



**AI4S**  
BSC AI 4 Science Fellowships

# Profiling and Optimizing AI Trainings

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

MINERVA AI Winter School, 26.02.2026

# 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



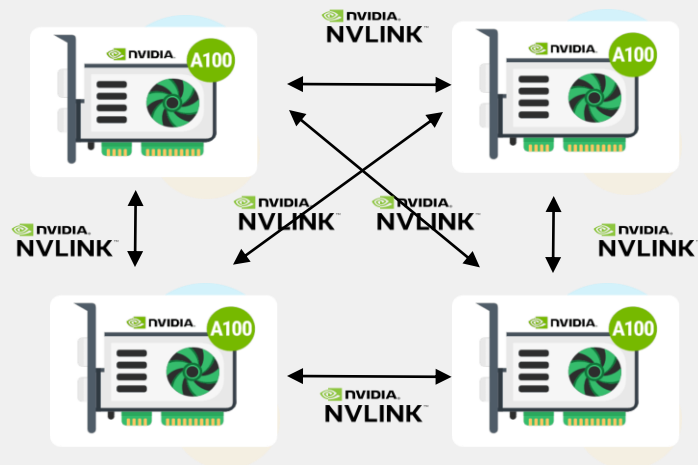
# Questions that need answers...

- 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

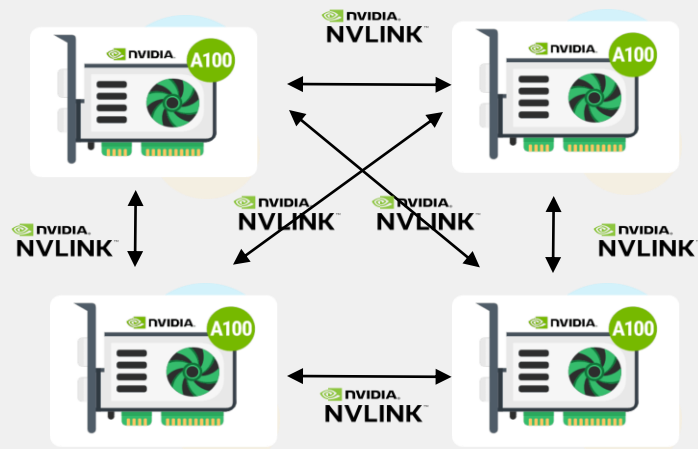
- NVLink cables  
(Leonardo 64GiB HBM2e NVLink 3.0 (200 GB/s))
- NVSwitch (>4 GPUs node) + NVLink
- PCIe (cpu-gpu, slow)



# How do GPUs communicate?

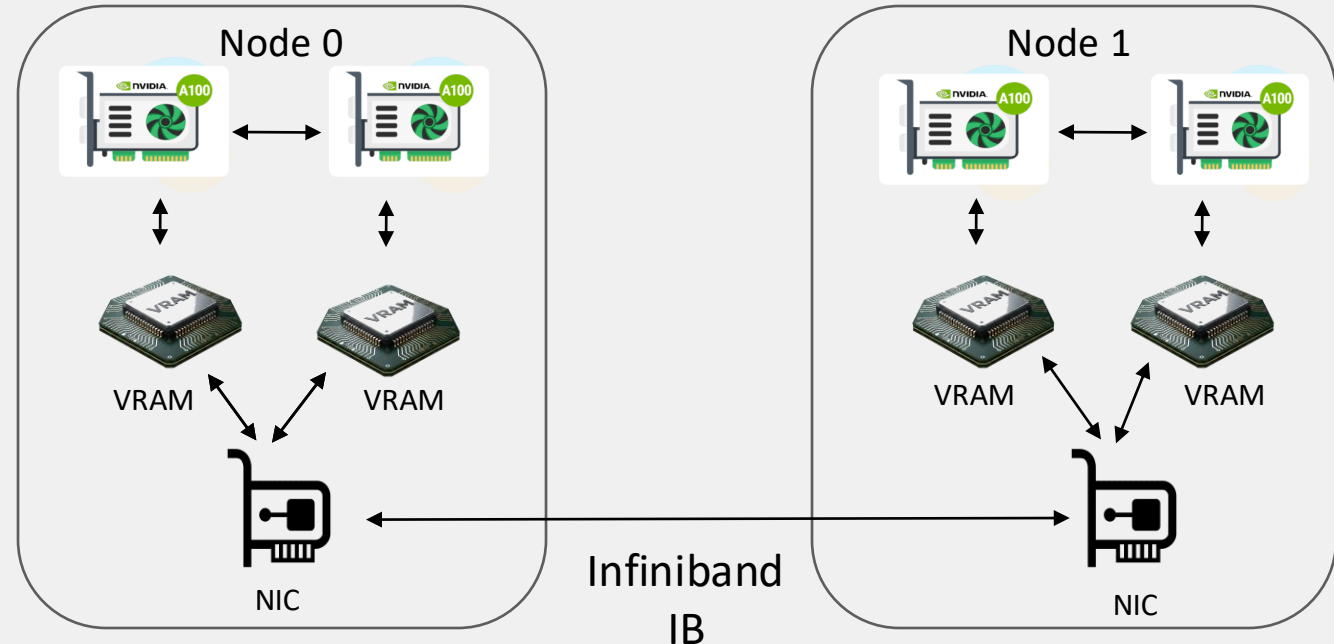
## Intra-Node

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



## Inter-Node

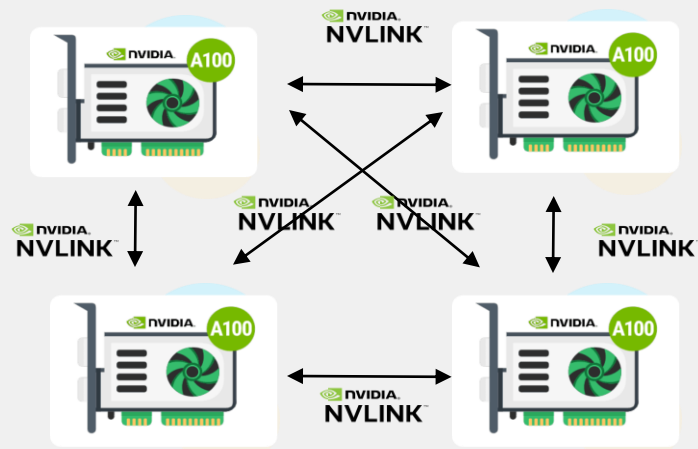
- InfiniBand GPU inter-node Remote Direct Memory Access  
(Leonardo 200 Gbps~25GB/s NVIDIA Mellanox HDR InfiniBand)
- Ethernet (fallback, slow)



