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 examples/tools can be accessed and reproduced **anytime** at IT4I clusters

- Please, setup your **VNC session** on a Karolina login node (strongly recommended!) or log in with X-Window system enabled
  - VNC session usually offer better UX For GUI tools than X11
  - https://docs.it4i.cz/general/accessing-the-clusters/graphical-user-interface/vnc/
  - RealVNC Viewer https://www.realvnc.com/en/connect/download/viewer/

- 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 the 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

Apply the knowledge!

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

BASIC TOOLS

Useful to get familiar with the machine

| `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

| `cpuinfo` # Intel MPI utility

| `cat /proc/meminfo`

- MemTotal: 196510848 kB -> 187 GB

Use HTOP tool for interactive jobs

| `htop –d 5` # delay 0.5s

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

```
  1  [|||||||||||||100.0%]   10 [|||||||||||||100.0%]   19 [|||||||||||||100.0%]   28 [|||||||||||||100.0%]
  2  [|||||||||||||100.0%]   11 [|||||||||||||100.0%]   20 [|||||||||||||100.0%]   29 [|||||||||||||100.0%]
  3  [|||||||||||||100.0%]   12 [|||||||||||||100.0%]   21 [|||||||||||||100.0%]   30 [|||||||||||||100.0%]
  4  [|||||||||||||100.0%]   13 [|||||||||||||100.0%]   22 [|||||||||||||100.0%]   31 [|||||||||||||100.0%]
  5  [|||||||||||||100.0%]   14 [|||||||||||||100.0%]   23 [|||||||||||||100.0%]   32 [|||||||||||||100.0%]
