ 633 816 676	00 20	1:01:45 10:000:009	00:01:45 400.000.000	0:01:45 60.000.003	00:01:45 800:000:000	00:01:46 000:000 000 	00:01:46 200 000 000	00:01:46 400:000:000	0:01:45 00:000 000 800 000	1346 00 000	00:01:47 009 009 000	
	x = Y Default Workspace												
rrent Trace	thread 11., n3) 113540 (main thread)	c10d:alireduce_	dee	a.	e1_	d <b>Harrison and</b>	d_ c10		d c10_			d c10_	
ch-trace-0-of-12.json (315 MB)		oneccl_bindings	dee		on_		d_ cne_		d one_ d one_			d one d one_	
Show timeline			tor							in an			
Download			tor	1. I.I.I.I.I.I.I.I.I.I.I.I.I.I.I.I.I.I.I			6						
Query (SQL)			on										
) Metrics ) Info and stats												11414	
into and stats													
nvert trace	Thread D thread 117239 (python3) 117239												
								A AA A		an de la companya de			
Switch to legacy UI Convert to .json													
Content to gaon													
ample Traces ^	∽ python3 0	1	Å	X		*	X	Å	Å	X		Å	
Open Android example	Thread 0 (main thread)			·			threading.py(973): _bootstrag		91				
Open Chrome example		threading av/1016/ botterns inner											
							threading.py(953): run torch/_inductor/compile_worker/subproc_pool torch/_inductor/compile_worker/subproc_pool	y(153): _read_thread .py(47): _recv_msg					
oport ^								· · · · · · · · · · · · · · · · · · ·					
Keyboard shortcuts													
Documentation	Thread 1												
Flags	Thread 2	11			1								
Report a bug	Thread 3		1			1		1	1			1	
Record metatrace	Current Selection												Ť
Plugins	Slice oneccl_bindings_for_pytorch::xpu::a	llreduce										Contextual O	Options
	Details						Arguments						
	Name oneccl_bindings_for_pyton Category cpu_op	ch::xpu::alfreduce					✓ args Concrete Insute[0] -						
	Category         Cpu, op           Start time         0.001.46.052.420.759           > Duration         Timead           Thread         thread 113540 (python3) (113540)					Concrete inputs(0) - Ex.ldx - 343793							
					External.ld - 107252 Input Dims[0][0] - 126674432								
	Process python3 [113540]					Input Strides[0][0] - 1	0::Half						
	SQL ID slice[241410] -						Record function id - 0						



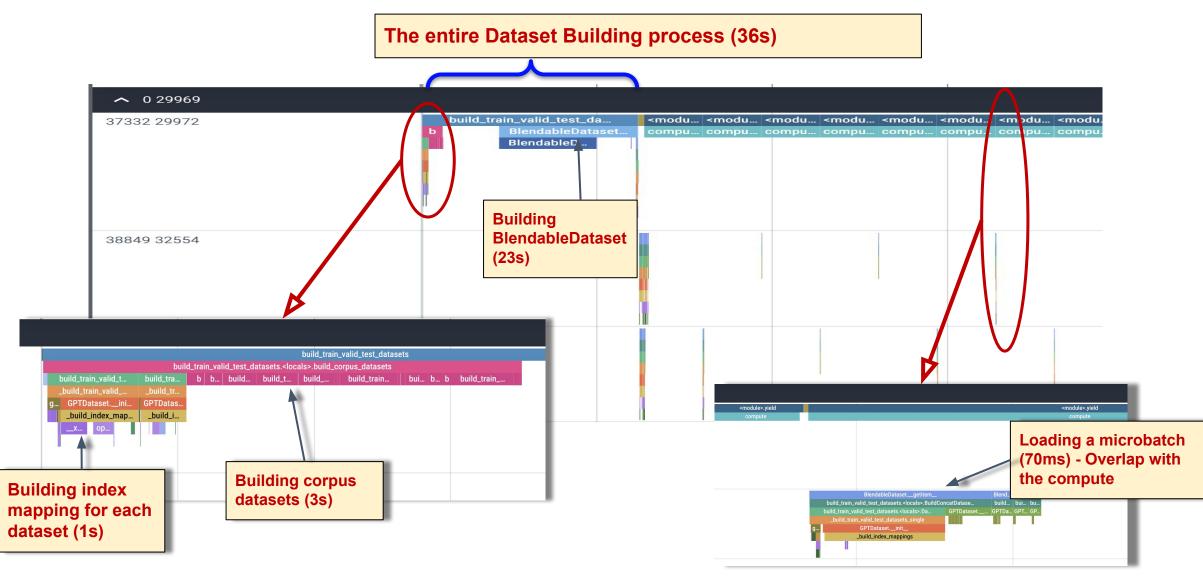
# Case 1: First step large overhead in pipeline parallelism



