VSB TECHNICAL UNIVERSITY OF OSTRAVA | IT4INNOVATIONS NATIONAL SUPERCOMPUTING CENTER

INTRODUCTION TO HIGH PERFORMANCE COMPUTING

PERFORMANCE ANALYSIS BASICS

Radim Vavřík

EUROPEAN UNION
European Structural and Investment Funds
Operational Programme Research,
Development and Education

MINISTRY OF EDUCATION, YOUTH AND SPORTS

OUTLINE

Performance analysis and optimisation

- Motivation
- Hardware aspects
- Development process
- Best-practices

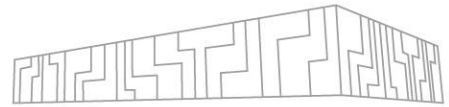
Performance tools and methodology

- Performance metrics
- CPU/GPU tools
- Live examples

POP CoE

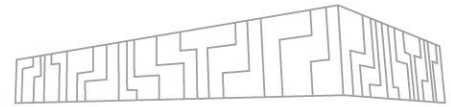
Cray-1 supercomputer, source: wikipedia.org

TECHNICAL NOTES



- All presented tools/examples can be accessed and reproduced at IT4I clusters **anytime**
- Please, setup your preferred GUI access:
  1. **VNC** - server on a Karolina login node + client on your local machine
     - How to? https://docs.it4i.cz/general/accessing-the-clusters/graphical-user-interface/vnc/
     - Recommended client https://www.realvnc.com/en/connect/download/viewer/
  2. OOD - Open OnDemand GUI via web browser, **IT4I VPN required**
     - How to? https://docs.it4i.cz/general/accessing-the-clusters/graphical-user-interface/ood/
     - Connection link https://ood-karolina.it4i.cz/
  3. **X11** - Log in via terminal with X-Window system enabled
     - How to? https://docs.it4i.cz/general/accessing-the-clusters/graphical-user-interface/x-window-system/
     - Usually worse UX for GUI apps due to network latency
- Most of the presented tools provide a **remote profiling**, e.g., generate output remotely from CLI while analysis can be done locally in GUI - not covered today

PERFORMANCE ANALYSIS



Who has any experience with a performance analysis tool?

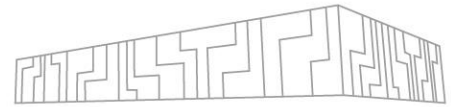
- What was the tool?

Objectives today?

- Not to become an expert analyst
- Not to reach an incredible performance improvement of example codes

- Rather to get idea about the domain and introduce some tools

EFFICIENT USE OF HPC



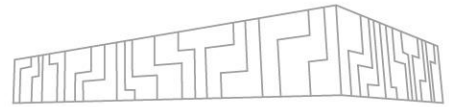
What does it mean?

- To get the most performance out of your hardware
- The process is called **Performance Optimisation**

Why should I care about performance?

- Industry – achieve goals faster and **cheaper**
- Academia – do **more science**
  - The trend in grant competition (resource allocation) is to prove performance, scalability, etc.

KEY INGREDIENTS



Know your application

- What does it compute? (domain, methods, algorithms)
- How is it parallelized? (programming models)
- What final performance is expected? (HW limits)

Know your hardware

- What are the target machines and how many? (laptop, workstation, cluster)
- Machine-specific optimisations?

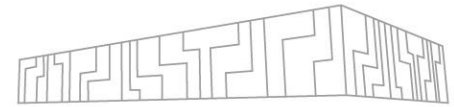
Know your tools

- Strengths and weaknesses of each tool? (easy-to-use vs detailed information)
- Learn how to use them (examples with problems/patterns)

Know your process

- Constant learning

HARDWARE ASPECTS OF PERFORMANCE



Filesystem

- I/O operations

Network

- internode communication

Memory subsystem

- NUMA effect

CPU cores

- thread/process affinity, pinning, caches

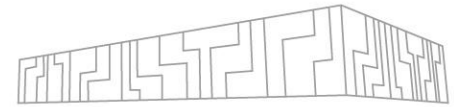
Vector registers

- vectorization, vector instructions

Accelerators

- GPU/MIC utilization, host-device data transfers

GET READY



Connect to Karolina login node via GUI

- **VNC** / OOD / X11

Submit an interactive job

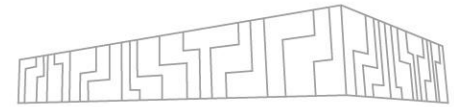
```
salloc -A ATR-25-5 --reservation=atr-25-5_2026-06-05T09:00_2026-06-05T12:30__qgpu -p qgpu -G 1 -n 16 -t 1:30:00
```

Tip!

- Use Adobe Acrobat Reader for copying the multi-line commands without line breaks

https://get.adobe.com/uk/reader/

BASIC TOOLS



Useful to get familiar with the machine

`lscpu`

`cat /proc/cpuinfo`

- processor : 71 -> 72 logical processors per node
- cpu cores : 18 -> 18 physical cores per socket
- siblings : 36 -> 36 logical processors per socket
- -> 2 hyperthreads per core
- -> 2 sockets per node

`cat /proc/meminfo`

- MemTotal: 196510848 kB -> 187 GiB # kiB in fact

`ml impi`

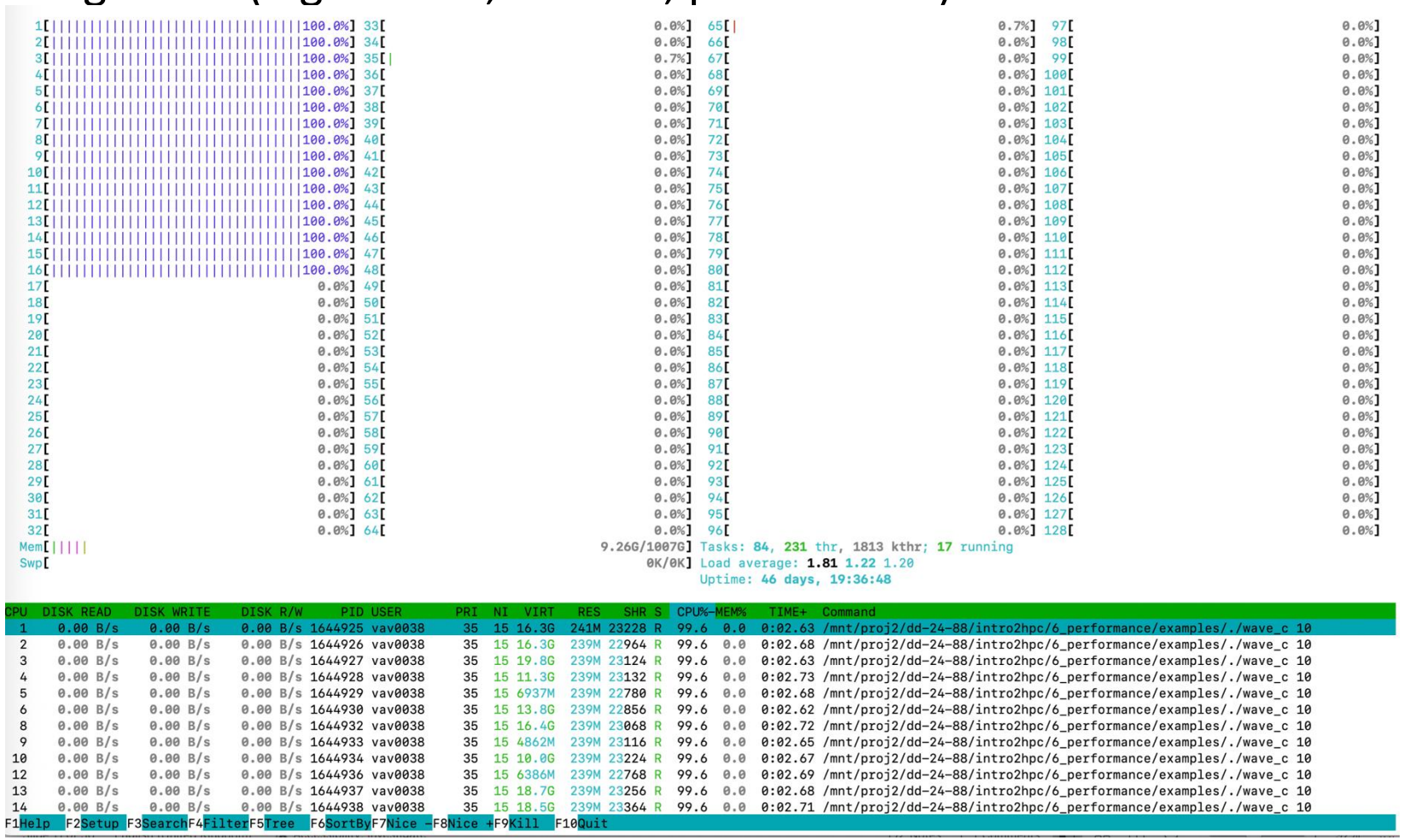
`cpuinfo` # Intel MPI utility, just single socket!

BASIC TOOLS

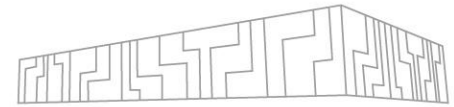
Use HTOP tool for interactive jobs

`htop -d 5` # delay 0.5s

- Configurable (e.g. core id, threads, process tree)