  6  [|||||||||||||100.0%]   15 [|||||||||||||100.0%]   24 [|||||||||||||100.0%]   33 [|||||||||||||100.0%]
  7  [|||||||||||||100.0%]   16 [|||||||||||||100.0%]   25 [|||||||||||||100.0%]   34 [|||||||||||||100.0%]
  8  [|||||||||||||100.0%]   17 [|||||||||||||100.0%]   26 [|||||||||||||100.0%]   35 [|||||||||||||100.0%]
  9  [|||||||||||||100.0%]   18 [|||||||||||||100.0%]   27 [|||||||||||||100.0%]   36 [|||||||||||||100.0%]
  Mem[|||||                             13.8G/187G]   Tasks: 79, 346 thr; 36 running
  Swp[                                      0K/0K]   Load average: 23.62 6.93 3.32
                                                     Uptime: 15 days, 12:06:32

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
11171 vav0038    35  15  296M 90236  7232 R 99.5  0.0  1:02.72 ../examples/wave_c 100
11203 vav0038    35  15  298M 90240  7244 R 99.5  0.0  1:03.07 ../examples/wave_c 100
11212 vav0038    35  15  322M 92280  7324 R 99.5  0.0  1:03.04 ../examples/wave_c 100
11162 vav0038    35  15  300M 90220  7272 R 99.5  0.0  1:03.10 ../examples/wave_c 100
11188 vav0038    35  15  323M 90236  7328 R 99.5  0.0  1:03.05 ../examples/wave_c 100
11207 vav0038    35  15  311M 92272  7296 R 99.5  0.0  1:03.04 ../examples/wave_c 100
11164 vav0038    35  15  326M 90232  7340 R 99.5  0.0  1:03.09 ../examples/wave_c 100
11195 vav0038    35  15  298M 90232  7232 R 99.5  0.0  1:03.09 ../examples/wave_c 100
11158 vav0038    35  15  319M 92284  7304 R 99.5  0.0  1:03.07 ../examples/wave_c 100
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```

