nsys2prv: translate Nsight System traces to Paraver

Marc Clascà

marc.clasca@bsc.es

Best Practices for Performance and Programmability

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

…but why?

Tools overview

Extrae and Paraver

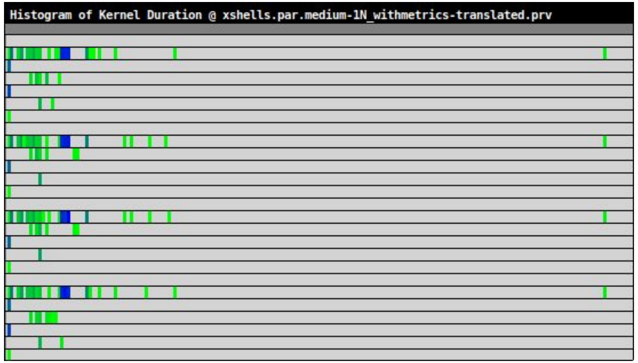
- Parallel performance analysis
- Configurable visualization
- Trace information -> Semantics
- Missing kernel names
- Limited to MPI distributed parallel model - **relevant for LLMs**

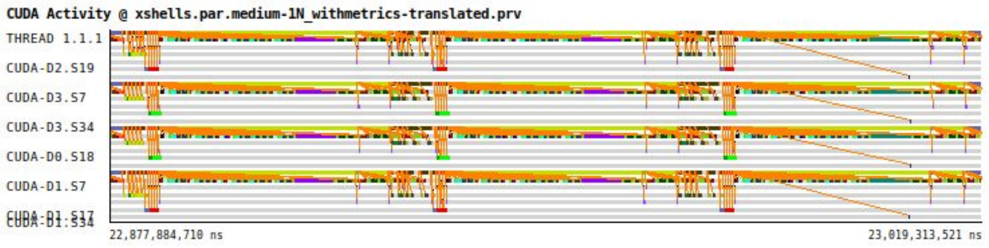
NVIDIA Nsight Systems/Compute

- NVIDIA libraries -> More information (for the moment)
- NOT intended to be a parallel profiler
  - Not good visualization for a lot of processes (imo)
  - Communication semantics is “hidden” in the UI
  - Different report for each process for multi-node
- … but outputs all information in SQLite and CSV format :)

nsys2prv tool: translate nsys traces to Paraver format

- Python script
- Reads data in CSV format from `nsys stats` command and SQLite DB
  - CUDA API calls
  - Kernels and CUDA Graphs, and associated execution information
  - Memory transfers: from/to relationship, size
  - MPI Calls
  - NVTX regions
  - GPU metrics
  - OpenACC
- Custom CFGs, more to come


Histogram of Kernel Duration @ xshells.par.medium-1N_withmetrics-translated.prv



Benefits and limitations

Benefits:

- Profit from all Nsight information + adapting it to Paraver semantics
- Easy data treatment with python
- Supports analyst work: fast tool feature integration -> first script, then to *Extrae*

