

AI4S
BSC AI 4 Science Fellowships

Profiling and Optimizing AI Trainings

Alexandros Paliouras – AI Engineer
Guillem Rovira Cortiada – AI Engineer
Barcelona Supercomputer Center



Agenda

1. Introduction
 - Systems' Topology
 - GPU communication (NCCL)
2. NVIDIA Nsight Systems
 - Nsys command
 - Nsight client
3. Paraver
 - Nsys2prv
 - Paraver client
4. Exercises



Agenda

1. Introduction

- **Systems' Topology**
- **GPU communication (NCCL)**

2. NVIDIA Nsight Systems

- Nsys command
- Nsight client

3. Paraver

- Nsys2prv
- Paraver client

4. Exercises

Why code profiling?

"You can have 1.000.000 GPU hours for FREE!..."



Why code profiling?

"You can have 1.000.000 GPU hours for FREE!..."

Only condition:

- Prove that all these hours will be used efficiently





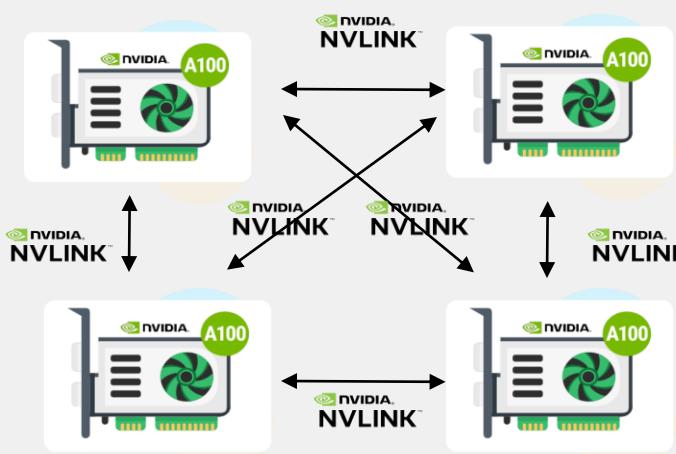
Questions that need answers...

- Is my GPU "starving" for data ?
 - Maybe the data loader is taking longer than expected
- Is the batch size optimized for memory?
 - Small batch sizes might be underutilizing the available
- How much time is spent on Communication vs. Computation?
 - Communication between gpus is blocking unrelated computations
- Is "Tensor Parallelism" (TP) causing too much intra-node latency?
 - Maybe the TP degree I have chosen is too high my hardware topology, and the available bandwidth is limiting
- Where is the "Bubble" in my Pipeline?
 - Pipeline parallelism (PP) introduces GPU dependencies, slowing down out training

How do GPUs communicate?

Intra-Node

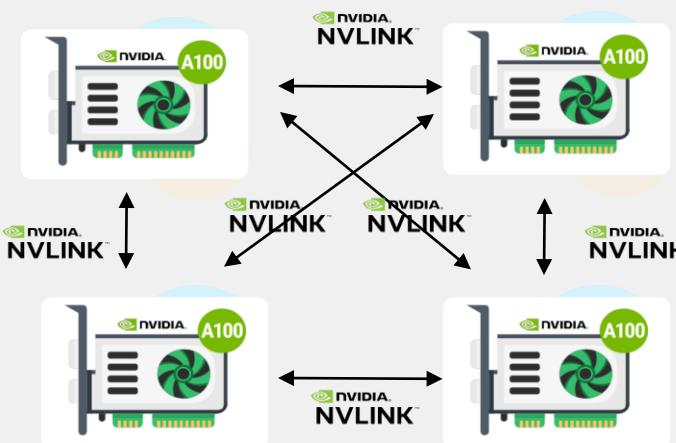
- NVLink cables
(Leonardo 64GiB HBM2e NVLink 3.0 (200 GB/s))
- NVSwitch (>4 GPUs node) + NVLink
- PCIe (cpu-gpu, slow)



How do GPUs communicate?

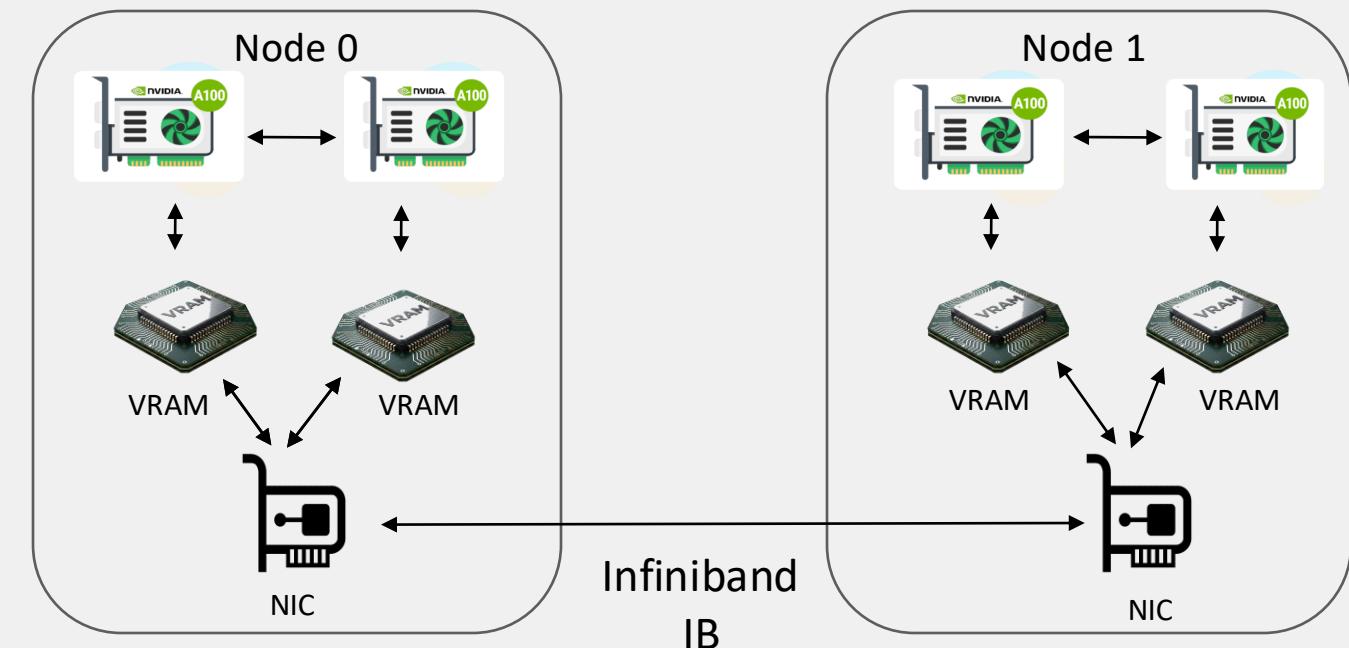
Intra-Node

- NVLink cables
(Leonardo 64GiB HBM2e NVLink 3.0 (200 GB/s))
- NVSwitch (>4 GPUs node) + NVLink
- PCIe (cpu-gpu, slow)



Inter-Node

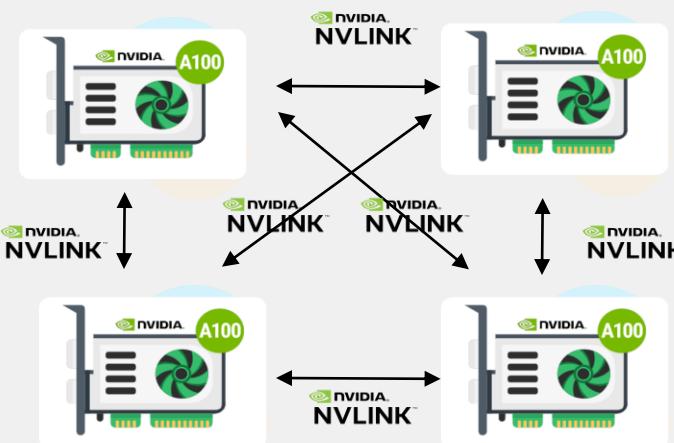
- InfiniBand GPU inter-node Remote Direct Memory Access
(Leonardo 200 Gbps~25GB/s NVIDIA Mellanox HDR InfiniBand)
- Ethernet (fallback, slow)



How do GPUs communicate?

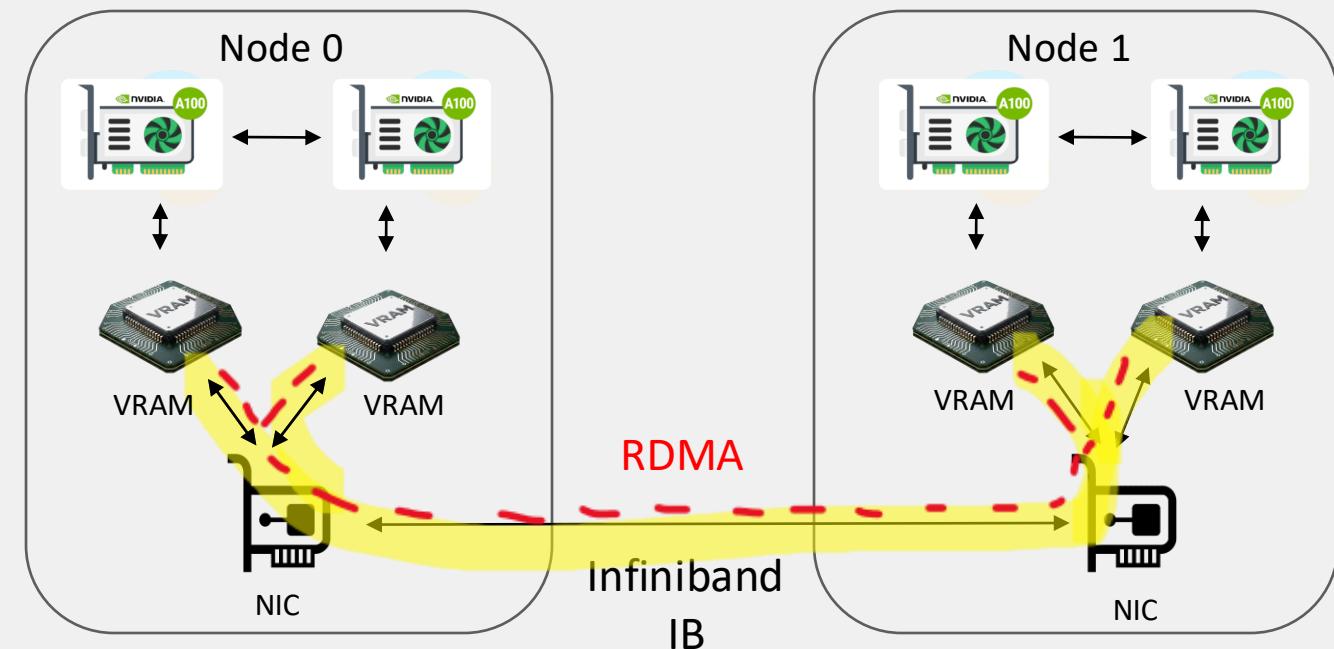
Intra-Node

- NVLink cables
(Leonardo 64GiB HBM2e NVLink 3.0 (200 GB/s))
- NVSwitch (>4 GPUs node) + NVLink
- PCIe (cpu-gpu, slow)



