nsys2prv: from *Nsight Systems* to Paraver traces

Marc Clascà Ramírez
marc.clasca@bsc.es
48th VI-HPS Tuning Workshop

Introduction

When do we need to jump to Nsight Systems?

What can we see with these translated traces?

How do we translate traces?

BSC Tools overview

Extrae

- System level parallel performance analysis
- Timestamped events, configurable semantics
- CUDA support improving in progress
- Requires MPI for distributed memory applications

Paraver

- Configurable visualizations via DSL
- Suitable for large number of resources

NVIDIA Nsight Systems

- Comprehensive workload-level performance
- System level information: different runtimes and hardware metrics
- Typical behaviors to study: synchronization, parallelization, data movement
- Trace visualization integrated, usable up to ~8 processes

NVIDIA Nsight Compute

- Detailed CUDA kernel performance
- Isolated kernel execution information: requires replaying
- Typical behaviors to study: GPU utilization, kernel implementation, memory access

BSC Tools overview

System level overview

Individual kernel level

Extrae

- System level parallel performance analysis
- Timestamped events, configurable semantics
- CUDA support improving in progress
- Requires MPI for distributed memory applications

Paraver

- Configurable visualizations via DSL
- Suitable for large number of resources

NVIDIA Nsight Systems

- Comprehensive workload-level performance
- System level information: different runtimes and hardware metrics
- Typical behaviors to study: synchronization, parallelization, data movement
- Trace visualization integrated, usable up to ~8 processes

NVIDIA Nsight Compute

- Detailed CUDA kernel performance
- Isolated kernel execution information: requires replaying
- Typical behaviors to study: GPU utilization, kernel implementation, memory access

BSC Tools overview

Proprietary, rich information and metrics

Open-source, under development

Extrae

- System level parallel performance analysis
- Timestamped events, configurable semantics
- CUDA support improving in progress
- Requires MPI for distributed memory applications

Paraver

- Configurable visualizations via DSL
- Suitable for large number of resources

NVIDIA Nsight Systems

- Comprehensive workload-level performance
- System level information: different runtimes and hardware metrics
- Typical behaviors to study: synchronization, parallelization, data movement
- Trace visualization integrated, usable up to ~8 processes

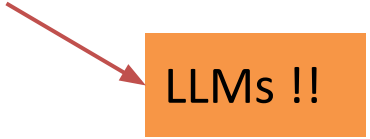
NVIDIA Nsight Compute

- Detailed CUDA kernel performance
- Isolated kernel execution information: requires replaying
- Typical behaviors to study: GPU utilization, kernel implementation, memory access

Highly customizable, large-scale parallel traces

When should we use Nsight Systems?

- Distributed parallel runtimes not supported by other tracers
- Overview of GPU metrics
- Instrumentation with NVTX
- Mix of multiple programming models
- Anything that we don’t understand from our own tool…



Nsight Systems profiling

Basic profiling session

```
$> nsys profile --gpu-metrics-devices=cuda-visible -t cuda,nvtx -o ./llm_all
python TestLLAMA.py
```

- -t (API tracing) cuda,nvtx,openmp,mpi,openacc…
- –gpu-metrics-devices Obtain GPU hardware metrics, predefined set for device
- -o name of the report, allows %q{ENV_VAR}

This should output an .nsys-rep file

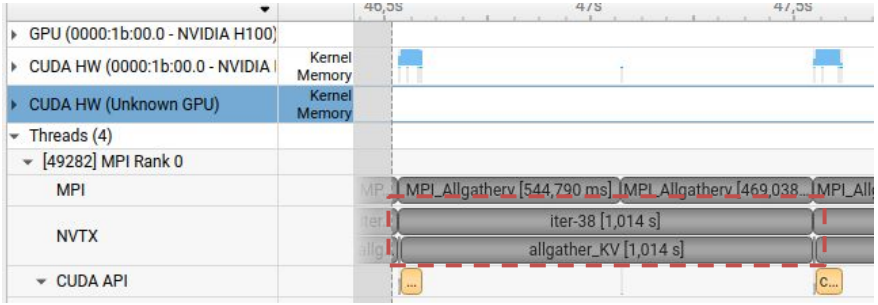
```
$> ls
 llm_all.nsys-rep
```

Instrumentation with NVTX

- Instrument application source code with markers and code regions.
- API also provides python decorators

```
from torch.cuda import nvtx

nvtx.range_push("region X")
# your Python code
nvtx.range_pop()
```




```
import nvtx

@nvtx.annotate("f()", color="purple")
def f():
  for i in range(5):
    with nvtx.annotate("loop", color="red"):
      # Python code goes here
```

