

Monitoring GPU-accelerated Deep Learning models



Norwegian research infrastructure services

Hicham Agueny, PhD
Scientific Computing Group
IT-department, UiB/NRIS

Motivation

- It does not seem that my code runs on **GPU** [Graphics Processing Unit].
- I don't see any **performance gain** when running on GPU.
- It seems my code does something strange;
 - How can I check what is going on ?
 - How can I **identify "hot spots"** in my code ?
 - **Would it be relevant** to run my code on GPU?

Outline

I. Overview of GPU-architecture [GPU: Graphics Processing Unit]

II. Basic tools for GPU-usage

III. Profiling Deep Learning with PyTorch Profiler

- What is PyTorch Profiler ?
- How to setup PyTorch Profiler?
- **Demo:** How to use PyTorch Profiler ?

Performance of a computer

Supercomputer

- The performance of a processor is measured by the quantity:

FLOPS (Floating-Point Operations Per Second).

- It is a measure of the speed of a computer to perform arithmetic operations.



Cluster



Computer
(or node/server)



Performance of a computer

Supercomputer

- The performance of a processor is measured by the quantity:

FLOPS (Floating-Point Operations Per Second).

- It is a measure of the speed of a computer to perform arithmetic operations.



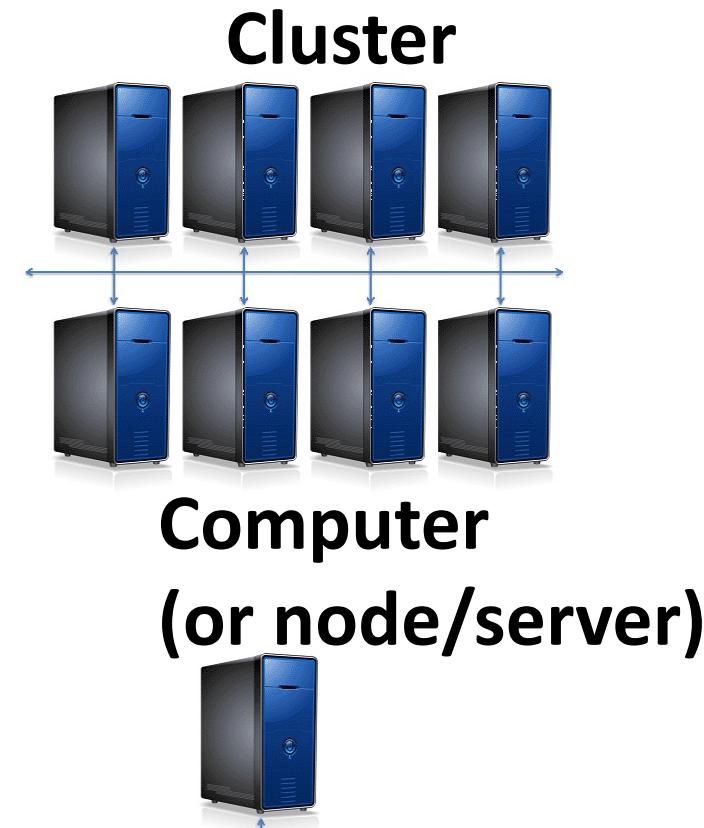
For a single processor:

$$\text{FLOPS} = (\text{Clock speed}) \times (\text{cores}) \times (\text{FLOPs/cycle})$$

=Peak performance

FLOP is a way of encoding real numbers (i.e. FP64 or FP32, FP16...)

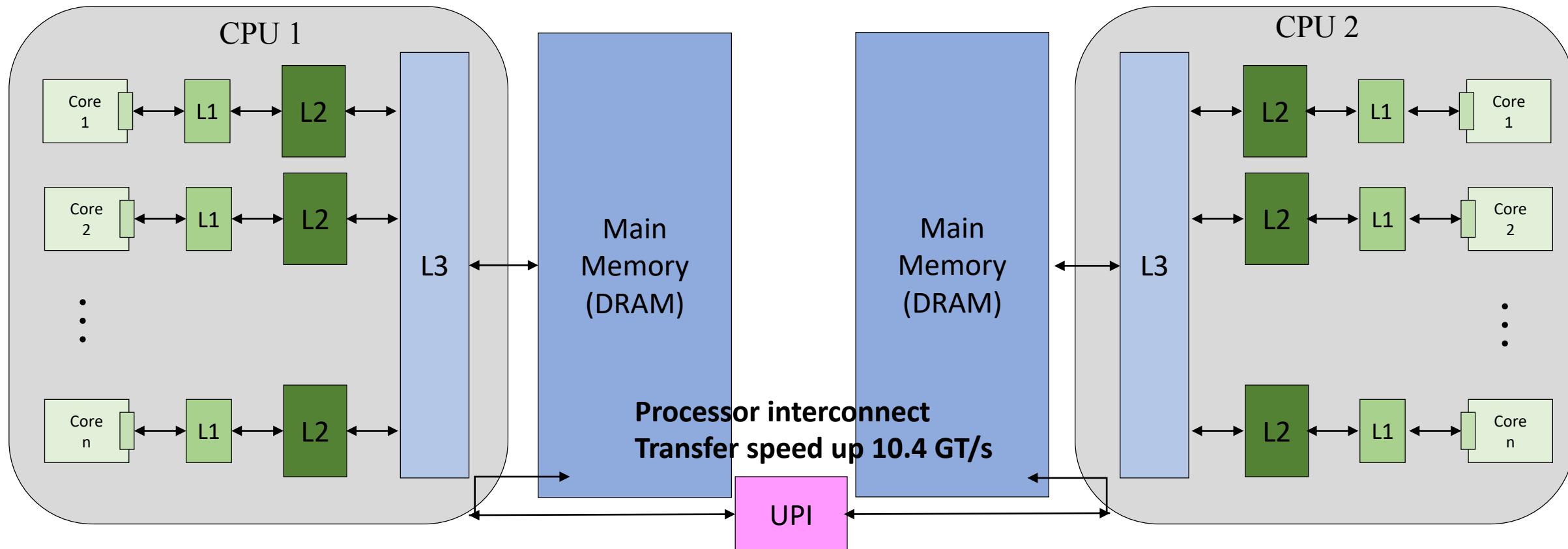
Exp. 1 PetaFLOPS = 10^{15} calculations per second.



CPU-Architecture

CPU-Architecture

Dual Socket Intel Xeon CPU

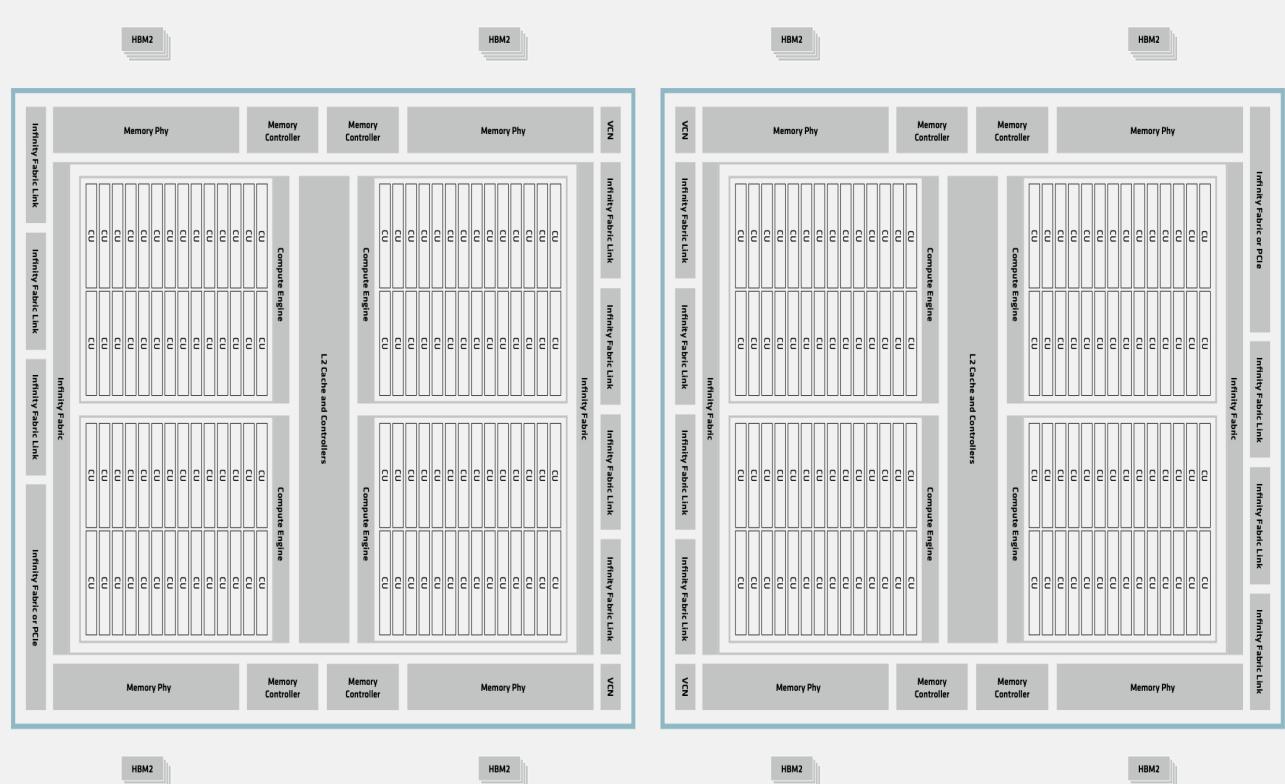


Access L1
Memory is
Faster than L2

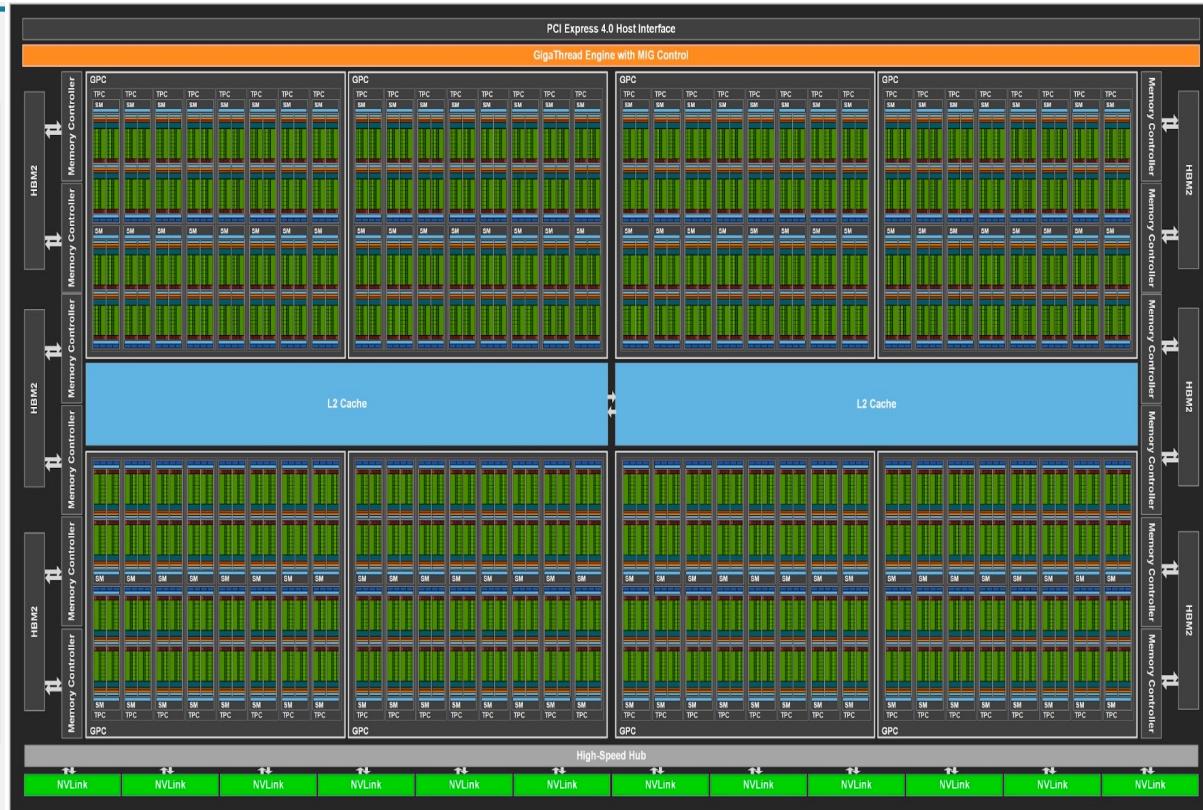
	Registers	L1 Cache	L2 Cache	L3 Cache	DRAM	Disk
Speed	1 cycle	~4 cycles	~10 cycles	~30 cycles	~200 cycles	10ms
Size	< KB per core	~32 KB per core	~256 KB per core	~35 MB per socket	~100 GB per socket	TB