BASIC TOOLS

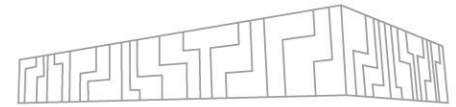


Similar tool for NVIDIA GPUs

```
watch -n 1 nvidia-smi
```

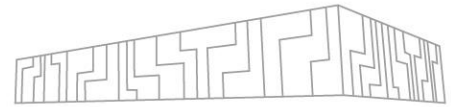
```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 550.54.15              Driver Version: 550.54.15      CUDA Version: 12.4     |
|-------------------------------+----------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|===============================+======================+======================|
|   0  NVIDIA A100-SXM4-40GB           Off |   00000000:0B:00.0 Off |                    0 |
| N/A   31C    P0             52W /  400W |       0MiB /  40960MiB |      0%      Default |
|                                         |                        |             Disabled |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

PERFORMANCE-AWARE DEVELOPMENT PROCESS



1. Develop correct functionality (testing helps)
2. Identify bottlenecks (performance limiters) using performance tools
3. Optimise bottlenecks until satisfied
    1. Build a hypothesis (ask a question)
    2. Explain the behavior (answer the question)
    3. Change the code (**double-check correct functionality**)
    4. Verify optimisations using profiling tools
4. Repeat until job done

BEST PRACTICES



- Do not optimise your code prematurely!
- Focus on main computational time-consuming phases (hotspots), omit preprocessing/postprocessing phases if applicable
- The 80/20 rule:
  - Programs typically spend 80% of their time in 20% of the code
  - Programmers typically spend 20% of their effort to get 80% of the total speedup possible for the application
- Keep track of your optimisation progress over time
- Always use compute nodes for profiling (**not login nodes - shared**)
- **Use SW libraries!**

SOFTWARE LIBRARIES

General-purpose math libraries

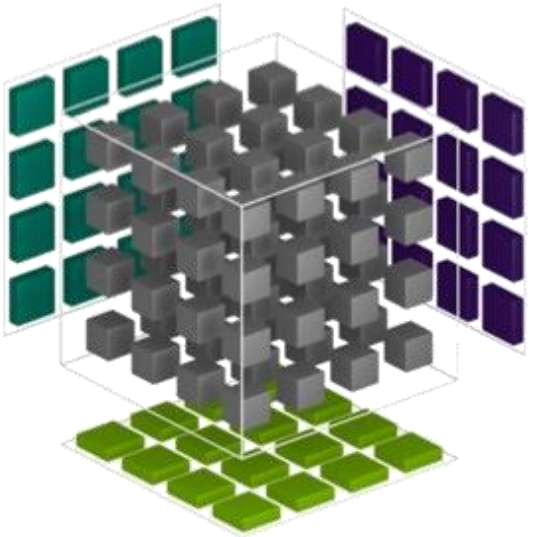
- BLAS (MKL, OpenBLAS, ATLAS, cuBLAS, ...)
- LAPACK (MKL, OpenBLAS, ATLAS, cuSolver, ...)
- FFT (MKL, cuFFT, ...)
- ...

Domain-specific libraries

- Chemistry, Bio, Geo, Physics, CAE, Big data, ML/DL

HW-specific libraries

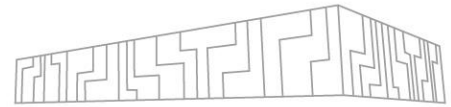
- GPU/MIC, Intel/AMD/IBM



Optimized implementation

- Usually much better performance than a custom code
- Do NOT reinvent a wheel!
- (But avoid overkill)

PERFORMANCE METRICS



Execution time (time, time.h, ...)

- real 0m10.245s (elapsed real time)
- user 0m19.890s (user CPU time using OMP_NUM_THREADS=2)
- sys 0m0.285s (system CPU time)

Processor speed (flop/s) and Memory throughput (GB/s)

- Calculated operations per time (e.g. c = a + b + c -> 2 operations)
- Transferred bytes per time (e.g. c = a + b + c -> 3 RD + 1 WR * 8 bytes)

Speedup and Efficiency

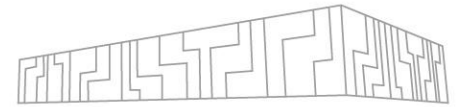
- $S_P = T_1 / T_P$
- $E_P = S_P / P$

Scalability

- Strong/weak scaling

Others: portability, programming ability, etc.

PEAK PERFORMANCE EXAMPLE



- The theoretical HW limits (Rpeak), e.g. AMD EPYC 7H12 (Rome)

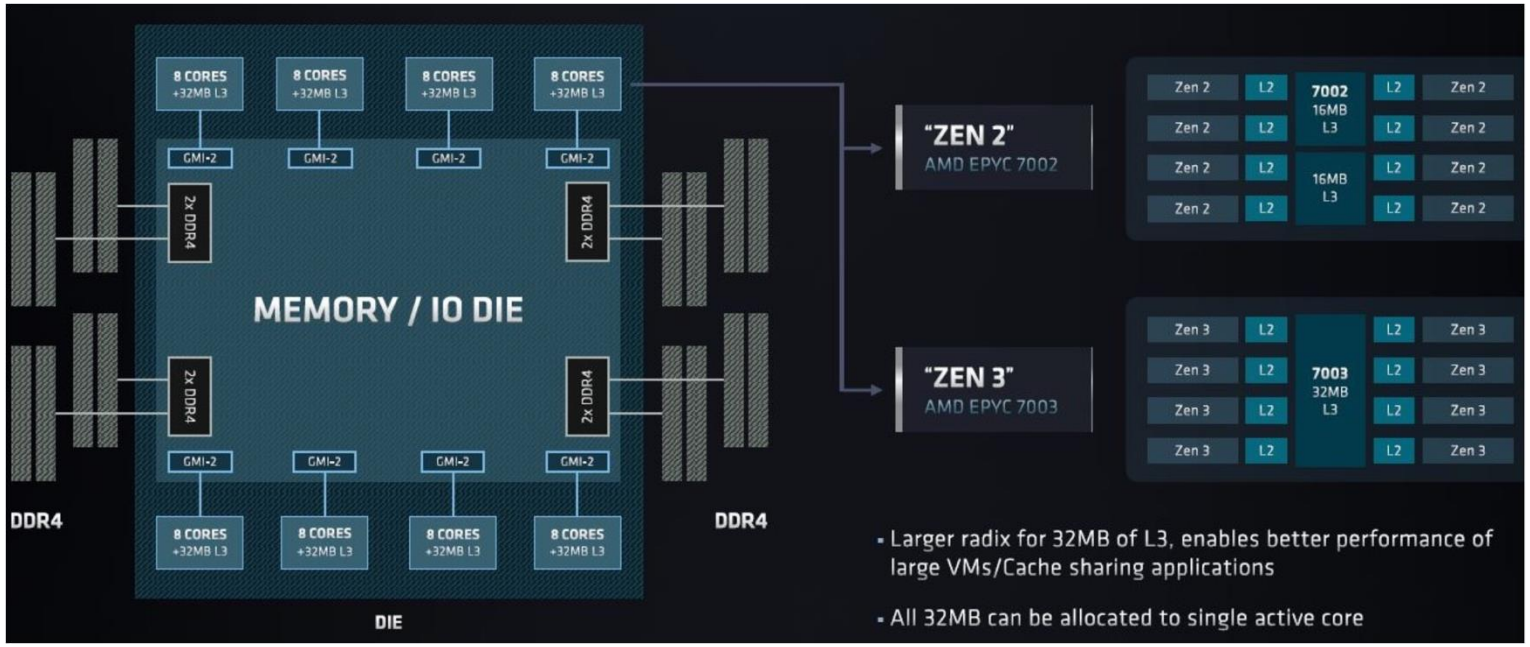
Processor speed:

Number of compute nodes (Karolina-size machine)	720
Number of sockets (CPUs) per node	2
Frequency	2.6 GHz
Number of cores per socket	64
FMA instructions (**a** * **b** + **c**)	2
FMA units per core	2
SIMD (AVX2 256b) = 4x double precision	4
	3 833 856 Gflop/s
	3.8 Pflop/s
	(2.6 Tflop/s per socket)

PEAK PERFORMANCE EXAMPLE

Memory bandwidth:

Number of compute nodes (Karolina-size machine)	720
Number of sockets (CPUs) per node	2
# channels per socket	8
DDR4 bus width	8 B