Finer grain profiling with NVTX

```
$> nsys profile --gpu-metrics-devices=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.

We can also start and stop the profiler directly with the API:

```
import torch.cuda

if epoch == 2:
  torch.cuda.cudart().cudaProfilerStart()

# train code

if epoch == 3:
  torch.cuda.cudart().cudaProfilerStop()
```

GPU metrics overview

Metrics definition and CUPTI raw equivalent

- GPC Clock Frequency: GPC graphics clock, “Boost Clock” or “Base Clock”
  - `gpc__cycles_elapsed.avg.per_second`
- SM Issue rate: SM inst. issue rate. Each SM can issue 4 instructions per cycle.
  - `sm__inst_executed_realtime.avg.pct_of_peak_sustained_elapsed`
- Tensor Active: Cycles the tensor pipe is active (% of peak)
  - `sm__pipe_tensor_cycles_active_realtime.avg.pct_of_peak_sustained_elapsed`
- NVLink BW: Number of bytes sent/recvd via NVLink for each type of packet (% of peak)
  - `NVL{RX,TX}.TriageCompute.nvl{rx,tx}__bytes_packet_{type}.avg.pct_of_peak_sustained_elapsed`
- PCIe Bandwidth: Number of bytes sent/recvd by GPU (% of peak)
  - `PCI.TriageCompute.pci__{read,write}_bytes.avg.pct_of_peak_sustained_elapsed`
- Compute warps in flight: Number of compute shader warps in flight (% of peak)
  - `TPC.TriageCompute.tpc__warps_active_shader_cs_realtime.avg`

GPU metrics overview

Metrics definition and CUPTI raw equivalent

- GPC Clock Frequency: GPC graphics clock, “Boost