# How do GPUs communicate?

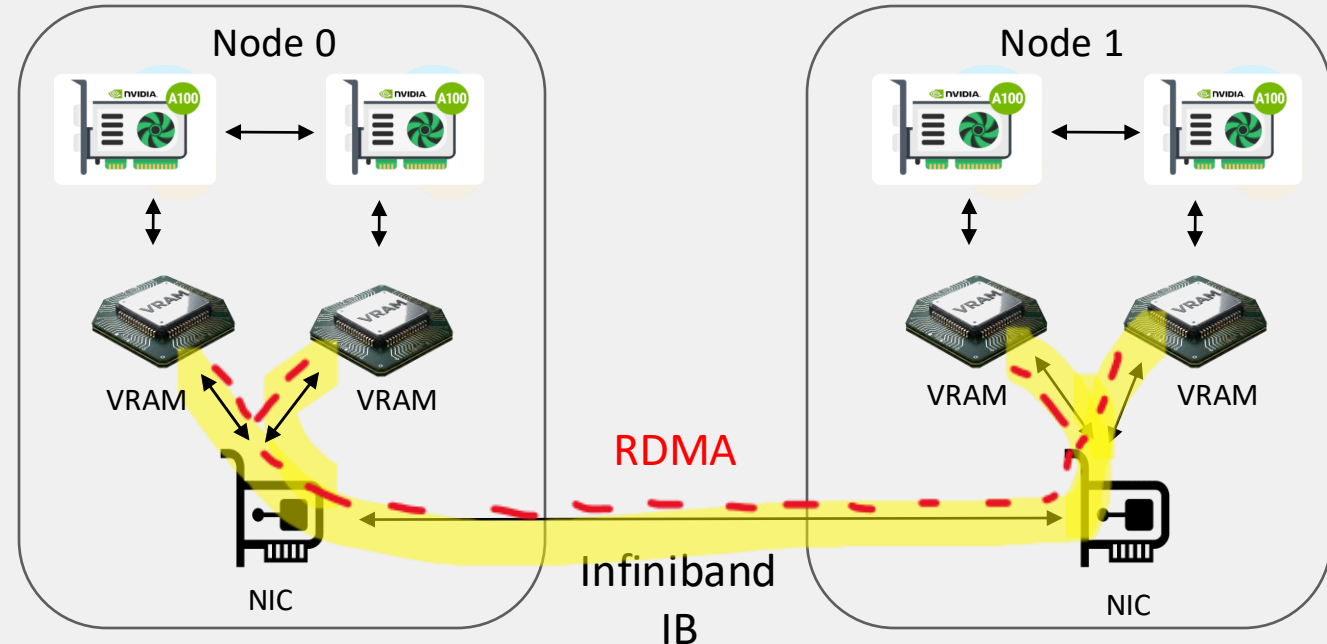
## Intra-Node

- 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)



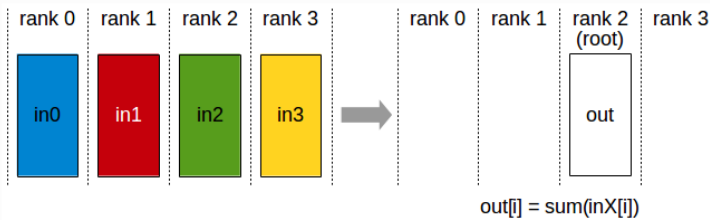
# How do GPUs communicate?

## NCCL: NVIDIA Collective Communication Library

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

### Reduce

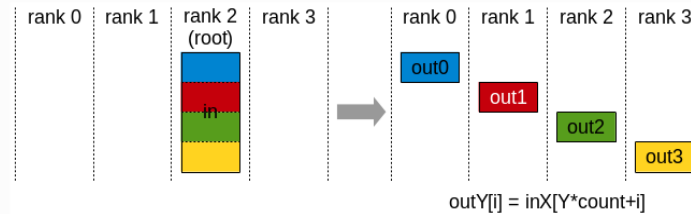
The Reduce operation performs the same operation as AllReduce, but stores the result only in the receive buffer of a specified root rank.



Reduce operation: one rank receives the reduction of input values across ranks.

### Scatter

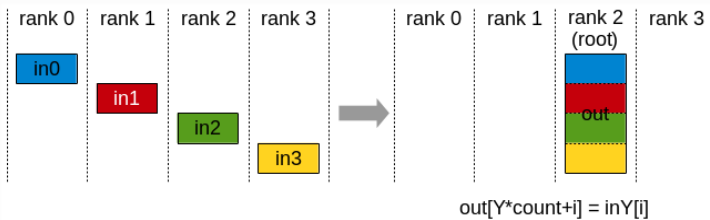
The Scatter operation distributes a total of  $N*k$  values from the root rank to  $k$  ranks, each rank receiving  $N$  values.



Scatter operation: root rank distributes data to all ranks.

### Gather

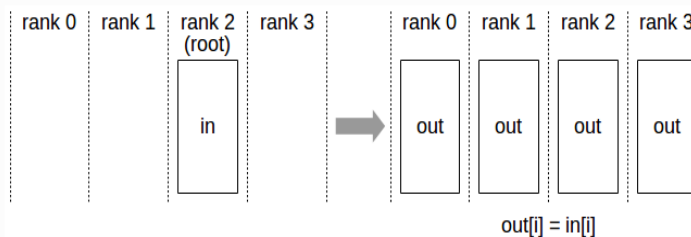
The Gather operation gathers  $N$  values from  $k$  ranks into an output buffer on the root rank of size  $k*N$ .



Gather operation: root rank receives data from all ranks.

### Broadcast

The Broadcast operation copies an  $N$ -element buffer from the root rank to all the ranks.



Broadcast operation: all ranks receive data from a "root" rank.

# How do GPUs communicate?

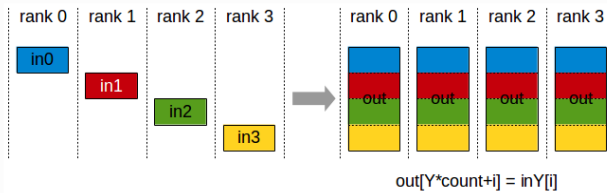
## NCCL: NVIDIA Collective Communication Library

- 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\*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.

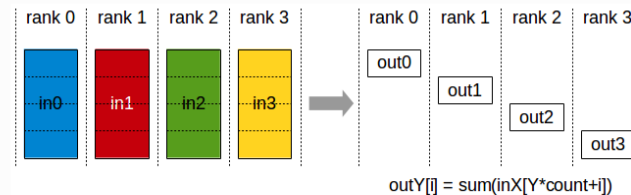


AllGather operation: each rank receives the aggregation of data from all ranks in the order of the ranks.

### 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.

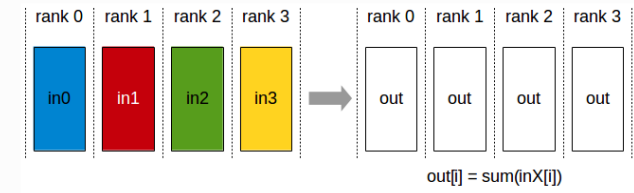


Reduce-Scatter operation: input values are reduced across ranks, with each rank receiving a subpart of the result.

### 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  $out[i] = in0[i] + in1[i] + \dots + in(k-1)[i]$ .

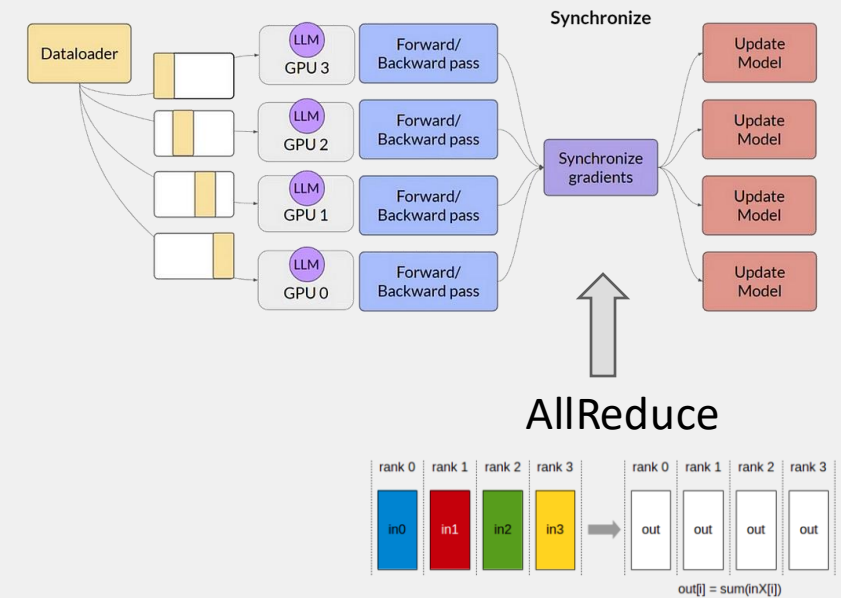


All-Reduce operation: each rank receives the reduction of input values across ranks.