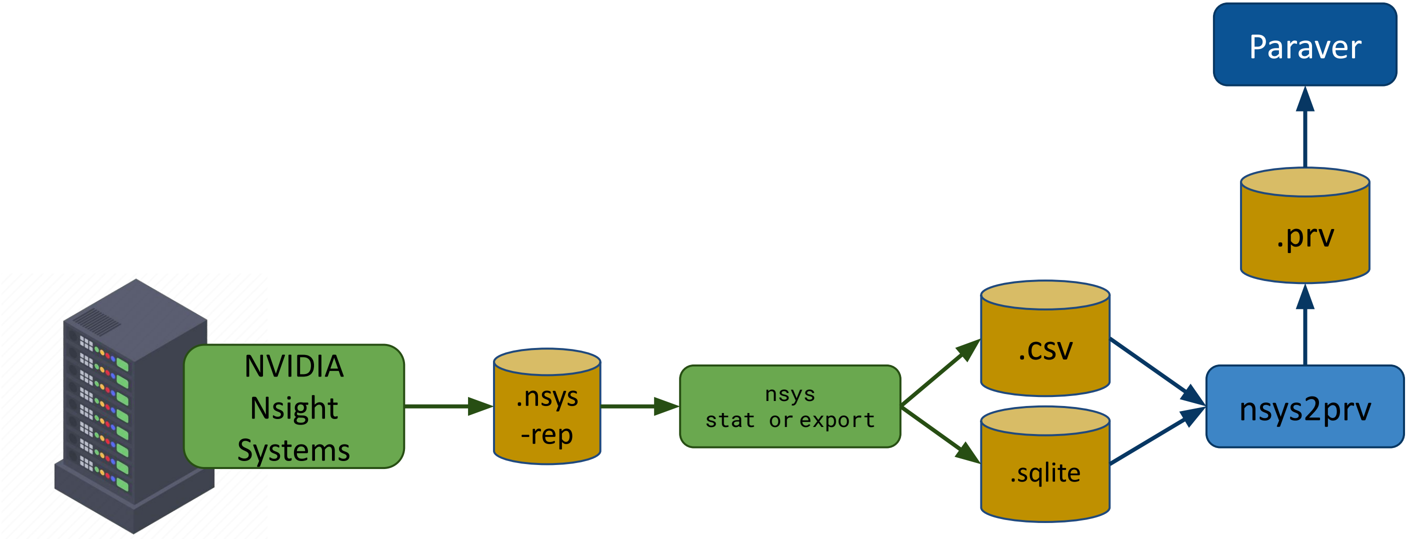
Main enabler for performance analysis: reduce gaps of different tools

Limitations:

- Nsight Systems CSV output format not documented
- SQLite schema briefly documented, and rapidly updated/changed
- Still it’s a lot of information, requires manual parsing
- Somehow unstable, lots of edge cases to find yet
- (still) Limited to NVIDIA software stack

Usage

Workflow of the translation



Usage

First, we need to get the Nsight Systems profiling with nsys profile.

```
$> nsys profile --gpu-metrics-device=cuda-visible -t cuda,nvtx -o ./llm_all
--capture-range=nvtx --env-var=NSYS_NVTX_PROFILER_REGISTER_ONLY=0
--nvtx-capture=RANGE_NAME python TestLLAMA.py
```

In this example, we ask the profiler to only trace during the "RANGE_NAME" NVTX range, to get a trace for our phase of interest.

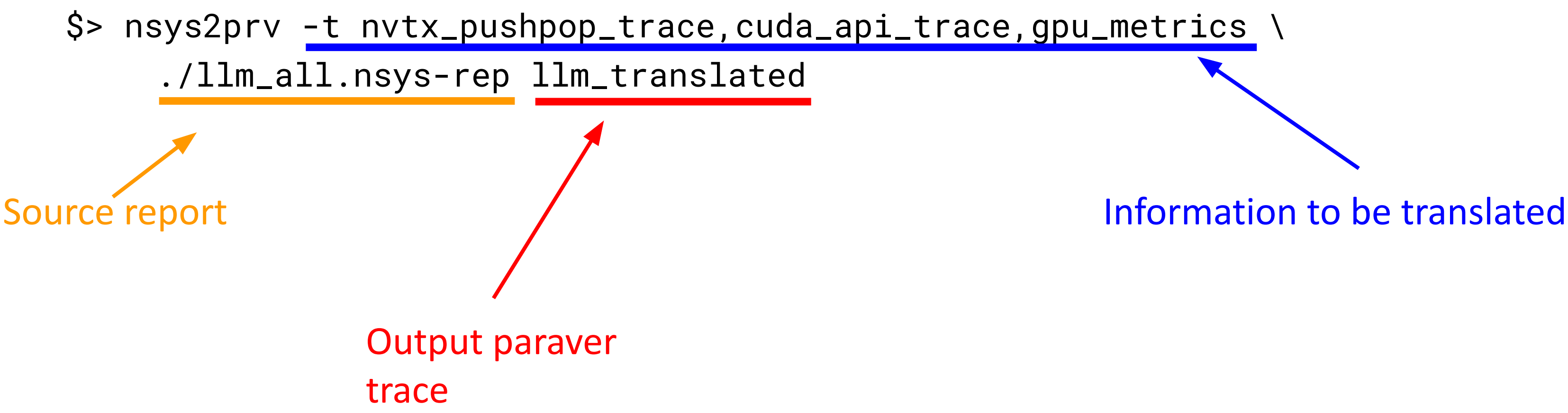
This should output an .nsys-rep file

```
$> ls
 llm_all.nsys-rep
```

Usage

Now the translator digests the report and outputs a Paraver trace.