Inter-Node

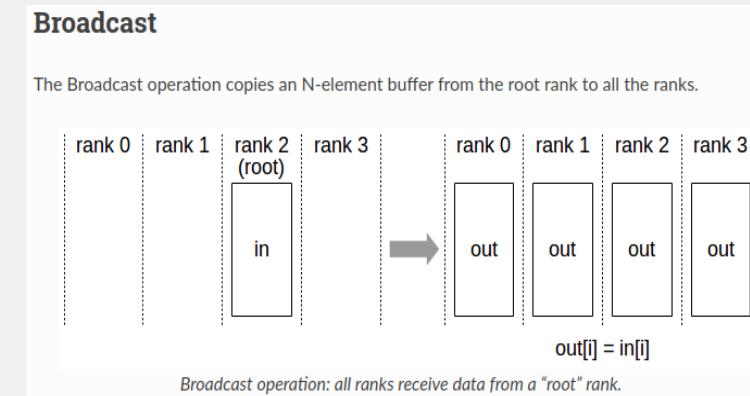
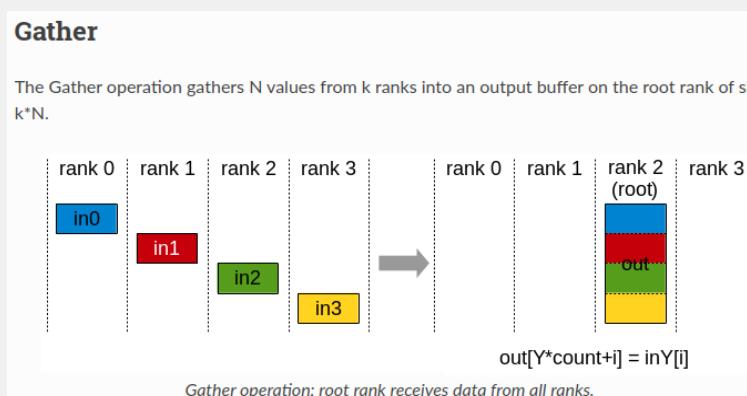
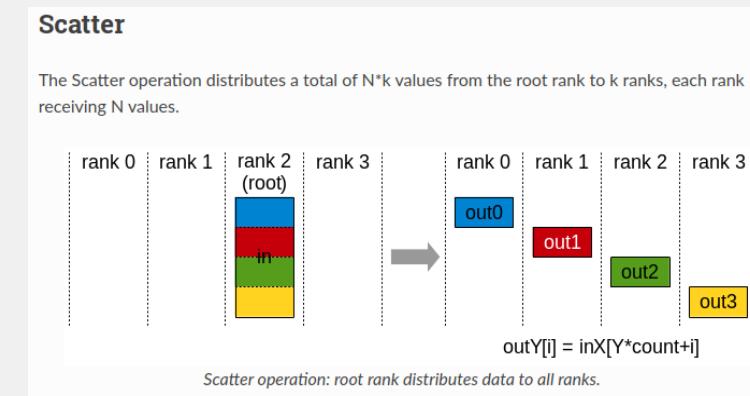
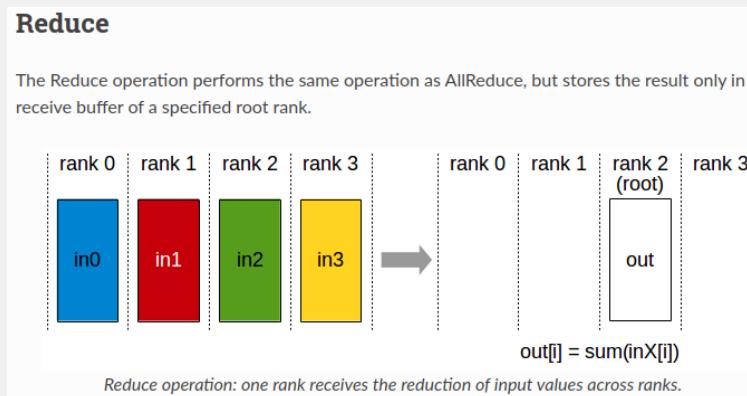
- InfiniBand GPU inter-node Remote Direct Memory Access
(Leonardo 200 Gbps~25GB/s NVIDIA Mellanox HDR InfiniBand)
- Ethernet (fallback, slow)



How do GPUs communicate?

NCCL: NVIDIA Collective Communication Library

- Implements multi-GPU and multi-node communication primitives optimized for NVIDIA GPUs and networking.



How do GPUs communicate?

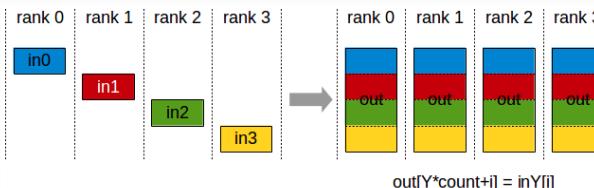
NCCL: NVIDIA Collective Communication Library

- Implements multi-GPU and multi-node communication primitives optimized for NVIDIA GPUs and networking.

AllGather

The AllGather operation gathers N values from k ranks into an output buffer of size $k \times N$, and distributes that result to all ranks.

The output is ordered by the rank index. The AllGather operation is therefore impacted by a different rank to device mapping.

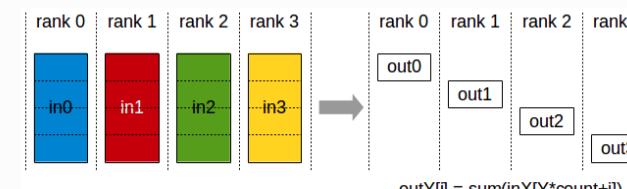


AllGather operation: each rank receives the aggregation of data from all ranks in the order of the ranks.

ReduceScatter

The ReduceScatter operation performs the same operation as Reduce, except that the result is scattered in equal-sized blocks between ranks, each rank getting a chunk of data based on its rank index.

The ReduceScatter operation is impacted by a different rank to device mapping since the ranks determine the data layout.

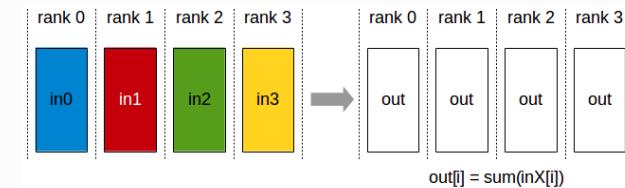


Reduce-Scatter operation: input values are reduced across ranks, with each rank receiving a subpart of the result.

AllReduce

The AllReduce operation performs reductions on data (for example, sum, min, max) across devices and stores the result in the receive buffer of every rank.

In a sum allreduce operation between k ranks, each rank will provide an array in of N values, and receive identical results in array out of N values, where $out[i] = in0[i] + in1[i] + \dots + in(k-1)[i]$.

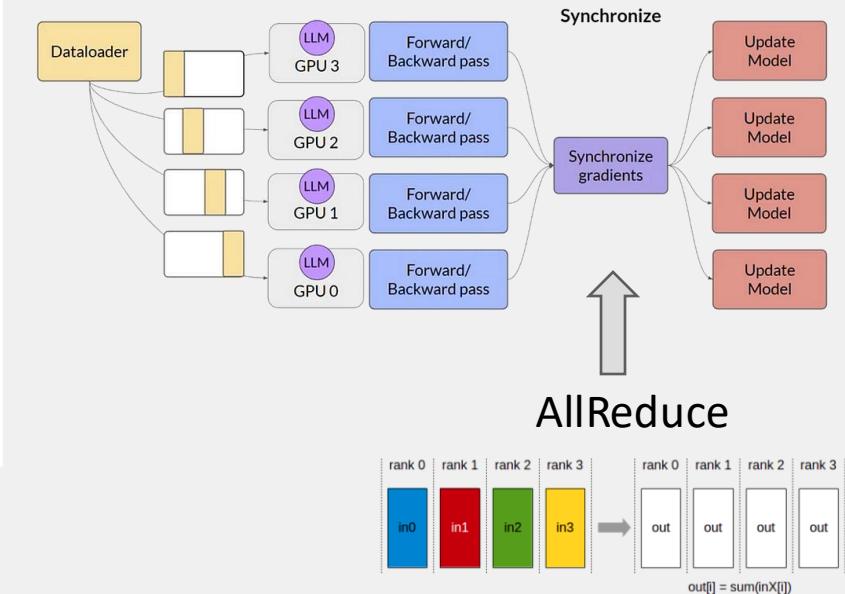


All-Reduce operation: each rank receives the reduction of input values across ranks.

DDP Communication

Collective	Role
AllReduce	Aggregates gradients across all GPUs after backward pass. This is the default and the most performance-critical operation in DDP.
Broadcast	Used at initialization to broadcast model weights from a master rank to all other ranks, ensuring consistent start states before training.

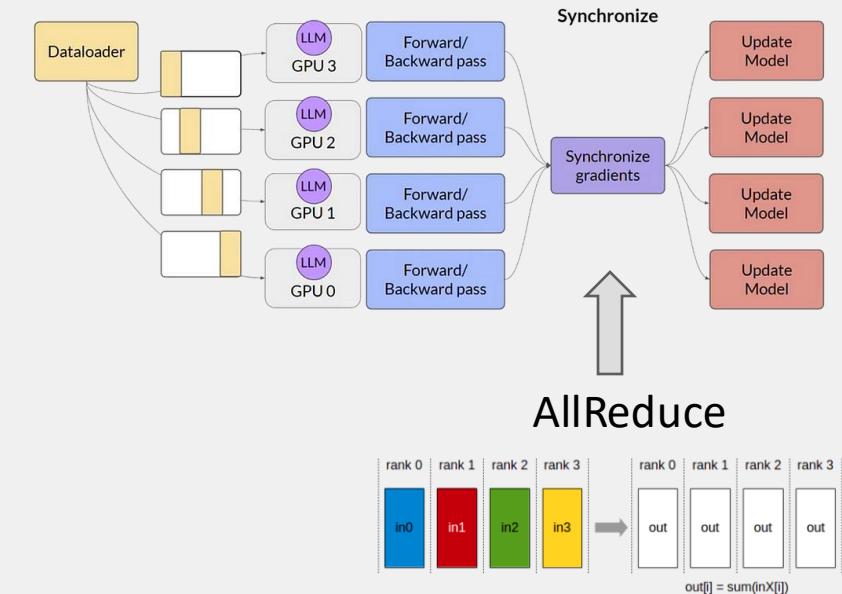
Distributed Data Parallel (DDP)



DDP Communication

Collective	Role
AllReduce	Aggregates gradients across all GPUs after backward pass. This is the default and the most performance-critical operation in DDP.
Broadcast	Used at initialization to broadcast model weights from a master rank to all other ranks, ensuring consistent start states before training.