# DDP Communication

Collective	Role
<b>AllReduce</b>	Aggregates gradients across all GPUs after backward pass. This is the default and the most performance-critical operation in DDP.
<b>Broadcast</b>	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)



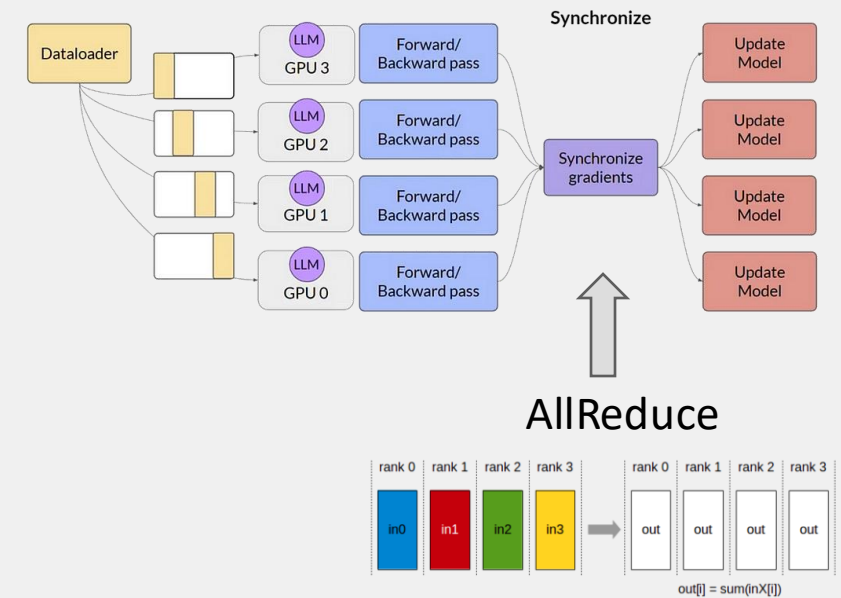
# DDP Communication

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

Trace Pattern:

Forward GEMM → Backward GEMM → AllReduce (bucketed)

## Distributed Data Parallel (DDP)



\*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.

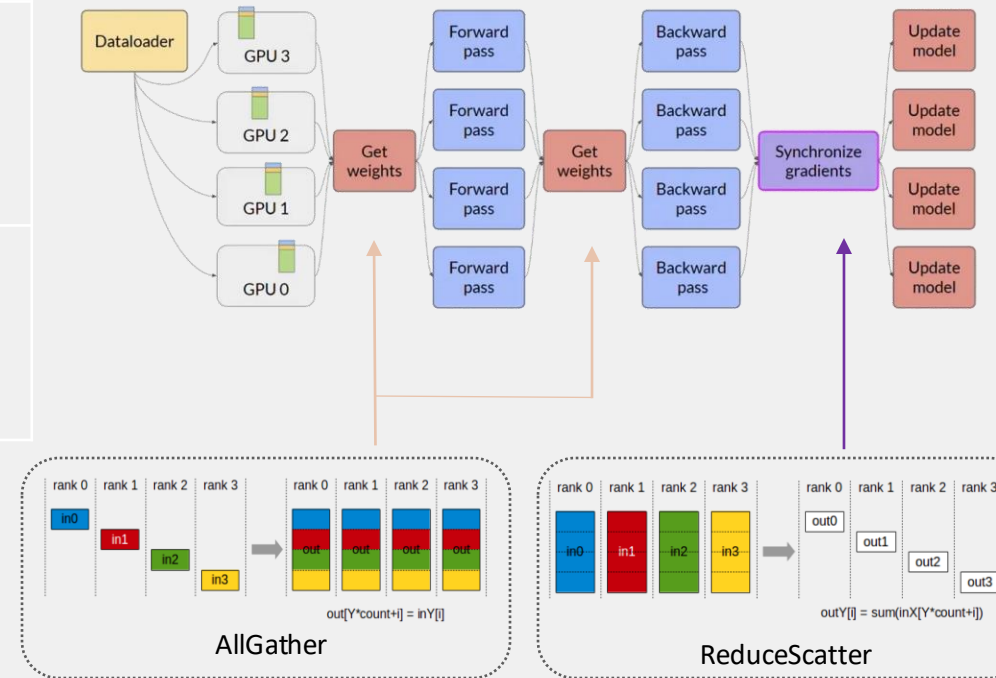
Dynamic states can vary in this example 8-20GB or more

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

# FSDP Communication

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

## Fully Sharded Data Parallel (FSDP)



# FSDP Communication

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

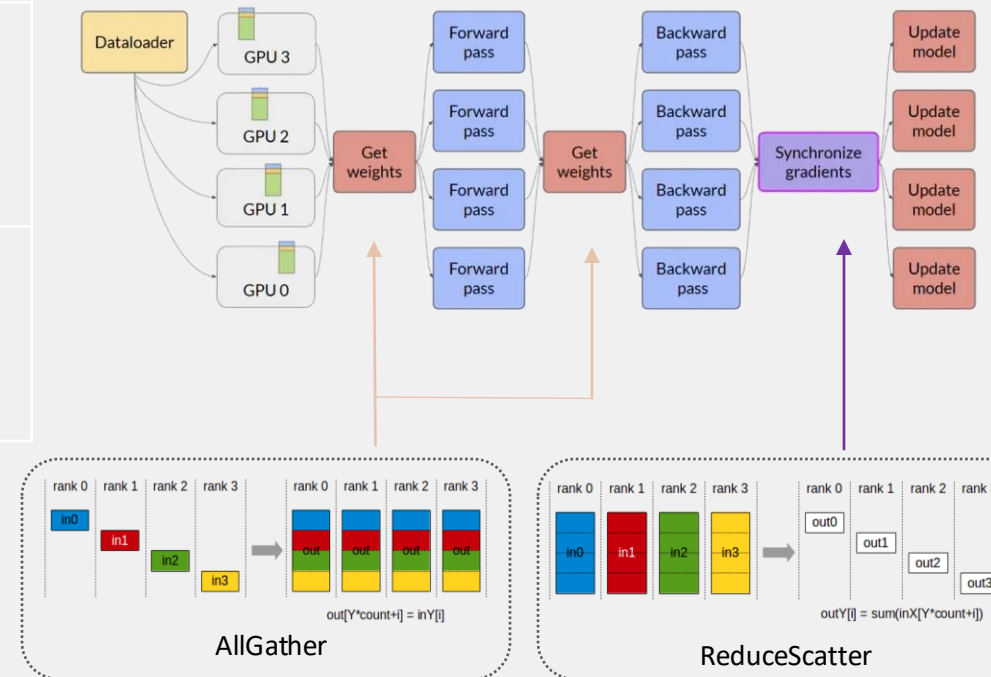
Trace Pattern:

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

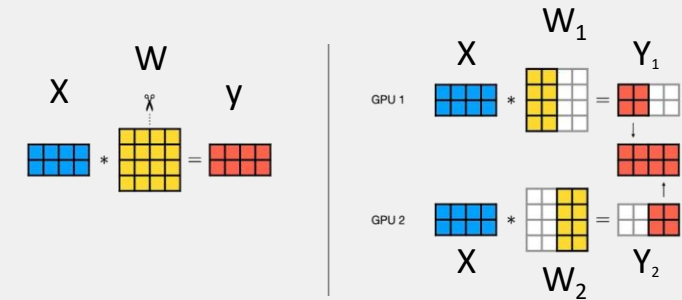
ZeRO3: AllGather → Forward GEMM → AllGather → Backward GEMM → ReduceScatter [Similar to FSDP]