GPU-Architecture

GPU-Architecture

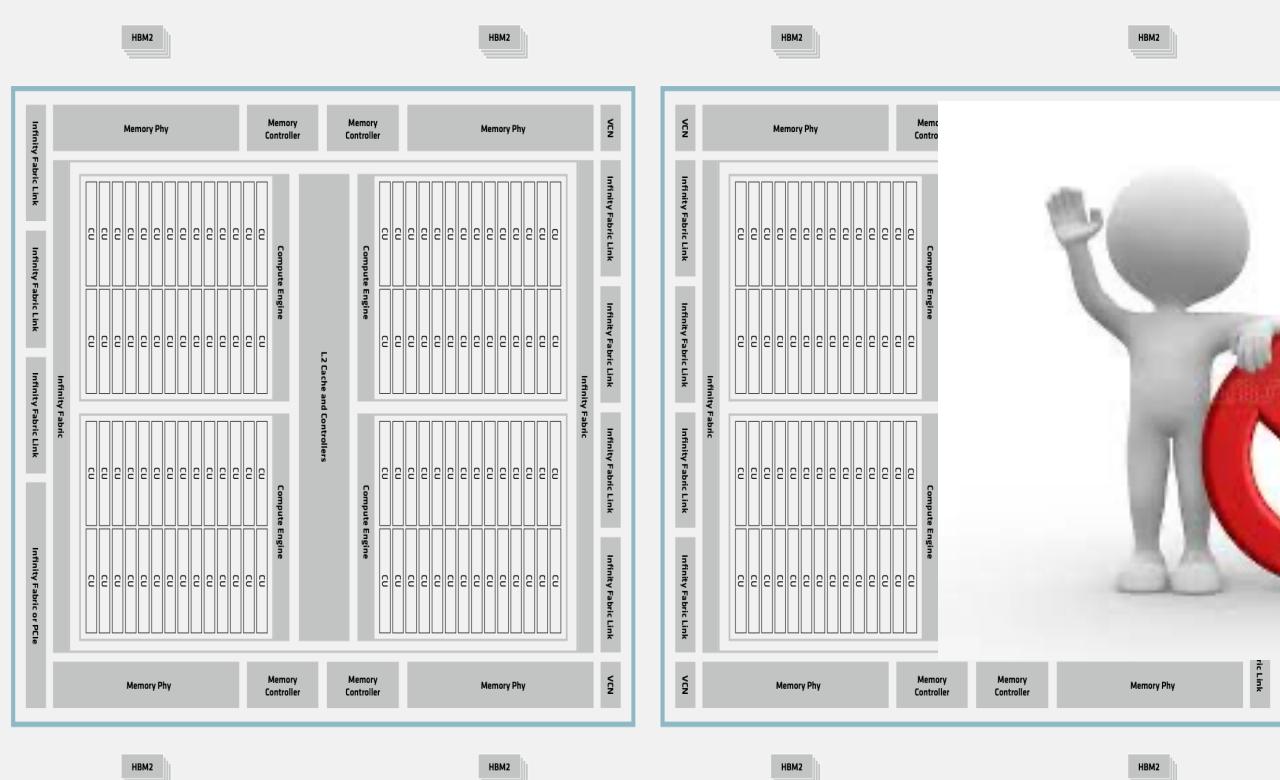


AMD GPU MI250X

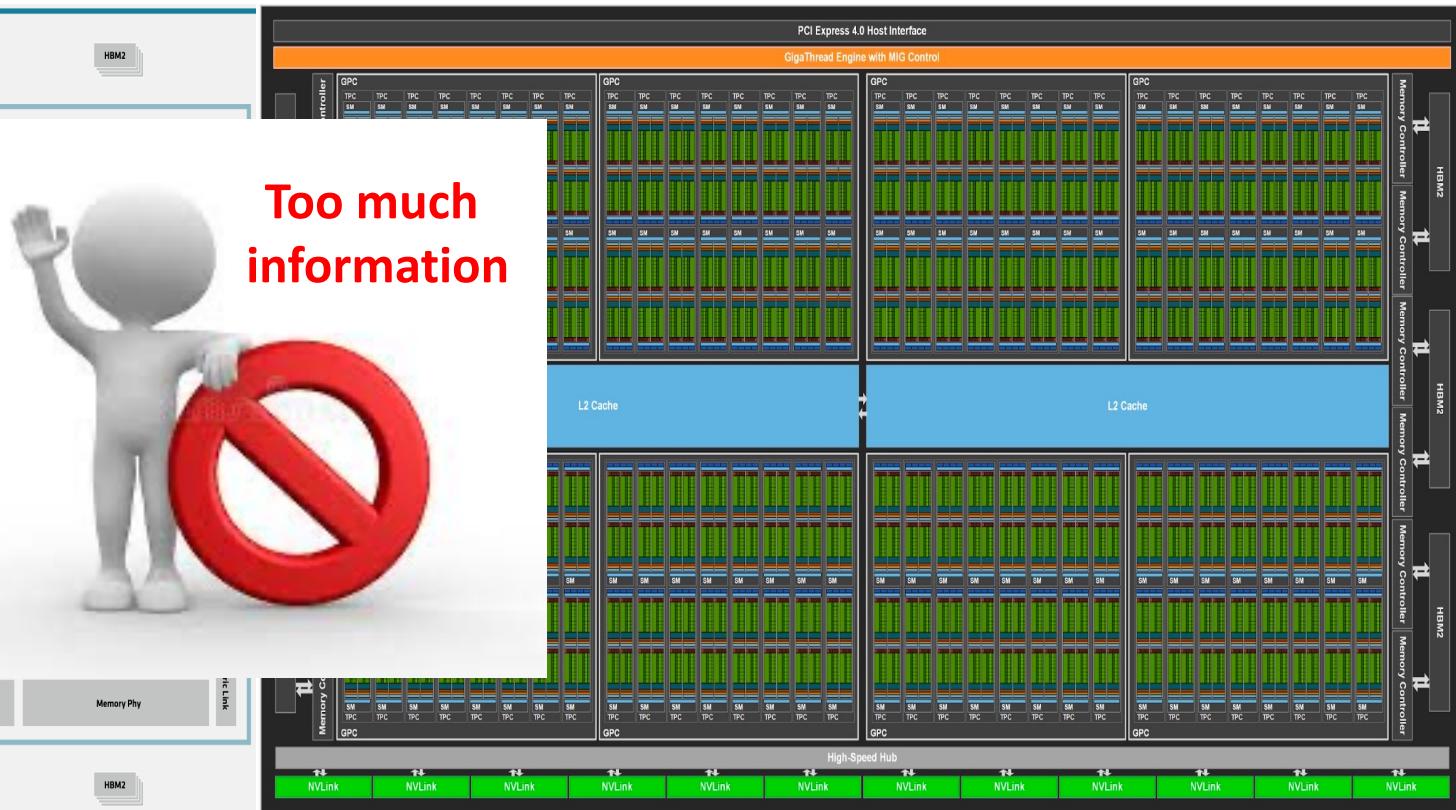


NVIDIA GPU GA100

GPU-Architecture

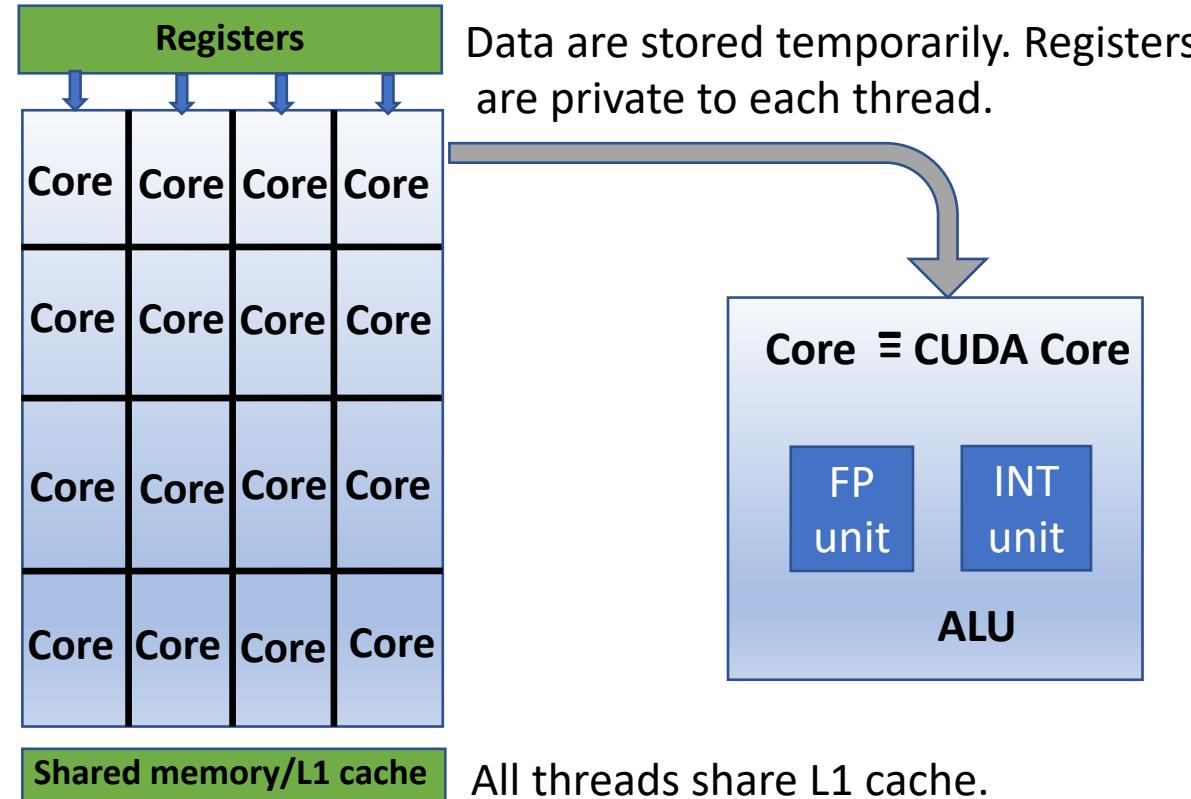
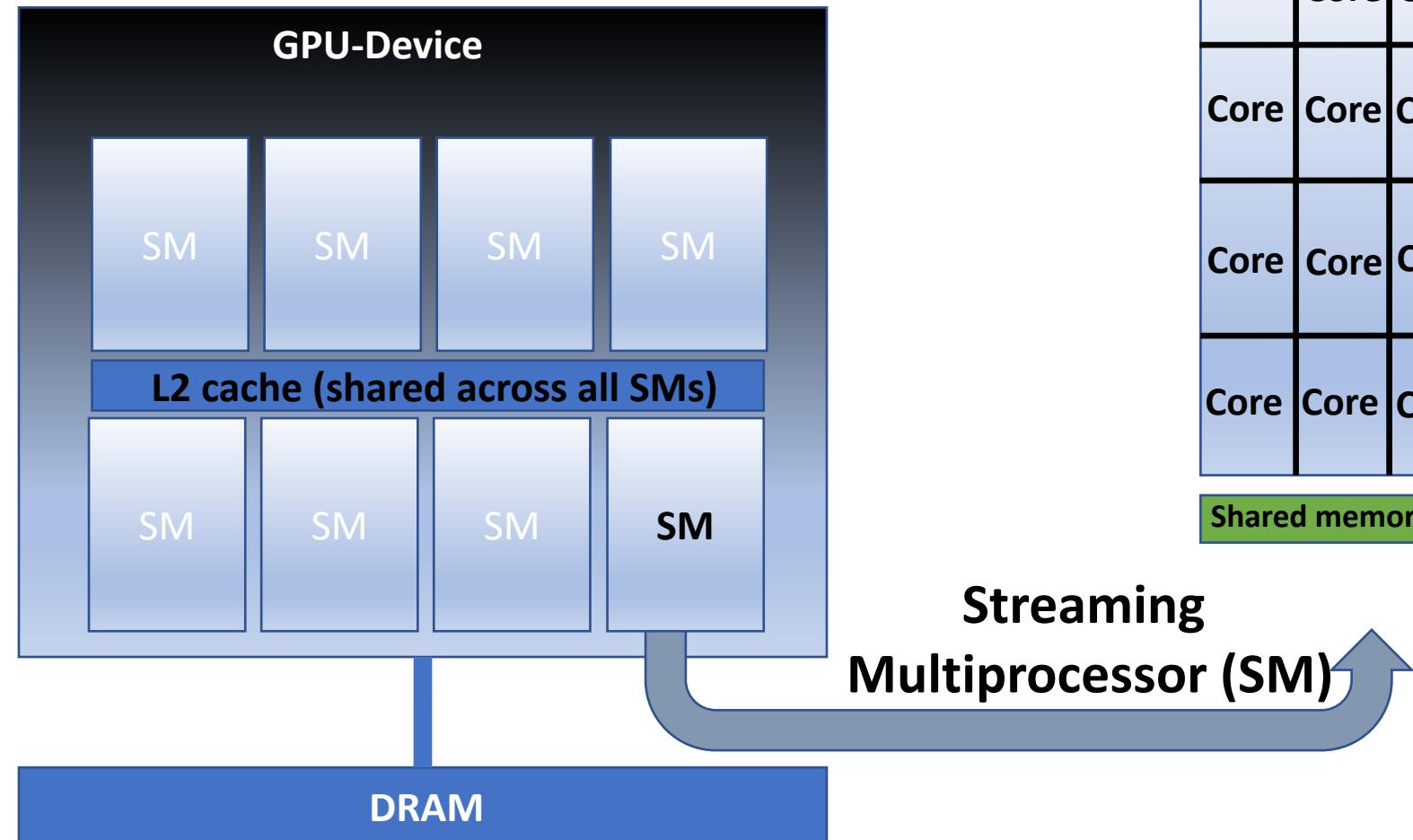