Frequency	3200 MT/s



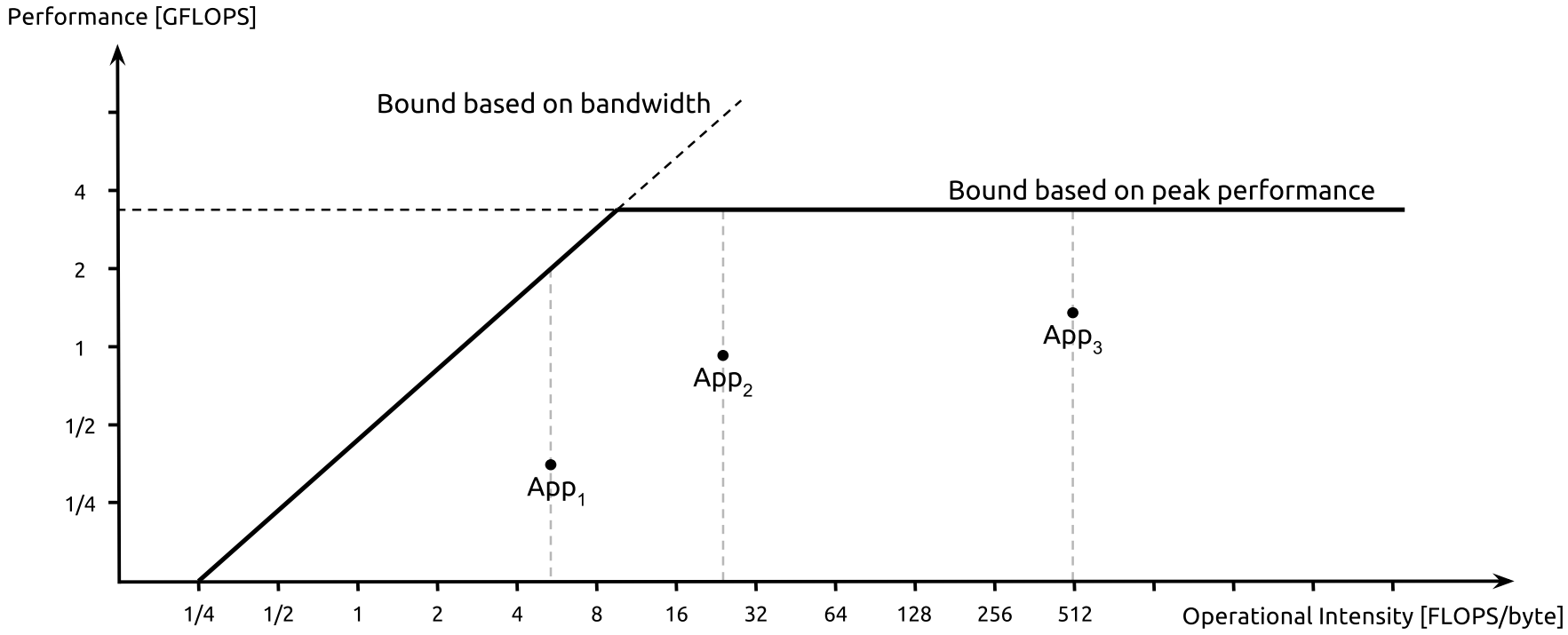
294 912 000 MB/s

294 TB/s

(204 GB/s per socket)

ROOFLINE MODEL

- Shows the performance of an algorithm (application) with respect to the HW limits of the architecture
- Identify if an algorithm is **compute bound** or **memory bound**
- Based on **Operational intensity** - a ratio of FLOPS (arithmetic operations) performed with required amount of data (operands)



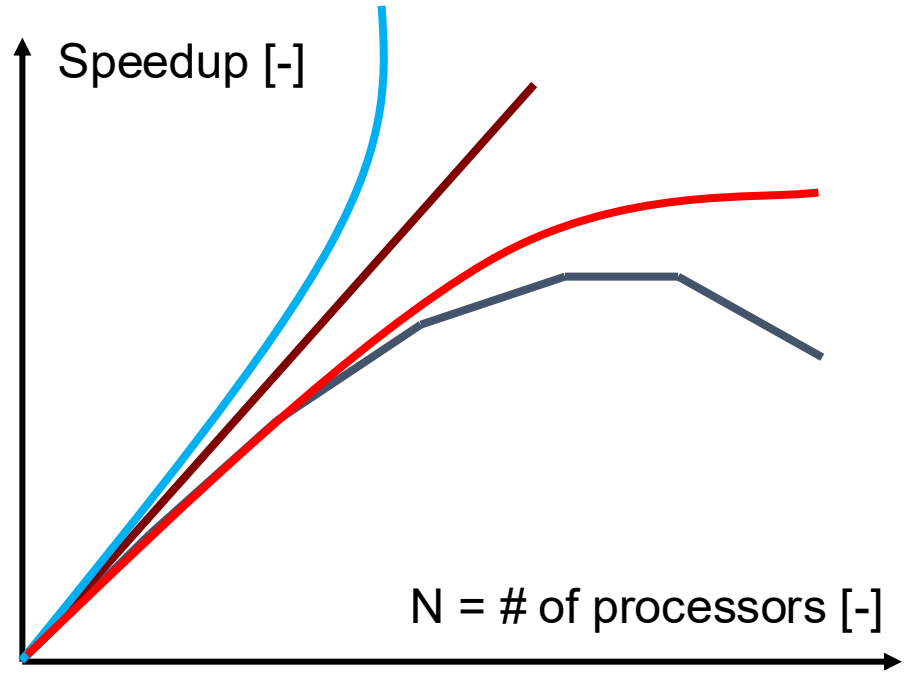
SPEEDUP

- **Speedup** – a ratio of a serial execution time to a parallel execution time

speedup on N processors - $S_N = \frac{T_1}{T_N}$

- T_1 - execution time on 1 processor
- T_N - execution time on N processors

- Linear speedup $S_N = N$
- Sub-linear speedup $S_N < N$
  - Communication
  - Load imbalance
  - Decomposition overhead
- Super-linear speedup $S_N > N$
  - Cache
  - Algorithm



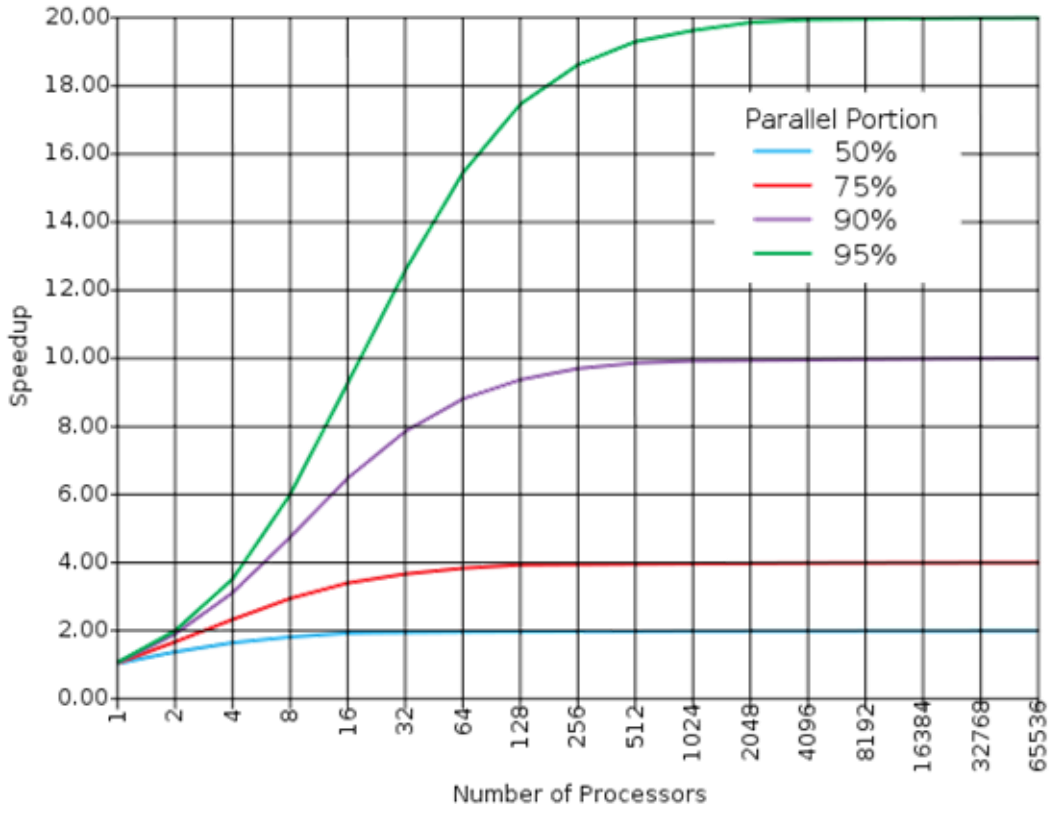
SCALABILITY

- **Scalability** - the ability to maintain performance gain when system and problem size increase

- **Amdahl's law** – maximum achievable speedup is limited by the serial portion of the code

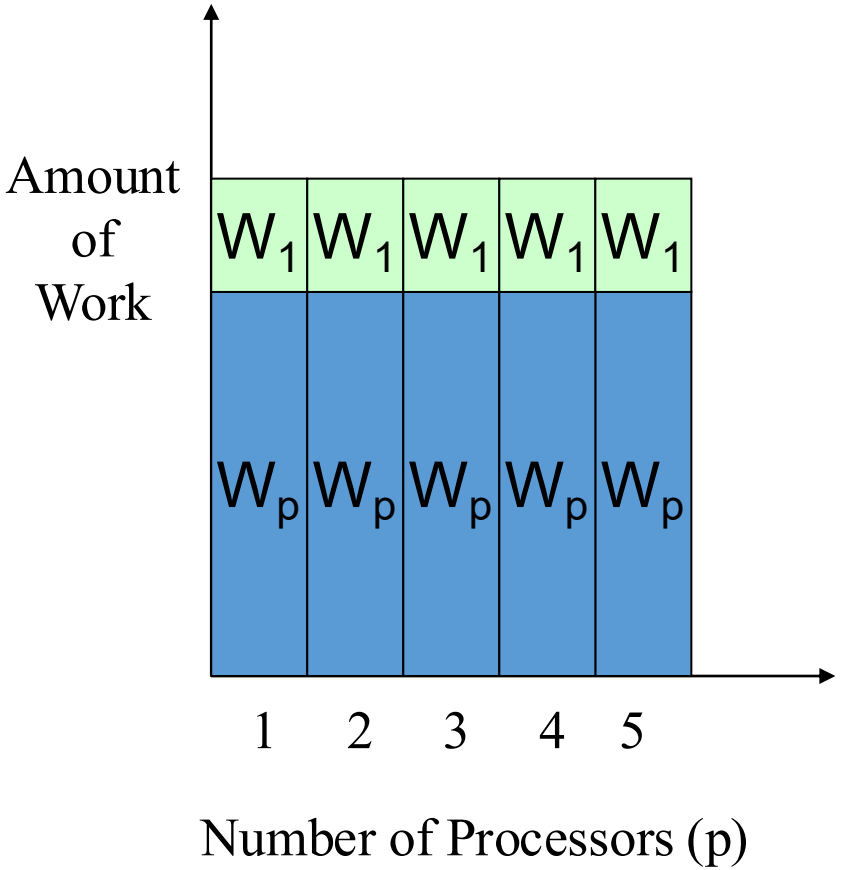
X% (parallel)	Y% (ser.)

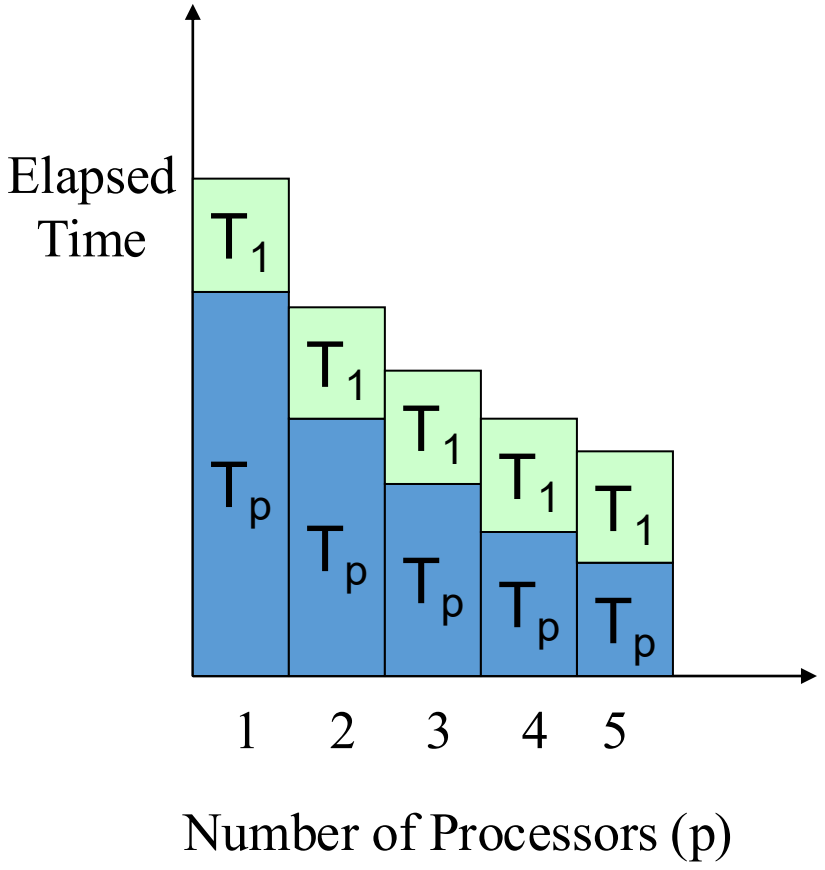
$$S_{MAX} = \frac{1}{\frac{Y}{100}}$$



SCALABILITY

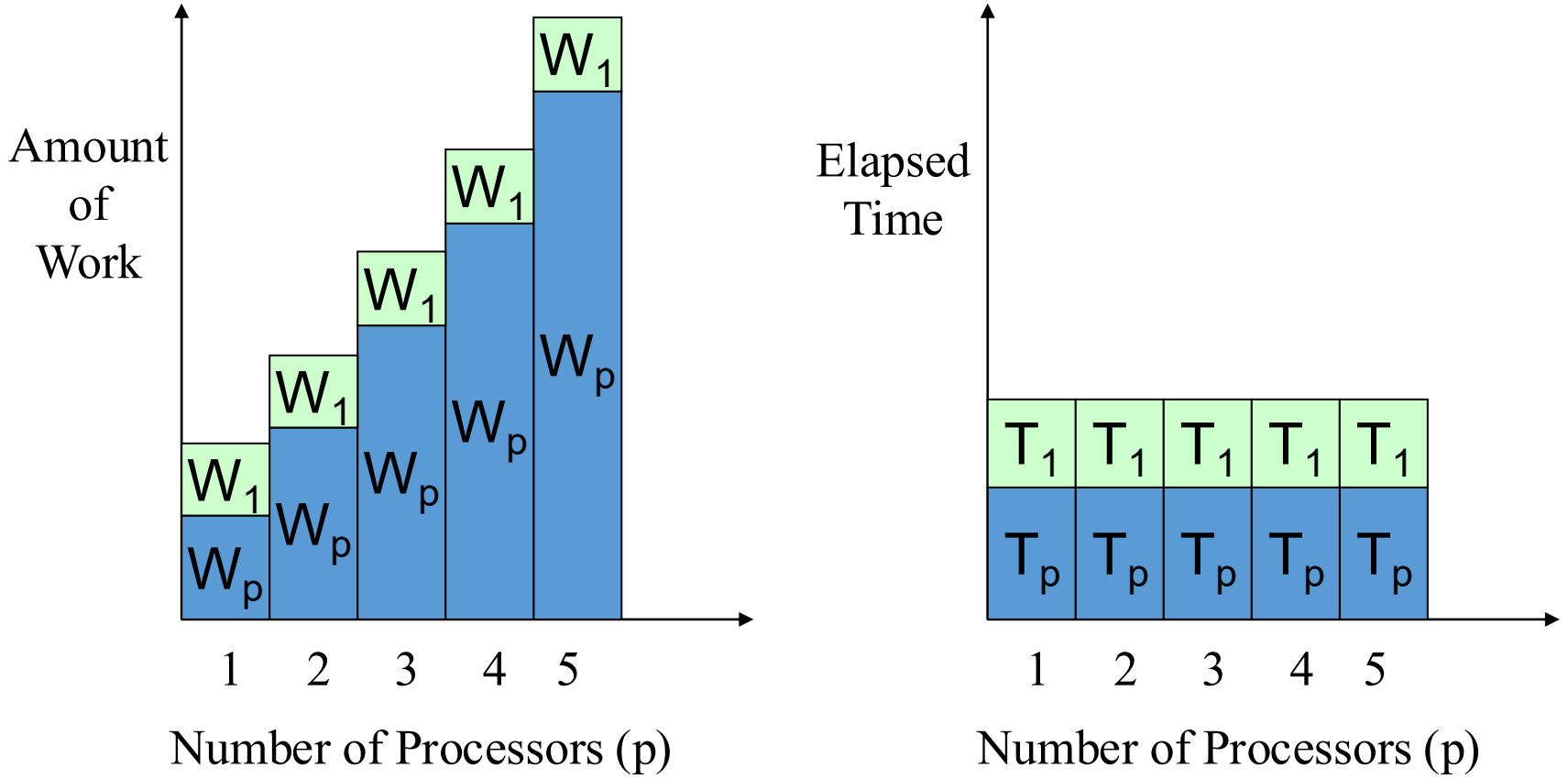
- **Strong scaling** - how the processing time varies with the number of processors for **a fixed total problem size**





SCALABILITY

- **Weak scaling** - how the processing time varies with the number of processors for **a fixed problem size per processing unit**



CLASSIFICATION OF PERFORMANCE TOOLS

- There are many tools that can be classified by the implemented approach

Data collecting mechanism

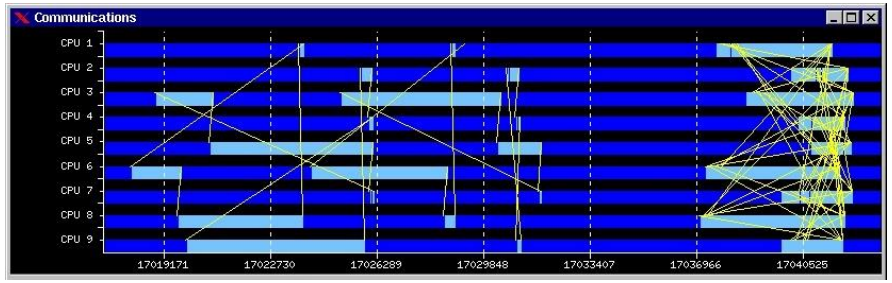
- **Sampling** - automatically collect data per time unit
- **Instrumentation** - manually/automatically add instructions to the source code to collect data - intrusive

Form of data presentation

- **Reports** - general overview of the whole application
- **Profiling** - accumulated characteristics of metrics
- **Tracing** - details about selected events - intrusive

Analysis of the collected data

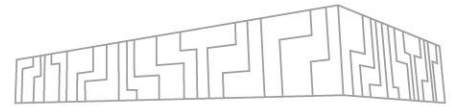
- **Online** - during the execution - rare
- **Post mortem** - after the execution



Example of a trace, source: tools.bsc.es

Modeling - simulate state, ideal network, HW failure, etc.

PERFORMANCE TOOLS - CPU



- Single-node/parallel, architecture, language, programming model, focus (instrumentation, correctness checking, etc.)

Proprietary tools – licenses usually available on clusters

- Linaro (~~ARM~~ (~~Allinea~~)) Performance Report
- Linaro (~~ARM~~ (~~Allinea~~)) MAP
- Intel Application Performance Snapshot
- Intel Vtune
- AMD µProf
- Vampir

Open-source tools (VI-HPS)

- BSC tools (Extrae/Paraver)
- JSC tools (Score-P/Scalasca/Cube)
- MAQAO
- https://www.vi-hps.org/tools/tools.html (guide)

GPU PROFILING – TOOLS

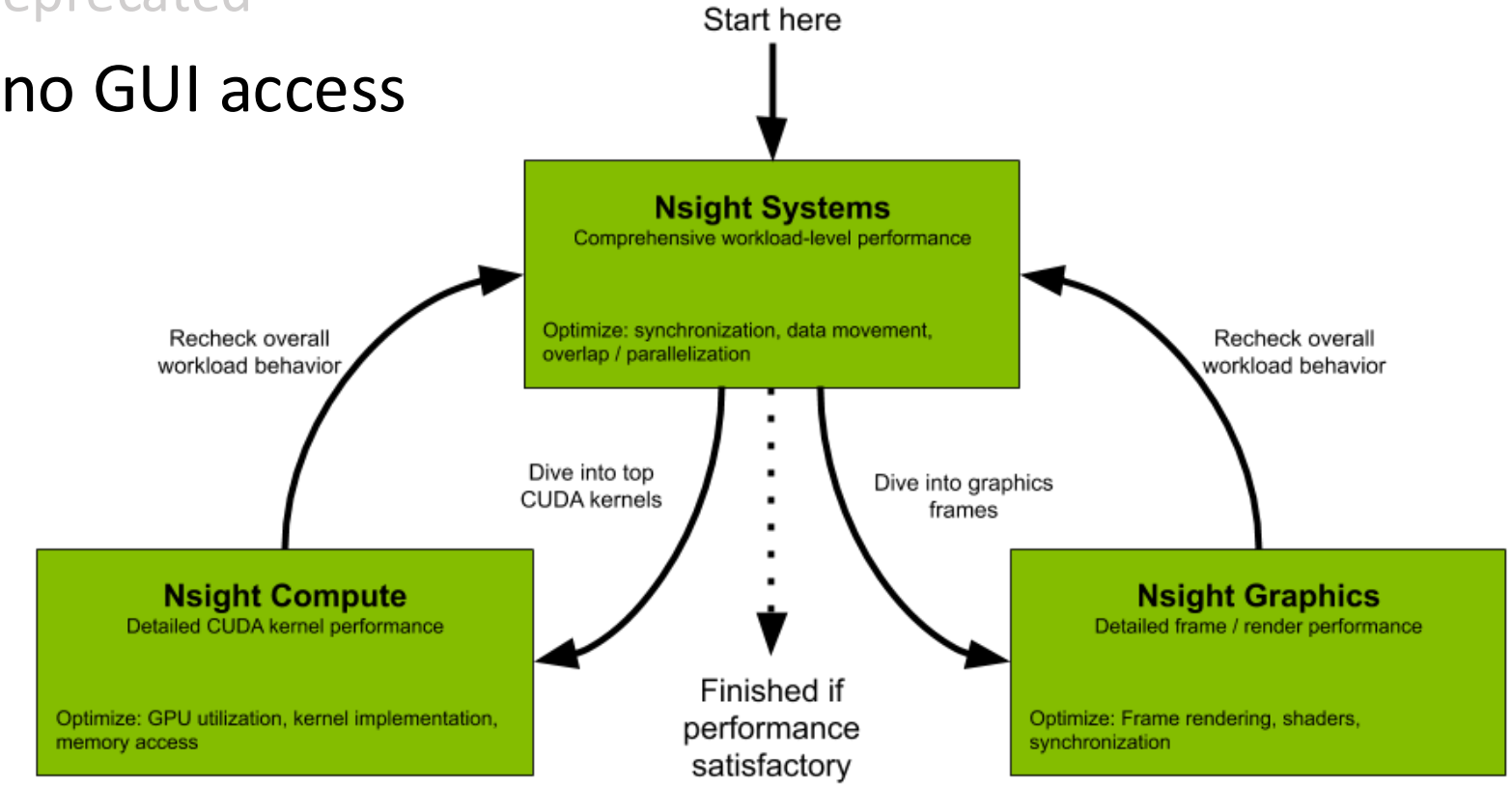
GUI tools

- NVIDIA Nsight Systems – **system-level** profiling
- NVIDIA Nsight Compute – **CUDA kernel-level** profiling
- NVIDIA Visual Profiler - deprecated

Command-line tools - for no GUI access

(e.g. in batch jobs)

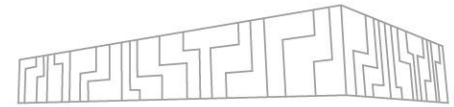
- NVIDIA nsys
- NVIDIA ncu
- AMD ROC-profiler
  - analogous to nsys
  - Chrome for visualization
- NVIDIA nvprof
  - deprecated



Nsight tools, source: nvidia.com

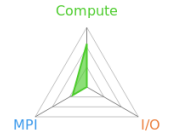