- The “-t” flag tells which information do we want to translate.



```
$> nsys2prv -t nvtx_pushpop_trace,cuda_api_trace,gpu_metrics \
      ./llm_all.nsys-rep llm_translated
```

Source report

Output paraver trace

Information to be translated

Usage (multi-report translation)

When running multi-node, we are forced to get multiple reports from Nsight Systems. We can give all reports to *nsys2prv*:

- Enabling analysis of LLMs at scale!

```
$> nsys2prv -t nvtx_pushpop_trace,cuda_api_trace,gpu_metrics \
     -m ./llm_0.nsys-rep ./llm_1.nsys-rep ./llm_2.nsys-rep … llm_translated
```

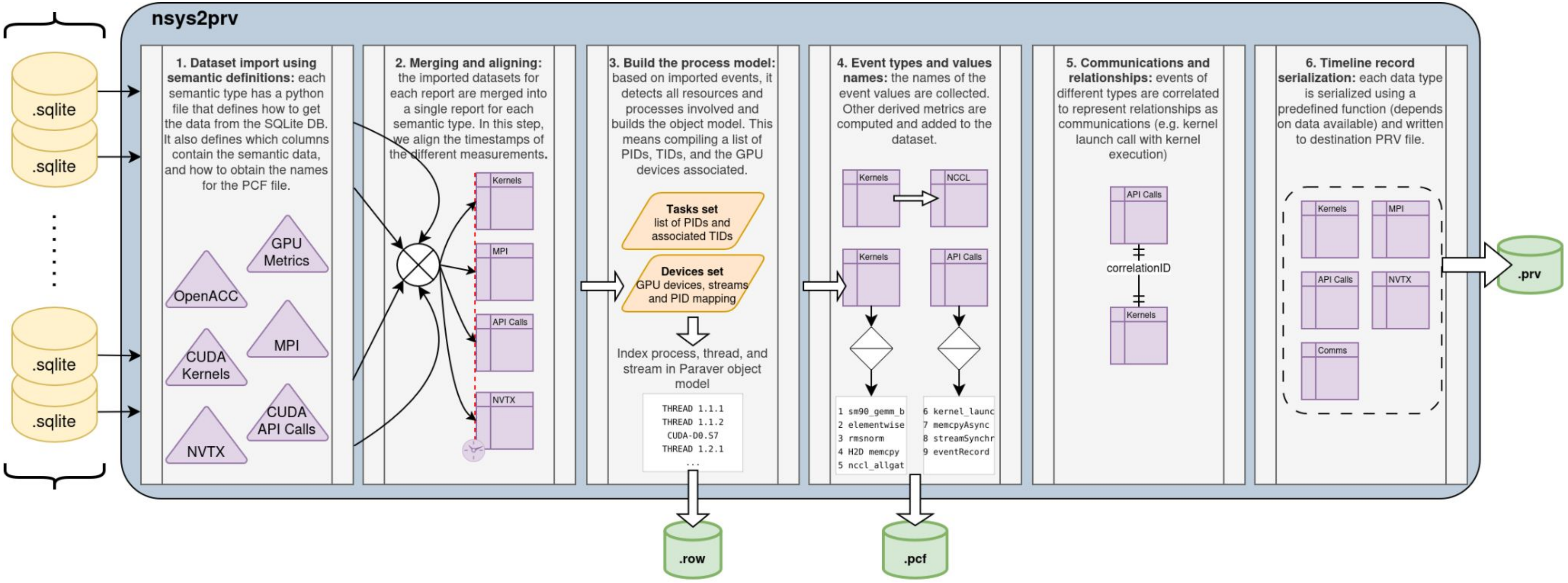
Multi-report flag

Source reports

- ⚠️ Inter-node clock synchronization still not implemented

Usage

Translation internal process



1. Dataset import using semantic definitions: each semantic type has a python file that defines how to get the data from the SQLite DB. It also defines which columns contain the semantic data, and how to obtain the names for the PCF file.

2. Merging and aligning: the imported datasets for each report are merged into a single report for each semantic type. In this step, we align the timestamps of the different measurements.