AMD GPU MI250X



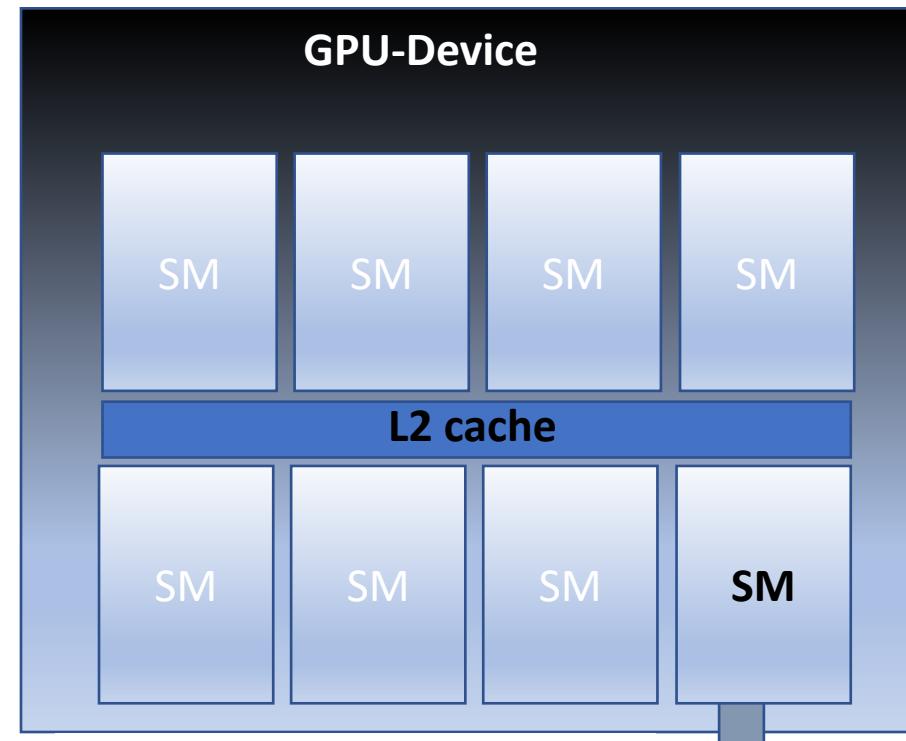
NVIDIA GPU GA100

Architecture of NVIDIA-GPU devices (simplified version)



- Core includes ALU. Various logical operations for computations (FP32, FP64), e.g. addition, multiplication and ...
- ALU executes SIMD instructions. (processing multiple data with the same operation concurrently (in parallel))

Architecture of NVIDIA-GPU devices (simplified version)

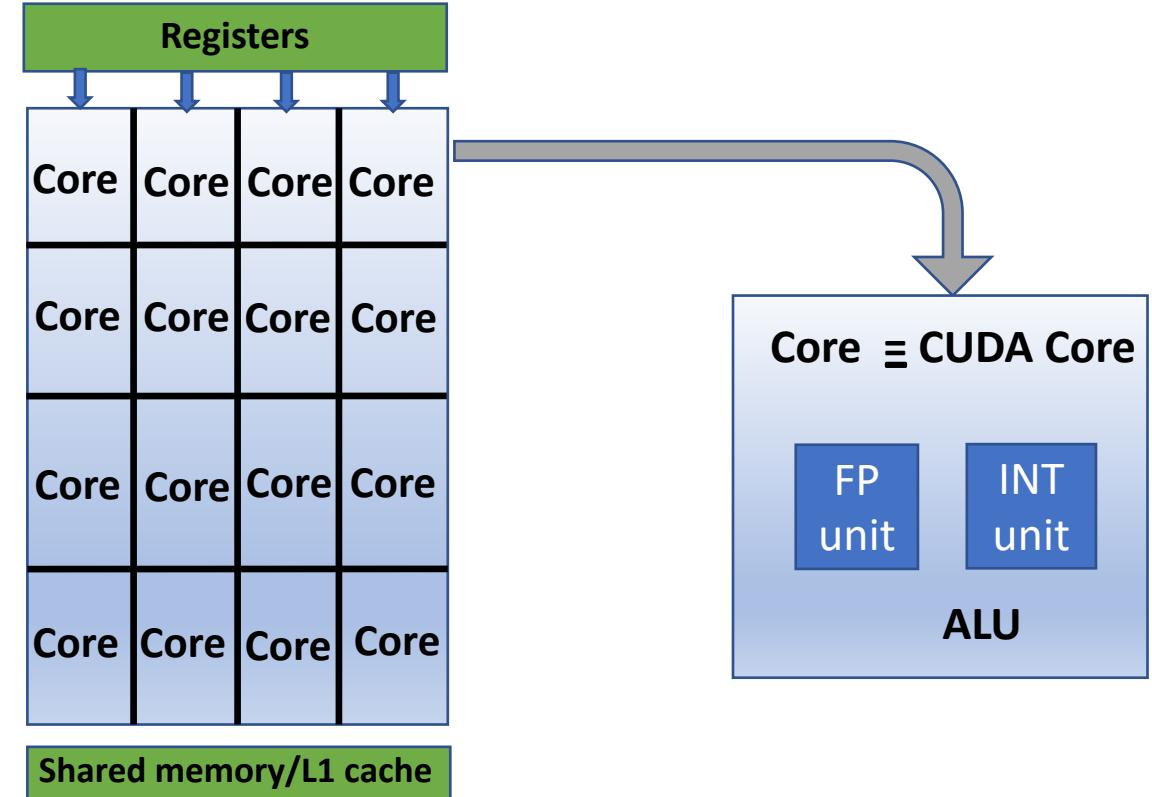


Streaming
Multiprocessor (SM)

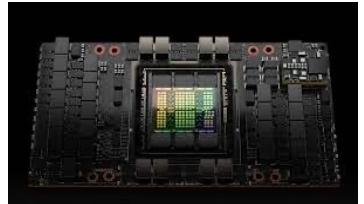
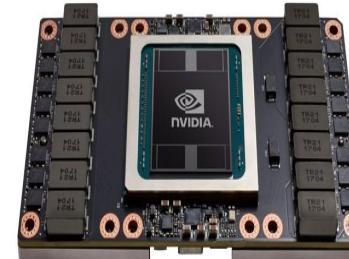
For Tesla A100 GPU: 108 SMs, each SM has:

64 FP32 cores ===== A total of 6912 FP32 CUDA cores

64 INT32 cores ===== A total of 6912 INT32 CUDA cores.

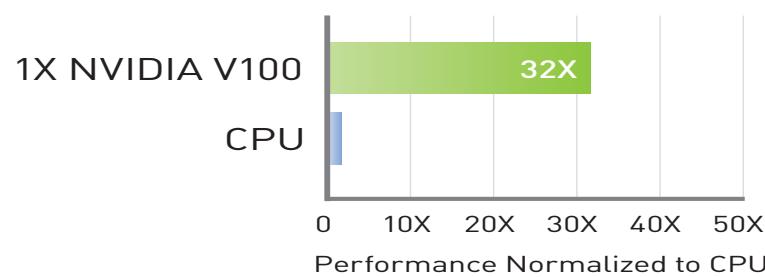


NVIDIA GPU characteristics



Architecture	NVIDIA P100 (Pascal)	NVIDIA GV100 (Volta)	NVIDIA GA100	NVIDIA GH100
SMs	56	84	128	144
FP32 CUDA cores per SM	64	64	64	128
NVIDIA CUDA cores	3584	5376	8192	18432
Tensor cores/GPU	NA	672	512	576
Peak performance	9.3 TFLOPS	15.7 TFLOPS	39 TFLOPS	66.9 TFLOPS
Transistors	15.3 billion	21.1 billion	54.2 billion	80 billion

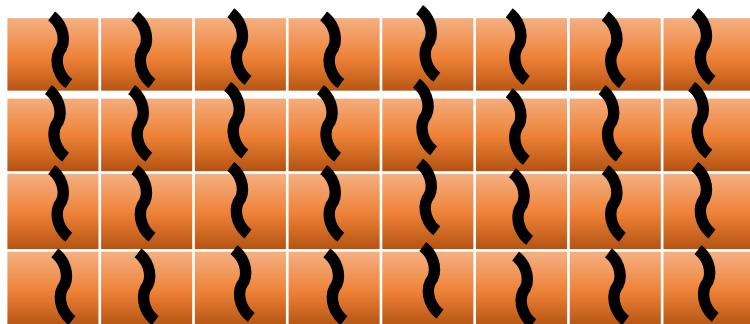
32X Faster Training Throughput
than a CPU¹



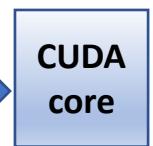
Software scheme

Execution on GPU

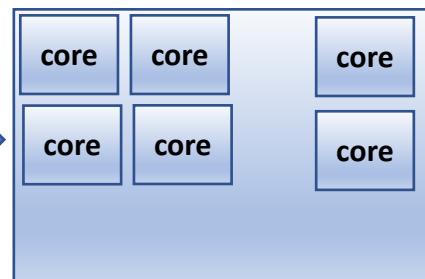
Hardware scheme



CUDA block
1024 threads
32 warps



is executed on

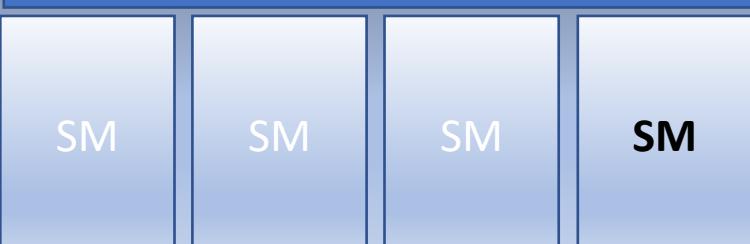
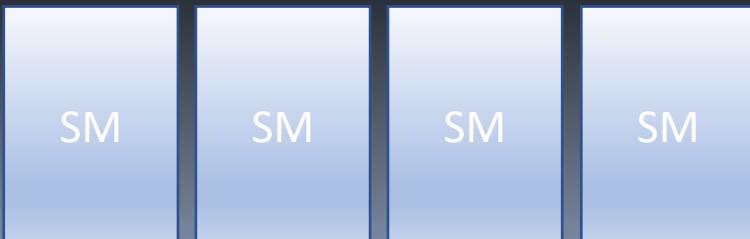


Streaming
Multiprocessor
(SM)

Grid of gangs
(CUDA kernel grid)



GPU-Device



Mapping a matrix into a GPU-device

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

Each thread executes a different piece
of data item (data parallelism).