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

OPTIMISATION TIPS

- Do not optimise your code prematurely!
- Focus on main computational time-consuming phases (hotspots), omit preprocessing/postprocessing phases
- 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

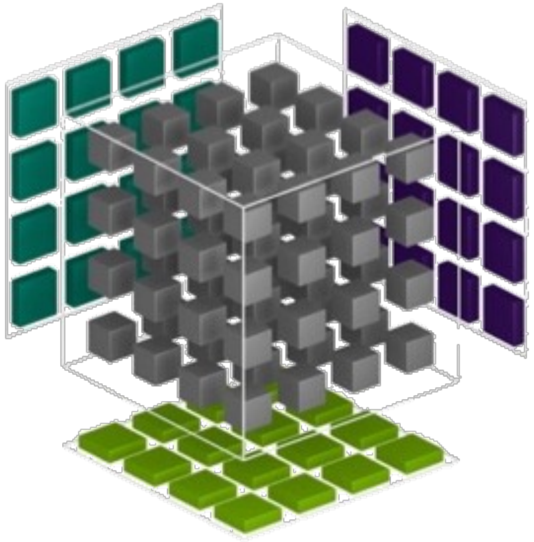
- BLAS (MKL, OpenBLAS, ATLAS, cuBLAS, ...)
- LAPACK (MKL, OpenBLAS, ATLAS, cuSolver, ...)
- FFT (MKL, cuFFT, ...)
- ...

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 vs weak scaling

Others: portability, programming ability, etc.

PEAK PERFORMANCE EXAMPLE

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

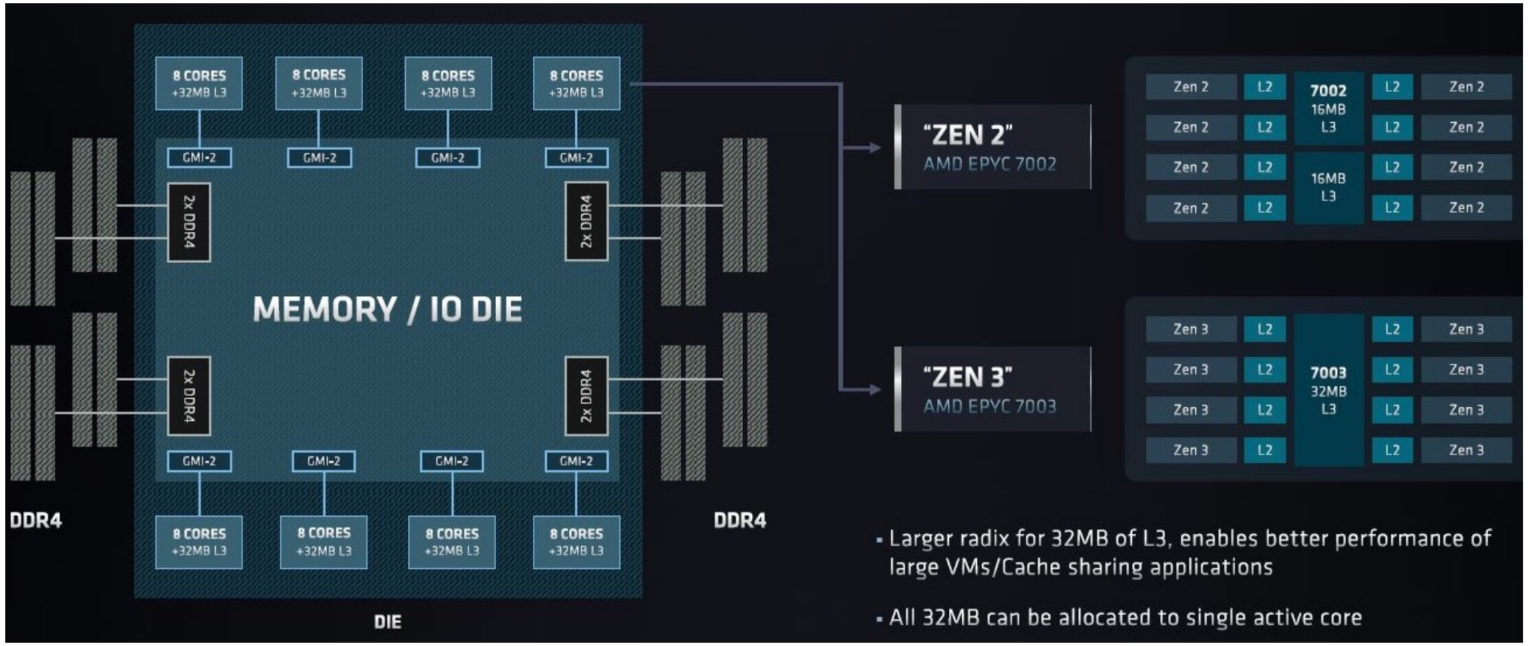
Processor speed:

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)



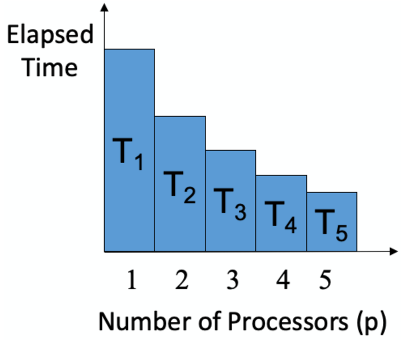
SPEEDUP EXAMPLE

- Assume the perfect speedup $\mathbf{S_P = P}$, perfect efficiency $\mathbf{E_P = 1}$ (100%)

Strong scaling

$S_P = T_1 / T_P$ $S_{16} = T_1 / T_{16} = 32 / 2 = 16$

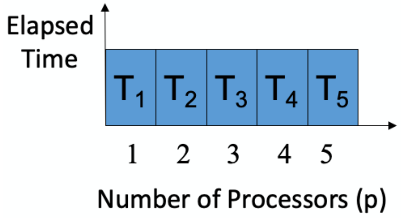
$E_P = S_P / P$ $E_{16} = S_{16} / 16 = 16 / 16 = 1$



Weak scaling

$S_P = T_1 / T_P$ $S_{16} = T_1 / T_{16} = 32 / 32 = 1$

$E_P = S_P / P$ $E_{16} = S_{16} / 16 = 1 / 16 = 0.0625$



- Perfect E = 6.25 % ? Not very intuitive, alternative:

$E_P = T_1 / T_P$ $E_{16} = T_1 / T_{16} = 32 / 32 = 1$

- "Perfect speedup" $\mathbf{S_P = 1}$

$S_P = 1 / E_P = T_P / T_1$ $S_{16} = T_{16} / T_1 = 32 / 32 = 1$

CLASSIFICATION OF PERFORMANCE TOOLS

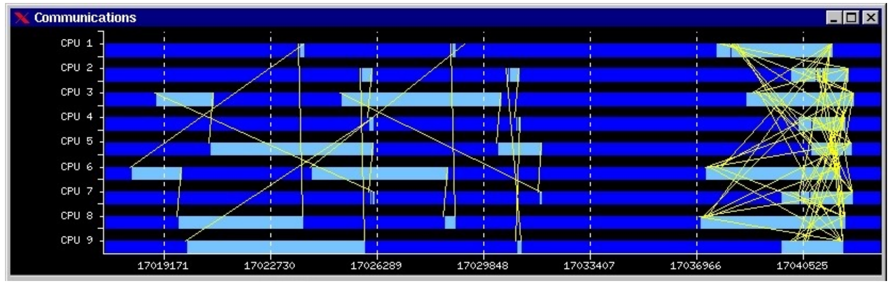