3. Build the process model: based on imported events, it detects all resources and processes involved and builds the object model. This means compiling a list of PIDs, TIDs, and the GPU devices associated.

Tasks set list of PIDs and associated TIDs

Devices set GPU devices, streams and PID mapping

Index process, thread, and stream in Paraver object model

```
THREAD 1.1.1
THREAD 1.1.2
CUDA-D0.S7
THREAD 1.2.1
...
```

4. Event types and values names: the names of the event values are collected. Other derived metrics are computed and added to the dataset.

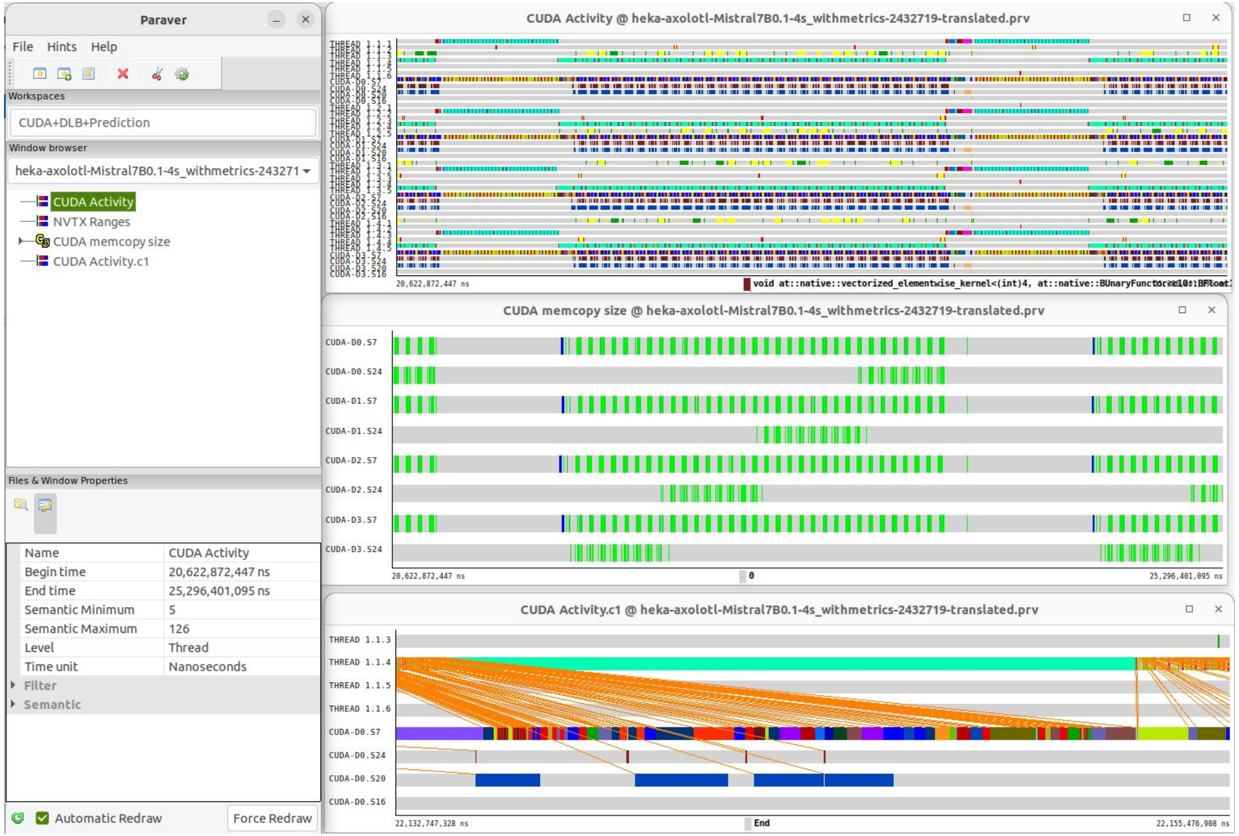
```
1 sm90_gemm_b
2 elementwise
3 rmsnorm
4 H2D memcpy
5 nccl_allgat
```

```
6 kernel_launc
7 memcpyAsync
8 streamSynchr
9 eventRecord
```

5. Communications and relationships: events of different types are correlated to represent relationships as communications (e.g. kernel launch call with kernel execution)