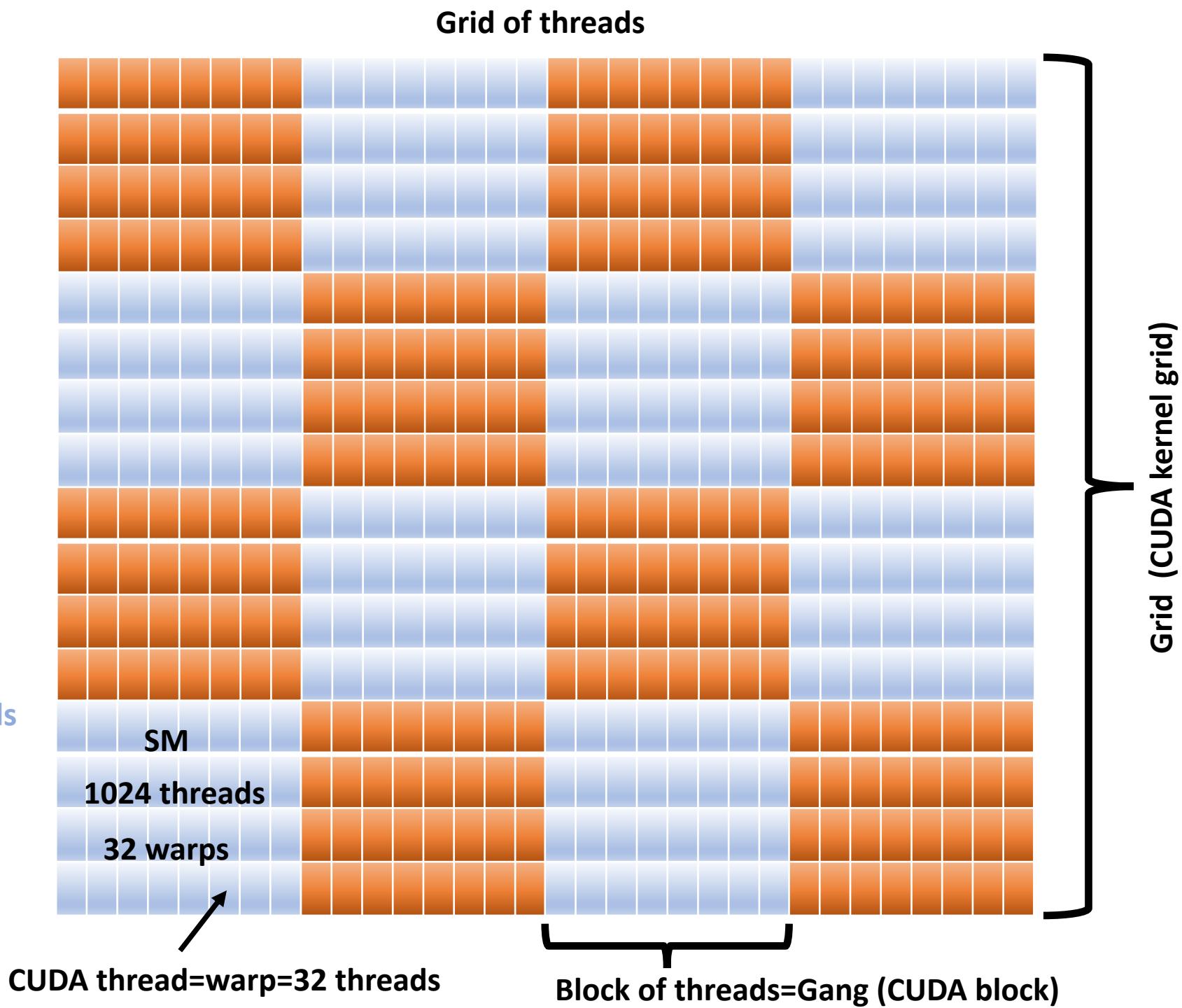
Each CUDA block has 1024 threads
(32 warps. A warp=collection of 32 threads
are executed simultaneously by a SM)

Execution on hardware

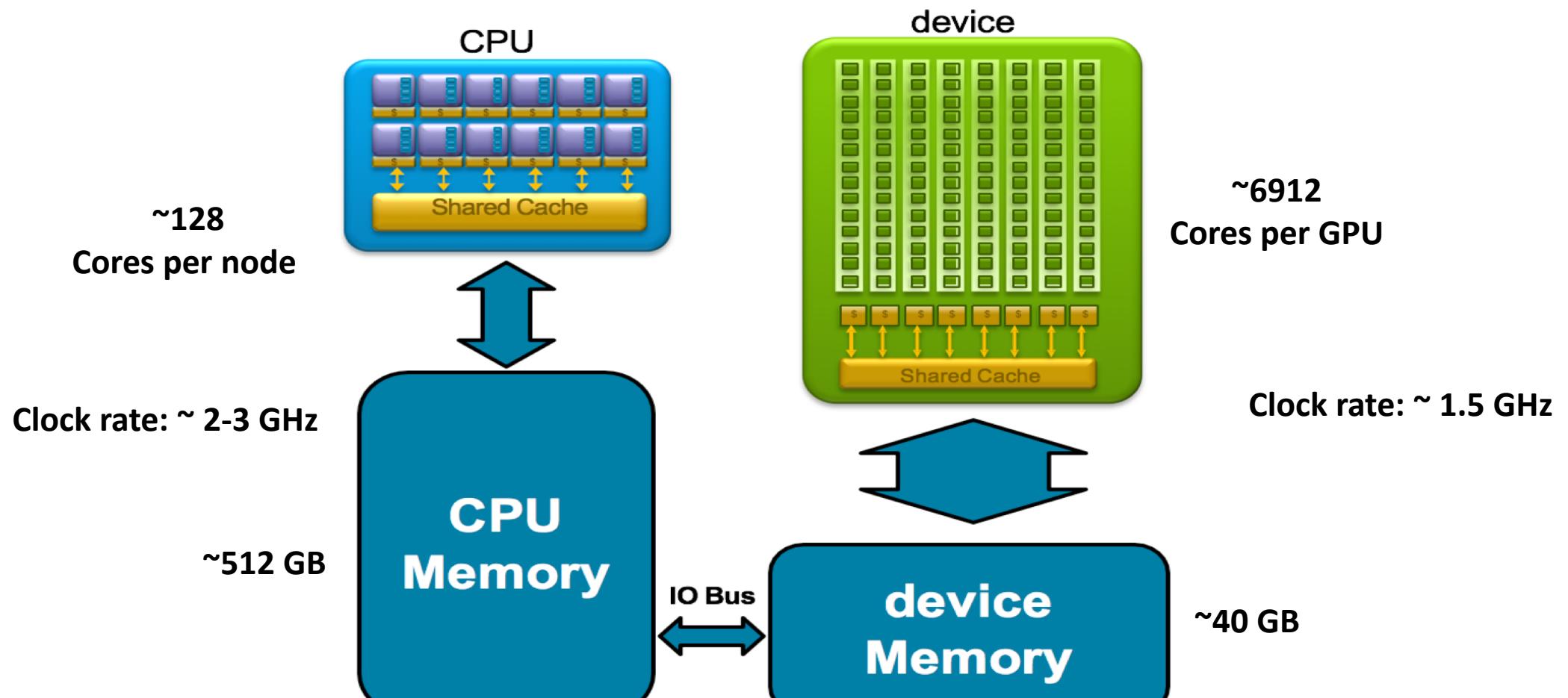
CUDA thread

CUDA block

CUDA grid



CPU vs GPU



Adapted from NVIDIA

**High throughput
Low latency**

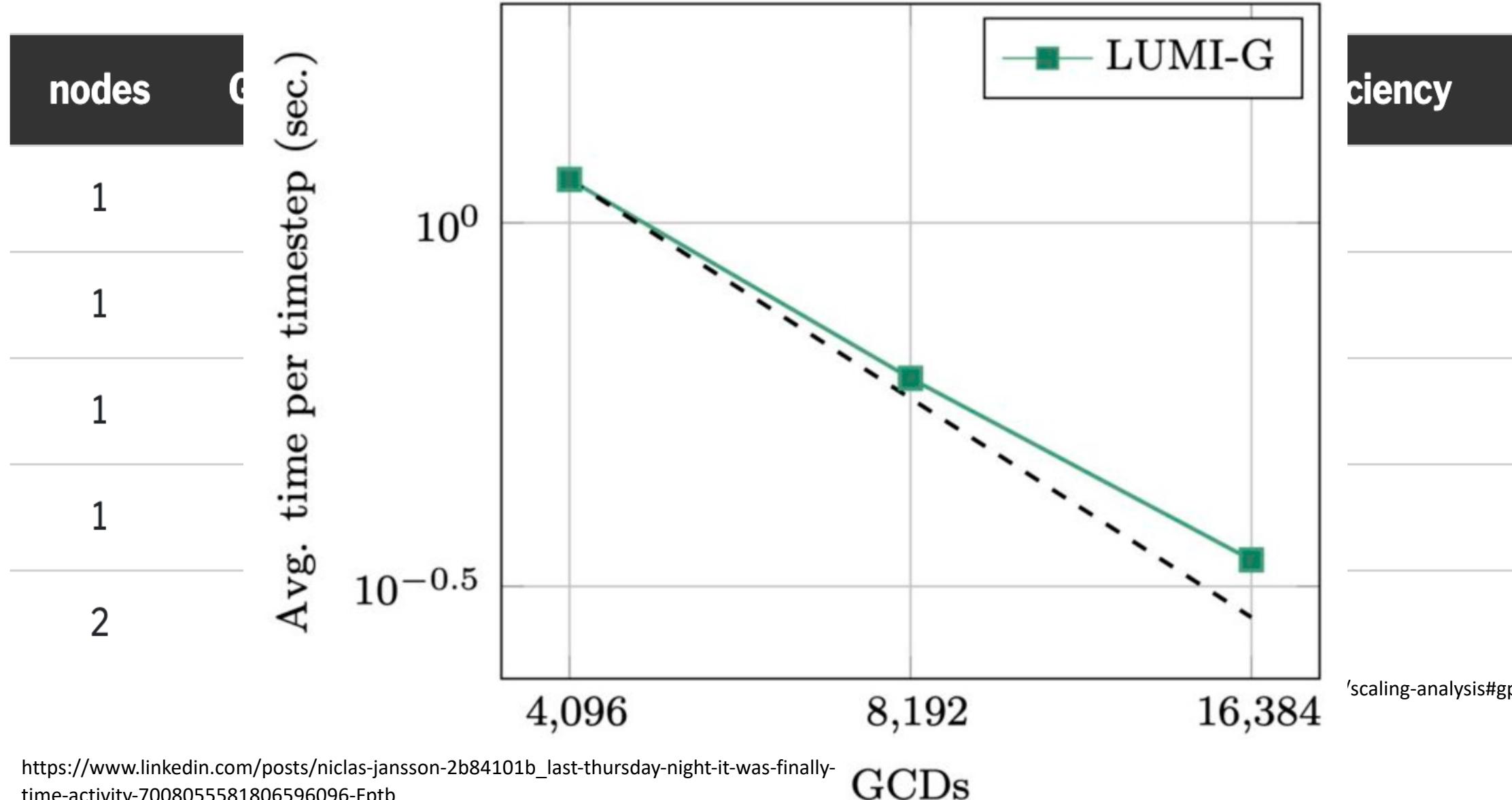
Scaling analysis

nodes	GPUs	execution time	speed-up ratio	parallel efficiency
1	1	212	1.0	100%
1	2	140	1.5	75%
1	3	110	1.9	64%
1	4	105	2.0	50%
2	8	145	1.5	18%

<https://researchcomputing.princeton.edu/support/knowledge-base/scaling-analysis#gpus>

Scaling analysis

Cylinder Re 50k, 113M el., 7th order poly.