- There are many tools that can be classified by the implemented approach

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



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

Analysis of the collected data

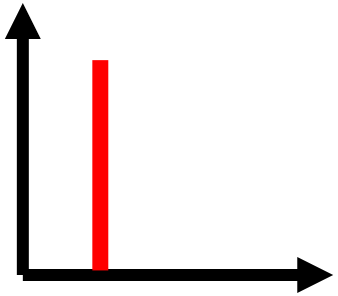
- Online - during the execution - rare
- Post mortem - after the execution

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

TYPES OF PROFILES

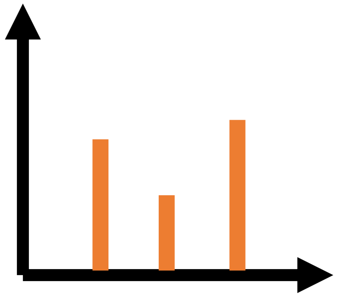
Hotspot

- One function corresponds to more 80% of the runtime
- Large speed-up potential
- Best optimisation scenario



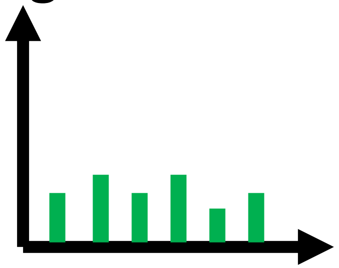
Spike

- The application spends most of the time in a few functions
- Speed-up potential depends on the aggregated time
- Variable optimisation time



Flat

- Runtime split evenly among many functions, each one with a very small runtime
- Little speed-up potential without algorithmic changes
- Worst optimisation scenario



PERFORMANCE TOOLS - CPU

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

Proprietary tools – licenses usually available on clusters

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

Open-source tools (VI-HPS)

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

PERFORMANCE TOOLS – GPU

GUI tools

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

Command-line tools – useful if you cannot use GUI (e.g. batch job)

- NVIDIA nvprof - deprecated
- NVIDIA nsys
- AMD ROC-profiler – analogous to nvprof (Chrome for visualization)

GET READY

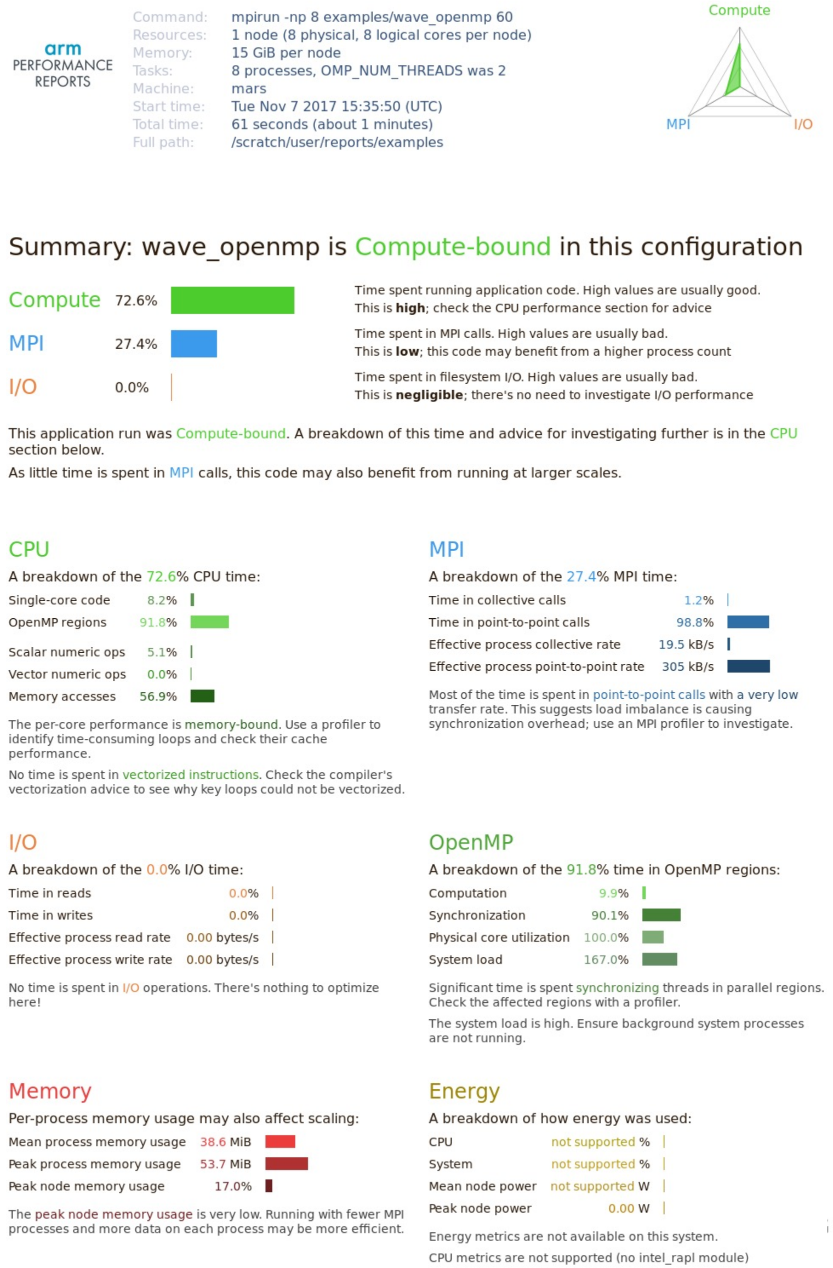
Submit an interactive job