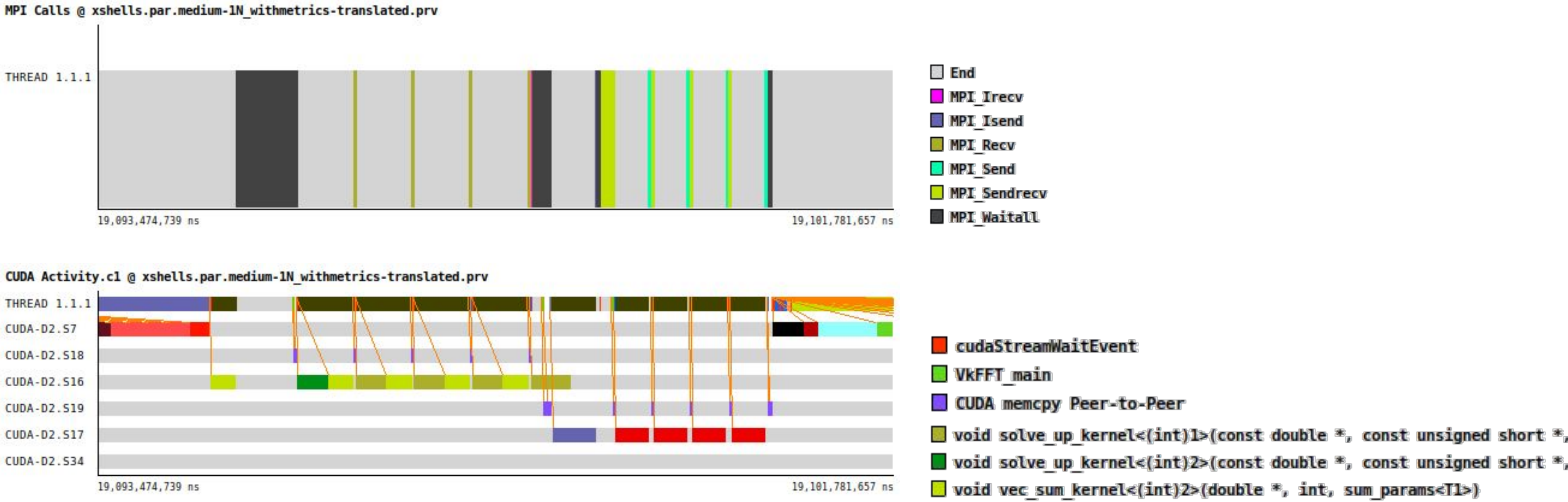
6. Timeline record serialization: each data type is serialized using a predefined function (depends on data available) and written to destination PRV file.