# MegatronLM Communication

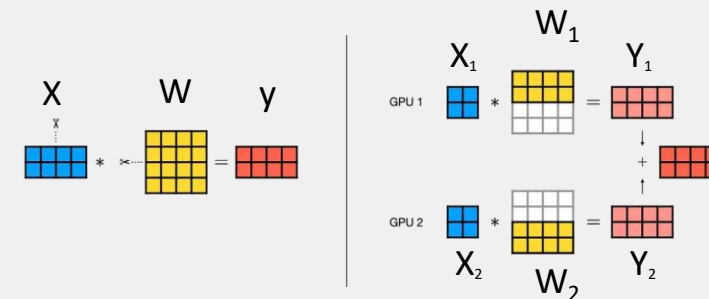
Parallelism Type	Collective Used	Forward	Backward
Data Parallel	AllReduce	-	After backward on weight updates
Pipeline Parallel	Send / Recv (P2P)	Transfer activations (outputs) between pipeline stages	Transfer gradients between pipeline stages
Tensor Parallel	-	ColumnParallelLinear -> RowParallelLinear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

Linear Layer:  $Y = W * X + b$

Column Parallel Linear



Row Parallel Linear



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

# 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 out 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"

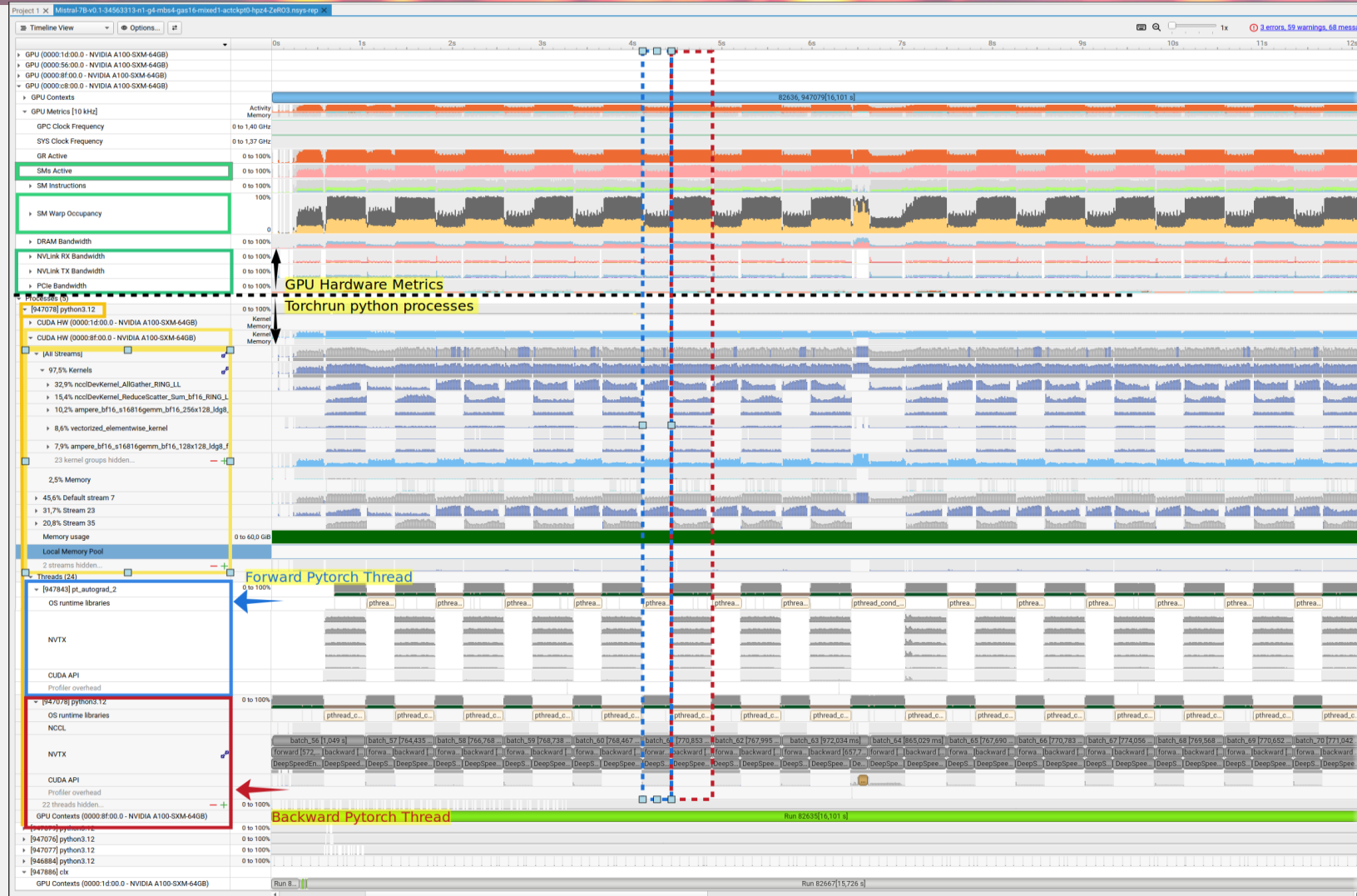
**Stream Processors (SM):** A compute unit inside the GPU. The A100 has 108 SMs.

**GPU Thread:** Smallest execution unit

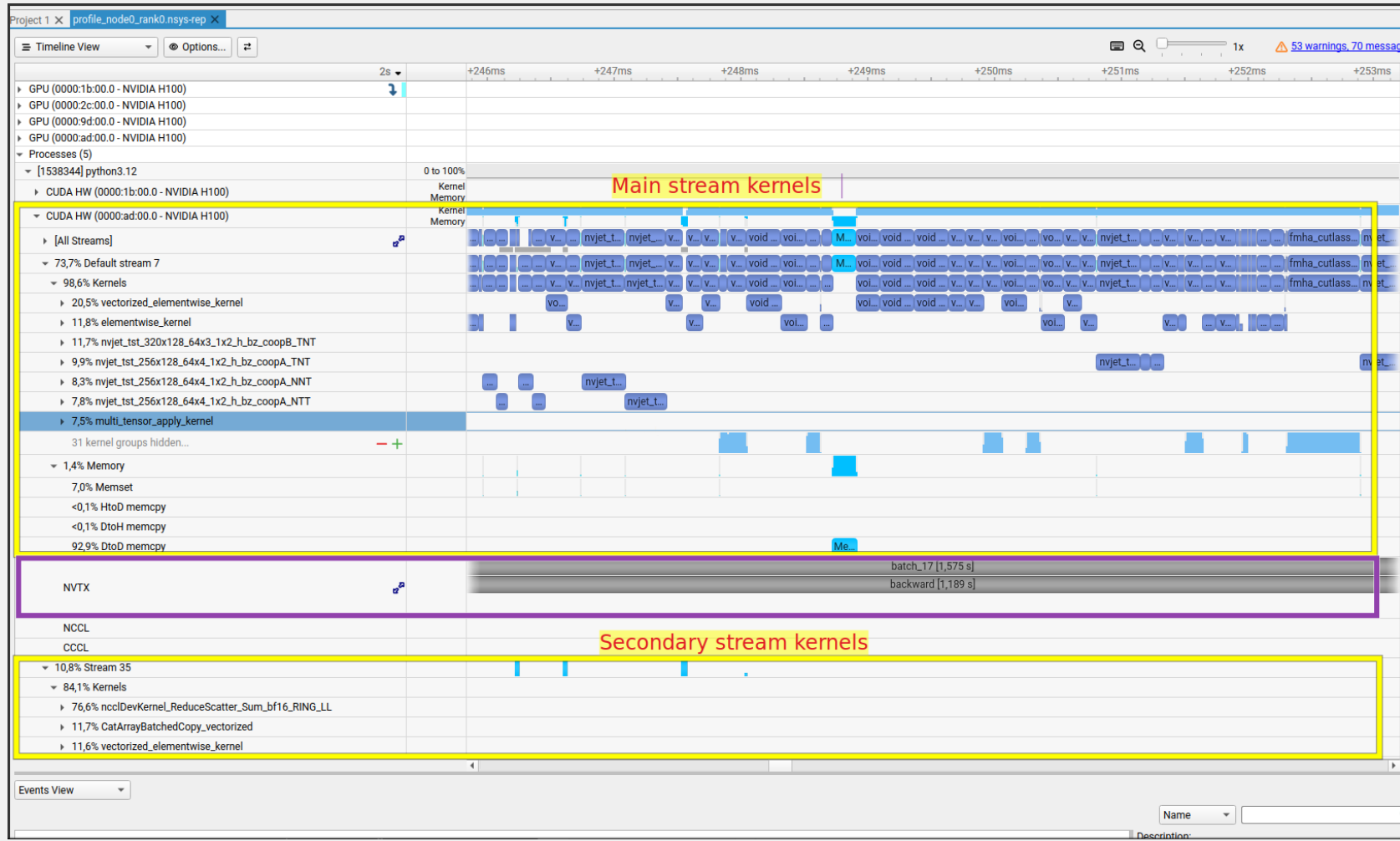
**Warp:** Group of GPU threads that execute together (commonly 32)