Basic tools for GPU-usage

Basic NVIDIA tools

\$nvidia-smi

\$nvidia-smi stats

\$nvidia-smi dmon

Demo

Basic tools for viewing GPU-usage

Overview of GPU-usage: \$nvidia-smi

```
[hicham@c7-8 ~]$ nvidia-smi  
Sun Jan  8 17:53:37 2023
```

```
+-----+-----+-----+-----+-----+-----+  
NVIDIA-SMI 515.43.04    Driver Version: 515.43.04    CUDA Version: 11.7 |  
+-----+-----+-----+-----+-----+-----+  
GPU  Name      Persistence-M| Bus-Id      Disp.A  | Volatile Uncorr. ECC |  
Fan  Temp     Perf  Pwr:Usage/Cap| Memory-Usage | GPU-Util  Compute M. |  
|                           |          |            | MIG M. |  
+-----+-----+-----+-----+-----+-----+-----+-----+  
  0  Tesla P100-PCIE... Off | 00000000:39:00.0 Off | 0 |  
N/A  28C     P0   33W / 250W | 325MiB / 16384MiB | 39%     Default | N/A |  
+-----+-----+-----+-----+-----+-----+-----+-----+
```

```
+-----+-----+-----+-----+-----+-----+  
Processes:  
 GPU  GI  CI      PID  Type  Process name      GPU Memory |  
 ID   ID              |           Usage |  
+-----+-----+-----+-----+-----+-----+-----+-----+  
  0  N/A  N/A  43560  C  python      323MiB |  
+-----+-----+-----+-----+-----+-----+-----+
```

	Value	Description
Driver Version	515.43.04	
CUDA Version	11.7	
GPU	0	GPU index
Name	Tesla P100	GPU name
Temp	28C	GPU-Temperature
Perf	P0	GPU-Performance
Persistance-M	Off	Persistance mode
Pwr:Usage/Cap	33W/250W	Ratio of GPU-Power usage
Bus-Id	00000000:39:00.0	
Disp.A	Off	Display active
Mem Usage	325/16384 MiB	Memory used
Volatile Uncorr.ECC	0	Counter Uncorrelated error
GPU-Util	39%	GPU utilisation
Compute M.	Default	Compute memory

Basic tools for viewing GPU-usage

Device statistic: \$nvidia-smi stats

\$nvidia-smi stats -d gpuUtil

```
0, gpuUtil , 1673214742538187, 0
0, gpuUtil , 1673214743709403, 0
0, gpuUtil , 1673214743876909, 0
0, gpuUtil , 1673214744044158, 0
0, gpuUtil , 1673214744211676, 8
0, gpuUtil , 1673214744379109, 4
0, gpuUtil , 1673214744546360, 0
0, gpuUtil , 1673214744713449, 13
0, gpuUtil , 1673214744880577, 57
0, gpuUtil , 1673214745047676, 46
0, gpuUtil , 1673214745214772, 0
0, gpuUtil , 1673214745381932, 60
0, gpuUtil , 1673214745549000, 68
0, gpuUtil , 1673214745716307, 0
0, gpuUtil , 1673214745883405, 0
0, gpuUtil , 1673214746050521, 69
0, gpuUtil , 1673214746217605, 59
0, gpuUtil , 1673214746384700, 0
0, gpuUtil , 1673214746551961, 9
0, gpuUtil , 1673214746718809, 87
0, gpuUtil , 1673214746885861, 32
0, gpuUtil , 1673214747052886, 0
0, gpuUtil , 1673214747219955, 22
0, gpuUtil , 1673214747387022, 82
0, gpuUtil , 1673214747554122, 23
0, gpuUtil , 1673214747721127, 0
0, gpuUtil , 1673214747888199, 65
0, gpuUtil , 1673214748055238, 62
0, gpuUtil , 1673214748222243, 0
0, gpuUtil , 1673214748389417, 0
0, gpuUtil , 1673214748556383, 80
```

To display
GPU utilisation
&
Memory utilisation

\$nvidia-smi stats -d memUtil

```
0, memUtil , 1673214794670094, 0
0, memUtil , 1673214794837087, 0
0, memUtil , 1673214795004095, 0
0, memUtil , 1673214795171159, 0
0, memUtil , 1673214795338176, 3
0, memUtil , 1673214795505181, 0
0, memUtil , 1673214795672219, 0
0, memUtil , 1673214795839287, 8
0, memUtil , 1673214796006305, 25
0, memUtil , 1673214796175765, 11
0, memUtil , 1673214796342801, 0
0, memUtil , 1673214796509822, 26
0, memUtil , 1673214796676819, 20
0, memUtil , 1673214796843886, 0
0, memUtil , 1673214797010928, 0
0, memUtil , 1673214797177935, 30
0, memUtil , 1673214797344972, 16
0, memUtil , 1673214797511999, 0
0, memUtil , 1673214797679049, 6
0, memUtil , 1673214797846118, 29
0, memUtil , 1673214798013127, 10
0, memUtil , 1673214798180164, 0
0, memUtil , 1673214798347203, 15
0, memUtil , 1673214798514211, 31
0, memUtil , 1673214798681248, 0
0, memUtil , 1673214798848289, 4
0, memUtil , 1673214799015297, 30
0, memUtil , 1673214799182335, 11
0, memUtil , 1673214799349448, 0
0, memUtil , 1673214799516412, 11
0, memUtil , 1673214799683451, 30
```

To display more information: "\$nvidia-smi stats -h"

Basic tools for viewing GPU-usage

Device monitoring: \$nvidia-smi dmon

#	gpu	pwr	gtemp	mtemp	sm	mem	enc	dec	mclk	pclk
#	Idx	W	C	C	%	%	%	%	MHz	MHz
0	31	28	-	-	9	0	0	0	715	1189
0	31	28	-	-	0	0	0	0	715	1189
0	31	28	-	-	0	0	0	0	715	1189
0	26	28	-	-	0	0	0	0	715	455
0	26	28	-	-	0	0	0	0	715	455
0	30	28	-	-	2	0	0	0	715	1189
0	31	28	-	-	8	4	0	0	715	1189
0	173	30	-	-	32	13	0	0	715	1328
0	36	30	-	-	3	1	0	0	715	1328
0	164	31	-	-	77	31	0	0	715	1328
0	137	30	-	-	0	0	0	0	715	1328
0	37	30	-	-	43	17	0	0	715	1328
0	183	32	-	-	46	19	0	0	715	1328
0	36	30	-	-	0	0	0	0	715	1328
0	37	31	-	-	88	35	0	0	715	1328
0	157	33	-	-	34	14	0	0	715	1328
0	36	31	-	-	0	0	0	0	715	1328
0	37	32	-	-	84	34	0	0	715	1328
0	160	33	-	-	36	14	0	0	715	1328
0	37	31	-	-	0	0	0	0	715	1328
0	37	32	-	-	84	33	0	0	715	1328
0	109	33	-	-	83	32	0	0	715	1328
0	36	31	-	-	0	0	0	0	715	1328

SM: Streaming Multiprocessor

Select GPU utilisation

\$nvidia-smi pmon -s u

Select memory usage

\$nvidia-smi pmon -s m

Display more information

\$nvidia-smi pmon -h

DEMO

How to access GPU metrics in **HPC systems** using the command-line **nvidia-smi**

Method 1

The command **nvidia-smi** is available from a **GPU node**:

- Login to Betzy cluster: **\$ssh username@betzy.sigma2.no**
- Identify GPU partitions: **\$sinfo -p accel**
- Submit a job: **\$sbatch job.slurm**
- Which node: **\$squeue -u username**
- ssh to the node e.g. **\$ssh b5304**
- Run the command: **\$nvidia-smi**
- For more options: **\$nvidia-smi -h**

Method 2

Run the command **nvidia-smi** interactively

- Submit a job: **\$sbatch job.slurm**
- Copy paste the **JobID** to the syntax below:

```
$for j in {1..10}; do srun --jobid=JobID
--interactive --pty nvidia-smi; sleep 2; done
```

Profiling Deep Learning with **PyTorch Profiler**

- What is PyTorch Profiler ?
- How to setup PyTorch Profiler
- **Demo:** How to use PyTorch Profiler?

What is PyTorch Profiler ?

- Advanced tool for analysing Deep Learning models.
- Dynamical tool built inside PyTorch.
- Collection of metrics during training and inference.

What is PyTorch Profiler ?

- Advanced tool for analysing Deep Learning models.
- Dynamical tool built inside PyTorch.
- Collection of metrics during training and inference.
 - GPU usage
 - Tensor cores usage
 - GPU Kernel view
 - Memory view
 - Trace view

What is PyTorch Profiler ?

- Advanced tool for analysing Deep Learning models.
 - Dynamical tool built inside PyTorch.
 - Collection of metrics during training and inference.
 - GPU usage
 - Tensor cores usage
 - GPU Kernel view
 - Memory view
 - Trace view
- 
- [View in a web browser](#)**
- Identify **bottlenecks** [memory and GPU]
- Identify functions that use the **majority of time**

What is PyTorch Profiler ?

- Advanced tool for analysing Deep Learning models.
- Dynamical tool built inside PyTorch.
- Collection of metrics during training and inference.
 - GPU usage
 - Tensor cores usage
 - GPU Kernel view
 - Memory view
 - Trace view



View in a web browser

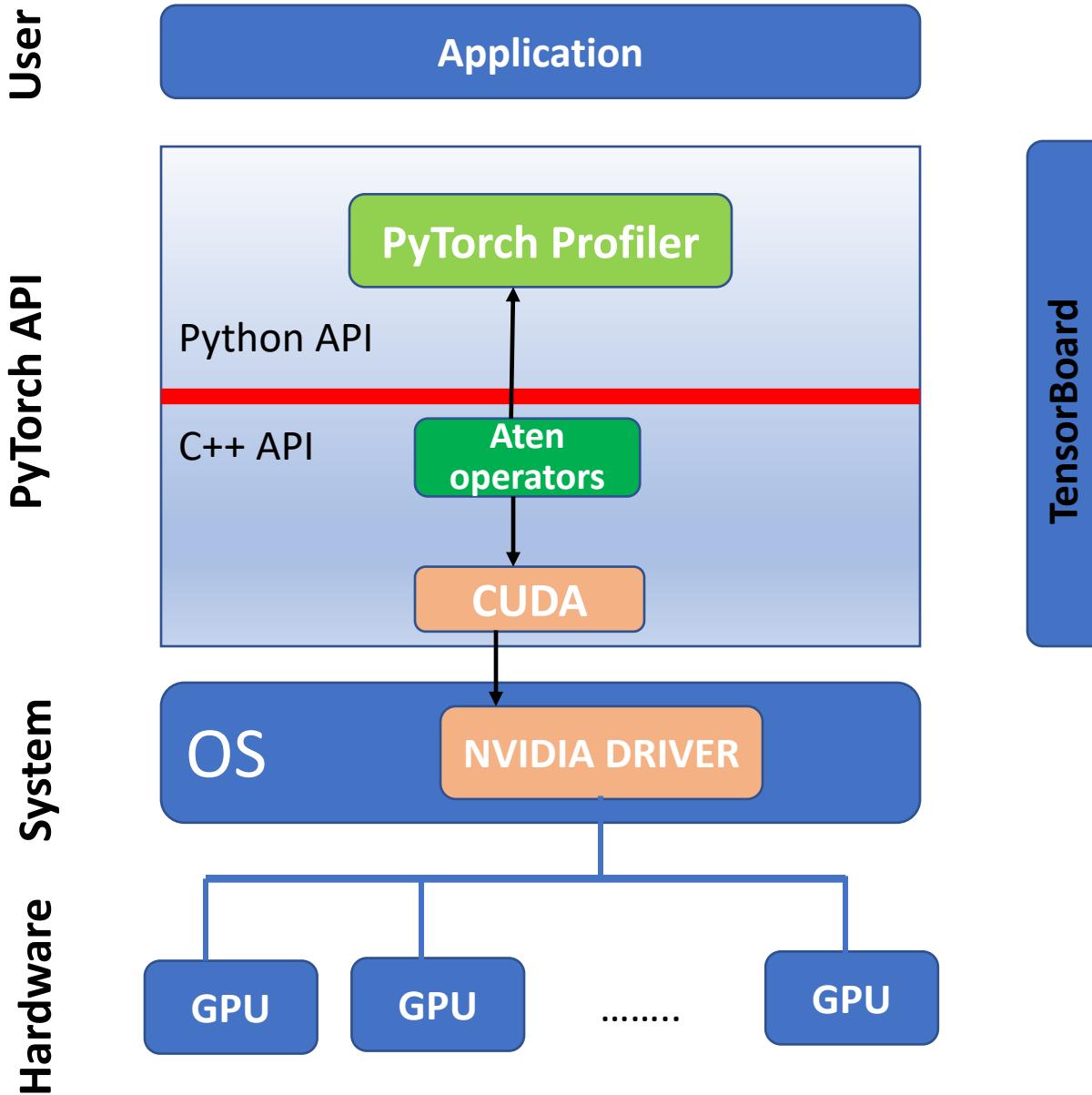
Identify bottlenecks [memory and GPU]

Identify functions that use the majority of time



Improve Performance

Architecture of PyTorch Profiler (simplified version)



Setup in HPC systems:

Setup in HPC systems:

Installing PyTorch

- **Method 1:** Loading a module
- **Method 2:** Virtual environment
- **Method 3:** Singularity container

Installing TensorBoard Plugin

```
$ python -m pip install torch_tb_profiler
```

Setup in HPC systems:

- **Method 3: Singularity container**

- **Pull the PyTorch container image**

e.g. from NVIDIA NGC container:

<https://catalog.ngc.nvidia.com/orgs/nvidia/containers/pytorch>

[the host system must have the CUDA driver installed and the container must have CUDA]

\$singularity pull docker://nvcr.io/nvidia/pytorch:22.12-py3

- **Launch singularity container**

\$singularity exec --nv pytorch_22.12-py3.sif python main.py

NVIDIA PyTorch container contains the following software:

- [CUDA](#)
- [cuBLAS](#)
- [NVIDIA cuDNN](#)
- [NVIDIA NCCL](#) (optimized for [NVLink](#))
- [RAPIDS](#)
- [NVIDIA Data Loading Library \(DALI\)](#)
- [TensorRT](#)
- [Torch-TensorRT](#)

<https://catalog.ngc.nvidia.com/orgs/nvidia/containers/pytorch>

Setup in HPC systems:

Installating TensorBord in a virtual environment

Step0: load a python model, create & activate Virt. Env.