```
qsub -q R1221596 -l select=1:mpiprocs=16 -IX     # use -q qnvidia later
```

```
qsub -q R1221596 -l select=1:mpiprocs=32:ngpus=2 -IX # if there are enough GPUs
```

ARM PERFORMANCE REPORTS

- Global high-level overview of the application
- No source code or recompilation required
- Run: **perf-report** mpirun -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 - EXAMPLE

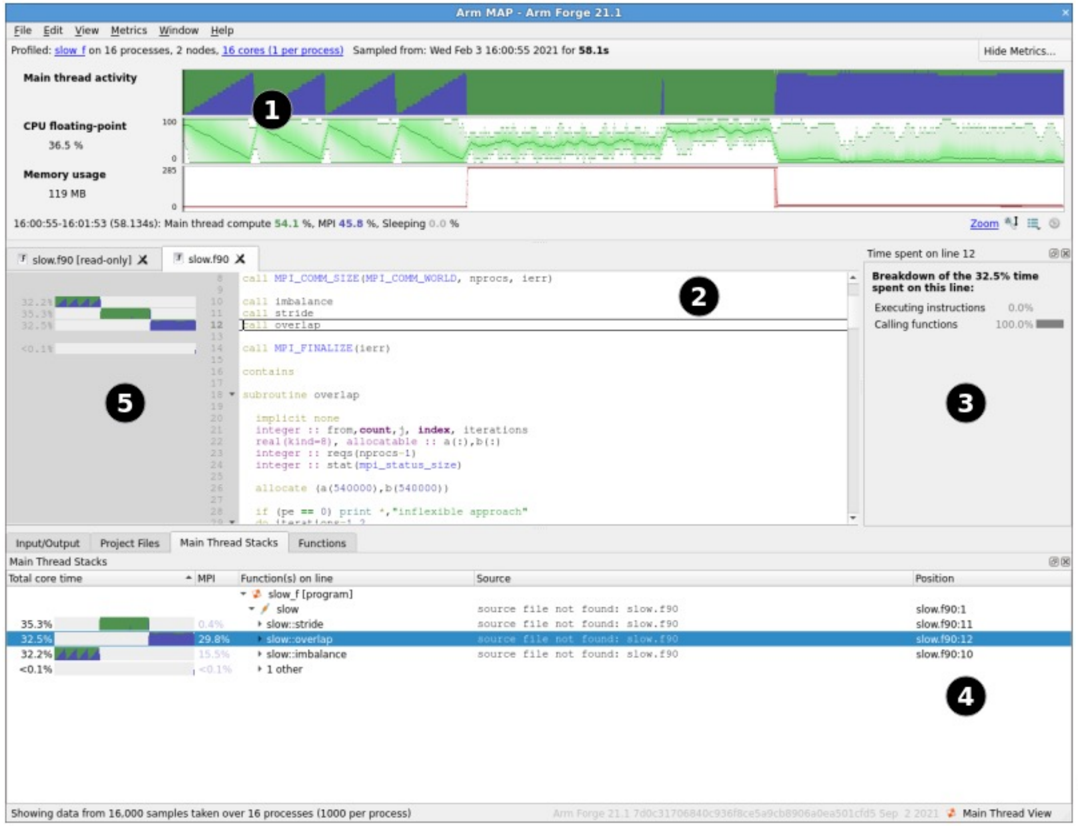
```
ml Forge/21.1.3 impi/2019.9.304-iccifort-2020.4.304
ml show Forge
cp -r /apps/all/Forge/21.1.3/examples ~/forge_examples
cd ~/forge_examples
make