- **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

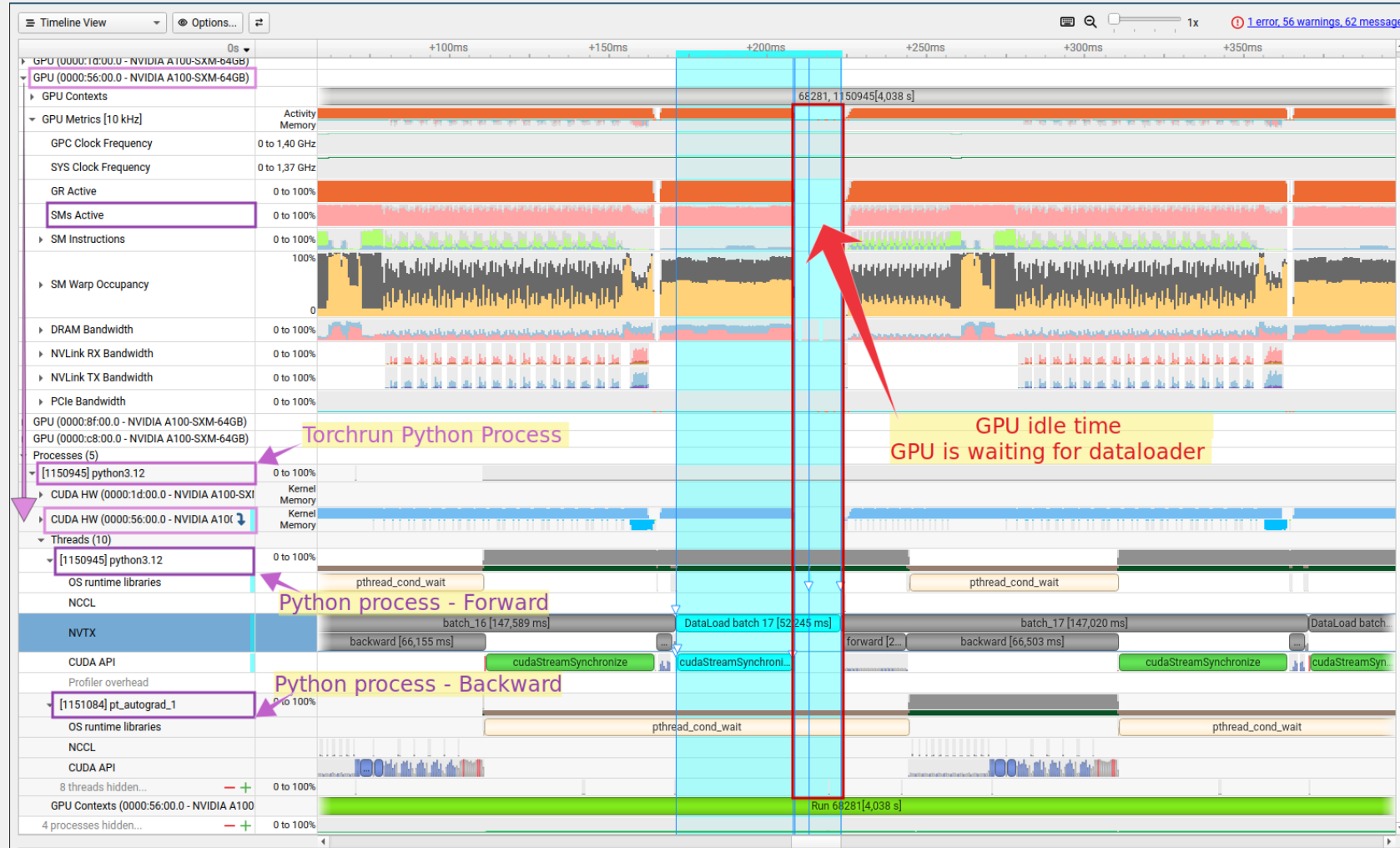
# Nvidia Nsight Systems - Events View



# Nvidia Nsight Systems - Events View



# Nvidia Nsight Systems – Idle time



# Nvidia Nsight Systems – Stats View

# CUDA API Summary

[illegible]