Find a python module: `$module avail python`

Load a python module .e.g.:

`$module load Python/3.9.6-GCCcore-11.2.0`

`$mkdir Myenv`

`$python -m venv Myenv`

`$source Myenv/bin/activate`

Step1: Install via pip wheel packages

Install PyTorch Profiler TensorBoard Plugin:

`$python -m pip install torch_tb_profiler`

DEMO:Profiling a Resnet 18 model

DEMO: Profiling a Resnet 18 model

```
with torch.profiler.profile(  
    activities=[  
        torch.profiler.ProfilerActivity.CPU,  
        torch.profiler.ProfilerActivity.CUDA],  
    schedule=torch.profiler.schedule(  
        wait=1,  
        warmup=1,  
        active=2),  
    on_trace_ready=torch.profiler.tensorboard_trace_handler('./out', worker_name='profiler'),  
    record_shapes=True,  
    profile_memory=True,  
    with_stack=True  
) as prof:
```

Lines of code to enable profiling with PyTorch Profiler

Output of profiling will be saved In ./out directory

```
#training step for each batch of input data  
for step, data in enumerate(trainloader):  
    :  
    if step +1>= 10:  
        break  
    prof.step()
```

Training loop

Submit an application: \$sbatch job.slurm

```
#!/bin/bash -l
#SBATCH --job-name=pyt-profiler
#SBATCH --account=nn9987k
#SBATCH --time=00:10:00
#SBATCH --partition=accel
#SBATCH --nodes=1           #nbr of nodes
#SBATCH --ntasks=1          #nbr of tasks
#SBATCH --cpus-per-task=1   #nbr of threads (or cores)
#SBATCH --gpus=1             #total nbr of gpus
#SBATCH --gpus-per-node=1   #nbr of gpus per node
#SBATCH --mem=4G             #main memory
#SBATCH -o PyTprofiler.out

#define paths
Mydir=/cluster/projects/nn9987k/PyTorchProfiler
MyContainer=${Mydir}/Container/pytorch_22.12-py3.sif
MyExp=${Mydir}/examples

#specify bind paths by setting the environment variable
export SINGULARITY_BIND="${MyExp},$PWD"

#TF32 is enabled by default in the NVIDIA NGC TensorFlow and PyTorch containers
#To disable TF32 set the environment variable to 0
export NVIDIA_TF32_OVERRIDE=0

#to run singularity container
singularity exec --nv ${MyContainer} python3 ${MyExp}/resnet18_profiler_api.py

echo
echo "--Job ID:" $SLURM_JOB_ID
echo "--total nbr of gpus" $SLURM_GPUS
```

Viewing with Tensorboard plugin

Viewing with Tensorboard plugin

1- Load a python module, e.g.:

```
$module load Python/3.9.6-GCCcore-11.2.0
```

2- Activate the virtual environment:

```
$source Myenv/bin/activate
```

3- Launch the TensorBoard Plugin:

```
$tensorboard --logdir=./out --bind_all
```

TensorBoard 2.11.0 at <http://login-2.betzy.sigma2.no:6006/>

4- Open a new terminal for local port forwarding

```
$ssh -L 6006:login-2.betzy.sigma2.no:6006 user@betzy.sigma2.no
```

5- View on a browser [chrome, firefox]

```
http://localhost:6006/
```

```
(env) [hicham@login-2.BETZY /cluster/projects/nn9997k/hich/PyTorchProfiler/Jobs]$ tensorboard --logdir=./out-4batch --bind_all
TensorFlow installation not found - running with reduced feature set.
/cluster/projects/nn9997k/hich/PyTorchProfiler/env/lib/python3.9/site-packages/tensorboard_data_server/bin/server: /lib64/libc.so.6: version `GLIBC_2.18' not found
(required by /cluster/projects/nn9997k/hich/PyTorchProfiler/env/lib/python3.9/site-packages/tensorboard_data_server/bin/server)
I0110 13:10:04.112582 139899703727872 plugin.py:429] Monitor runs begin
I0110 13:10:04.113618 139899703727872 plugin.py:444] Find run directory /cluster/projects/nn9997k/hich/PyTorchProfiler/Jobs/out-4batch
I0110 13:10:04.114232 139899821176576 plugin.py:493] Load run out-4batch
I0110 13:10:04.140846 139899821176576 loader.py:571 started all processing
TensorBoard 2.11.0 at http://login-2.betzy.sigma2.no:6006/ (Press CTRL+C to quit)
I0110 13:10:08.541492 139899821176576 plugin.py:497] Run out-4batch loaded
I0110 13:10:08.541845 139899745675008 plugin.py:467] Add run out-4batch
```

GPU Usage View

TensorBoard PYTORCH_PROFILER INACTIVE UPLOAD ⚙️ ⏪ ⏴ ⚙️ ?

NORMAL DIFF

Runs Xresult Views Overview

Workers worker0

Spans 1

Configuration

Number of Worker(s) 1 Device Type GPU

GPU Summary

GPU 0: Tesla P100-PCIE-16GB

Name	Memory	Compute Capability
Memory	15.9 GB	6.0
Compute Capability		71.23 %
GPU Utilization		70.04 %
Est. SM Efficiency		44.79 %
Est. Achieved Occupancy		

Execution Summary

Category	Time Duration (us)	Percentage (%)
Average Step Time	109,570	100
Kernel	78,052	71.23
Memcpy	20,258	18.49
Memset	67	0.06
Runtime	0	0
DataLoader	0	0
CPU Exec	5,391	4.92
Other	5,802	5.3

Step Time Breakdown

Step Time (microseconds)

Step

Kernel Memcpy Memset Runtime DataLoader CPU Exec Other

Category	Time Duration (us)	Percentage (%)
Kernel	78,052	71.23
Memcpy	20,258	18.49
Memset	67	0.06
Runtime	0	0
DataLoader	0	0
CPU Exec	5,391	4.92
Other	5,802	5.3

Legend:

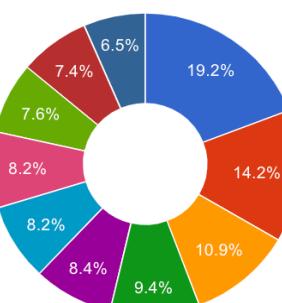
- Kernel
- Memcpy
- Memset
- Runtime
- DataLoader
- CPU Exec
- Other

GPU Kernel View

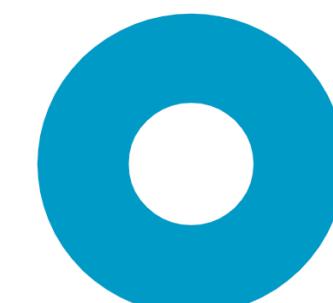
TensorBoard PYTORCH_PROFILER INACTIVE UPLOAD ⚙️ ⚙️ ⚙️ ⚙️

Kernel View

All kernels Top kernels to show 10

Total Time (us) 

Kernel Name	Time (%)
cudnn_maxwell_scudnn_128x32_stri...	19.2%
maxwell_sgemm_32x128_nt	14.2%
maxwell_sgemm_32x128_nn	10.9%
void at::native::(anonymous namespace)::max_pool_backward_n...	9.4%
cudnn_maxwell_scudnn_winograd_1...	8.4%
cudnn_maxwell_scudnn_winograd_1...	8.2%
cudnn_maxwell_scudnn_128x64_stri...	8.2%
cudnn_maxwell_scudnn_128x128_stri...	7.6%
void	7.4%
at::native::vectorized_elementwise_k...	6.5%

Tensor Cores Utilization 

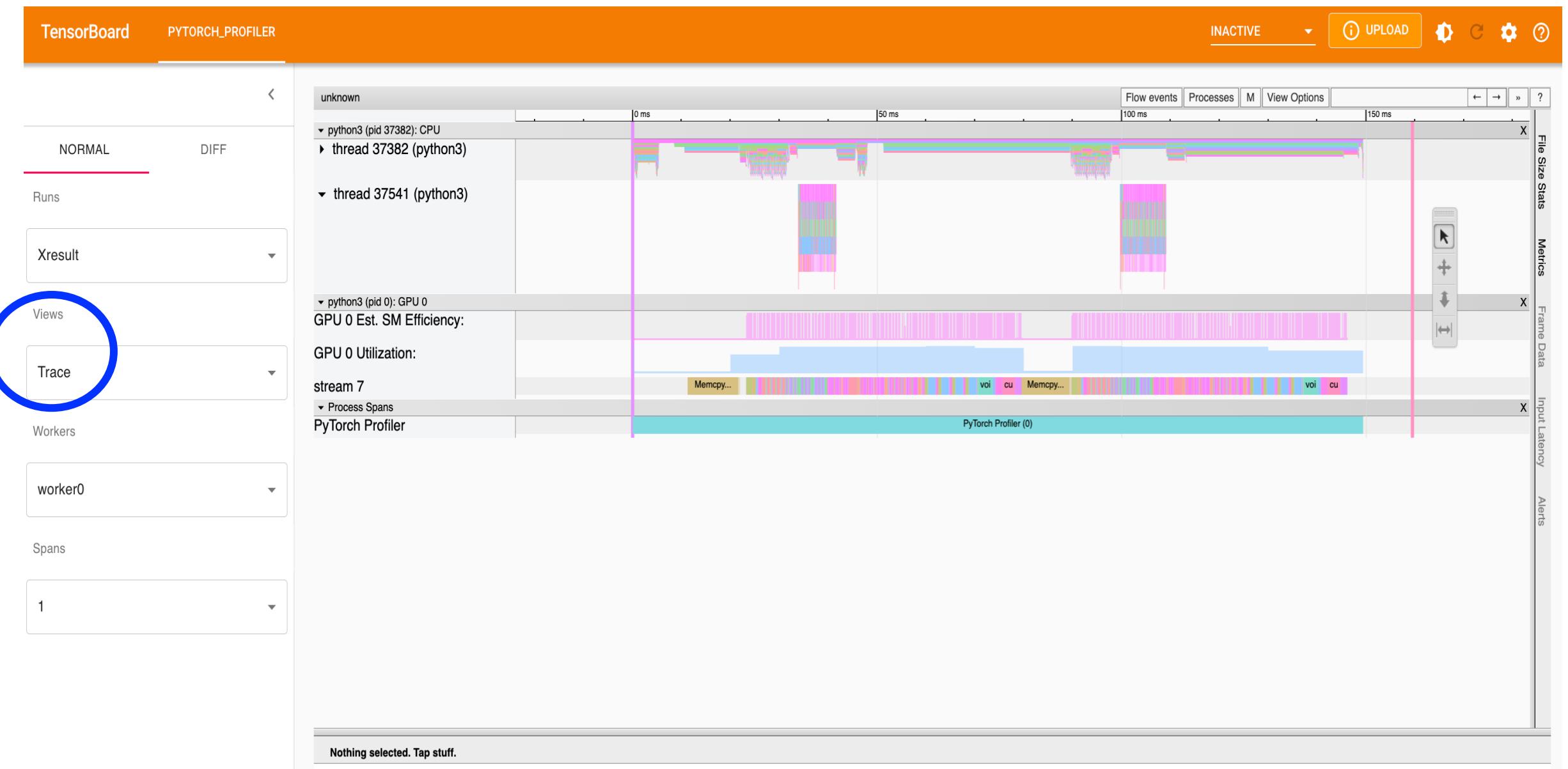
Category	Count
Not Using Tensor Cores	10
Using Tensor Cores	0