Distributed Data Parallel (DDP)



Trace Pattern:

Forward GEMM → Backward GEMM → AllReduce (bucketed)

*GEMM: CUDA kernel for General Matrix Multiplication



Larger Model VRAM requirements - example

- **Training static states:**
 - Weights
 - Gradients
 - Optimizer states
- **ADAM optimizer:**
 - Master Weights (FP32)
 - Momentum & variance (FP32)
for each model parameter
- **Dynamic or live states:**
 - Activations (depends batch size, hidden layers, sequence length, etc.)
 - Activation gradients
 - Communication & precision casting buffers
 - Etc.

Component	Per Billion Params	Mistral 7B Total	Equal partitions (4 GPUs)
Model weights	x2bytes → 2 GB (FP16/BF16)	14 GB	3.5 GB / GPU
Gradients	x2bytes → 2 GB (FP16/BF16)	14 GB	3.5 GB / GPU
Optimizer States	3x4bytes → 12 GB (Adam)	84 GB	21 GB / GPU
Total Static	16 GB	112 GB	28 GB / GPU

Dynamic states can vary in this example 8-20GB or more



FSDP Communication

Collective

Role

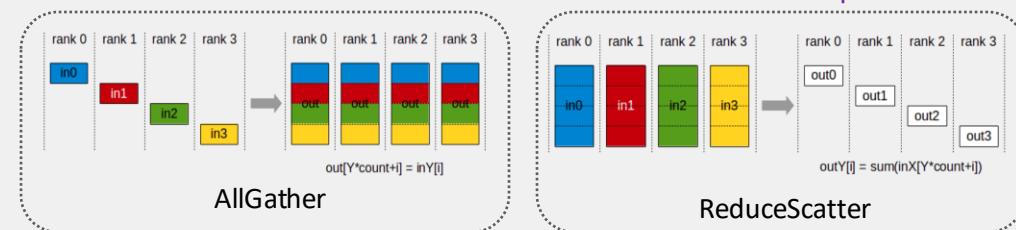
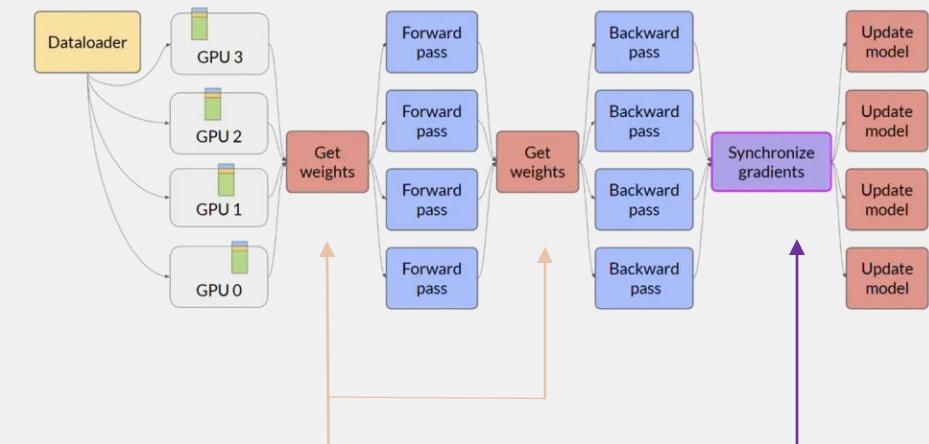
AllGather

During forward and backward pass, before computing a layer, the full parameter shard is assembled across GPUs, so that each GPU can compute activations, attention matrices, etc.
(Q-K-V projection weights, embeddings, MLP weights)

ReduceScatter

After backward pass, gradient shards are reduced and scattered back across GPUs.

Fully Sharded Data Parallel (FSDP)





FSDP Communication

Collective	Role
AllGather	During forward and backward pass, before computing a layer, the full parameter shard is assembled across GPUs, so that each GPU can compute activations, attention matrices, etc. (Q-K-V projection weights, embeddings, MLP weights)
ReduceScatter	After backward pass, gradient shards are reduced and scattered back across GPUs.

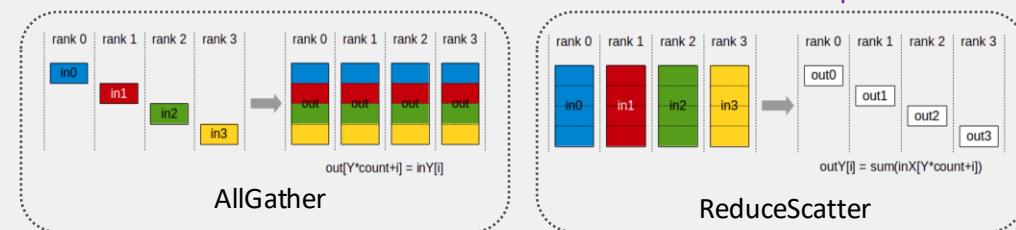
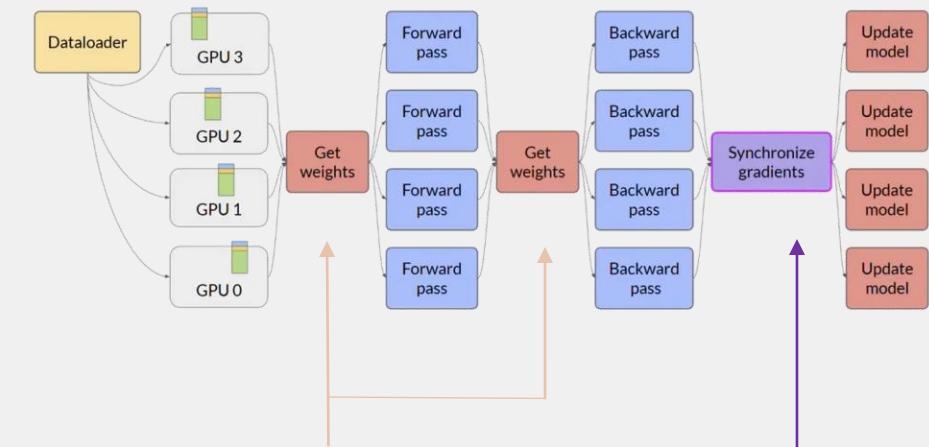
Trace Pattern:

AllGather → Forward GEMM → AllGather → Backward GEMM → ReduceScatter

FSDP compared to DDP:

- More frequent collectives
- Smaller collective sizes
- Higher sensitivity to latency
- More opportunity for communication-computation overlap

Fully Sharded Data Parallel (FSDP)





Deepspeed Communication - ZeRO1

Feature	ZeRO Stage 1
Parameter storage	Replicated
Gradient storage	Replicated
Optimizer state	Sharded
Forward communication	None
Backward communication	AllReduce
Optimizer communication	Minimal
Memory efficiency	Medium
Communication frequency	Low

Trace Pattern:

ZeRO1: Backward GEMM → AllReduce (large buckets) [Similar to DDP]

Deepspeed Communication - ZeRO2

Feature	ZeRO Stage 1	ZeRO Stage 2
Parameter storage	Replicated	Replicated
Gradient storage	Replicated	Sharded
Optimizer state	Sharded	Sharded
Forward communication	None	None
Backward communication	AllReduce	ReduceScatter
Optimizer communication	Minimal	Minimal
Memory efficiency	Medium	High
Communication frequency	Low	Medium

Trace Pattern:

ZeRO1: Backward GEMM → AllReduce (large buckets) [Similar to DDP]

ZeRO2: Backward GEMM → ReduceScatter



Deepspeed Communication - ZeRO3

Feature	ZeRO Stage 1	ZeRO Stage 2	ZeRO Stage 3
Parameter storage	Replicated	Replicated	Sharded
Gradient storage	Replicated	Sharded	Sharded
Optimizer state	Sharded	Sharded	Sharded
Forward communication	None	None	AllGather (per layer)
Backward communication	AllReduce	ReduceScatter	AllGather, ReduceScatter
Optimizer communication	Minimal	Minimal	Minimal
Memory efficiency	Medium	High	Very High
Communication frequency	Low	Medium	High

Trace Pattern:

ZeRO1: Backward GEMM → AllReduce (large buckets) [Similar to DDP]

ZeRO2: Backward GEMM → ReduceScatter

ZeRO3: AllGather → Forward GEMM → AllGather → Backward GEMM → ReduceScatter [Similar to FSDP]

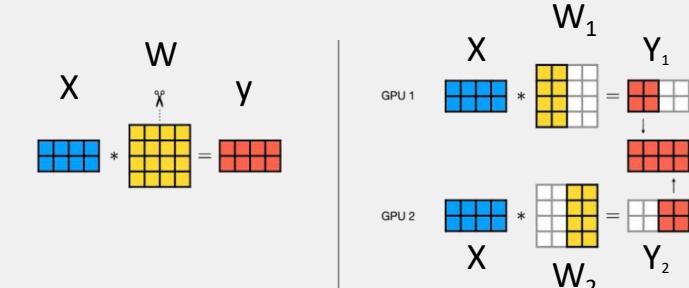
MegatronLM Communication

Parallelism Type	Collective Used	Forward	Backward
Data Parallel	AllReduce	-	After backward on weight updates
Pipeline Parallel	Send / Recv (P2P)	Transfer activations (outputs) between pipeline stages	Transfer gradients between pipeline stages
Tensor Parallel	-	ColumnParallelLinear -> RowParallelLinear	-
	AllGather	ColumnParallelLinear -> ColumnParallelLinear (rare)	-
	AllReduce	RowParallelLinear -> ColumnParallelLinear	ColumnParallelLinear layer input gradients only * (all other gradients are computed locally on each rank)

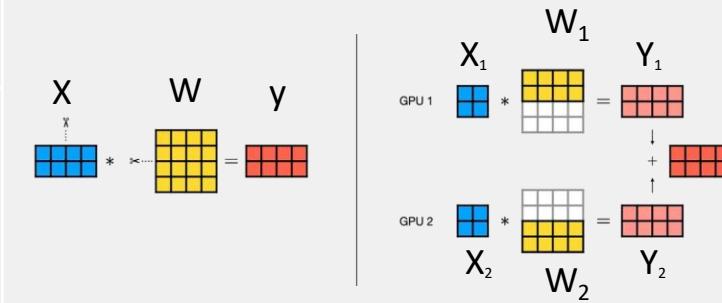
* $dX = dY W^T$, $W = [W_1; W_2; \dots; W_{TP-ranks}]$, $Y = [Y_1, Y_2, \dots, Y_{TP-ranks}]$

$$\text{Linear Layer: } Y = W^T X + b$$

Column Parallel Linear



Row Parallel Linear





Agenda

1. Introduction
 - Systems' Topology
 - GPU communication (NCCL)
2. NVIDIA Nsight Systems
 - **Nsys command**
 - **Nsight client**
3. Paraver
 - Nsys2prv
 - Paraver client
4. Exercises



Nvidia Nsight Systems - Terminology

Kernel: function executed on the GPU.