## NVTX Push/Pop Range Summary

Time	Total Time	Instances	Avg	Med	Min	Max	StdDev	Range
13.0%	37,532 s	80	469,152 ms	458,900 ms	449,230 ms	663,615 ms	44,154 ms	backward
13.0%	36,728 s	80	459,228 ms	458,664 ms	449,196 ms	482,026 ms	8,899 ms	DeepSpeedEngine.backward
9.6%	26,201 s	80	327,668 ms	318,668 ms	303,196 ms	572,411 ms	56,867 ms	forward
9.0%	23,045 s	80	234,569 ms	207,520 ms	202,271 ms	569,380 ms	58,177 ms	DeepSpeedEngine.forward
4.0%	13,551 s	6,740	200,940 ms	123,429 ms	39,896 ms	649,429 ms	56,194 ms	PartitionParameterCoordinator.fetch_sub_module
3.0%	9,000 s	22,280	386,611 ms	143,163 ms	124,181 ms	6,492 ms	403,795 ms	DeepSpeedOptimizer_Stage3_create_reduce_and_remove_grad_hooks<locals> wrapper<locals> reduce_partition_and_remove_grads
2.0%	6,761 s	33,680	200,738 ms	135,097 ms	51,518 ms	1,430 ms	172,159 ms	DeepSpeedZelloOffload_register_deepspeed_module<locals> _run_before_backward_function
2.0%	5,794 s	7920	731,546 ms	742,024 ms	330,995 ms	6,368 ms	250,202 ms	DeepSpeedOptimizer_Stage3__reduce_and_partition_spg_grads
1.0%	4,965 s	6,640	76,808 ms	100,967 ms	15,563 ms	2,419 ms	97,163 ms	PartitionParameterCoordinator.release_sub_module
1.0%	4,530 s	18,256	249,292 ms	201,948 ms	168,182 ms	1,822 ms	1,612 ms	PartitionParameterCoordinator_all_gather_params
1.0%	4,197 s	4	1,049 s	1,049 s	1,049 s	1,049 s	131,009 ms	batch_56
1.0%	3,932 s	18,256	215,371 ms	169,891 ms	147,229 ms	69,115 ms	1,021 ms	int_convert_to_deepspeed_param<locals> all_gather_coalesced
1.0%	3,888 s	4	972,114 ms	972,122 ms	971,871 ms	972,341 ms	205,121 ms	batch_convert
1.0%	3,460 s	4	864,954 ms	864,973 ms	864,796 ms	865,074 ms	124,211 ms	batch_64
1.0%	3,244 s	34,764	93,316 ms	80,227 ms	600 ms	2,379 ms	111,768 ms	PartitionParameterCoordinator_release_param
1.0%	3,125 s	4	781,242 ms	781,274 ms	781,088 ms	781,330 ms	100,742 ms	batch_71
1.0%	3,121 s	33,760	92,506 ms	45,163 ms	24,679 ms	1,424 ms	110,691 ms	DeepSpeedZelloOffload_register_deepspeed_module<locals> _post_forward_module_hook
1.0%	3,099 s	4	774,700 ms	774,519 ms	774,519 ms	774,870 ms	178,747 ms	batch_75
1.0%	3,096 s	4	773,897 ms	773,941 ms	773,623 ms	774,082 ms	215,668 ms	batch_67
1.0%	3,084 s	4	770,972 ms	770,978 ms	770,880 ms	771,052 ms	87,657 ms	batch_70
1.0%	3,083 s	4	770,676 ms	770,729 ms	770,462 ms	770,783 ms	149,377 ms	batch_66
1.0%	3,082 s	4	770,604 ms	770,614 ms	770,334 ms	770,853 ms	279,124 ms	batch_61
1.0%	3,062 s	4	770,513 ms	770,584 ms	770,141 ms	770,742 ms	264,341 ms	batch_69
1.0%	3,078 s	4	769,414 ms	769,401 ms	769,226 ms	769,566 ms	142,748 ms	batch_68
1.0%	3,077 s	4	769,204 ms	769,296 ms	769,038 ms	769,585 ms	247,710 ms	batch_73
1.0%	3,075 s	4	768,741 ms	768,762 ms	768,640 ms	768,799 ms	71,803 ms	batch_72
1.0%	3,073 s	4	768,366 ms	768,335 ms	768,054 ms	768,738 ms	292,214 ms	batch_59

# CUDA GPU Kernel Summary

[illegible]

# Nvidia Nsight Systems – nsys command

## 'nsys' command

```
[apaliour@login05 ~]$ module purge
[apaliour@login05 ~]$ module load cuda/12.6
[apaliour@login05 ~]$ which nsys
/leonardo/prod/opt/compilers/cuda/12.6/nv/bin/nsys
[apaliour@login05 ~]$ nsys
usage: nsys [--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

```
[apaliour@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,nccl,cublas : Trace GPU kernels, NVTX markers, OS runtime, cuDNN, UCX, NCCL, cuBLAS
# --force-override 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-override=true
--output=/exercise_2_DeepSpeed/profiler/Mistral-7B-v0.1-34563983-n1-g4-mbs4-gas16-mixed1-actckpt0-hp24-2eR03-badcomm1/nsys/profile_node%q(SLURM_NODEID)
Singularity exec
--nv
--bind /leonardo
--bind /leonardo work
/leonardo work/tra26_minwinc/containers/ai-profiling-workshop.sif
accelerate launch
--config file ./exercise_2_DeepSpeed/accelerate_config.yaml-H806t1r6yDfrb
--rdzv 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
--data-loader-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

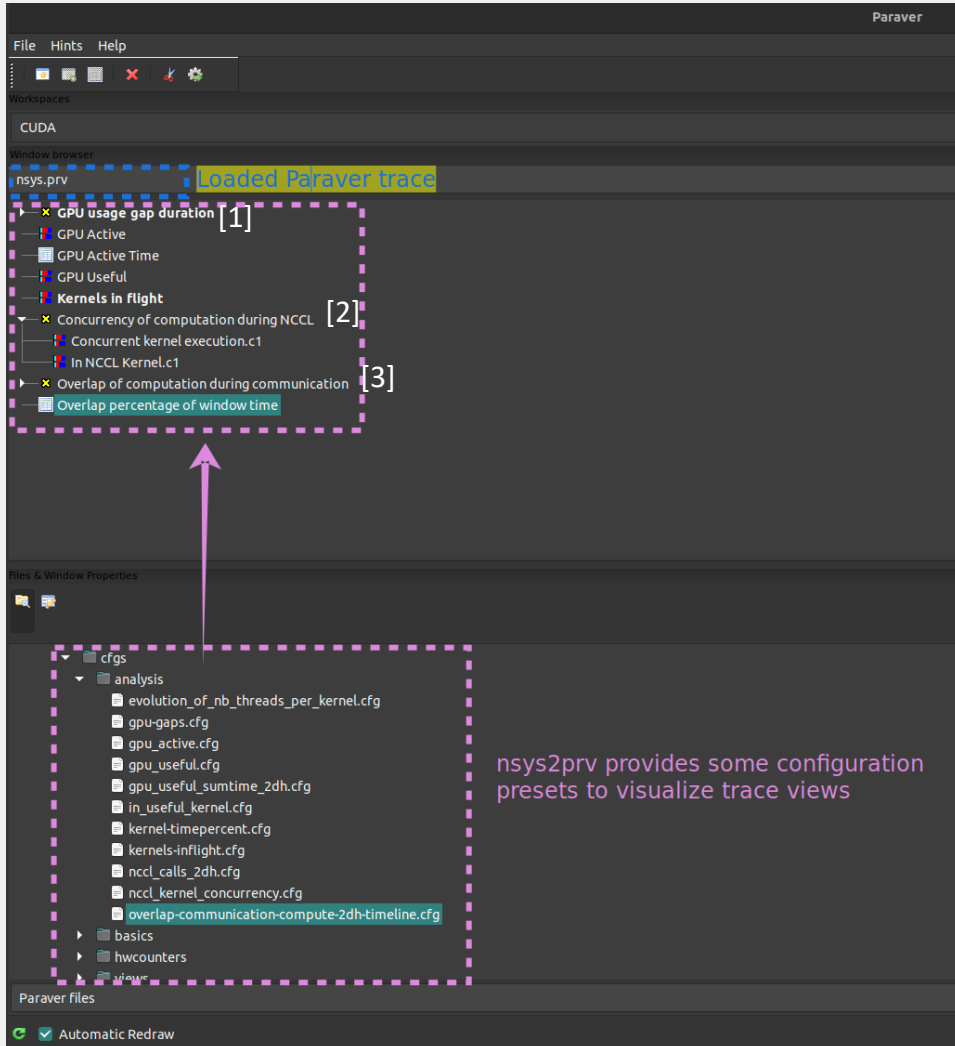
## 3. Paraver

- **Nsys2prv**
- **Paraver client**

## 4. Demo

## 5. Hands-on Exercises

# Paraver



File Hints Help

Workspaces

CUDA

Window browser

nsys.prv **Loaded Paraver trace**

- GPU usage gap duration [1]
- GPU Active
- GPU Active Time
- GPU Useful
- Kernels in flight
- Concurrency of computation during NCCL [2]
  - Concurrent kernel execution.c1
  - In NCCL Kernel.c1
- Overlap of computation during communication [3]
  - Overlap percentage of window time