kernels from one stage to another



# **Case 2: I/O in Megatron-DeepSpeed**





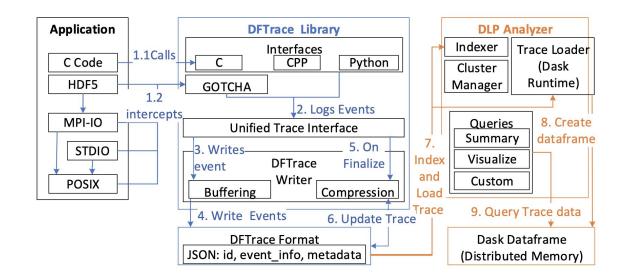
#### Existing issues for torch.profiler on Aurora

When profiling on Intel XPU backends, enabling ProfilerActivity.XPU can occasionally lead to hangs in multi-process or multi-node runs. To avoid this issue:

- **Single-Node Profiling:** Use activities=[ProfilerActivity.CPU, ProfilerActivity.XPU] and collect a full trace on **one** node (or a single process per node) to capture XPU kernel timings safely without multi-node synchronization issues .
- Multi-Node Profiling: For distributed workloads spanning multiple nodes, omit ProfilerActivity.XPU and rely solely on CPU events (ProfilerActivity.CPU) to prevent hanging barriers in the Kineto profiler's XPU hooks.
- ./test\_dtensor\_1d.py works fine for multiple node with ProfilerActivity.XPU
- ./test\_miniGPT.py hangs on 2+ nodes

#### **DFTracer - A multilevel I/O profiler with application context**

- Works with Python applications (multiple threads)
- Able to profile I/O in the context of application functions
- Able combine with results from other tools
- With small overhead



https://github.com/LLNL/dftracer.git

DFTracer: An Analysis-Friendly Data Flow Tracer for AI-Driven Workflows.

Devarajan, H., L. Pottier, K. Velusamy, H. Zheng, I. Yildirim, O. Kogiou, W. Yu, A. Kougakas, X.-H. Sun, J. S. Yeom, and K. Mohror, SC24: International Conference for High Performance Computing, Networking, Storage and Analysis



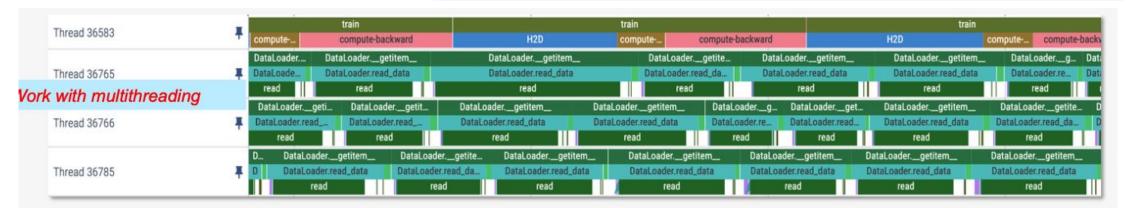
#### **Multilevel profiling feature in dftracer**

from dftracer.logger import dft\_fn as Profile
dlp\_data = Profile("DataLoader")
class DataLoader(datasets.ImageFolder):
 @dlp\_data.log
 def preprocess(self, sample, target):
 if self.transform is not None:
 Easy to use
 sample = self.transform(sample)
 if self.target\_transform is not None:
 target = self.target\_transform(target)
 return sample, target

@dlp\_data.log def read\_data(self, index): path, target = self.samples[index] return self.loader(path), target