  - `gpc__cycles_elapsed.avg.per_second`
- SM Issue rate: SM inst. issue rate. Each SM can issue 4 instructions per cycle.
  - `sm__inst_executed_realtime.avg.pct_of_peak_sustained_elapsed`
- Tensor Active: Cycles the tensor pipe is active (% of peak)
  - `sm__pipe_tensor_cycles_active_realtime.avg.pct_of_peak_sustained_elapsed`
- NVLink BW: Number of bytes sent/recvd via NVLink for each type of packet (% of peak)
  - NV…eComput…{type…_elapsed
- PCIe Bandwidth: Num…vd by…
  - PCI.TriageCompute.pci__…of_peak_sustained_elapsed
- Compute warps in flight: Number of compute shader warps in flight (% of peak)
  - `TPC.TriageCompute.tpc__warps_active_shader_cs_realtime.avg`

For reference: https://docs.nvidia.com/cupti/main/main.html#enumeration

Metric name

Unit instances aggregation operation (sum, max, min, avg), called *rollup*

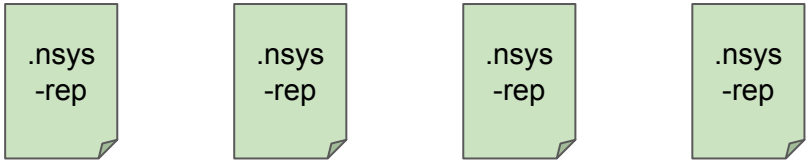
Submetric, metric transformation (e.g. events per second or *% of peak sustained rate achieved during unit elapsed cycles*)

What can we see/do?

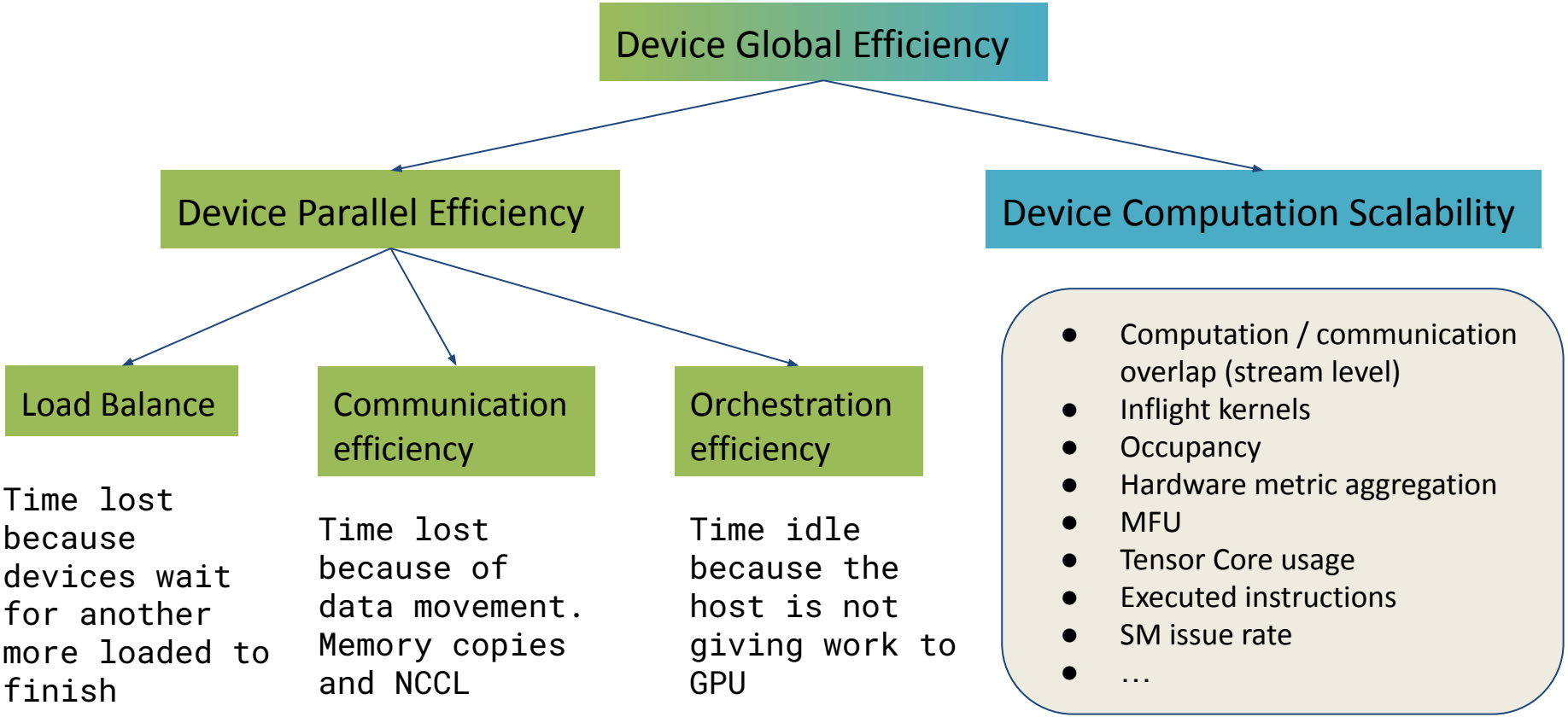
- Multi-node, large-scale runs traces merged
- Analysis of NCCL behavior
- POP efficiency model!
- Derived metrics
  - Attribution of GPU metrics to regions, kernels…
  - Relationship between CUDA API calls and kernels
  - Aggregations (of streams) per task or application

Multi-node profiling