LINARO PERFORMANCE REPORTS

- Global high-level overview of the application
- No source code or recompilation required
- Run: **perf-report** srun -n <#procs> <app>
- Auto-generated text and HTML output
- Report summary (Compute, MPI, Input/Output)
- CPU, MPI, I/O, OpenMP, Memory, Energy, Accelerator breakdown sections
- Advanced configuration through command line flags possible



arm PERFORMANCE REPORTS

Command:	mpirun -np 8 examples/wave_openmp 60
Resources:	1 node (8 physical, 8 logical cores per node)
Memory:	15 GiB per node
Tasks:	8 processes, OMP_NUM_THREADS was 2
Machine:	mars
Start time:	Tue Nov 7 2017 15:35:50 (UTC)
Total time:	61 seconds (about 1 minutes)
Full path:	/scratch/user/reports/examples



Summary: wave_openmp is Compute-bound in this configuration

Compute	72.6%	Time spent running application code. High values are usually good. This is **high**; check the CPU performance section for advice
MPI	27.4%	Time spent in MPI calls. High values are usually bad. This is **low**; this code may benefit from a higher process count
I/O	0.0%	Time spent in filesystem I/O. High values are usually bad. This is **negligible**; there's no need to investigate I/O performance

This application run was Compute-bound. A breakdown of this time and advice for investigating further is in the CPU section below.

As little time is spent in MPI calls, this code may also benefit from running at larger scales.

CPU

A breakdown of the 72.6% CPU time:

Single-core code	8.2%
OpenMP regions	91.8%
Scalar numeric ops	5.1%
Vector numeric ops	0.0%
Memory accesses	56.9%

The per-core performance is memory-bound. Use a profiler to identify time-consuming loops and check their cache performance.

No time is spent in vectorized instructions. Check the compiler's vectorization advice to see why key loops could not be vectorized.

MPI

A breakdown of the 27.4% MPI time:

Time in collective calls	1.2%
Time in point-to-point calls	98.8%
Effective process collective rate	19.5 kB/s
Effective process point-to-point rate	305 kB/s

Most of the time is spent in point-to-point calls with a very low transfer rate. This suggests load imbalance is causing synchronization overhead; use an MPI profiler to investigate.

I/O

A breakdown of the 0.0% I/O time:

Time in reads	0.0%
Time in writes	0.0%
Effective process read rate	0.00 bytes/s
Effective process write rate	0.00 bytes/s

No time is spent in I/O operations. There's nothing to optimize here!

OpenMP

A breakdown of the 91.8% time in OpenMP regions:

Computation	9.9%
Synchronization	90.1%
Physical core utilization	100.0%
System load	167.0%

Significant time is spent synchronizing threads in parallel regions. Check the affected regions with a profiler.

The system load is high. Ensure background system processes are not running.

Memory

Per-process memory usage may also affect scaling:

Mean process memory usage	38.6 MiB
Peak process memory usage	53.7 MiB
Peak node memory usage	17.0%

The peak node memory usage is very low. Running with fewer MPI processes and more data on each process may be more efficient.

Energy

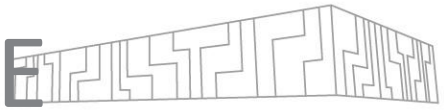
A breakdown of how energy was used:

CPU	not supported %
System	not supported %
Mean node power	not supported W
Peak node power	0.00 W

Energy metrics are not available on this system.

CPU metrics are not supported (no intel_rapl module)

LINARO PERFORMANCE REPORTS - EXAMPLE

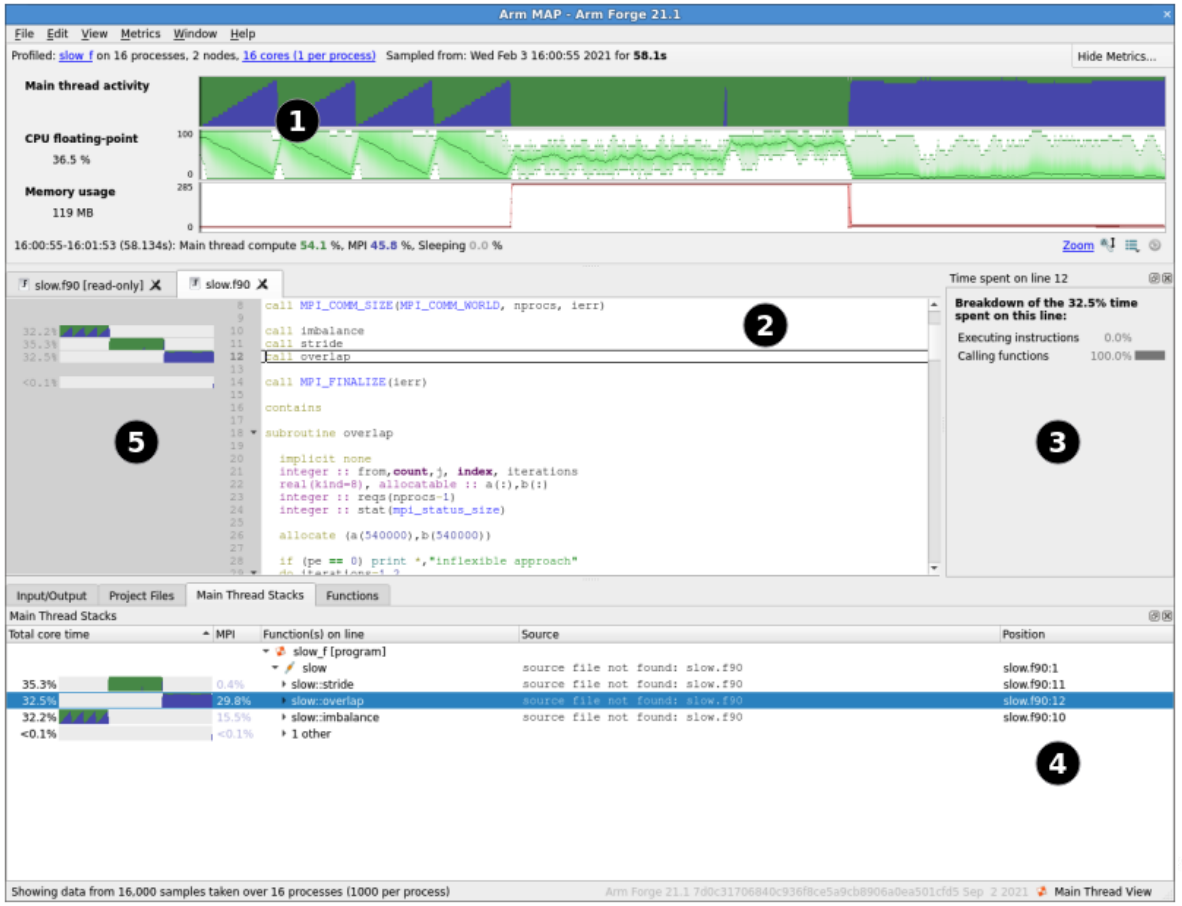