- ampere_bf16_gemm → matrix multiplication
- attn_fwd_kernel → attention
- ncclDevKernl_AllReduce → communication

CUDA API: CPU-GPU interface, set of functions that allow our program to launch kernels, allocate GPU memory, copy data, synchronize execution, etc.

- cudaMalloc, cudaMemcpy, cudaStreamSynchronize, etc.

NVTX: lightweight annotation API to create custom markers and labels in the code. Used in torch.cuda.nvtx.

- "Batch X", "forward", "backward", "data loading X"

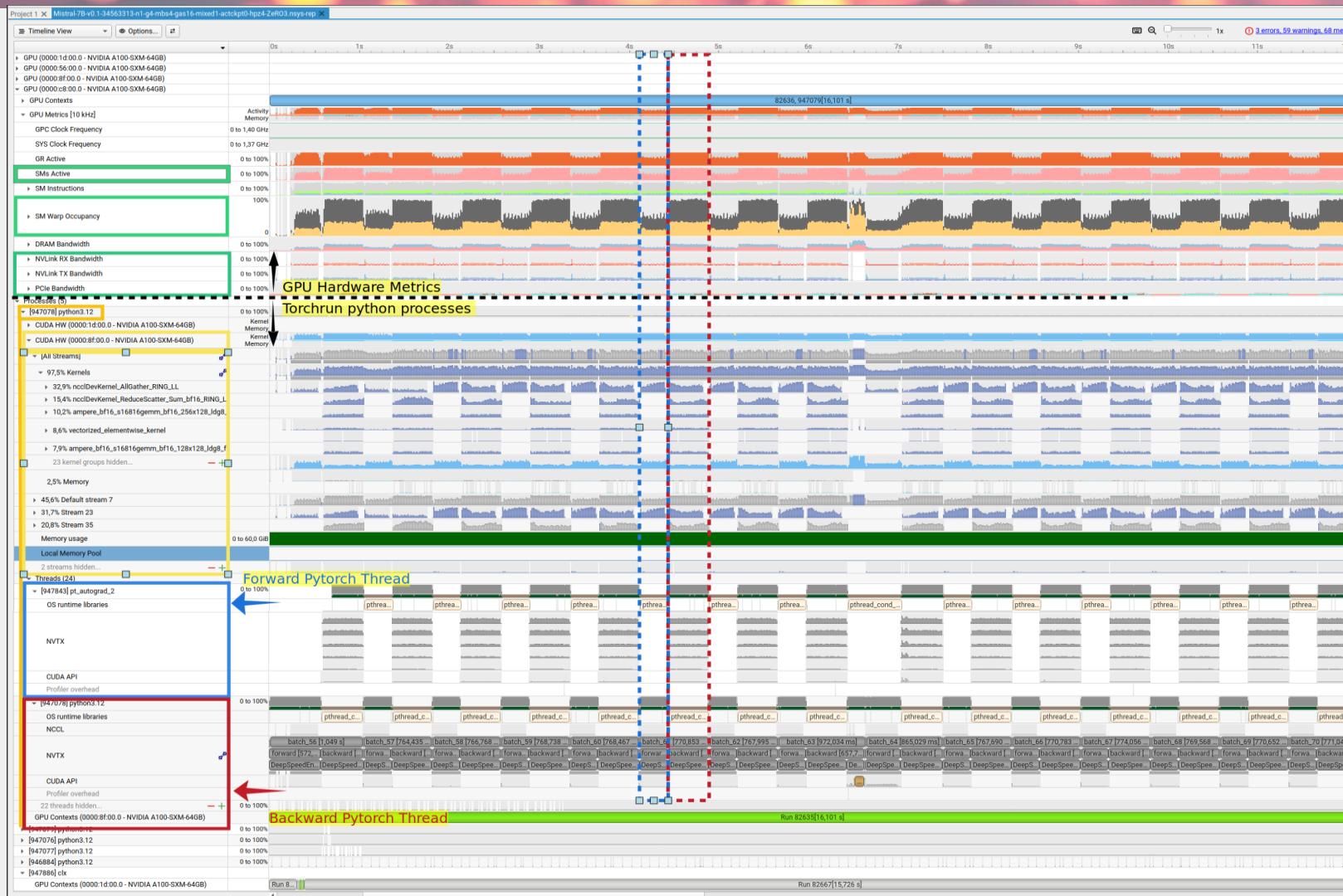
Stream Processors (SM): A compute unit inside the GPU. The A100 has 108 SMs.

GPU Thread: Smallest execution unit

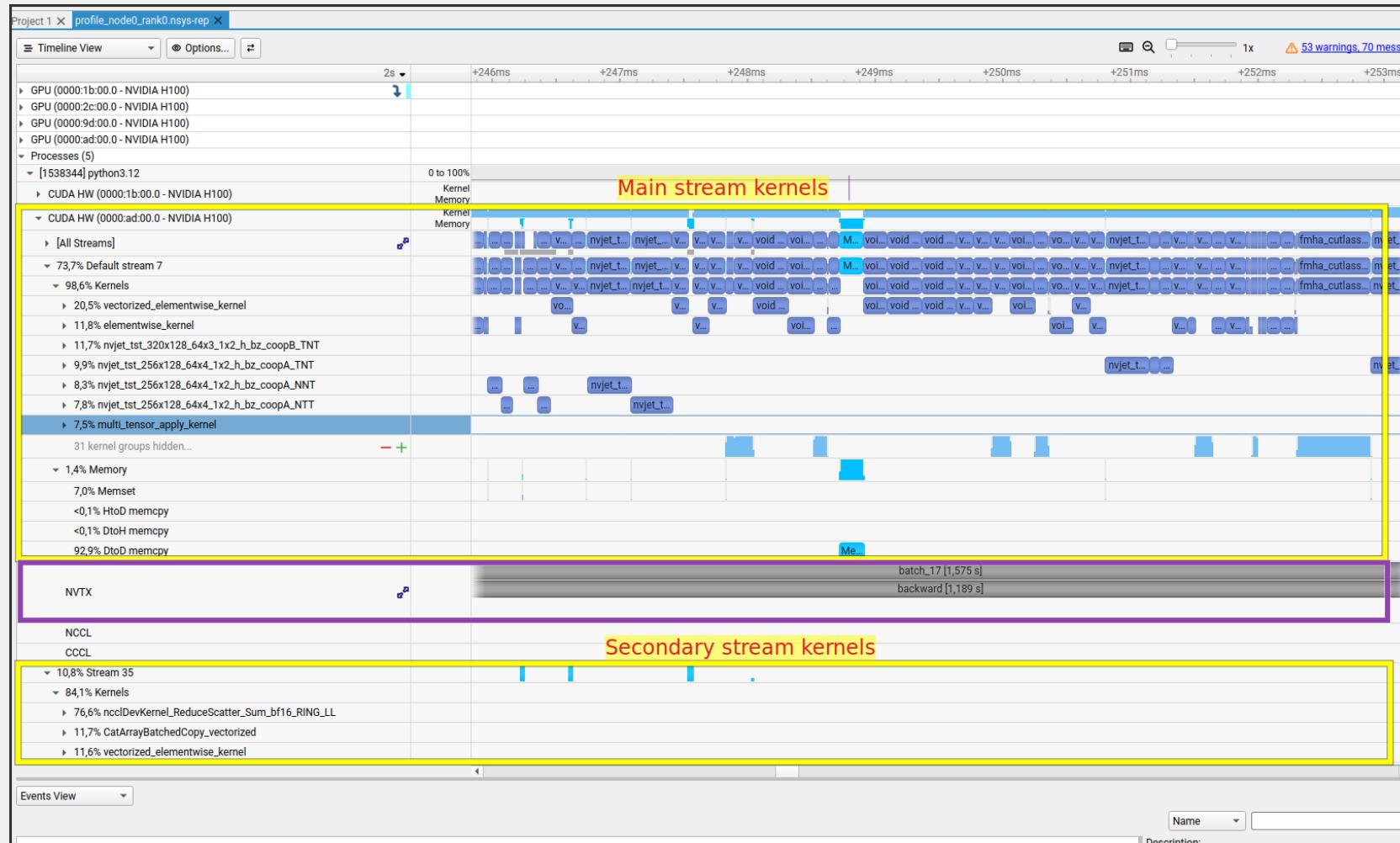
Warp: Group of GPU threads that execute together (commonly 32)

- **SM Active %:** Percentage of time SMs are doing work.
- **SM Instructions Tensor Active:** Tensor core activity for accelerate tensor multiplication
- **Warp Occupancy:** How many warps are ready to run per SM
- **(CUDA) Stream:** Execution queue