- Prefix “nsys profile” before your binary, after the launcher
  - `srun <args> nsys profile ./your_app <args> …`
- Nsight Systems creates one report for every profiler instance
- Use SLURM environment variables to tune nsys parameters, like:
  - Report name: `-o report_N${SLURM_NODENAME}_${SLURM_PROCID}`
  - Which GPUs to get metrics: `–gpu-metrics-devices=$SLURM_LOCALID`

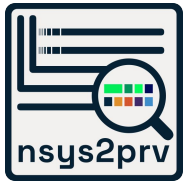


Methodology - Efficiency model



nsys2prv - What is it?

- **Translate** performance data acquired by Nsight Systems into Paraver **timestamped records**.
- **Merge** multiple *.nsys-rep* reports, coming from a **multi-node execution**, into a single trace.
- And we provide all **predefined configuration files** for Paraver within the package to display all metrics described in the article and in this presentation



Package released on PyPI
🔗 https://pypi.org/project/nsys2prv/

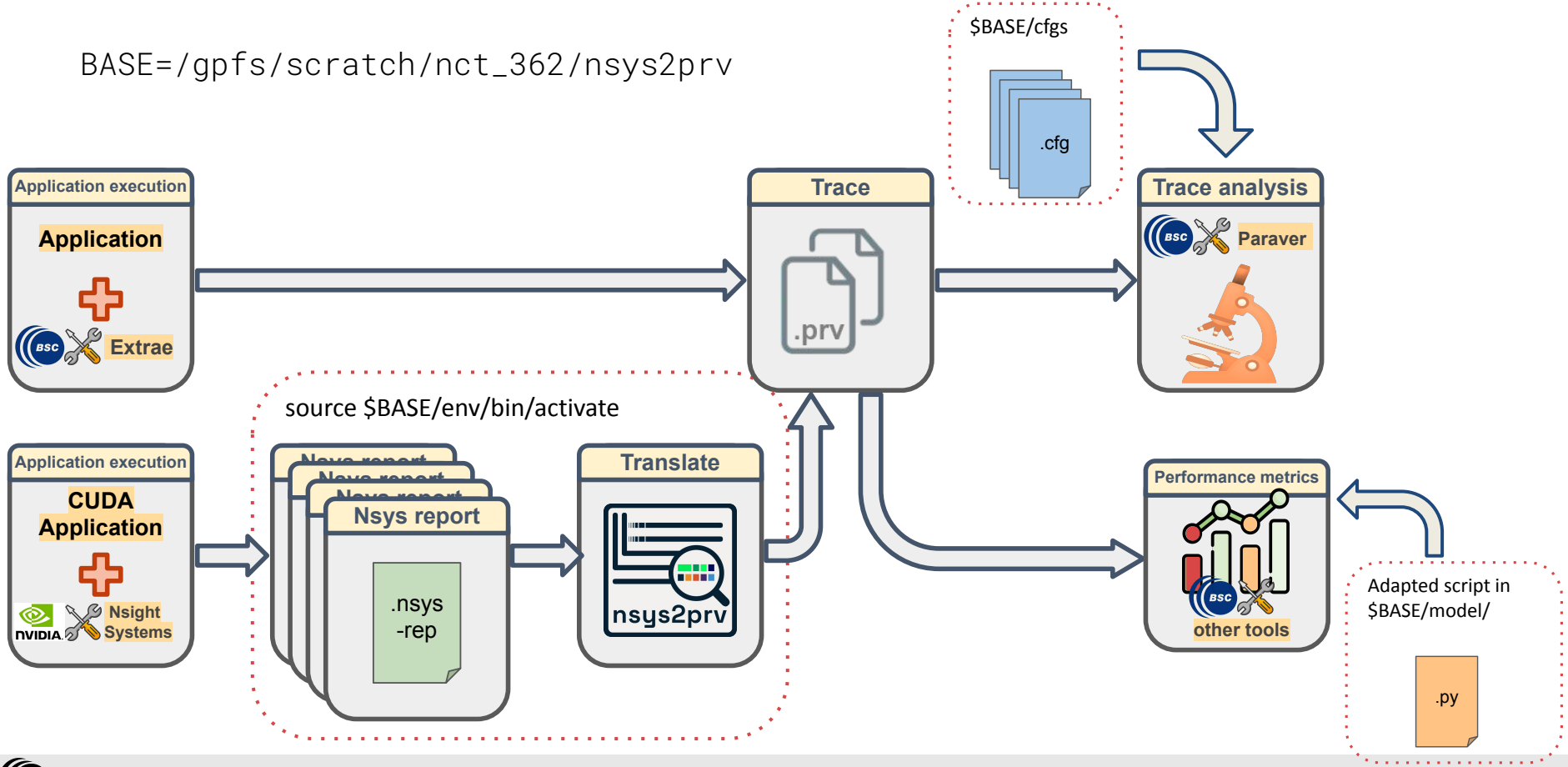
Source code publicly available on
🔗 https://gitlab.pm.bsc.es/beppp/nsys2prv

📂 Installed on the shared folder of MN5!

```
$> 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
```

Translation workflow

BASE=/gpfs/scratch/nct_362/nsys2prv



How do we translate a trace?

```
$> module load intel mkl impi hdf5 python/3.12.1 sqlite3
$> source /gpfs/scratch/nct_362/nsys2prv/env/bin/activate

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