```
ml Forge/23.1.2 OpenMPI/5.0.8-GCC-14.3.0
ml show Forge
cp -r /apps/all/Forge/23.1.2/examples ~/forge_examples
cd ~/forge_examples
make -f wave.makefile
srun -n 16 ./wave_c 10

mkdir perf_reports && cd perf_reports
perf-report srun -n 16 ../wave_c 10
firefox wave_c_16p_1n_YYYY-MM-DD_hh-mm.html &          # on login node
make -f openmp.makefile
OMP_NUM_THREADS=8 perf-report srun -n 2 -c 8 ../wave_openmp 10
firefox wave_openmp_2p_1n_8t_YYYY-MM-DD_hh-mm.html &
```

LINARO MAP

- Low overhead sampling profiler for localisation of bottlenecks
- No recompilation required, only debugging symbols are useful (-g)

1. Metrics view (CPU, MPI, I/O, memory, vectorization)
2. Source code viewer
3. Selected lines view
4. Output, files, callpaths
5. Sparkline charts

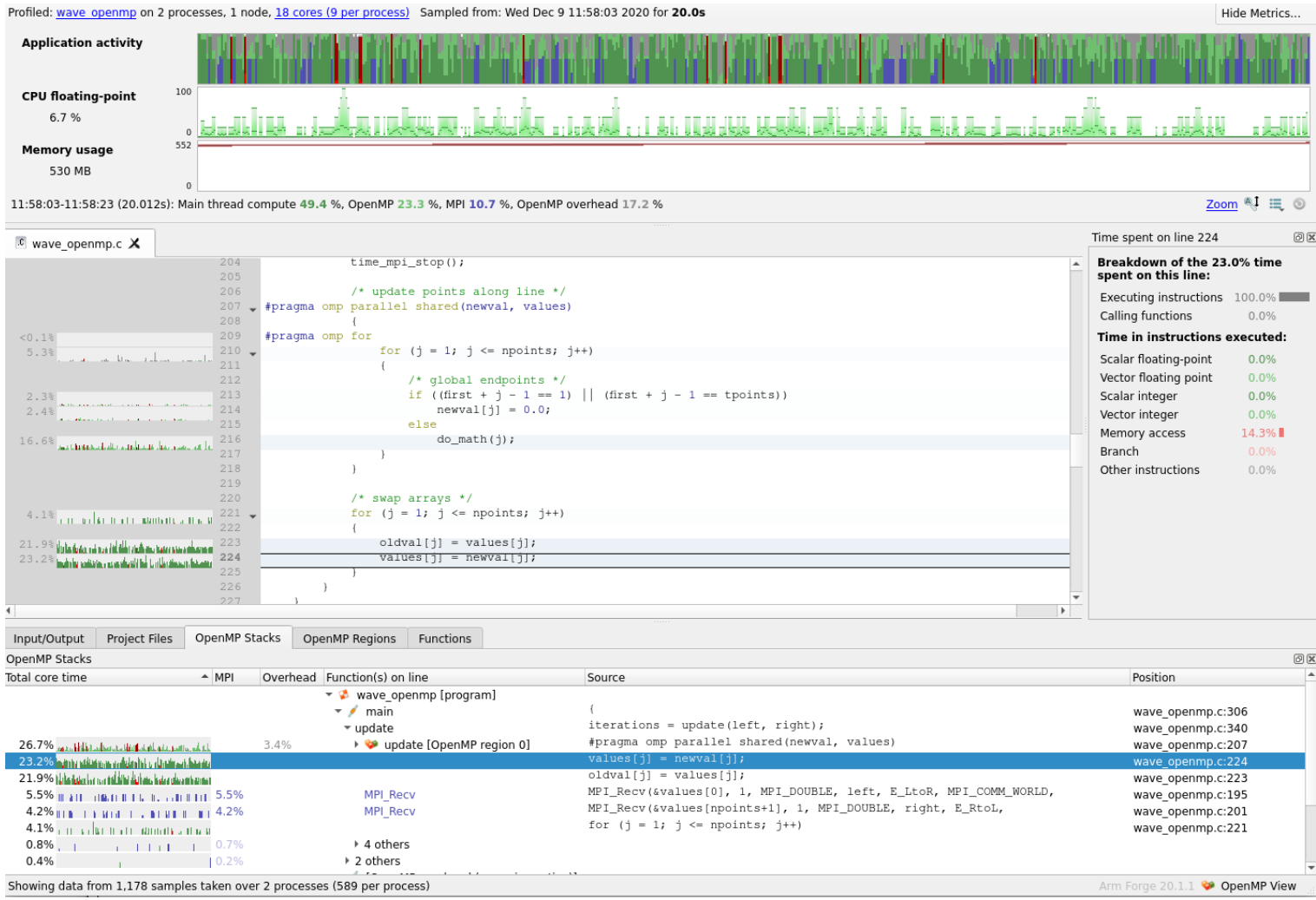