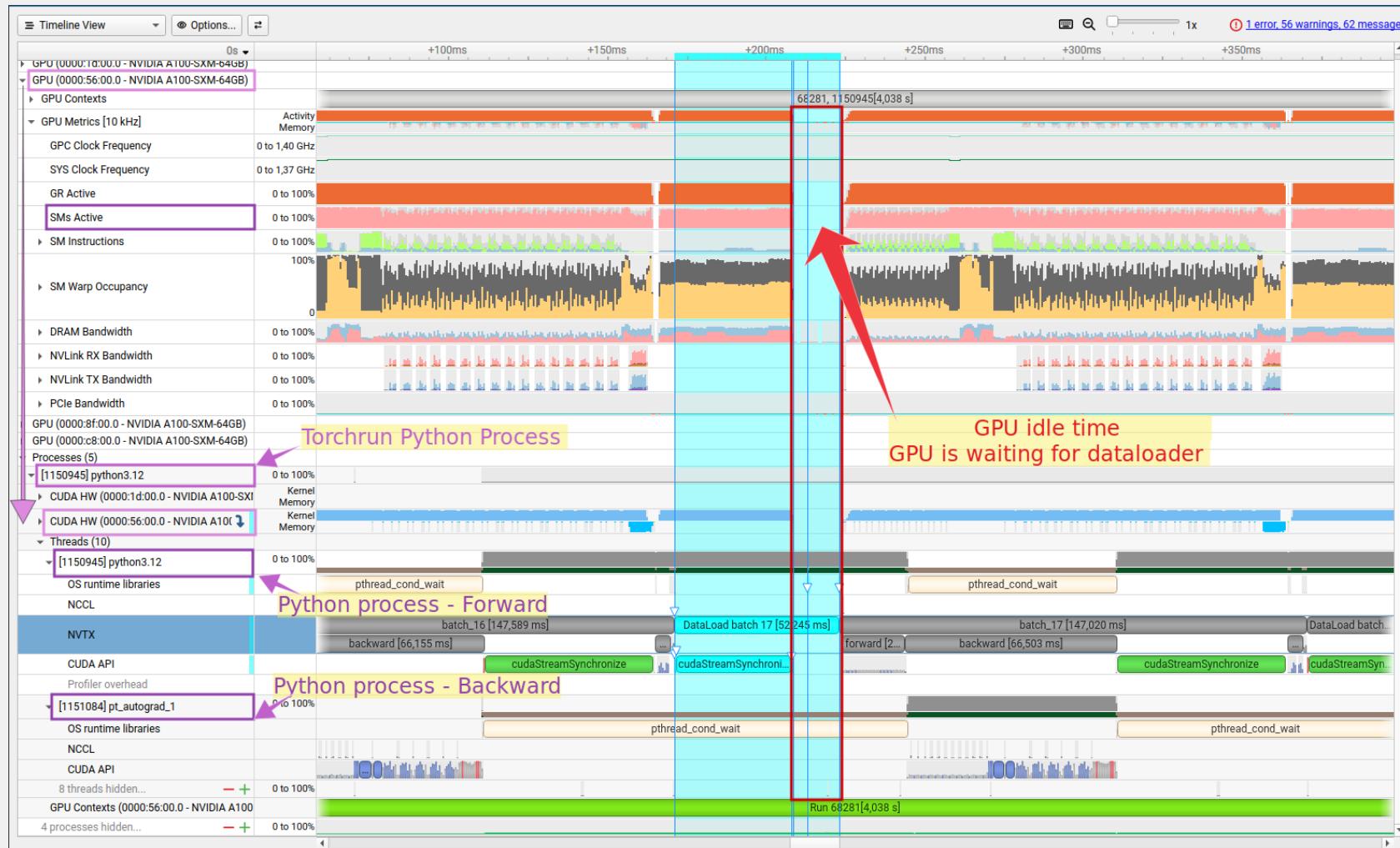
Nvidia Nsight Systems - Events View



Nvidia Nsight Systems - Events View



Nvidia Nsight Systems – Idle time





Nvidia Nsight Systems – Stats View

CUDA API Summary

NVTX Push/Pop Range Summary

CUDA GPU Kernel Summary



Nvidia Nsight Systems – nsys command

'nsys' command

```
[apalior@login05 ~]$ module purge
[apalior@login05 ~]$ module load cuda/12.6
[apalior@login05 ~]$ which nsys
/leonardo/prod/opt/compilers/cuda/12.6/none/bin/nsys
[apalior@login05 ~]$ nsys
usage: nsys [l--version] [--help] <command> [<args>] [application] [<application args>]

The most commonly used nsys commands are:
  profile      Run an application and capture its profile into a nsys-rep file.
  launch       Launch an application ready to be profiled.
  start        Start a profiling session.
  stop         Stop a profiling session and capture its profile into a nsys-rep file.
  cancel       Cancel a profiling session and discard any collected data.
  service      Launch the Nsight Systems data service.
  stats        Generate statistics from an existing nsys-rep or SQLite file.
  status       Provide current status of CLI or the collection environment.
  shutdown     Disconnect launched processes from the profiler and shutdown the profiler.
  sessions list List active sessions.
  export       Export nsys-rep file into another format.
  analyze     Identify optimization opportunities in a nsys-rep or SQLite file.
  recipe      Run a recipe for multi-node analysis.
  nvprof      Translate nvprof switches to nsys switches and execute collection.

Use 'nsys --help <command>' for more information about a specific command.

To run a basic profiling session:  nsys profile ./my-application
For more details see "Profiling from the CLI" at https://docs.nvidia.com/nsight-systems
```

'nsys profile' sub-command

```
[apalior@login05 ~]$ nsys profile --help

usage: nsys profile [<args>] [application] [<application args>]

# =====#
# NSYS Configuration
# =====#
# Recommended NSYS options for ML training profiling:
#   --trace=cuda,nvtx,osrt,cudnn,ucx,cublas : Trace GPU kernels, NVTX markers, OS runtime, cuDNN, UCX, NCCL, cuBLAS
#   --force-overwrite true                  : Overwrite existing profile files
#   --cuda-memory-usage=true               : Track CUDA memory allocations and usage
#   --gpuctxsw=true                      : Track GPU context switches (optional, rarely a bottleneck)
#   --gpu-metrics-devices=all             : Collect GPU metrics (SM utilization, memory throughput, etc.)
#   --gpu-metrics-frequency=10000          : Sample GPU metrics at 10KHz for fine granularity
#   --capture-range=cudaProfilerApi       : Use cudaProfiler start/stop for precise capture (requires NVTX markers in code)
#   --capture-range-end=stop              : End capture when cudaProfilerStop is called
#   --sample=cpu                          : CPU sampling for host-side bottlenecks
#   --backtrace=dwarf                   : Detailed backtraces for CPU samples
#   --cudabacktrace=kernel              : Collect backtraces for CUDA kernel launches
#   --stats=true                         : Generate summary statistics
#   --export=sqlite                      : Export results in SQLite format for advanced analysis
#   --output=<path>                     : Set output file path (already used)
# =====#
```

'nsys profile' example

```
nsys profile
  -trace=cuda,nvtx,osrt,cudnn,cublas
  -cuda-memory-usage=true
  -gpuctxsw=true
  -gpu-metrics-devices=all
  -gpu-metrics-frequency=10000
  -capture-range=cudaProfilerApi
  -capture-range-end=stop
  -cudabacktrace=kernel
  -stats=true
  -force-overwrite=true
  -output=/exercise_2_DeepSpeed/profiler/Mistral-7B-v0.1-34563983-n1-q4-mbs4-gas16-mixed1-actckpt0-hpz4-ZeR03-badcomm1/nsys/profile_node%{SLURM_NODEID}
singularity exec
  -n
  --bind /leonardo
  --bind /Leonardo
  /leonardo_work/tra26_minwinc/containers/ai-profiling-workshop.sif
    accelerate launch
      --config_file ./exercise_2_DeepSpeed/accelerate_config.yaml-H80Gt1r6yDfrb
      --rdzy_backend=c10d
      --machine_rank 0
      -m exercise_2_DeepSpeed.train
      --data_path /leonardo_work/tra26_minwinc/DATA/alpaca-cleaned/alpaca_data_cleaned.json
      --model_path /leonardo_work/tra26_minwinc/models/Mistral-7B-v0.1
      --epochs 1
      --no-validation
      --profile
      --data-sample 5000
      --dataloader-num-workers 8
      --batch-size 4
      --gradient-accumulation-steps 16
      -deepspeed-config ./exercise_2_DeepSpeed/ds_configs/stage-3/ds_config_mixed-precision_no-activation-checkpointing_comm-overhead.json
```