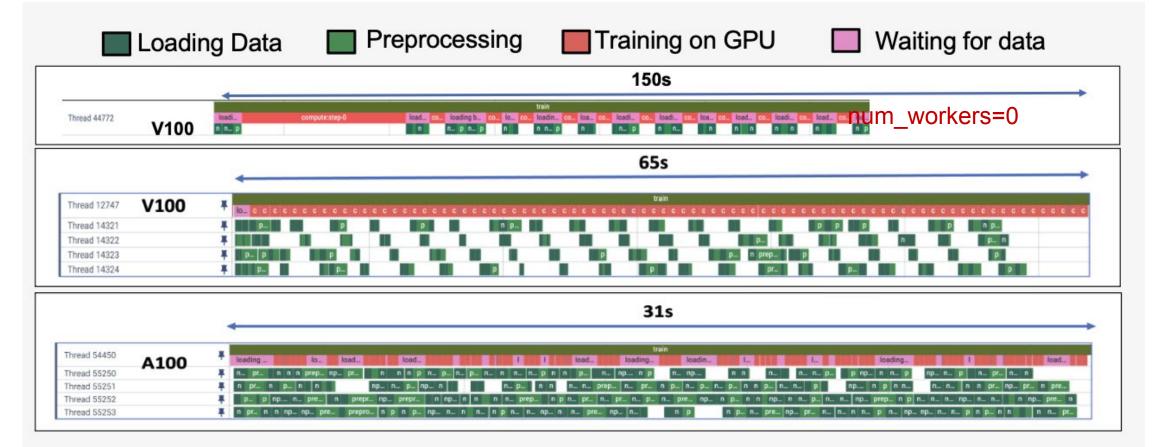
@dlp\_data.log def \_\_getitem\_\_(self, index): sample, target = self.read\_data(index) return sample, target

Name	Wall duration (ms)	
	400698.238	
DataLoadergetitem	142903.235	
DataLoader.read_data	134743.702	
read	110562.695	
DataLoader.preprocess	7762.795	
close	2356.922	
open64	2335.804	
fxstat64	22.914	
lseek64	10.171	





#### I/O bottleneck for UNet3D shown through dftracer



Timeline trace for training the UNet3D workload on a single GPU on JLSE@ALCF.



#### Performance issue revealed through profiling

_train.iter	_train.iter	_train.iter	_train.it_	DLIOBend DLIOBend _train_l
Tor., TorchD., Torch., TorchData., HDF, HDFSRe, HDFSR, HDFSReade., H, HDFS, HDF, HDFSRea., p., pread pre., pread	TorchDa_ TorchD_ HDFSRea_ HDFSRe HDFSR_ HDFSR pread pread	TorchDa TorchD. HDFSRea HDFSR HDFSR HDFSR pread pread	To Torch HD HDFSR H HDF5 p pread	TorchDat HDF5Read HDF5Re
HDF_ HDFSReade_ HDF5Rea_ HDF	SReader read HDF	statute and a statute and a second	Read_HDF5H	IOFSR. HD HDFS. H pre. P
Tore Tore Toreh Toreh HDF5_HDF5_HDF5R_HDF5R_ HD_HD_HDF_HDFH Pfprprprepre	HDFSR_ HDF_ H	DFS_ HDFS_ HDFS HDF_ HDF_ HDF	Datase Torc Reader HDF5 SReade	Torch T HDFSR H HDF
TorchDTorchDTorchDatTorchT H0F5ReHDF5ReadHDF5R H0F5RH0F5ReH0F5Re pread_preadpreadpreadpreadpreadpreadpread_preadpreadpreadpreadpreadpreadpread_pread_preadpreadpreadpread_pr	Rea HDF5Re HDF SRe HDF5. HD	hDa TorchDatase 5Re HDF5Reader F5R HDF5Reade read pread	HDF5Read HD	chDatase FSReader DFSReade pread



#### PyTorch

Name	Wall duration (ms)		
	20049.987		
TorchDatasetgetitem	5042.087		
HDF5Reader.read_index	5030.107		
HDF5Reader.get_sample	4202.46		
pread	4066.861		

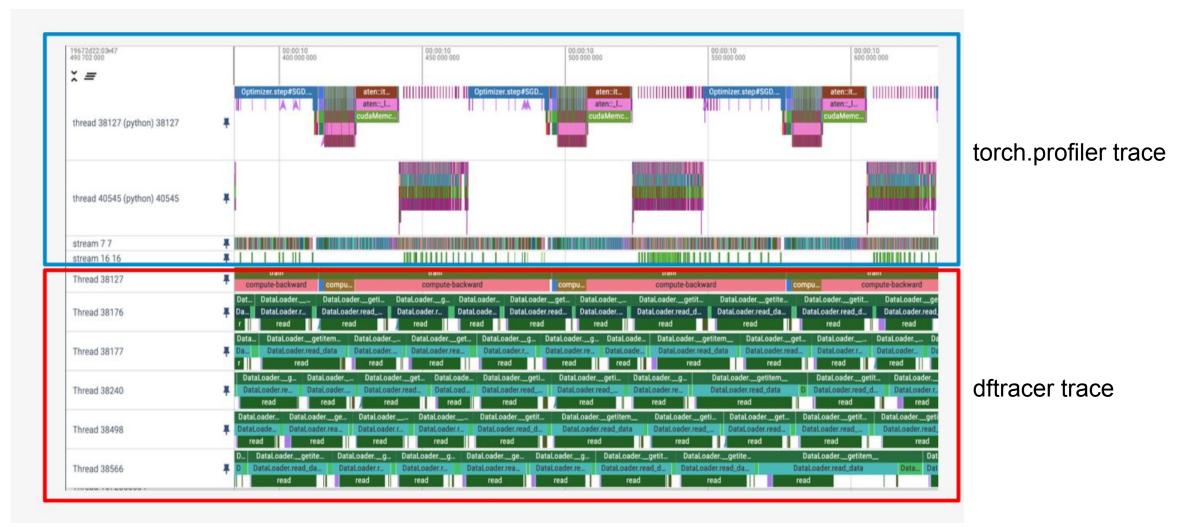
Name	Wall duration (ms)				
	17441.186				
HDF5Reader.get_sample	7735.878				
HDF5Reader.open	3764.007				
pread	3134.925				
HDF5Reader.close	2466.615				