```
map
map srun -n <#procs> <app> [args]
map --profile srun -n <#procs> …
map <profile.map>
perf-report <profile.map>
```

LINARO MAP

- All charts are timelines
  - Horizontal axis time
- Vertical axis are processes
- Useful code is green
- MPI is blue
- Breakout recalculated

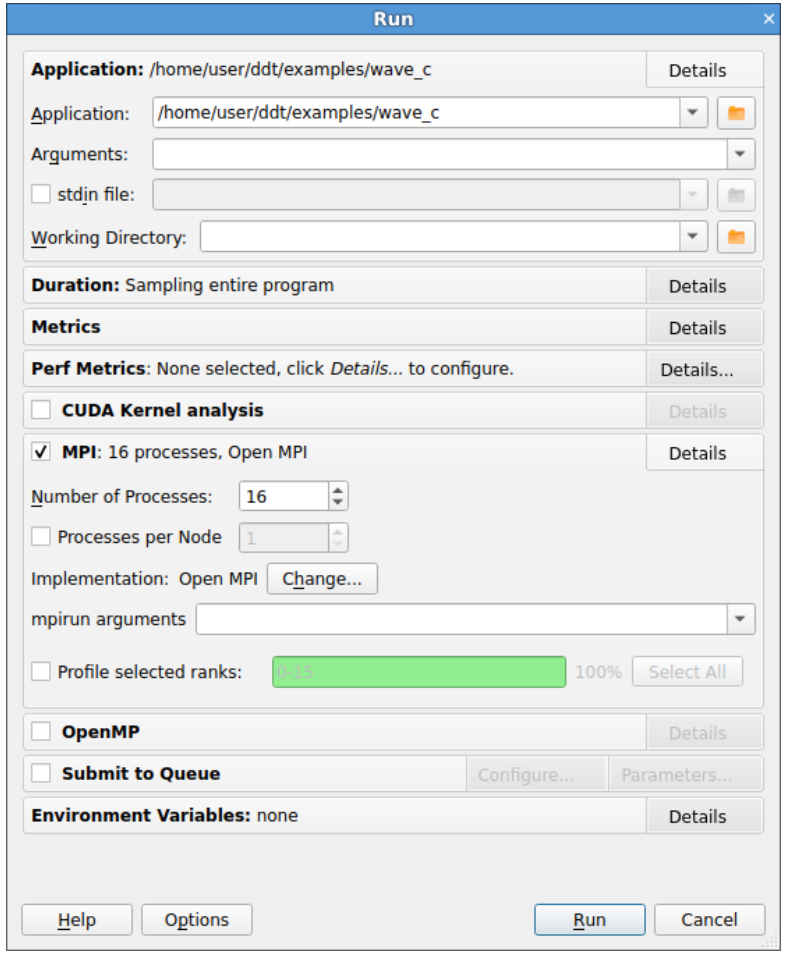
  when zooming
- Multiple presets available
  - CPU
  - MPI
  - I/O
  - memory
  - …



LINARO MAP - EXAMPLE

```
ml Forge/23.1.2 OpenMPI/4.1.6-GCC-12.2.0-CUDA-12.4.0
mkdir ~/forge_examples/map && cd ~/forge_examples/map
OMP_NUM_THREADS=8 map srun -n 2 -c 8 ../wave_openmp 10
```

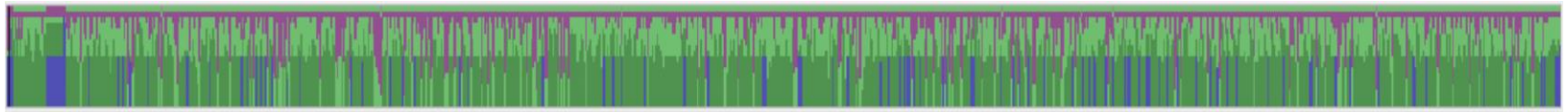
- Optionally limit duration
- Optionally adapt metrics
- Click Run
- Use the User guide!



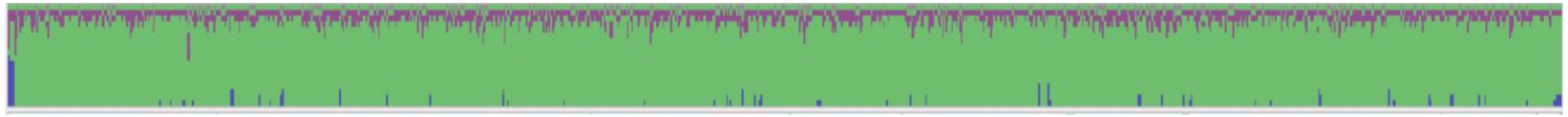
LINARO MAP - EXAMPLE

- A large section of blue means all the processes in MPI calls - try to reduce these. Triangular shape indicates load imbalance.

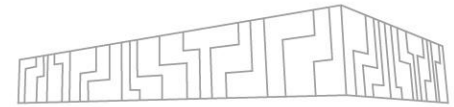
- A large section of dark green means all the processes in single-threaded computations - try to avoid.



- A large sections of light green - OpenMP regions being effectively used across all processes simultaneously.



NVIDIA NSIGHT SYSTEMS



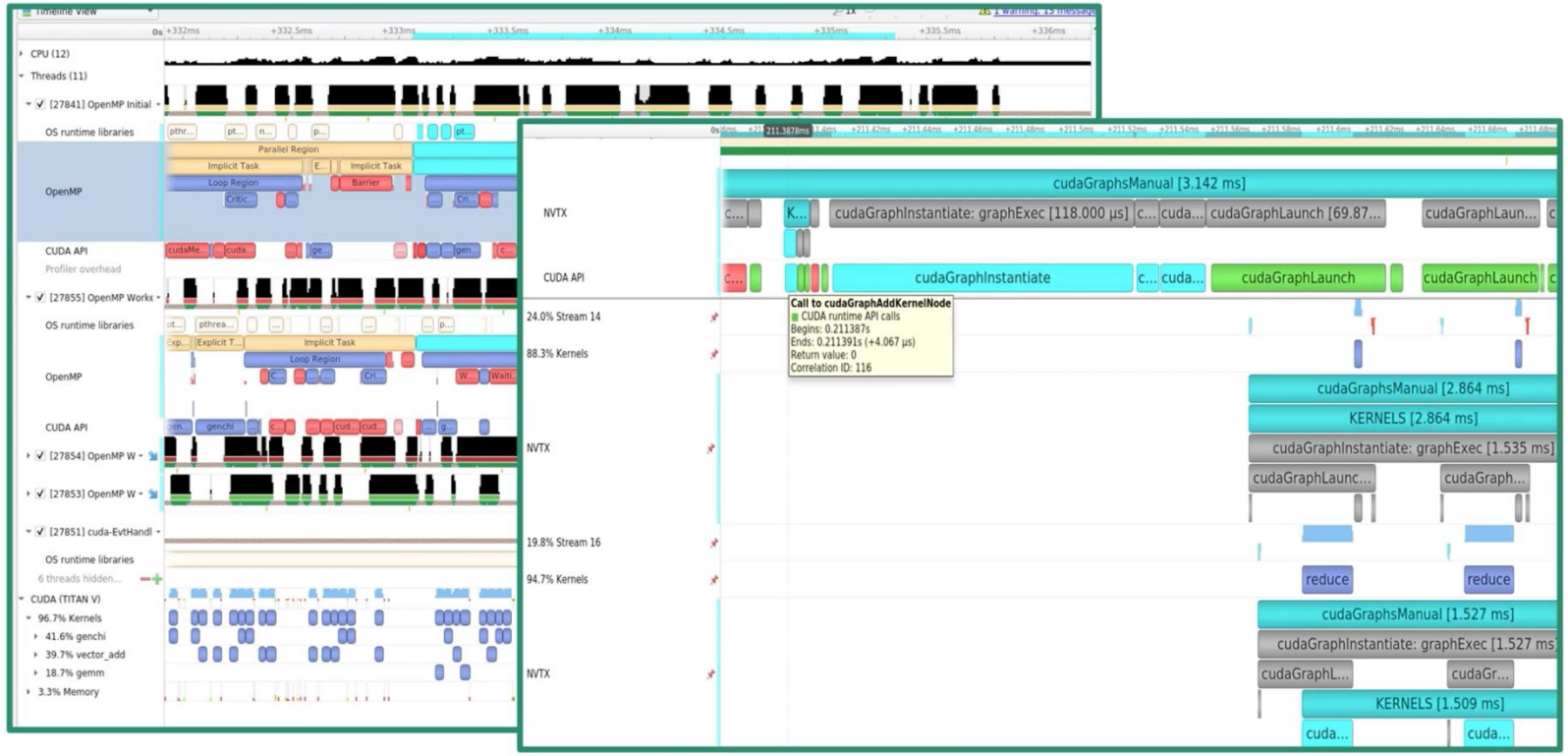
Scalable system-wide performance analysis tool

- Low-overhead multi-node, multi-GPU profiling
- Visualize millions of events on a very fast GUI timeline
- Assess on timeline to narrow down frames/areas of the app to focus
- Locate optimization opportunities, CPU/GPU bottlenecks
- or gaps of unused CPU and GPU time - idle
- Balance your workload across multiple CPUs and GPUs
- Expert system GPU utilization analysis
- Detailed information, documentation, free download
  https://developer.nvidia.com/nsight-systems

NVIDIA NSIGHT SYSTEMS

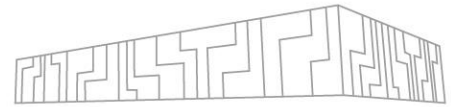
Multi-level information

- CPU cores utilization
- MPI calls
- OpenMP, Pthreas
- OS runtime calls
- NVTX
- CUDA API calls
- HtD / DtH data transfers
- CUDA kernels / streams
- OpenACC
- CUDA libraries (cuBLAS, …), GPU HW metrics, UCX, NIC, …



NVIDIA NSIGHT SYSTEMS

PROFILING WITH NSIGHT SYSTEMS



GUI profiling and analysis