Group By Kernel Name

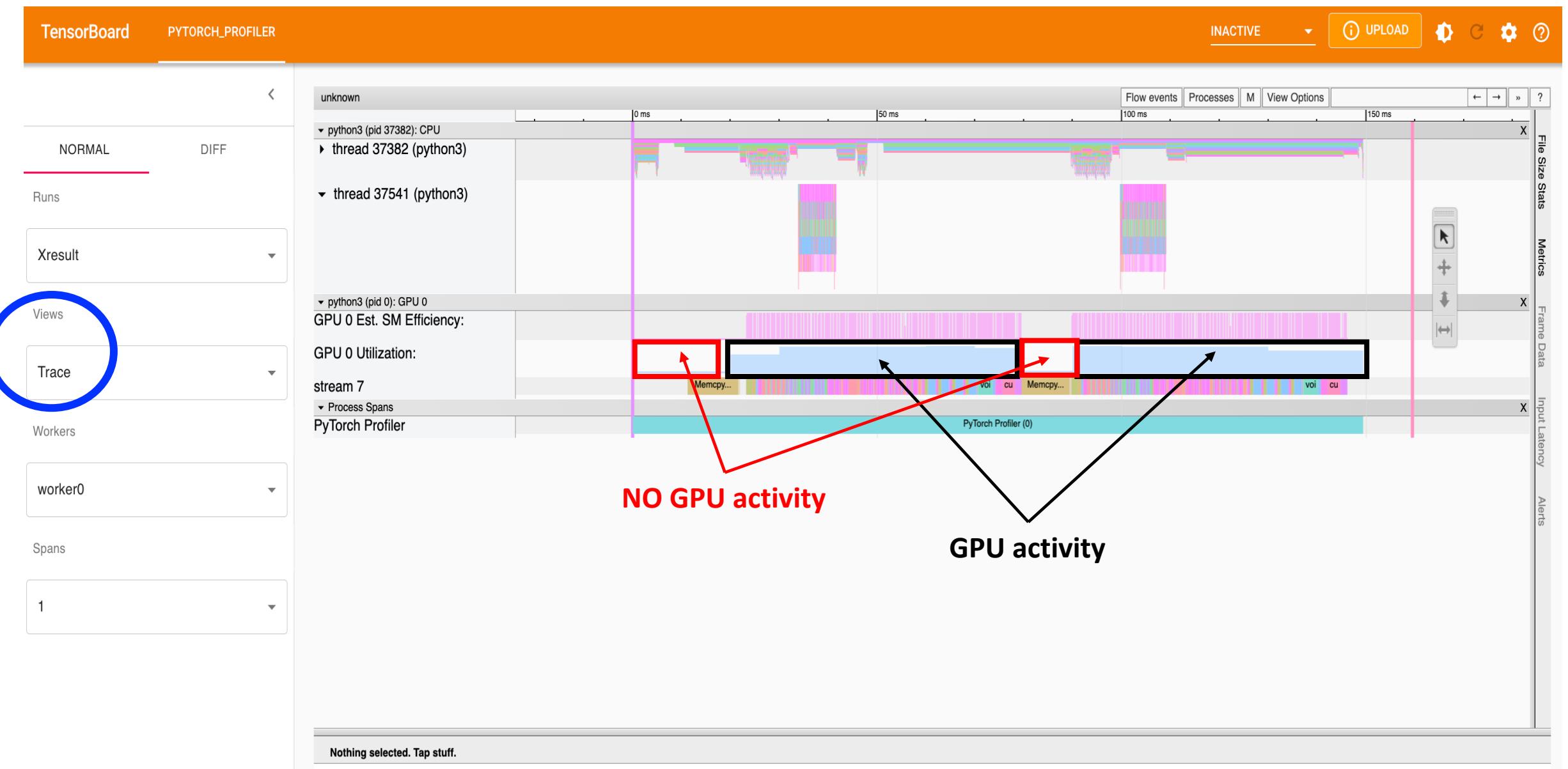
Search by Kernel Name

Name	Tensor Cores Used	Calls	Total Duration (us)	Mean Duration (us)	Max Duration (us)	Min Duration (us)	Mean Blocks Per SM	Mean Est. Occupancy (%)
cudnn_maxwell_scudnn_128x32_stridedB_splitK_medium_nn_v0	No	8	12302	1538	1548	1521	10	31
maxwell_sgemm_32x128_nt	No	24	9091	379	422	326	23.44	50
maxwell_sgemm_32x128_nn	No	18	6977	388	436	340	20.11	50
void at::native::(anonymous namespace)::max_pool_backward_nchw<float, float>(float const*, long const*, int, long, long, long, int, float*)	No	2	6009	3004	3005	3004	1792	100
cudnn_maxwell_scudnn_winograd_128x128_ldg1_ldg4_relu_tile148m_nt_v1	No	8	5373	672	681	657	32	25
cudnn_maxwell_scudnn_winograd_128x128_ldg1_ldg4_mobile_relu_tile148t_nt_v0	No	8	5282	660	671	646	32	25
cudnn_maxwell_scudnn_128x64_stridedB_splitK_large_nn_v0	No	2	5235	2618	2618	2617	4	25
cudnn_maxwell_scudnn_128x128_stridedB_splitK_medium_nn_v0	No	8	4888	611	736	413	4.43	11.81

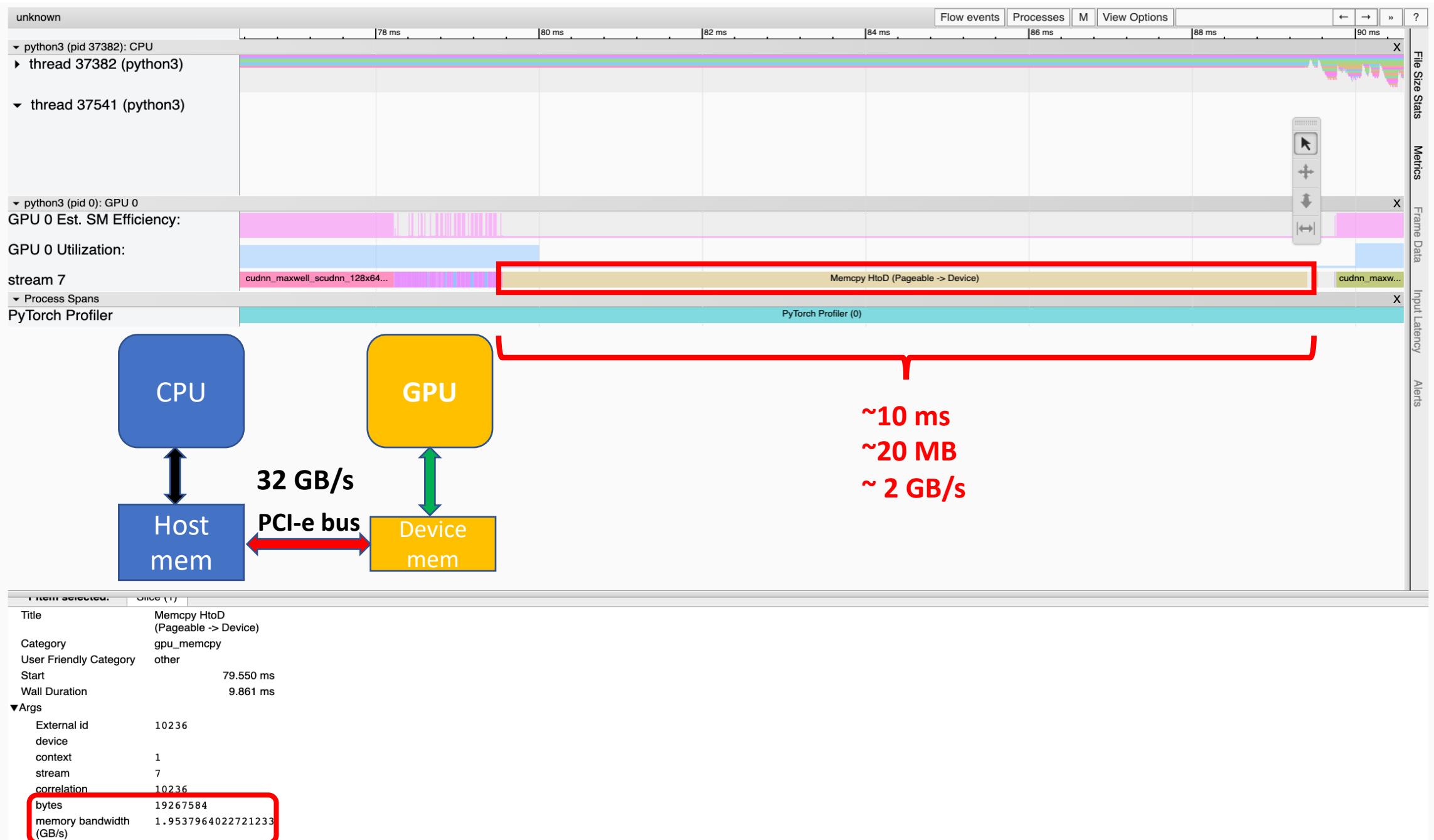
Trace View



Trace View



Trace View



Trace View



Cuda stream: a sequence of operations executed on a GPU (in the order)

queued
device
context 1
stream 7
correlation 9288
registers per thread 128
shared memory 10240
blocks per SM 4
warps per SM 16
grid [2, 1, 112]
block [128, 1, 1]
est. achieved occupancy % 25

A half-capacity of each SM

CUDA block has 32 warps

Mapping a matrix into a GPU-device

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

Each thread executes a different piece
of data item (data parallelism).

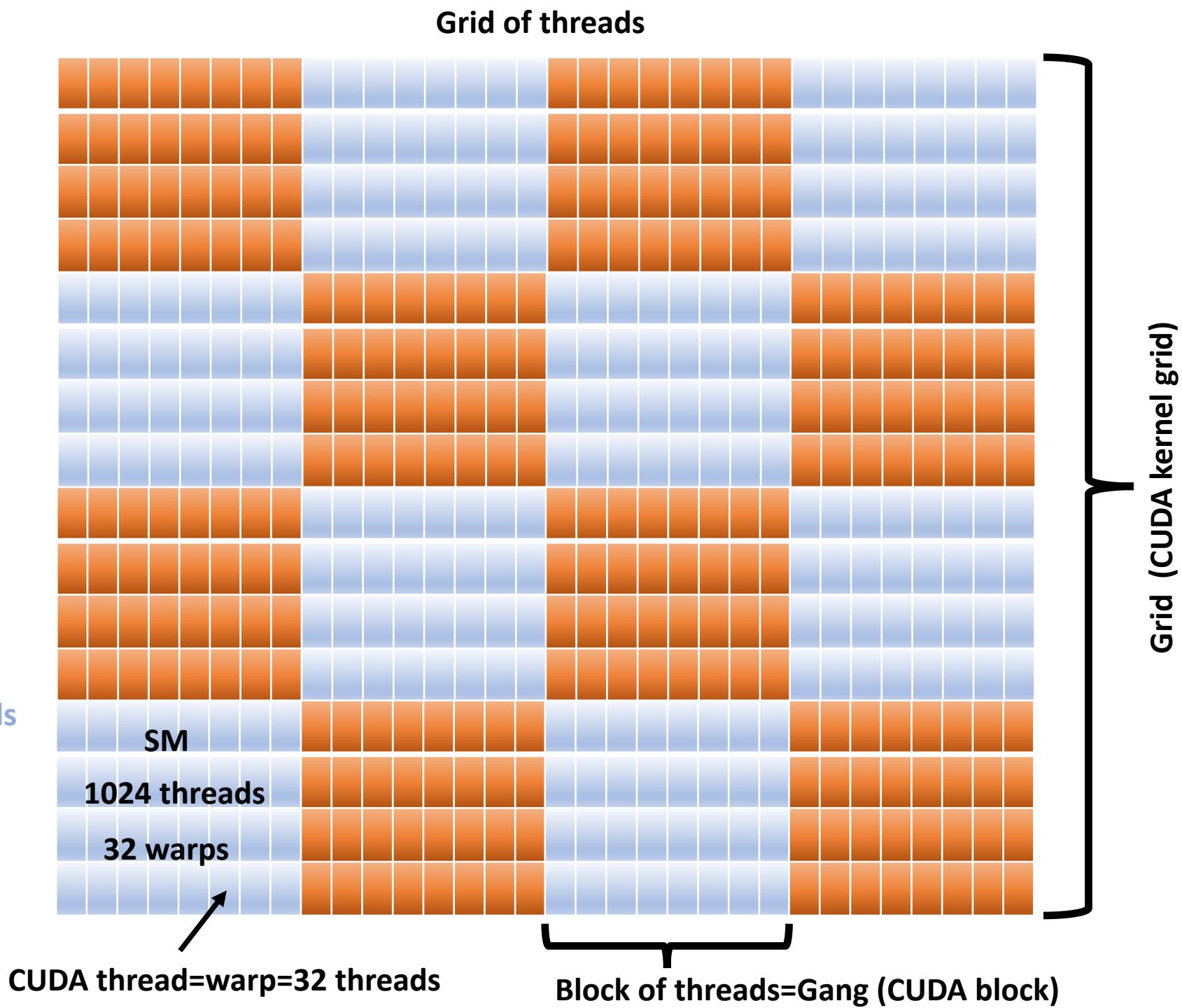
Each CUDA block has 1024 threads
(32 warps. A warp=collection of 32 threads
are executed simultaneously by a SM)