TensorFlow

Performance comparison of Torch DataLoader and tf.data for HDF5 datasets. TensorFlow data loader performs worse for HDF5 because of thread locking issue



#### Combining data loading trace with compute trace





#### **Current existing issues for dftracer on Aurora**

Aurora-specific workaround: The POSIX-level I/O interceptor in DFTracer can hang on Aurora's file system. If you encounter hangs during low-level POSIX tracing, disable it by setting:

export DFTRACER\_DISABLE\_IO=1

This turns off the problematic POSIX I/O hooks while still allowing higher-level function and MPI-IO tracing to proceed normally.



# Merging multiple timeline trace

#### Install utils

pip install git+https://github.com/zhenghh04/pyutils

#### Merging traces from different ranks

merge\_trace --inputs ./torch-trace-0-of-24.json ./torch-trace-12-of-24.json --output torch-trace-combine.json

#### Merging traces from different profilers (alignment might be needed)

merge\_trace --inputs ./torch-trace-0-of-24.json ./trace-0-of-24.pfw --output trace-0-of-24-combine.json



#### Hands on examples

$\leftrightarrow$ $\rightarrow$ C $\sim$ github.com/argonne-lcf,	/GettingStarted/tree/incite-hackath	on-2025/Profilers/pytorch_profiler 📮 🛱 🗍 🔞 🛞 🖸	* * 🚳 :					
Files	GettingStarted / Profilers / p	ytorch_profiler /	个 Тор					
	ြ miniGPT.sc	added dtensor example	1 hour ago					
<sup>₽</sup> incite-hackathon-2 + Q	requirements.txt	added pytorch profiler	3 hours ago					
Q Go to file	test_dtensor_1d.py	remove link	now					
> AI_frameworks	test_miniGPT.py	added pytorch profiler	3 hours ago					
> 📄 Aurora_Onboarding			21					
> 🛅 Compiling_and_Running	torch_setup.py	added pytorch profiler	3 hours ago					
> 🖿 DAOS	README.md		∅ :≡					
> 🛅 Debuggers	README.IIId		<i>ν</i> :=					
Profilers								
∽	P	rofiling Python applications with torch.profiler and dftracer						
> 📄 figures								
🗅 README.md		hor: Huihuo Zheng, <u>huihuo.zheng@anl.gov</u> red: May 7, 2025						
🗋 ds_config.json								
dtensor.sc		We here introduce two useful profiling tools for profiling Python applications, one is Torch profiler for compute and communication profiling,						
🗋 miniGPT.sc	and	and dftracer for I/O profiling. One can combine the traces from both profilers to get a holistic view of the application execution.						
requirements.txt	PyTorch Profiler							
test_dtensor_1d.py								
🗋 test_miniGPT.py		orch profiler and measure the time and memory consumption of the model's operators. It includes simple profiler API that is useful when ar needs to determine the most expensive operators in the model.						
🗋 torch_setup.py								
🗋 .gitkeep	1	. Import all necessary libraries						
> 📄 Workflows		from torch.profiler import profile, record_function, ProfilerActivity						
.gitignore								
🗋 .gitkeep	2	. Using profiler to analyze execution time PyTorch profiler is enabled through the context manager and accepts a number of parameters,						
🗋 README.md		some of the most useful are:						
		activities - a list of activities to profile:						

git clone -b incite-hackathon-2025 https://github.com/argonne-lcf/GettingStarted



# Acknowledgments

This research used resources of the Argonne Leadership Computing Facility, a U.S. Department of Energy (DOE) Office of Science user facility at Argonne National Laboratory and is based on research supported by the U.S. DOE Office of Science-Advanced Scientific Computing Research Program, under Contract No. DE-AC02-06CH11357.