Examples: LLM training

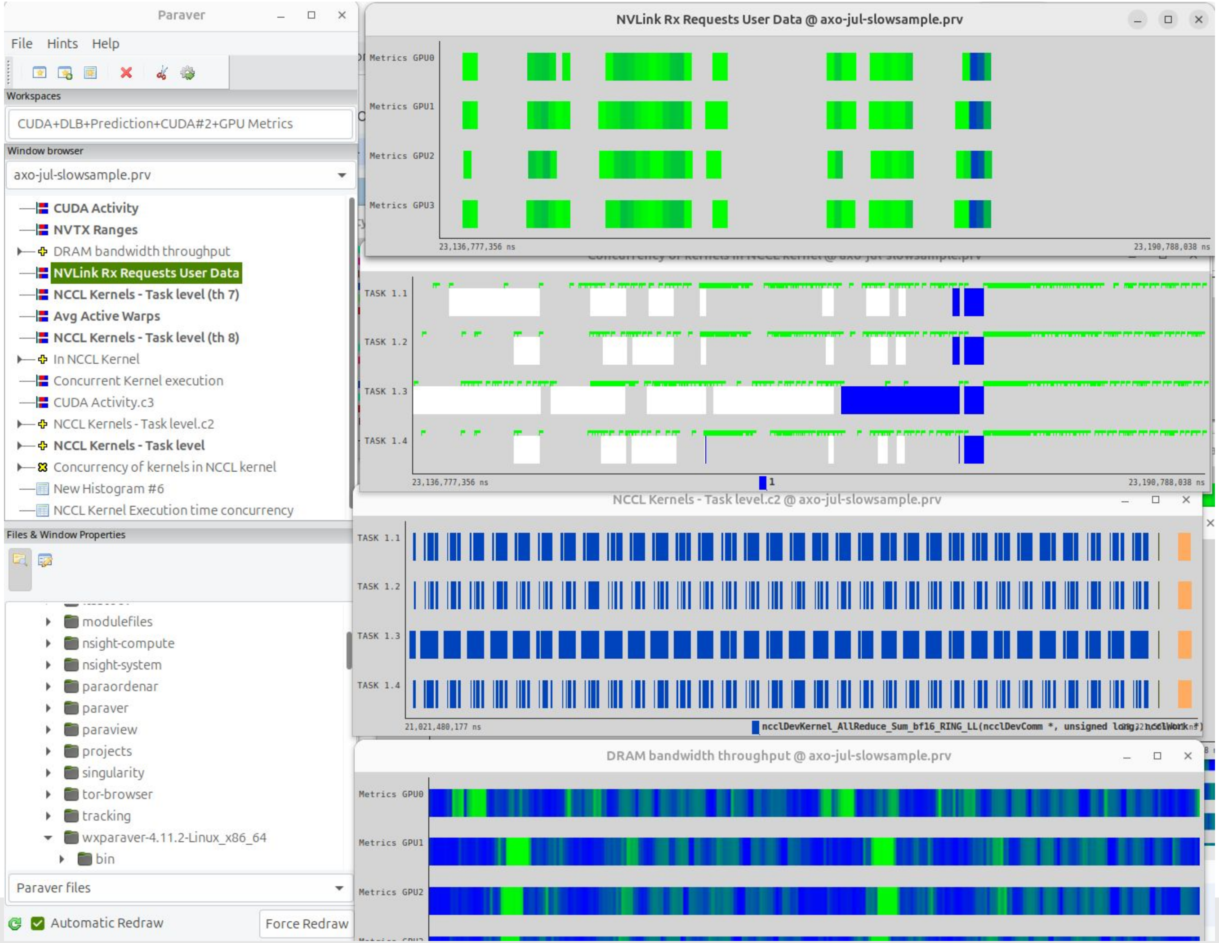


Name	CUDA Activity
Begin time	20,622,872,447 ns
End time	25,296,401,095 ns
Semantic Minimum	5
Semantic Maximum	126
Level	Thread
Time unit	Nanoseconds

Examples: SOD2D


MPI Calls @ xshells.par.medium-1N_withmetrics-translated.prv
THREAD 1.1.1
19,093,474,739 ns
19,101,781,657 ns
End
MPI_Irecv
MPI_Isend
MPI_Recv
MPI_Send
MPI_Sendrecv
MPI_Waitall
CUDA Activity.c1 @ xshells.par.medium-1N_withmetrics-translated.prv
THREAD 1.1.1
CUDA-D2.S7
CUDA-D2.S18
CUDA-D2.S16
CUDA-D2.S19
CUDA-D2.S17
CUDA-D2.S34
19,093,474,739 ns
19,101,781,657 ns
cudaStreamWaitEvent
VkFFT_main
CUDA memcpy Peer-to-Peer
void solve_up_kernel<(int)1>(const double *, const unsigned short *,
void solve_up_kernel<(int)2>(const double *, const unsigned short *,
void vec_sum_kernel<(int)2>(double *, int, sum_params<T1>)