Files & Window Properties

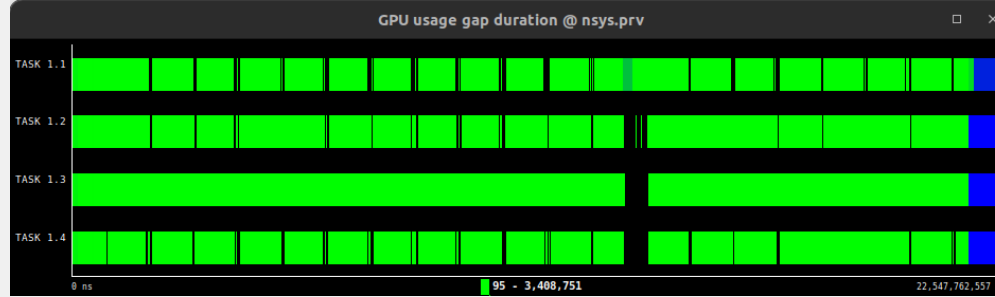
- cfgs
  - analysis
    - evolution\_of\_nb\_threads\_per\_kernel.cfg
    - gpu-gaps.cfg
    - gpu\_active.cfg
    - gpu\_useful.cfg
    - gpu\_useful\_sumtime\_2dh.cfg
    - in\_useful\_kernel.cfg
    - kernel-timepercent.cfg
    - kernels-inflight.cfg
    - nccl\_calls\_2dh.cfg
    - nccl\_kernel\_concurrency.cfg
    - overlap-communication-compute-2dh-timeline.cfg
  - basics
  - hwcounters

Paraver files

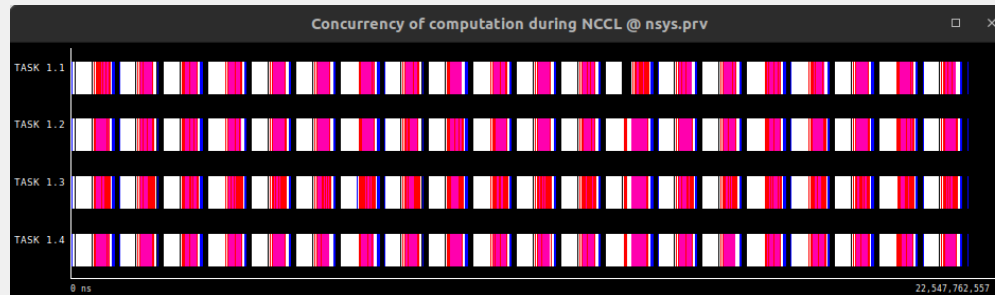
Automatic Redraw

nsys2prv provides some configuration presets to visualize trace views

[1]



[2]

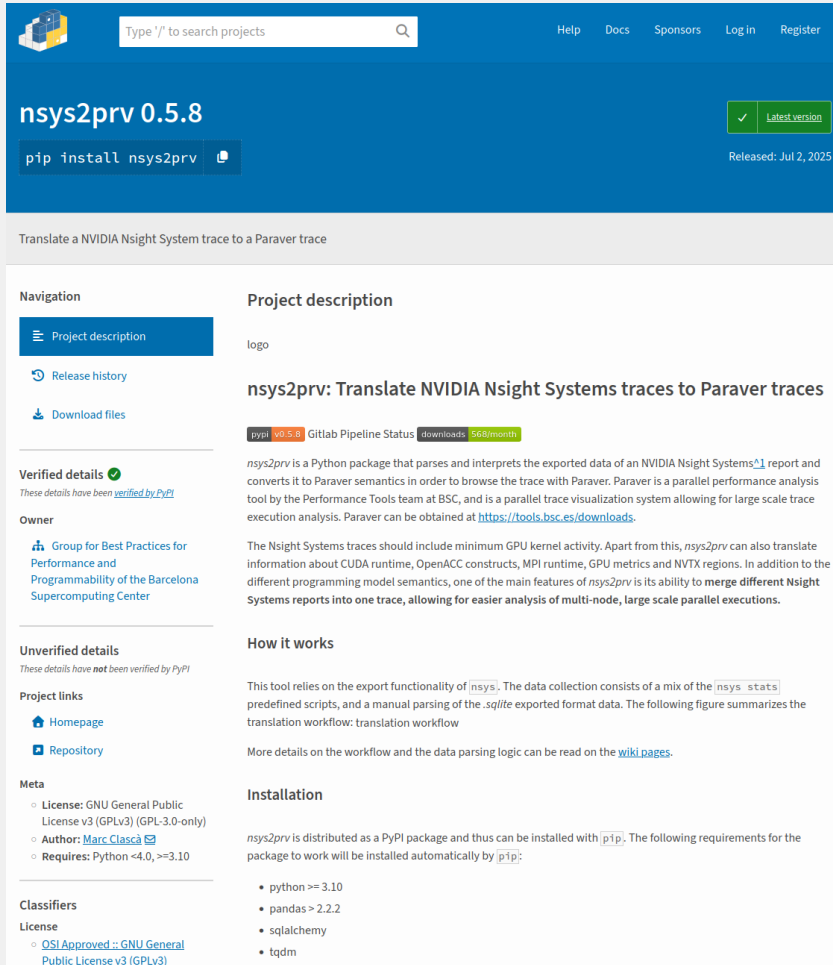


[3]

Overlap percentage of window time @ nsys.prv

	1	2	3	4
TASK 1.1	6.98 %	22.47 %	5.15 %	0.18 %
TASK 1.2	7.54 %	22.73 %	6.54 %	0.22 %
TASK 1.3	6.44 %	17.99 %	5.45 %	0.10 %
TASK 1.4	8.24 %	24.50 %	6.25 %	0.22 %
Num. Cells	4	4	4	4
Total	29.19 %	87.69 %	23.40 %	0.71 %
Average	7.30 %	21.92 %	5.85 %	0.18 %
Maximum	8.24 %	24.50 %	6.54 %	0.22 %
Minimum	6.44 %	17.99 %	5.15 %	0.10 %
StDev	0.67 %	2.40 %	0.57 %	0.05 %
Avg/Max	0.89	0.89	0.89	0.81

# nsys2prv – Trace translation



The screenshot shows the PyPI project page for nsys2prv 0.5.8. The page includes a navigation sidebar with links to Project description, Release history, and Download files. The main content area features a project description, verified details (verified by PyPI), owner information (Group for Best Practices for Performance and Programmability of the Barcelona Supercomputing Center), project links (Homepage, Repository), meta information (License: GNU General Public License v3, Author: Marc Clasca), and classifiers (License: OSI Approved :: GNU General Public License v3 (GPLv3)).