"nsys profile" options

Application to profile

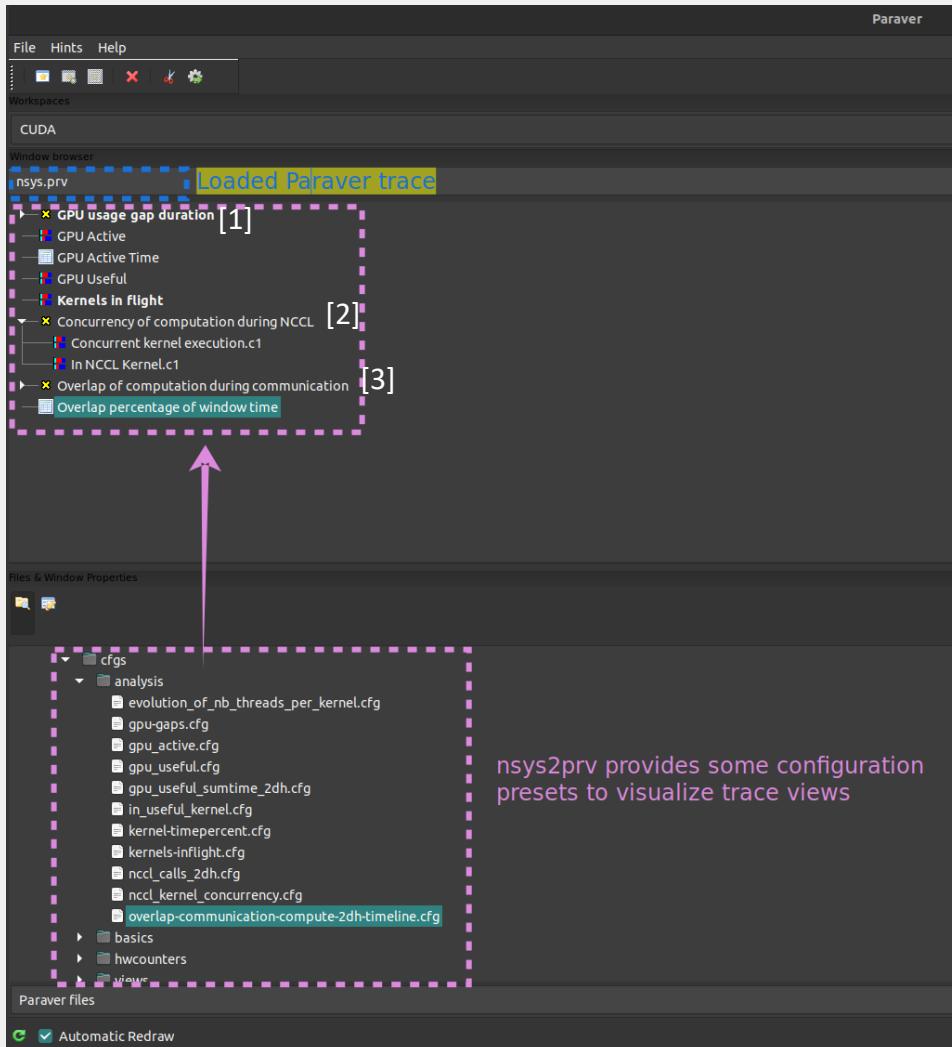


Agenda

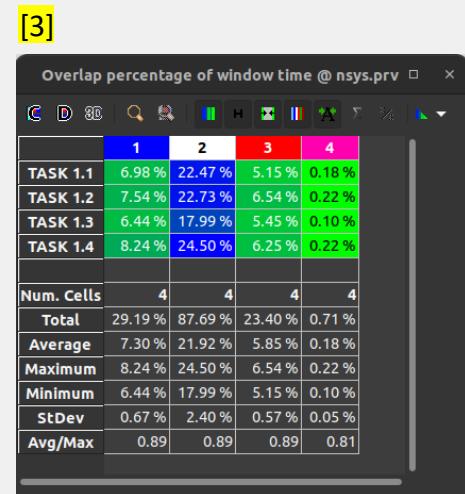
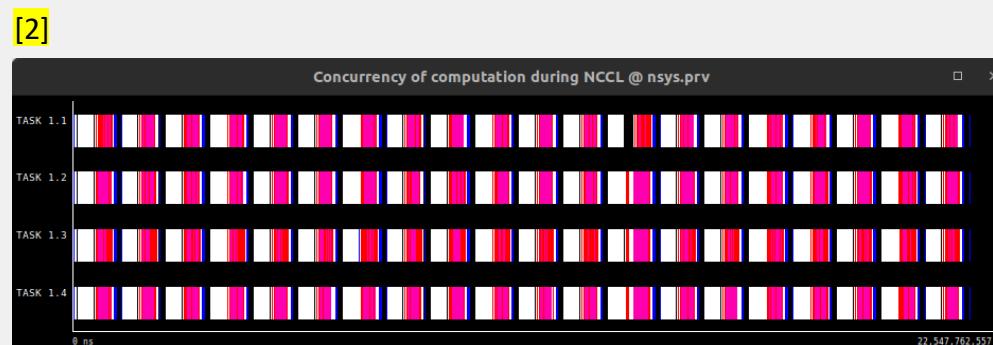
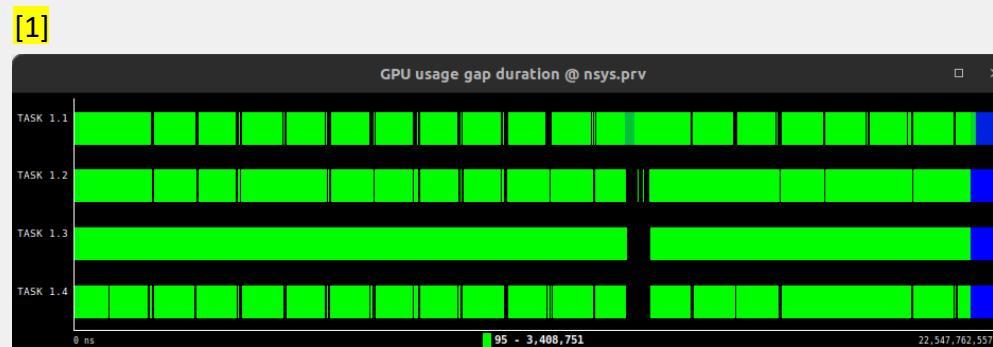
1. Introduction
 - NCCL
 - Systems' Topology
2. NVIDIA Nsight Systems
 - Nsys command
 - Nsight client
3. Paraver
 - **Nsys2prv**
 - **Paraver client**
4. Demo
5. Hands-on Exercises



Paraver



`nsys2prv` provides some configuration
presets to visualize trace views





nsys2prv – Trace translation

The screenshot shows the GitHub project page for nsys2prv 0.5.8. The page includes a search bar, navigation links (Help, Docs, Sponsors, Log in, Register), and a release section indicating it's the latest version (released Jul 2, 2025). A prominent button says "pip install nsys2prv". Below the header, there's a brief description: "Translate a NVIDIA Nsight System trace to a Paraver trace". The main content area has sections for Navigation (Project description, Release history, Download files), Project description (Logo, nsys2prv: Translate NVIDIA Nsight Systems traces to Paraver traces, PyPI v0.5.8, Gitlab Pipeline Status), Verified details (PyPI verification status), Owner (Group for Best Practices for Performance and Programmability of the Barcelona Supercomputing Center), Unverified details (PyPI verification status), Project links (Homepage, Repository), and Meta (License: GNU General Public License v3 (GPLv3) (GPL-3.0-only), Author: Marc Clascà, Requirements: Python >=4.0, >=3.10). The Project description section contains detailed information about the tool's purpose, how it works, and its installation requirements.

- Python cli tool developed by BSC.
- Translate Nvidia NSYS traces to Paraver traces.

```
> nsys2prv --help
nsys2prv v0.5.8
usage: nsys2prv [-h] [-v] [-f FILTER_NVTX] [-t TRACE] [-m] [--force-sqlite] [-s] [-z] source_rep [source_rep ...] output

Translate a NVIDIA Nsight System trace to a Paraver trace

positional arguments:
  source_rep          Nsight source report file
  output              Paraver output trace name

options:
  -h, --help           show this help message and exit
  -v, --version        Show version and exit. (default: None)
  -f FILTER_NVTX, --filter-nvtx FILTER_NVTX
                      Filter by this NVTX range (default: None)
  -t TRACE, --trace TRACE
                      Comma separated names of events to translate: [mpi_event_trace, nvtx_pushpop_trace, nvtx_startend_trace, cuda_api_trace, gpu_metrics, openacc, nccl, graphs] (default: None)
  -m, --multi-report  Translate multiple reports of the same execution into one trace. (default: False)
  --force-sqlite       Force Nsight System to export SQLite database (default: False)
  -s, --sort           Sort trace at the end (default: False)
  -z, --compress       Compress trace at the end with gzip (default: False)

The nsys executable needs to be in the PATH, or the environment variable NSYS_HOME needs to be set. If using postprocessing, the PARAVER_HOME variable needs to be set.
```



Agenda

1. Introduction

- Systems' Topology
- GPU communication (NCCL)

2. NVIDIA Nsight Systems

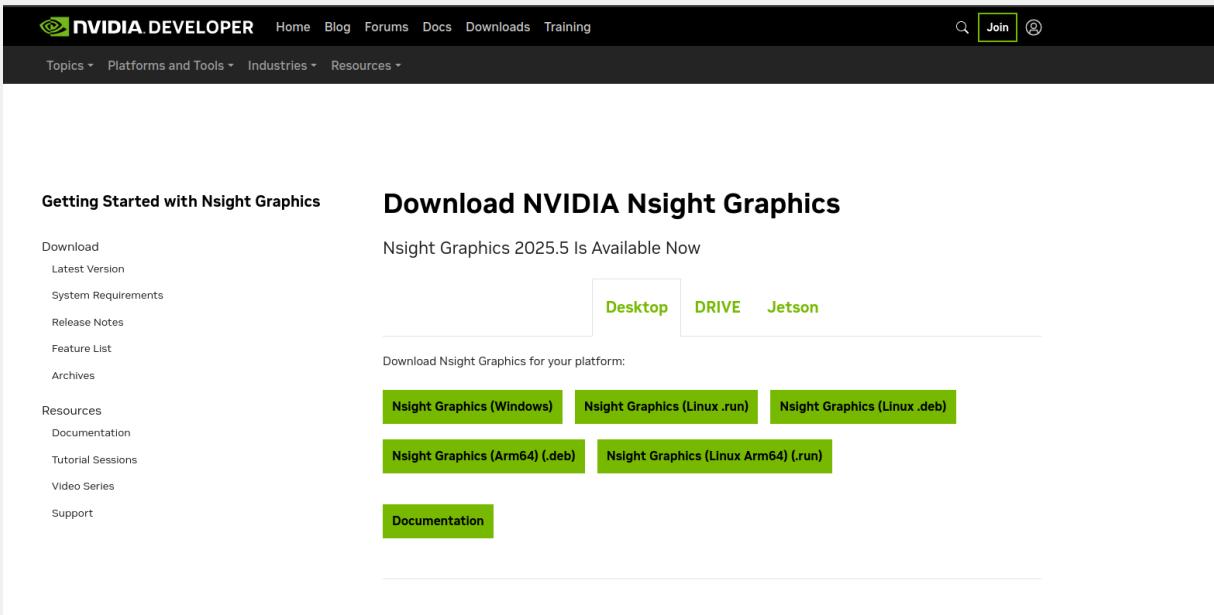
- Nsys command
- Nsight client

3. Paraver

- Nsys2prv
- Paraver client

4. Exercises

Download and setup Nsight



The screenshot shows the NVIDIA Developer website with the following navigation bar:

- NVIDIA DEVELOPER
- Home
- Blog
- Forums
- Docs
- Downloads
- Training

Search and user account icons are also present.

The main content area is titled "Getting Started with Nsight Graphics" and "Download NVIDIA Nsight Graphics". It states "Nsight Graphics 2025.5 Is Available Now". Below this, there are tabs for "Desktop", "DRIVE", and "Jetson". A section for "Download Nsight Graphics for your platform:" lists the following options:

- Nsight Graphics (Windows)
- Nsight Graphics (Linux .run)
- Nsight Graphics (Linux .deb)
- Nsight Graphics (Arm64) (.deb)
- Nsight Graphics (Linux Arm64) (.run)

A "Documentation" button is also visible.