mpirun -n 16 ./wave_c 10

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

ARM MAP

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

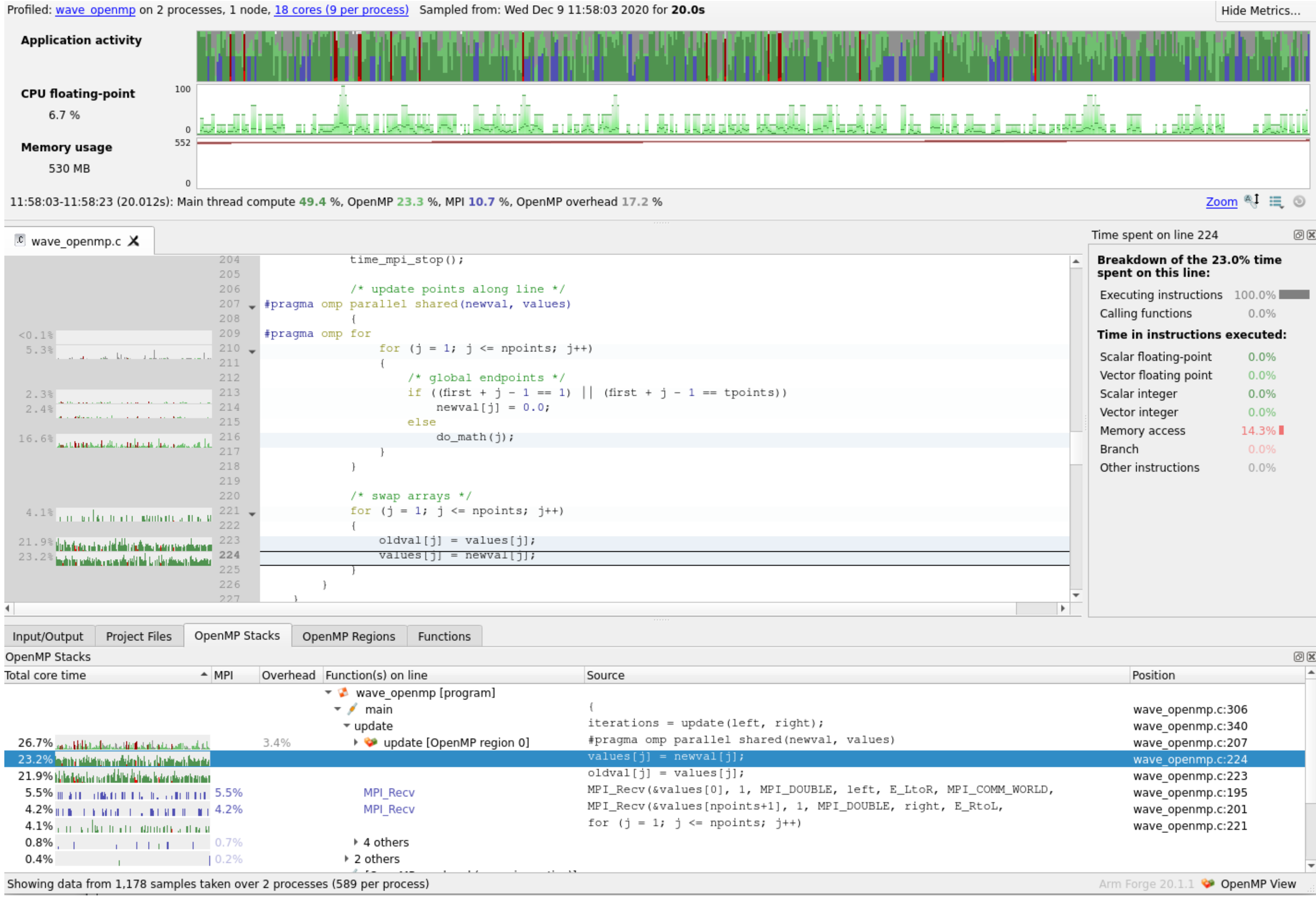
1. Metrics view (CPU, MPI, I/O, memory, vectorization)
2. Source code viewer
3. Selected lines view
4. Output, files, callpaths
5. Sparkline charts