- 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 PARAVR_HOME variable needs to be set.
```

# Agenda

## 1. Introduction

- Systems' Topology
- GPU communication (NCCL)

## 2. NVIDIA Nsight Systems

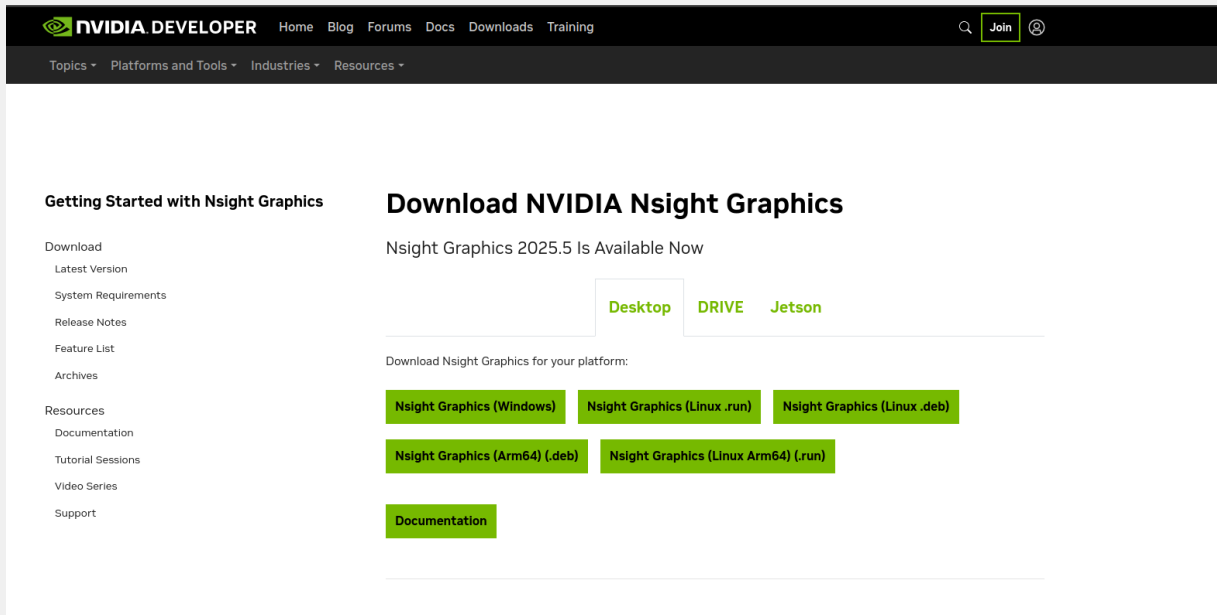
- Nsys command
- Nsight client

## 3. Paraver

- Nsys2prv
- Paraver client

## 4. Exercises

# Download and setup Nsight



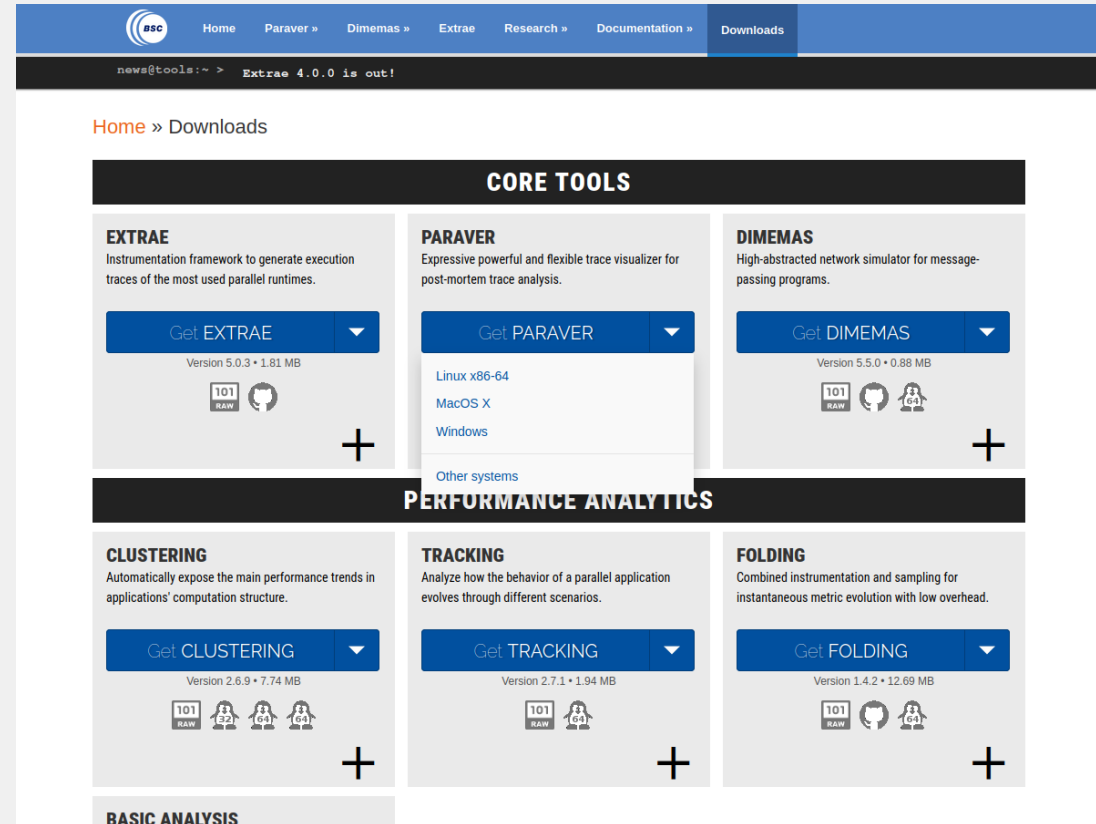
The screenshot shows the NVIDIA Developer website. The top navigation bar includes links for Home, Blog, Forums, Docs, Downloads, and Training. A search bar and a 'Join' button are also present. The main content area is titled 'Getting Started with Nsight Graphics' and 'Download NVIDIA Nsight Graphics'. It features a section for 'Nsight Graphics 2025.5 Is Available Now' with tabs for Desktop, DRIVE, and Jetson. Below this, there are download links for various platforms: Nsight Graphics (Windows), Nsight Graphics (Linux .run), Nsight Graphics (Linux .deb), Nsight Graphics (Arm64) (.deb), and Nsight Graphics (Linux Arm64) (.run). A 'Documentation' link is also visible.

1. Download Nvidia Nsight Systems  
@ <https://developer.nvidia.com/nsight-graphics/get-started>
2. Install on your computer

# Download and setup Paraver

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

```
> exercises
v nsys2prv
  v configs
    > analysis
    > basics
    > hwcounters
    > views
    $ nsys2prv-folder-sbatch.sh
    $ nsys2prv-folder.sh
```



The screenshot shows the BSC Downloads page. The navigation bar includes links for Home, Paraver, Dimemas, Extrae, Research, Documentation, and Downloads. A status bar at the top indicates 'news@tools:~ > Extrae 4.0.0 is out!'. The main content area is titled 'Home » Downloads' and is divided into two main sections: 'CORE TOOLS' and 'PERFORMANCE ANALYTICS'.

**CORE TOOLS**

- EXTRAE**: Instrumentation framework to generate execution traces of the most used parallel runtimes. Version 5.0.3 • 1.81 MB. Download button: Get EXTRAE. Operating system icons: Linux x86-64, MacOS X, Windows, and Other systems.
- PARAVER**: Expressive powerful and flexible trace visualizer for post-mortem trace analysis. Download button: Get PARAVER. Operating system icons: Linux x86-64, MacOS X, Windows, and Other systems.
- DIMEMAS**: High-abstracted network simulator for message-passing programs. Version 5.5.0 • 0.88 MB. Download button: Get DIMEMAS. Operating system icons: Linux x86-64, MacOS X, Windows, and Other systems.

**PERFORMANCE ANALYTICS**

- CLUSTERING**: Automatically expose the main performance trends in applications' computation structure. Version 2.6.9 • 7.74 MB. Download button: Get CLUSTERING. Operating system icons: Linux x86-64, MacOS X, Windows, and Other systems.
- TRACKING**: Analyze how the behavior of a parallel application evolves through different scenarios. Version 2.7.1 • 1.94 MB. Download button: Get TRACKING. Operating system icons: Linux x86-64, MacOS X, Windows, and Other systems.
- FOLDING**: Combined instrumentation and sampling for instantaneous metric evolution with low overhead. Version 1.4.2 • 12.69 MB. Download button: Get FOLDING. Operating system icons: Linux x86-64, MacOS X, Windows, and Other systems.

**BASIC ANALYSIS**

# Thank you



**Co-funded by  
the European Union**



**AI4S**  
BSC AI 4 Science Fellowships



**EuroHPC**  
Joint Undertaking

**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.**