- Download Nvidia Nsight Systems @ <https://developer.nvidia.com/nsight-graphics/get-started>
 - Install on your computer
3. Validate client is running

If in Linux, prefer to download ".deb" and install with:

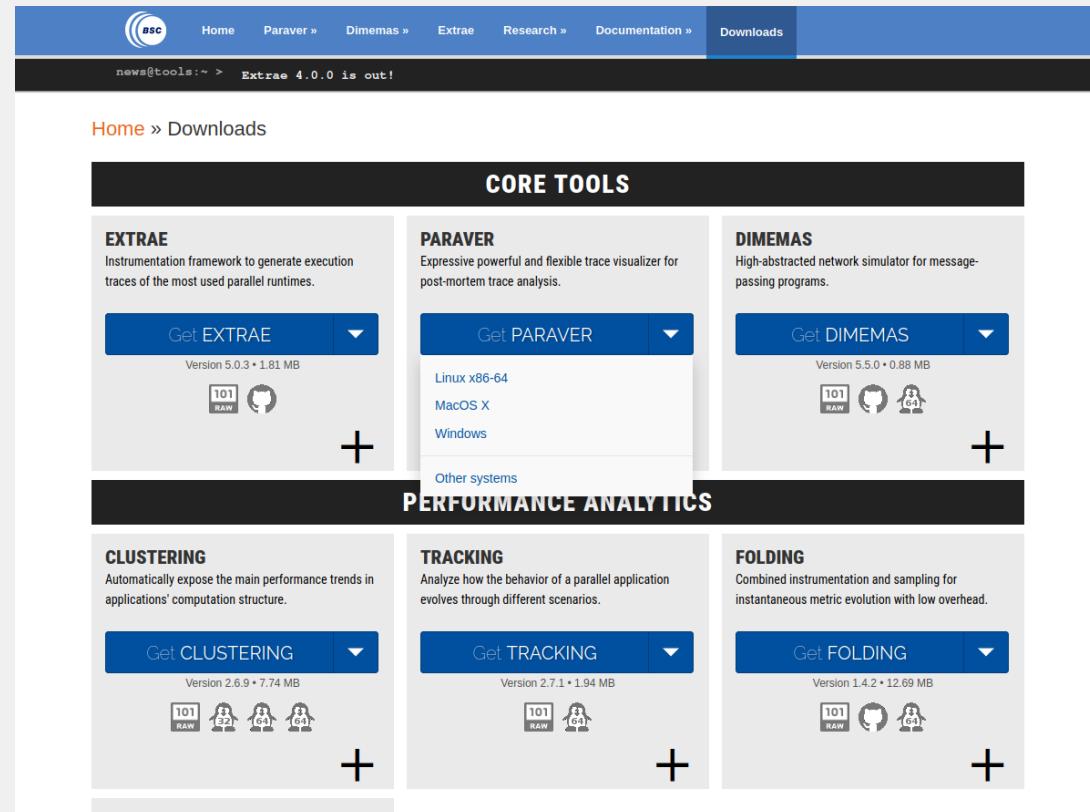
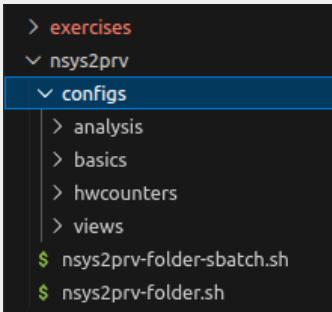
```
> sudo dpkg install NVIDIA_Nsight_Graphics_2025.5.0.25335.deb
```

You can check your CPU architecture by running:

```
> arch
```

Download and setup Paraver

1. Download Paraver
@ <https://tools.bsc.es/downloads>
2. Install on your computer
3. On Linux/MacOS
 - o [Optional] Move folder to ~/Applications
 - o Run "bin/wxparaver.bin" on terminal
4. On Windows
 - o Run bin/wxparaver.bin.exe
5. Download nsys2prv configuration files stored inside the workshop directory:
 - o sys2prv/configs



A screenshot of the BSC Tools Downloads page. The 'Paraver' section is highlighted. It shows download links for 'EXTRAE', 'PARAVER', and 'DIMEMAS' across various operating systems (Linux x86-64, Mac OS X, Windows, and Other systems). Below this, sections for 'CLUSTERING', 'TRACKING', and 'FOLDING' are shown, each with their own download links and system requirements. A 'BASIC ANALYSIS' section is also present.

Exercises

- Repository under path "/leonardo_work/tra26_minwincs/training-profiling-workshop"
- Copy to your home folder: `cp -R /leonardo_work/tra26_minwincs/training-profiling-workshop ~/`
- Check contents of the copied folder `cd ~/training-profiling-workshop/exercises && ls -la`

```
[apaliour@login07 ~]$ ls -la training-profiling-workshop/exercises/
total 52
drwxrwsrwx. 9 apaliour interactive 4096 feb 24 14:40 .
drwxrwsrwx. 8 apaliour interactive 4096 feb 22 19:49 ..
drwxrwsrwx. 23 apaliour interactive 4096 feb 25 01:36 .cache
drwxrwsrwx. 4 apaliour interactive 4096 feb 10 15:09 datasets
drwxrwsrwx. 2 apaliour interactive 4096 feb 24 09:59 exercise_0_Communication_Tests
drwxrwsrwx. 6 apaliour interactive 4096 feb 24 14:02 exercise_1_DDP
drwxrwsrwx. 7 apaliour interactive 4096 feb 25 01:41 exercise_2_DeepSpeed
drwxrwsrwx. 7 apaliour interactive 4096 feb 24 16:20 exercise_3_MegatronLM
-rw-rw-rw-. 1 apaliour interactive 0 feb 10 15:19 __init__.py
-rw-rw-rw-. 1 apaliour interactive 13560 feb 25 00:40 slurm.sh
drwxrwsrwx. 4 apaliour interactive 4096 feb 16 16:05 utils
```

- `bash slurm.sh --help`

```
Usage: slurm.sh [OPTIONS] [-- EXTRA_ARGS]

Options:
  -n, --nodes          Number of nodes (default: 1)
  -g, --gpus-per-node  GPUs per node (default: 4)
  -q, --queue           Queue/QOS for SLURM job (default: acc_bench)
  -a, --account         Account name (default: bsc99)
  -p, --partition       Partition on HPC (default: acc)
  -e, --exercise        Exercise number (0, 1, 2, or 3) (required)
  -h, --help             Show this help message

Extra arguments after '--' will be passed to the training script.

  --slow-dataloading      Enable slow dataloading example for exercise 1 (DDP) (default: disabled)
  --mixed-precision        Enable mixed precision training using 'bf16' (default: disabled)
  --micro-batch-size N    Set micro batch size to N (default: 4)
  --gradient-accumulation-steps N
                         Set gradient accumulation steps to N (default: 1)
  --activation-checkpointing
                         Enable activation/gradients checkpointing (default: disabled)
  --ds-hpz-partition N   Set DeepSpeed HPZ partition size to N, for exercise 2 (DeepSpeed) only. Refers to number of GPUs per model replica (default
                        : 2)
  --ds-stage2              Use DeepSpeed stage 2 partitioning instead of stage 3, for exercise 2 (DeepSpeed) only (default: stage 3)
  --tp N                  Set tensor parallelism to N, for exercise 3 (MegatronLM) only (default: 1, i.e. no tensor parallelism)
  --pp N                  Set pipeline parallelism to N, for exercise 3 (MegatronLM) only (default: 1, i.e. no pipeline parallelism)
  --no-profile            Disable profiling with NSYS (default: profiling enabled)
  --nsys2prv              After profiling, automatically translate NSYS output to Paraver traces using nsys2prv (default: disabled)

Example:
  slurm.sh -n 1 -g 4 -e 1 -a tra26_minwincs -p boost_usr_prod -- --mixed-precision --gradient-accumulation-steps 4
```



Exercises

- *Exercise 0* - Intra-node topology discovery and communication validation
 - "exercise_0_Communication_Tests/"
 - Run `bash slurm.sh -e 0`
- *Exercise 1* – Finetune **Llama-3.1-1B** with Accelerate & DDP
 - "exercise_1_DDP/"
 - Run `bash slurm.sh -e 1`
 - slow-dataloading, mixed-precision, gradient-accumulation-steps, micro-batch-size
- *Exercise 2* – Finetune **Mistral-7B-v0.1** with Accelerate & DeepSpeed
 - "exercise_2_DeepSpeed/"
 - Run `bash slurm.sh -e 2`
 - mixed-precision, gradient-accumulation-steps, micro-batch-size, activation-checkpointing, ds-hpz-partition, ds-stage2, ds-heavy-comm, ds-no-overlap
- *Exercise 3* – Pretrain **Mistral-7B-v0.1** with MegatronLM
 - "exercise_3_MegatronLM/"
 - Run `bash slurm.sh -e 3`
 - tp, pp, micro-batch-size, global-batch-size



Exercises

- After running each exercise (1,2,3) for the 1st time, inside its folder will be created the following folder:
 - *logs/* - containing slurm logs per JOB ID
 - *profiler/* - containing gpu monitoring plots and nsys profiles, also paraver traces is "--nsys2prv" enabled
 - *gpus-monitor/* - logs and plots of gpus' memory, utilization %, watt per node
 - *nsys/* - code profiling traces of type ".nsys-rep" and ".prv"



Exercise 0 – Intranode comm tests

```
bash slurm.sh -e 0
```

```
===== Test 1: PCIe P2P Read =====
Description: Test direct GPU-to-GPU PCIe read connectivity (P2P) using 'nvidia-smi topo -p2p r'.
Checking intra-node P2P read PCIe communication with 'nvidia-smi topo -p2p r'...
GPU0 GPU1 GPU2 GPU3
GPU0 X OK OK OK
GPU1 OK X OK OK
GPU2 OK OK X OK
GPU3 OK OK OK X

Legend:
X = Self
OK = Status Ok
CNS = Chipset not supported
GNS = GPU not supported
TNS = Topology not supported
NS = Not supported
U = Unknown
```

```
===== Test 2: PCIe P2P Write =====
Description: Test direct GPU-to-GPU PCIe write connectivity (P2P) using 'nvidia-smi topo -p2p w'.
Checking intra-node P2P write PCIe communication with 'nvidia-smi topo -p2p w'...
GPU0 GPU1 GPU2 GPU3
GPU0 X OK OK OK
GPU1 OK X OK OK
GPU2 OK OK X OK
GPU3 OK OK OK X

Legend:
X = Self
OK = Status Ok
CNS = Chipset not supported
GNS = GPU not supported
TNS = Topology not supported
NS = Not supported
U = Unknown
```

```
===== Test 3: NVLink P2P =====
Description: Test direct GPU-to-GPU NVLink connectivity using 'nvidia-smi topo -p2p n'.
Checking intra-node P2P NVLink communication with 'nvidia-smi topo -p2p n'...
GPU0 GPU1 GPU2 GPU3
GPU0 X OK OK OK
GPU1 OK X OK OK
GPU2 OK OK X OK
GPU3 OK OK OK X

Legend:
X = Self
OK = Status Ok
CNS = Chipset not supported
GNS = GPU not supported
TNS = Topology not supported
NS = Not supported
U = Unknown
```

```
===== Test 4: GPU Topology =====
Description: Show the full GPU, NIC, and CPU topology using 'nvidia-smi topo -m'. Useful for understanding all device interconnections.
Checking intra-node GPUs topology with 'nvidia-smi topo -m'...
ID GPU0 GPU1 GPU2 GPU3 NIC0 NIC1 NIC2 NIC3 CPU Affinity NUMA Affinity GPU NUMA
GPU0 X NV4 NV4 NV4 PXB NODE NODE NODE 0-15 0 N/A
GPU1 NV4 X NV4 NV4 NODE PXB NODE NODE 0-15 0 N/A
GPU2 NV4 NV4 X NV4 NODE NODE PXB NODE 0-15 0 N/A
GPU3 NV4 NV4 X NODE NODE NODE PXB 0-15 0 N/A
NIC0 PXB NODE NODE NODE X NODE NODE NODE
NIC1 NODE PXB NODE NODE NODE X NODE NODE
NIC2 NODE NODE PXB NODE NODE NODE X NODE
NIC3 NODE NODE NODE PXB NODE NODE NODE X
```