```
| ml CUDA/12.4.0 Qt5
| nsys-ui # On the allocated GPU compute node
```

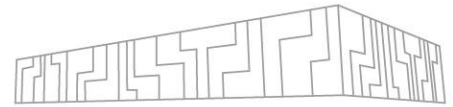
- File -> New Project
- Select target for profiling… -> acnXX.karolina.it4i.cz (allocated GPU node)
- Enter binary (**absolute path** with arguments if necessary)
- Select tracing modules (CPU, OS, CUDA, GPU, NVTX,…)
- Start

Cmd line profiling + GUI analysis

```
| nsys profile -t cuda,osrt --stats=true --gpu-metrics-device=0
  -o profile_name ./program
| nsys-ui # On login node
```

- File -> Open -> Select profile_name.nsys-rep

NVIDIA NSIGHT SYSTEMS - EXAMPLE



```
git clone https://code.it4i.cz/training/intro2hpc.git
ml CUDA/13.0.0
cd intro2hpc/5_gpu_accelerators/handson
vim vector_add.solution.cu # int count = 123456789;
nvcc -g -O2 -gencode arch=compute_80,code=sm_80
vector_add.solution.cu -o vector_add
./vector_add
```

Barbora sm_70
Karolina sm_80

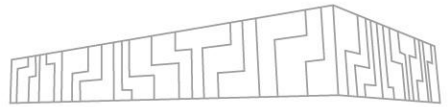
- Perform profiling of vector_add example:

```
nsys profile -t cuda,osrt --stats=true --gpu-metrics-
device=0 -o vector_add ./vector_add
```

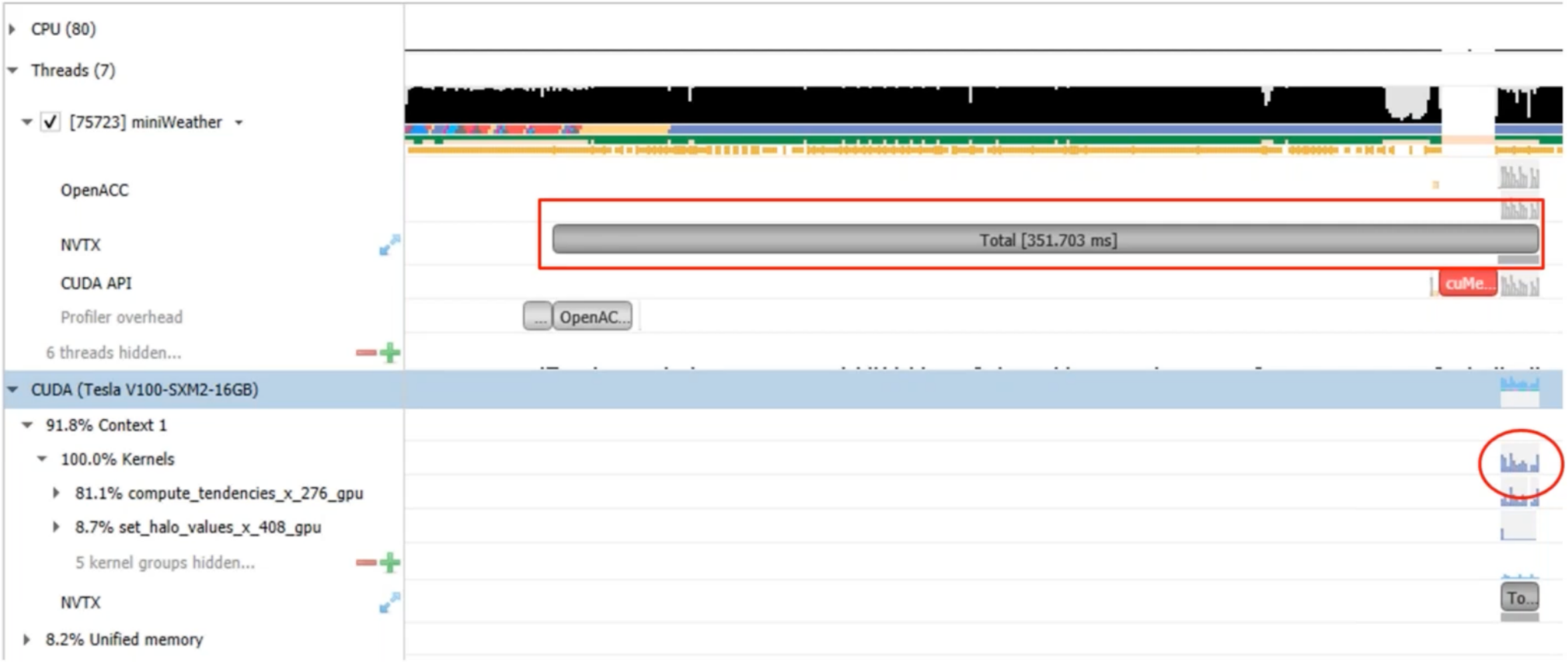
- An extra CUDA examples:

```
git clone https://github.com/NVIDIA/cuda-samples.git
```

ANALYSIS WITH NSIGHT SYSTEMS

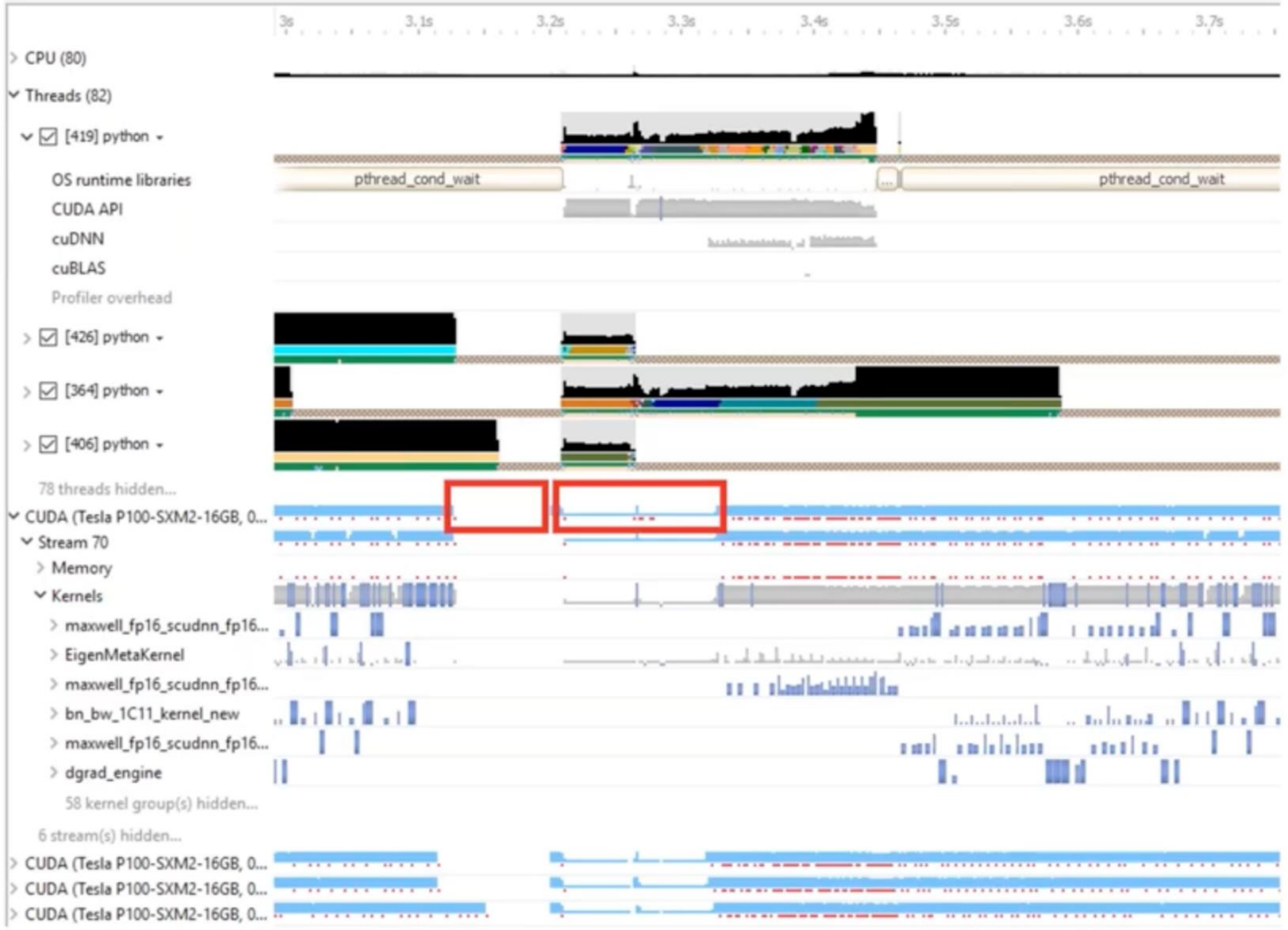


Only small portion of application accelerated (for real-world apps)



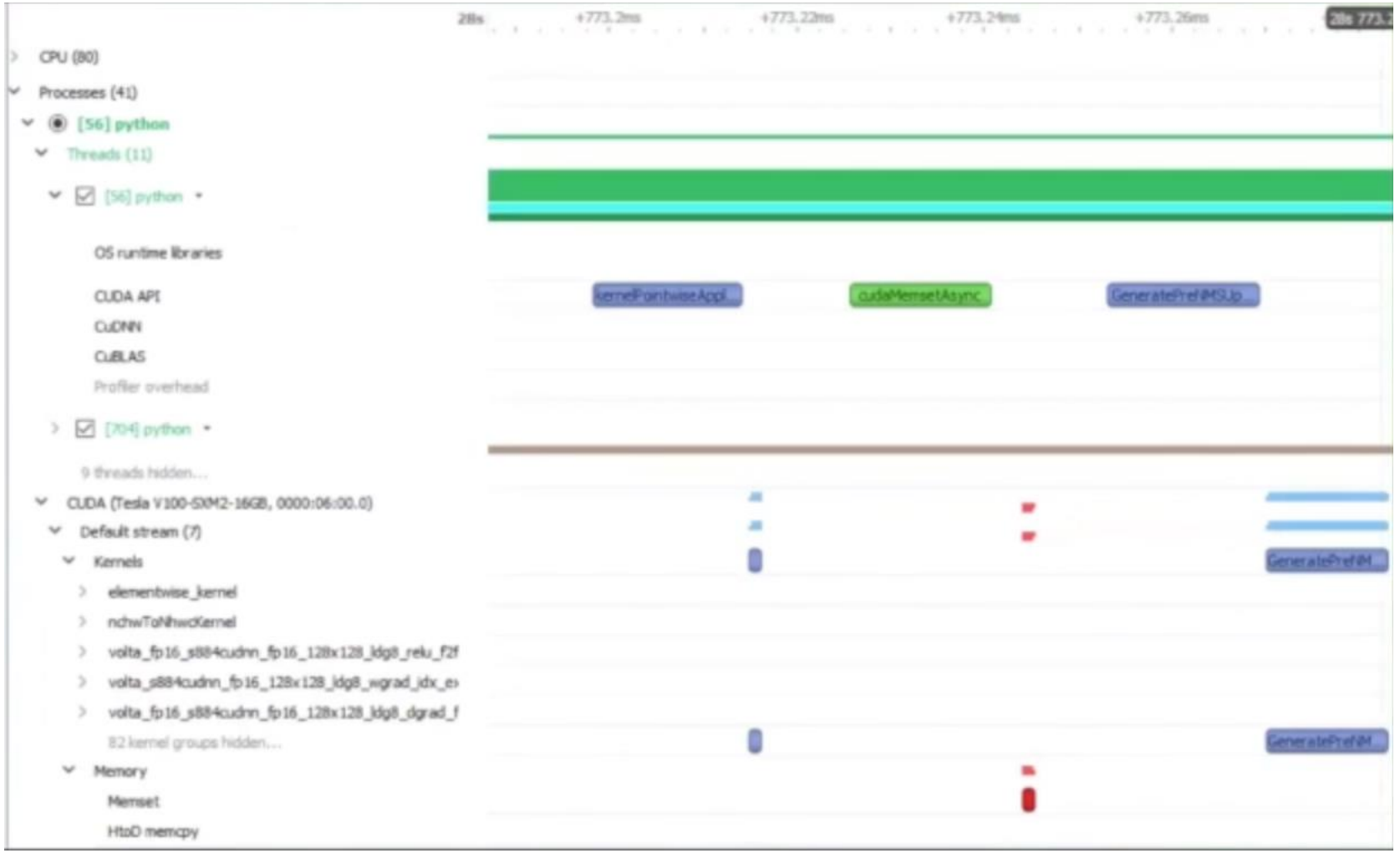
ANALYSIS WITH NSIGHT SYSTEMS

GPU idle/low utilization of detailed zoom (e.g. because of Pthread creation)



ANALYSIS WITH NSIGHT SYSTEMS

Fusion opportunities: CPU launch cost + small GPU work size -> GPU idle

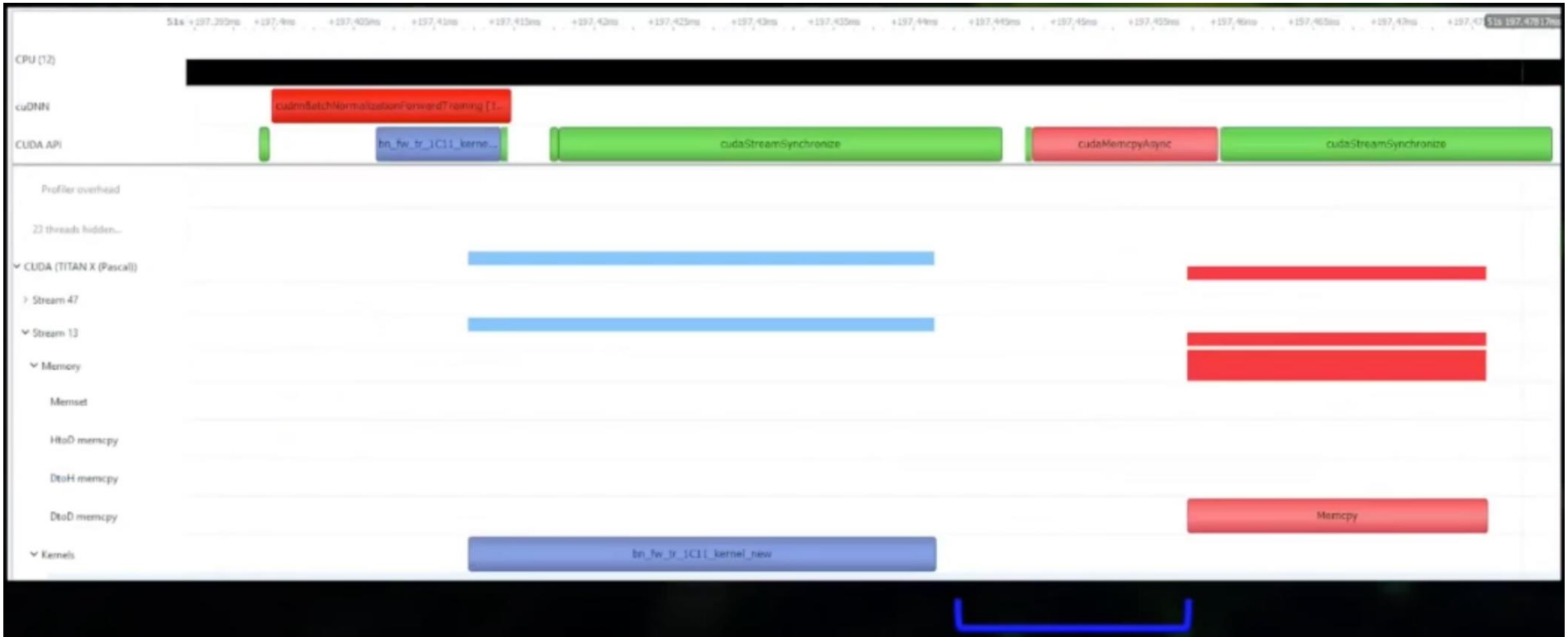


ANALYSIS WITH NSIGHT SYSTEMS

cudaMemcpyAsync behaving synchronously – DtH pageable memory -> Mitigate with pinned memory

ANALYSIS WITH NSIGHT SYSTEMS

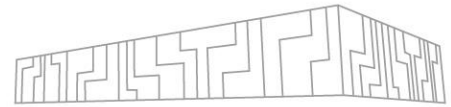
GPU idle caused by stream synchronization



POP COE

- An EuroHPC **Centre of Excellence** (CoE)
  - On **Performance Optimisation and Productivity**
  - Promoting **best practices in parallel programming**
- Providing **FREE Services** for EU **academic AND industrial codes in all domains!**
  - **Performance Assessment**: initial analysis to identify performance issues and recommend approaches to address them
  - **Proof-of-concept**: explore the potential benefit of proposed optimisations by applying them to selected regions of the applications
  - **Correctness-check**: evaluate the correctness of hybrid MPI + OpenMP applications
  - **Energy-efficiency study**: investigate improvements of energy consumption or efficiency
  - **Advisory study**: ongoing consultancy for customers that choose to implement proposed optimisations on their own

USEFUL LINKS



VI-HPS – Association of institutions developing tools and providing training

- Overview of the tools with a description: https://www.vi-hps.org/cms/upload/material/general/ToolsGuide.pdf

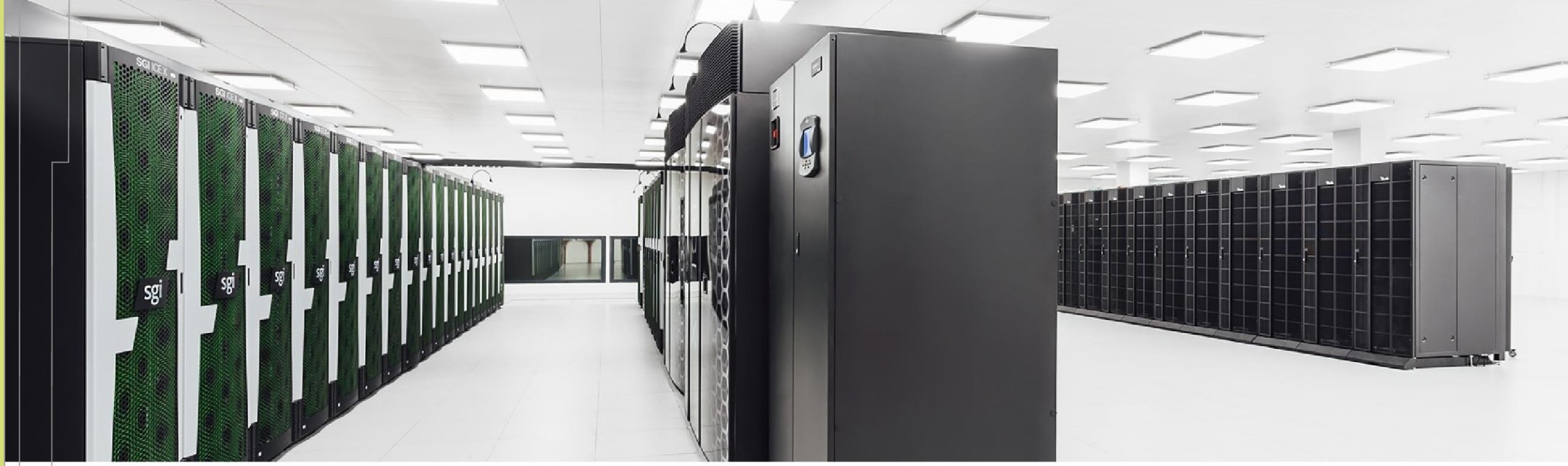
Nvidia tools for GPUs: Nsight Systems and Nsight Compute

Intel performance tools: VTune and Advisor

Database of code patterns and best practices developed in POP: co-design

Docs + further reading:

- https://docs.linaroforge.com/23.1.2/html/forge/performance_reports/index.html
- https://docs.linaroforge.com/23.1.2/html/forge/map/index.html
- https://software.intel.com/content/www/us/en/develop/articles/intel-advisor-roofline.html



Radim Vavřík
radim.vavrik@vsb.cz

IT4Innovations National Supercomputing Center
VSB – Technical University of Ostrava
Studentská 6231/1B
708 00 Ostrava-Poruba, Czech Republic
www.it4i.cz

VSB TECHNICAL UNIVERSITY OF OSTRAVA | IT4INNOVATIONS NATIONAL SUPERCOMPUTING CENTER

EUROPEAN UNION
European Structural and Investment Funds
Operational Programme Research, Development and Education

MINISTRY OF EDUCATION, YOUTH AND SPORTS