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



ARM MAP

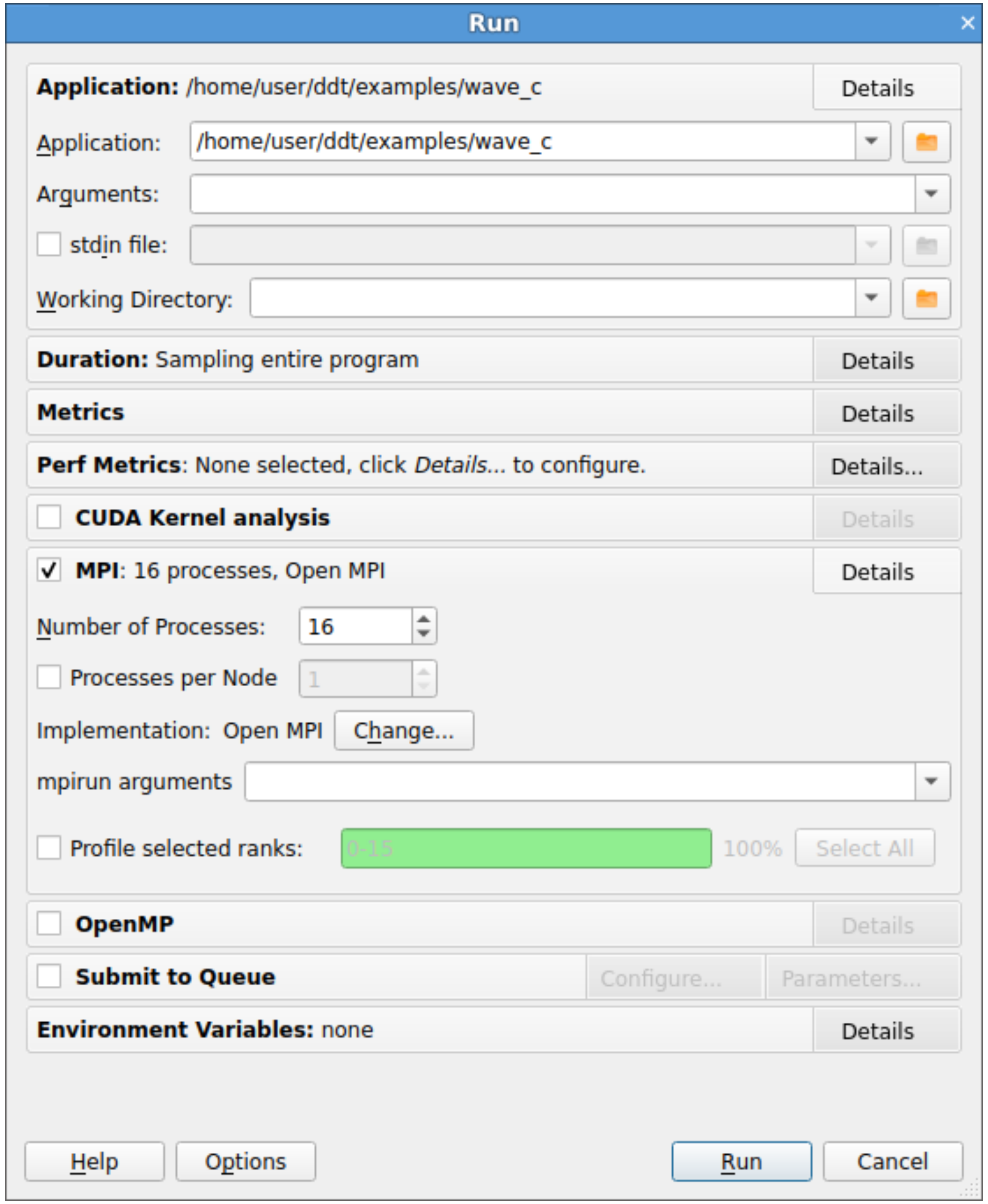
- All charts are timelines
  - Horizontal axis time
- Vertical axis are processes
- Useful code is green
- MPI is blue
- breakout recalculated when zooming
- Multiple presets available
  - CPU
  - MPI
  - I/O
  - memory
  - …