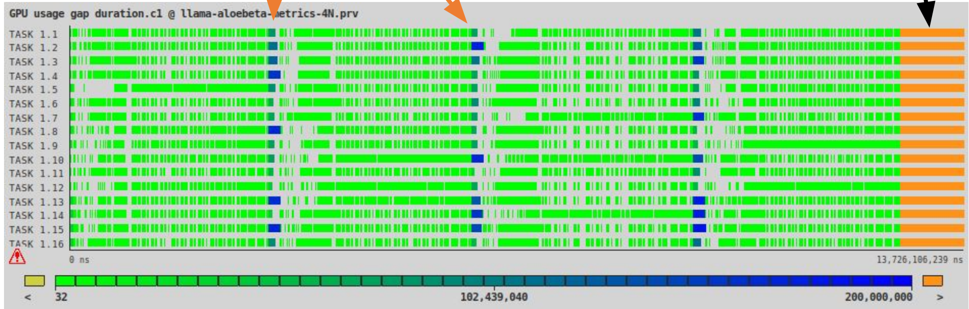
Examples: NCCL usage study


Paraver
File Hints Help
Workspaces
CUDA+DLB+Prediction+CUDA#2+GPU Metrics
Window browser
axo-jul-slowsample.prv
CUDA Activity
NVTX Ranges
DRAM bandwidth throughput
NVLink Rx Requests User Data
NCCL Kernels - Task level (th 7)
Avg Active Warps
NCCL Kernels - Task level (th 8)
In NCCL Kernel
Concurrent Kernel execution
CUDA Activity.c3
NCCL Kernels - Task level.c2
NCCL Kernels - Task level
Concurrency of kernels in NCCL kernel
New Histogram #6
NCCL Kernel Execution time concurrency
Files & Window Properties
modulefiles
nsight-compute
nsight-system
paraordenar
paraver
paraview
projects
singularity
tor-browser
tracking
wxparaver-4.11.2-Linux_x86_64
bin
Paraver files
Automatic Redraw
Force Redraw
NVLink Rx Requests User Data @ axo-jul-slowsample.prv
Metrics GPU0
Metrics GPU1
Metrics GPU2
Metrics GPU3
23,136,777,356 ns
23,190,788,038 ns
Concurrency of kernels in NCCL kernel @ axo-jul-slowsample.prv
TASK 1.1
TASK 1.2
TASK 1.3
TASK 1.4
23,136,777,356 ns
1
23,190,788,038 ns
NCCL Kernels - Task level.c2 @ axo-jul-slowsample.prv
TASK 1.1
TASK 1.2
TASK 1.3
TASK 1.4
21,021,480,177 ns
ncclDevKernel_AllReduce_Sum_bf16_RING_LL(ncclDevComm *, unsigned long, ncclWork *)
DRAM bandwidth throughput @ axo-jul-slowsample.prv
Metrics GPU0
Metrics GPU1
Metrics GPU2

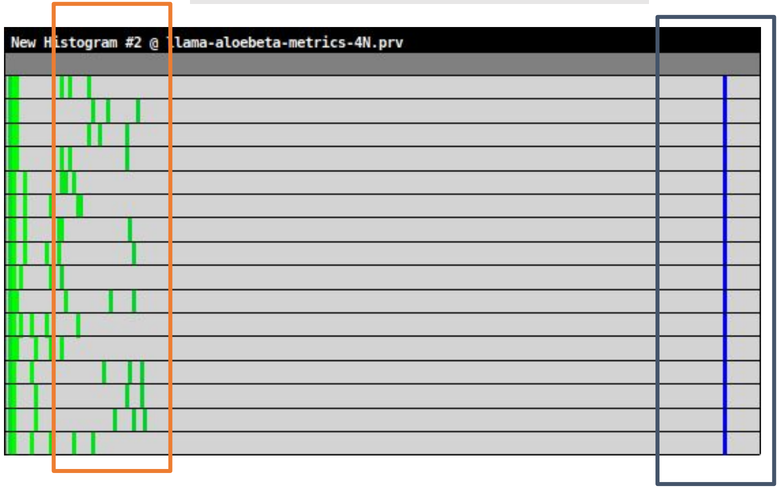
Examples: GPU gap analysis

Significative GPU usage gaps of ~200ms max found at the end of every micro-step, with differences between GPUs

Big GPU gap at the end of step. For further analysis, a complete trace of two steps is needed to have a clean view of what is happening in between 2 steps



Histogram shows relevant GPU gap regions, that then can be related to a timeline window