Legend:

- X = Self
- SYS = Connection traversing PCIe as well as the SMP interconnect between NUMA nodes (e.g., QPI/UPI)
- NODE = Connection traversing PCIe as well as the interconnect between PCIe Host Bridges within a NUMA node
- PHB = Connection traversing PCIe as well as a PCIe Host Bridge (typically the CPU)
- PXB = Connection traversing multiple PCIe bridges (without traversing the PCIe Host Bridge)
- PIX = Connection traversing at most a single PCIe bridge
- NVW = Connection traversing a bonded set of # NVLinks

NIC Legend:

- NIC0: mlx5_0
- NIC1: mlx5_1
- NIC2: mlx5_2
- NIC3: mlx5_3

```
===== Test 5: InfiniBand State =====
Description: Show the InfiniBand state using 'ibstat'. Useful for understanding the network interconnections.
Checking InfiniBand state with 'ibstat'...
CA 'mlx5_0'
  CA type: MT4123
  Number of ports: 1
  Firmware version: 20.39.2048
  Hardware version: 0
  Node GUID: 0x0000380001c2f7b8
  System Image GUID: 0x0000380001c2f7b8
  Port 1:
    State: Active
    Physical state: LinkUp
    Rate: 100
    Base lid: 10209
    LMC: 0
    SM lid: 5
    Capability mask: 0xa651e848
    Port GUID: 0x0000380001c2f7b8
    Link layer: InfiniBand
CA 'mlx5_1'
  CA type: MT4123
  Number of ports: 1
  Firmware version: 20.39.2048
  Hardware version: 0
  Node GUID: 0x0000380001c3356a
  System Image GUID: 0x0000380001c3356a
  Port 1:
    State: Active
    Physical state: LinkUp
    Rate: 100
    Base lid: 10209
    LMC: 0
    SM lid: 5
    Capability mask: 0xa651e848
    Port GUID: 0x0000380001c3356a
    Link layer: InfiniBand
CA 'mlx5_2'
  CA type: MT4123
  Number of ports: 1
  Firmware version: 20.39.2048
  Hardware version: 0
  Node GUID: 0x0000380001c3356a
  System image GUID: 0x0000380001c33568
  Port 1:
    State: Active
    Physical state: LinkUp
    Rate: 100
    Base lid: 6302
    LMC: 0
    SM lid: 5
    Capability mask: 0xa651e848
    Port GUID: 0x0000380001c3356a
    Link layer: InfiniBand
CA 'mlx5_3'
  CA type: MT4123
  Number of ports: 1
  Firmware version: 20.39.2048
  Hardware version: 0
  Node GUID: 0x0000380001c2f7ba
  System image GUID: 0x0000380001c2f7ba
  Port 1:
    State: Active
    Physical state: LinkUp
    Rate: 100
    Base lid: 6107
    LMC: 0
    SM lid: 5
    Capability mask: 0xa651e848
    Port GUID: 0x0000380001c2f7ba
    Link layer: InfiniBand
Communication tests completed!
```

Exercise 1a – DDP

1a. `bash slurm.sh -e 1 --slow-dataloading` vs `bash slurm.sh -e 1`

Description:

The "--slow-dataloading" example simulates a bad case of on-the-fly batch embedding process, without spawning data loading workers* nor using the "pinned memory"** for faster CPU-GPU copy.

Goal:

Recognize GPU idle times during data loading and compare with version that pre-computes batch embeddings, spawns parallel data loading workers and user "pinned memory" when copying CPU-GPU.

* "data loading workers": CPU threaded process to handle asynchronous batch preprocessing and GPU loading.

** "pin memory": Host RAM area which the OS prohibits from being swapped (reindexed or moved to disk), also "paged memory".



Exercise 1a – DDP [Optional]

Task:

Designate the part of code to profile and create our own profiling tagged sections.

Description:

When profiling a python code with nsys using the CUDA API, we need to insert some simple code lines designating the part of the code we want to profile, as well as, tag with custom names sub-section of interest.

Using the torch.cuda.nvtx library we can make that happen:

- `torch.cuda.cudart().cudaProfilerStart()` : Insert line where profiling starts collecting information
- `torch.cuda.cudart().cudaProfilerStop()` : Insert line where profiling stops
- `nvtx.range_push("Section name")` : the beginning of a custom tagged code section named "Section name"
- `nvtx.range_pop()` : the end of a custom tagged code section, must come after a `nvtx.range_push()`.

Tip – Can be avoided if `nvtx.range_push("Section name")` is used in a python "with" statement,

e.g. `with nvtx.range_push("Section name"):
 <tagged section code> ...`

ToDo:

Insert the beginning and end of the code to be profiled and add your own custom NVTX tags inside the epoch training loop of file "exercises/exercise_1_DDP/train_todo.py", inside function `train_epoch()`.

!! Make sure to rename `train.py->train_original.py` and `train_todo.py -> train.py` to run your own code. !!

Exercise 1b – DDP

1b. `bash slurm.sh -e 1` vs `bash slurm.sh -e 1 -- --mixed-precision`

Description:

Running all matrix multiplications during training under FP32 precision provides us with great accuracy and stability. However, as models scale up in parameters, FP32 computations become almost impossible to carry out in efficient time, as well as store in GPU VRAM. Mixed precision*, solves this issue by downscaling the precision of crucial computations down to FP16/BF16, bringing computation time and memory requirements down to almost half.

Goal:

Locate the use of FP32 and BF16 precision matrix multiplication kernels (GEMMs).

* **Mixed Precision:** The training technique of keeping models weight and gradients in FP16/BF16 for speed and memory optimization and the optimizer states in FP32 for training stability.



Exercise 1c – DDP

1c. `bash slurm.sh -e 1 -- --mixed-precision --gradient-accumulation-steps 16`

Description:

Gradient accumulation is part of set of training techniques that allows us to aggregate the gradients of multiple forward-backward iterations, before upgrading the model's weights. Along those lines, it also contributes by decreasing training time, since we do not need to communicate gradients or compute new weights as frequently. Thus, each aggregated batch becomes a micro-batch to the bigger effective global batch size.

Goal:

Recognize the essential difference in communications and weights' update done.



Exercise 2a – DeepSpeed

2a. `bash slurm.sh -e 2 -- --mixed-precision --gradient-accumulation-steps 16`

vs

`bash slurm.sh -e 2 -- --mixed-precision --gradient-accumulation-steps 16 --ds-stage2`

Description:

The major difference between DeepSpeed ZeRO2 and ZeRO3 is the partitioning of model weights. In ZeRO2 each device retains its own whole copy of the model weight, in this case in BF16 precision, while in ZeRO3 these weights are also sharded between ranks. This sharding, introduces a communication overhead during forward/backward passes, compared to ZeRO2.

Goal:

Compare traces of these two cases and estimate the overhead introduced when training in ZeRO3. Is the overhead worth the gains ?



Exercise 2b – DeepSpeed

2b. `bash slurm.sh -e 2 -n 2 -- --mixed-precision --gradient-accumulation-steps 16 --micro-batch-size 8 --ds-no-overlap`

vs `bash slurm.sh -e 2 -n 2 -- --mixed-precision --gradient-accumulation-steps 16 --micro-batch-size 8`

Description:

DeepSpeed ZeRO3 partitions all trainable parameters, model weights, gradients and optimizer states. However, that dictates the frequent communication between GPUs during forward and backward passes, especially when running in multi-node setups. In this example we will double the resources and the workload (2 nodes, 8 GPUs & $8 \times 16 \times 8 = 256$ global batch size). However, the in one trace communication is not being overlapped with computation. In both case ds-hpz-partition is 4. So, we employ also DDP, because in 8 GPUs we can fit 2 model copies.

Goal:

Compare traces of between ZeRO3 fine-tuning that does not overlap any communication with computations to the case where non-blocking communications are overlapped with computation wherever possible.

Tips: Looking for differences in communication and computation times. Nsight Stats view and Paraver communications overlap configuration can be of help.

*** hpx-partition-size:** Hierarchical Partition Size is a DeepSpeed parameter that denotes the number of GPUs among which one model copy is sharded

Exercise 3 – MegatronLM

3a. `bash slurm.sh -e 3 -- --pp 1 --tp 4 --nsvsys2prv`

vs `bash slurm.sh -e 3 -n 2 -- --pp 2 --tp 4 --nsvsys2prv`

Description:

MegatronLM, as mentioned earlier, follows a per layer sharding strategy which optimizes the communication requirements between GPUs using Pipeline and Tensor Parallelism. When PP=1, we shard all model layers' tensors between the GPUs inside only 1 node. When using PP=2, the first half of the layers' are assigned to one node and the rest are assigned to a second node. Setting TP=4, means that tensors per PP stage are sharded between 4 GPUs inside one node.

However, PP>1 introduces an inherent dependency between the different pipeline stages, according to which later stages of the model need the output of previous stages to perform a forward pass and equivalently need the gradients of the next stages to perform the backward pass. This dependency causes GPUs idle times, which are referred to as the "Pipeline Bubble".

Goal:

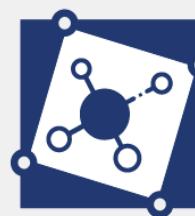
Compare Paraver trace of PP=1 and PP=2 and recognize the phenomenon of "Pipeline Bubble".

Tip: Run a communication-computation overlap configuration and locate activity delays between different groups of GPUs

Thank you



Co-funded by
the European Union



AI4S
BSC AI 4 Science Fellowships



This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 101182737. The JU receives support from the Digital Europe Programme.