Execution on hardware

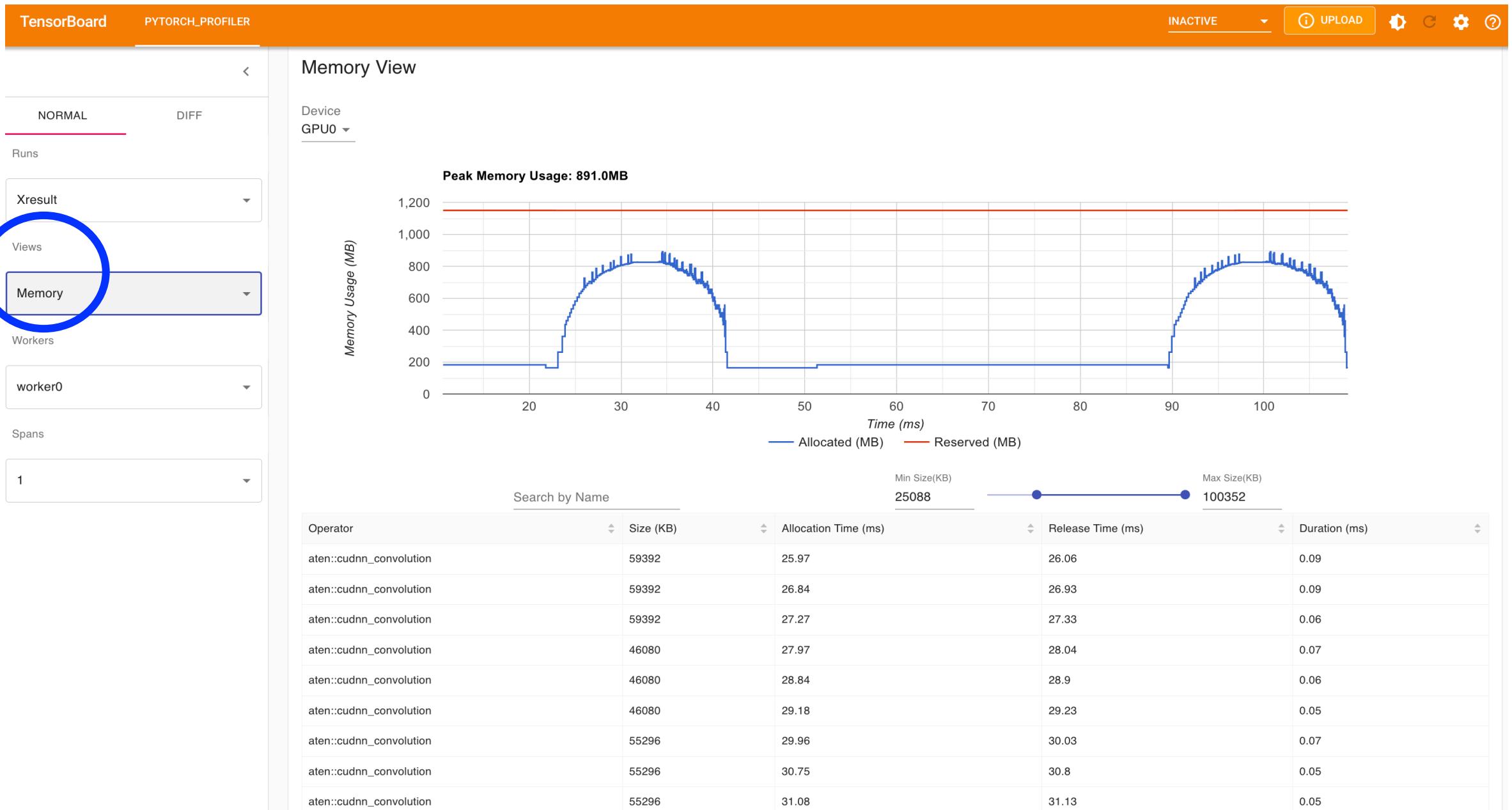
CUDA thread

CUDA block

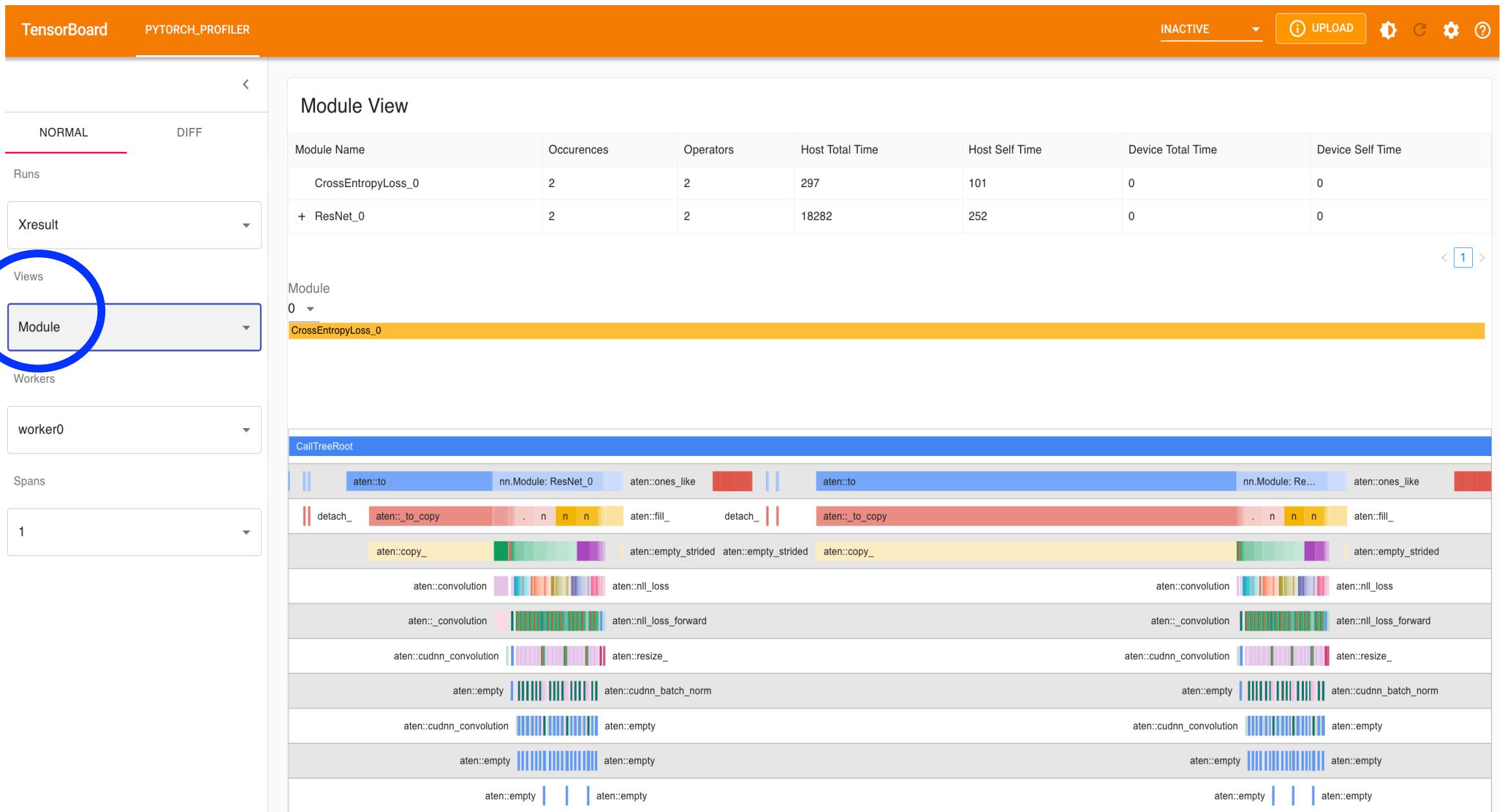
CUDA grid



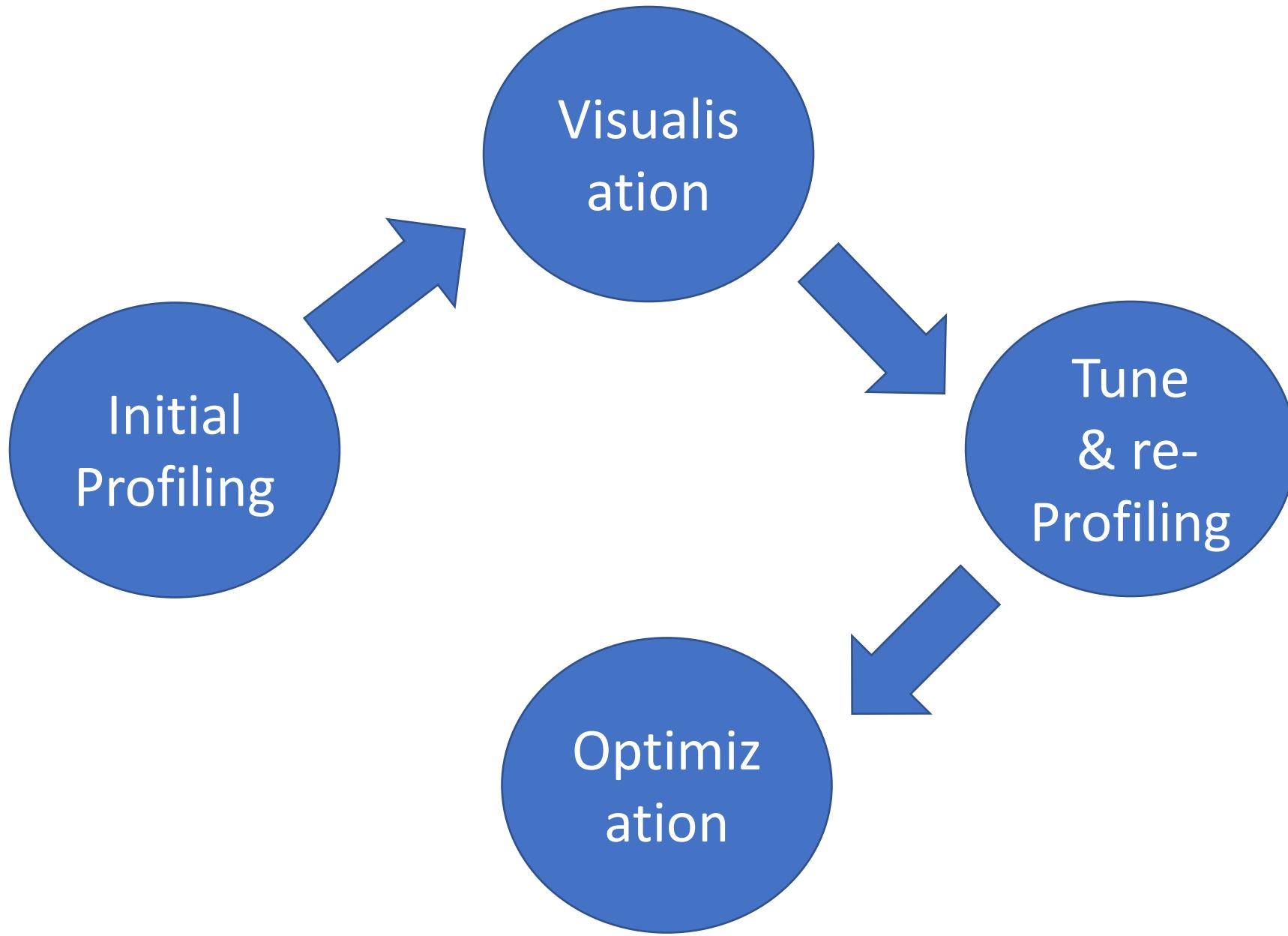
Memory View



Module View



Cycle of Profiling



Conclusion

Conclusion

- Basic understanding of GPU architecture.
- Profiling analysis becomes necessary.
 - Identify bottlenecks.
 - Improve the performance.
 - Well-utilisation of GPUs.
- Scaling analysis.

I stop HERE



Slides and DEMO are available here

cd /cluster/project/nn9987k