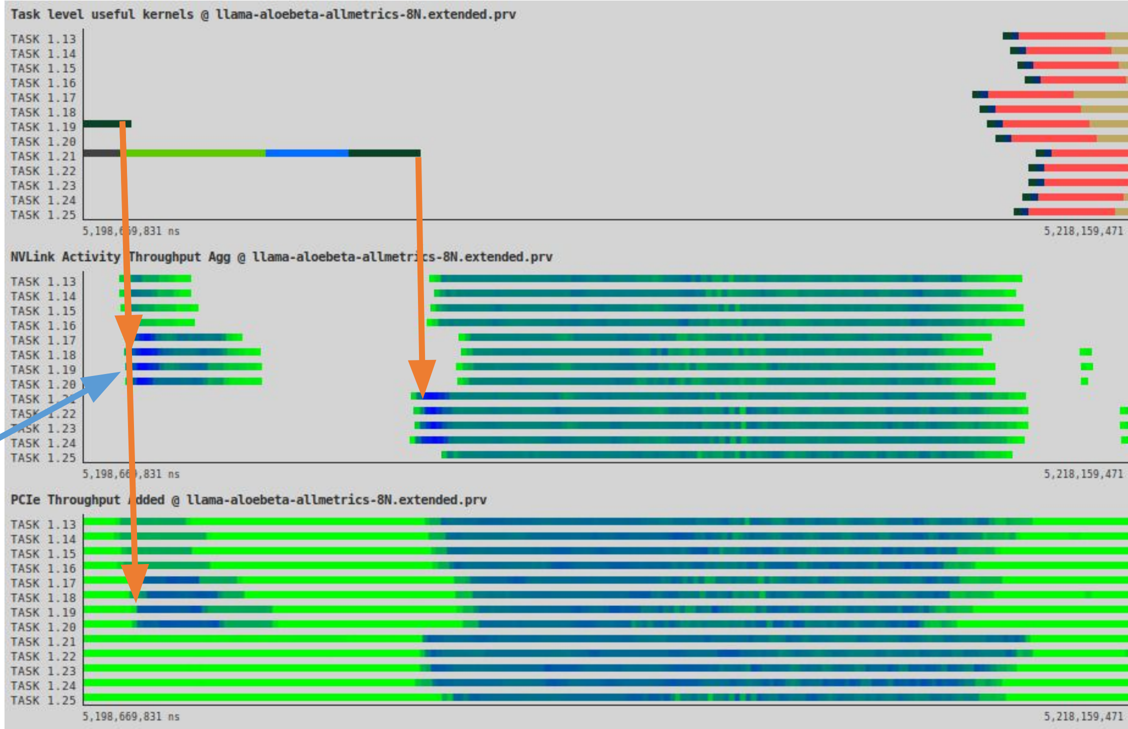


Examples: detailed communication behavior

Compute kernels

NVL Activity throughput aggregated

PCIe throughput aggregated


Task level useful kernels @ llama-aloebeta-allmetrics-8N.extended.prv
TASK 1.13
TASK 1.14
TASK 1.15
TASK 1.16
TASK 1.17
TASK 1.18
TASK 1.19
TASK 1.20
TASK 1.21
TASK 1.22
TASK 1.23
TASK 1.24
TASK 1.25
5,198,669,831 ns
5,218,159,471
NVLink Activity Throughput Agg @ llama-aloebeta-allmetrics-8N.extended.prv
5,198,669,831 ns
5,218,159,471
PCIe Throughput Added @ llama-aloebeta-allmetrics-8N.extended.prv
5,198,669,831 ns
5,218,159,471

When a full node reaches the synchronization point, some communication actually happens intra and inter-node, transmitting some data in advance

Let’s get into it!

Common steps:

```
$> module load intel mkl python/3.12.1 sqlite3 #required modules in MN5
$> module load nvidia-hpc-sdk # or ..
$> export /
NSYS_HOME=/gpfs/scratch/nct_310/nsight-systems/opt/nvidia/nsight-systems-cli/2024.6.1
```

🟢 For the stable version:

```
$> source /gpfs/scratch/nct_310/nsys2prv/env/bin/activate
```

🟡 For the prerelease version with latest features:

```
$> source /gpfs/scratch/nct_310/nsys2prv-pre/env/bin/activate
```

Resources

Repo with set of Paraver CFGs:

🔗 https://pm.bsc.es/gitlab/beppp/nsys2prv/-/tree/main/cfgs

Documentation:

🔗 https://pm.bsc.es/gitlab/beppp/nsys2prv/-/wikis/home

Current status of translation features:

🔗 https://pm.bsc.es/gitlab/beppp/nsys2prv/-/wikis/Translation%20status

PIP package:

🔗 https://pypi.org/project/nsys2prv/

Thank you!

Translate a lot of traces and file a lot of bug reports!