ARM MAP - EXAMPLE

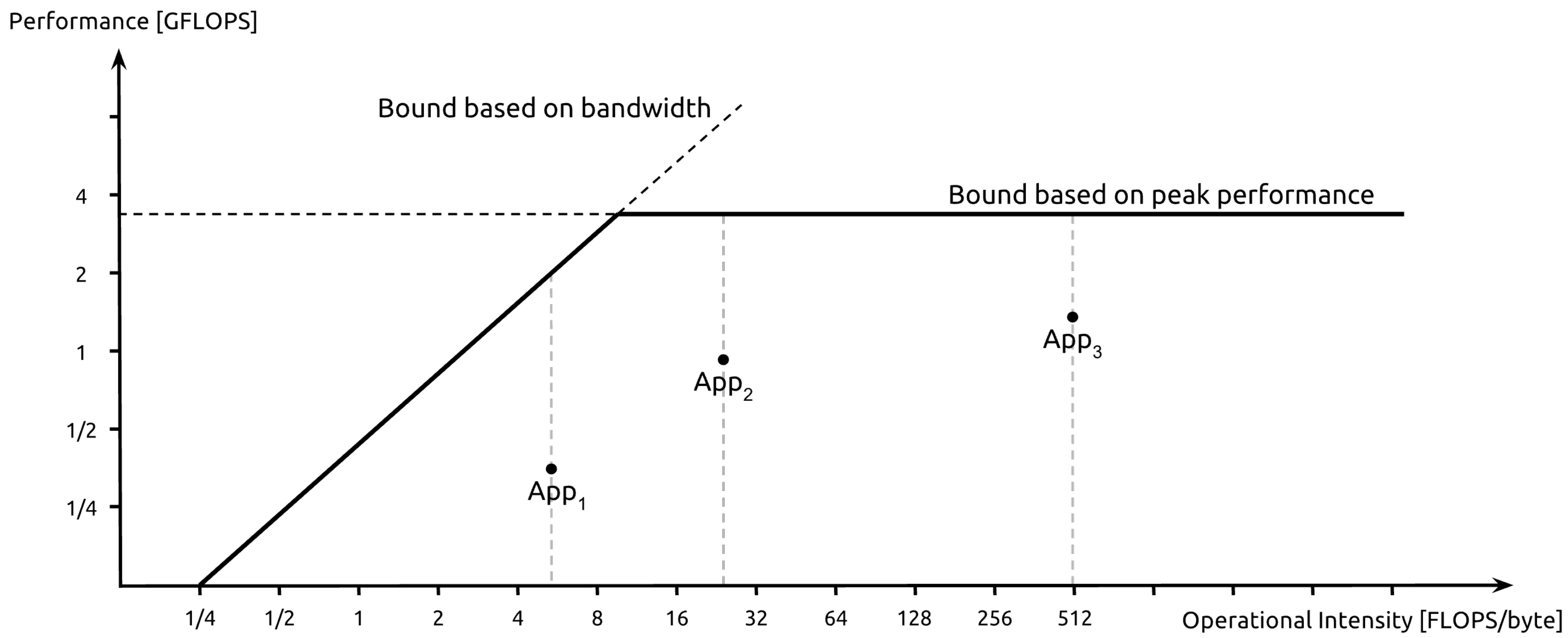
- `ml Forge/21.1.3 impi/2019.9.304-iccifort-2020.4.304`
- `mkdir ~/forge_examples/map && cd ~/forge_examples/map`
- `OMP_NUM_THREADS=8 map mpirun -n 2 ../wave_openmp 10`

- Optionally limit duration
- Optionally adapt metrics
- Click Run



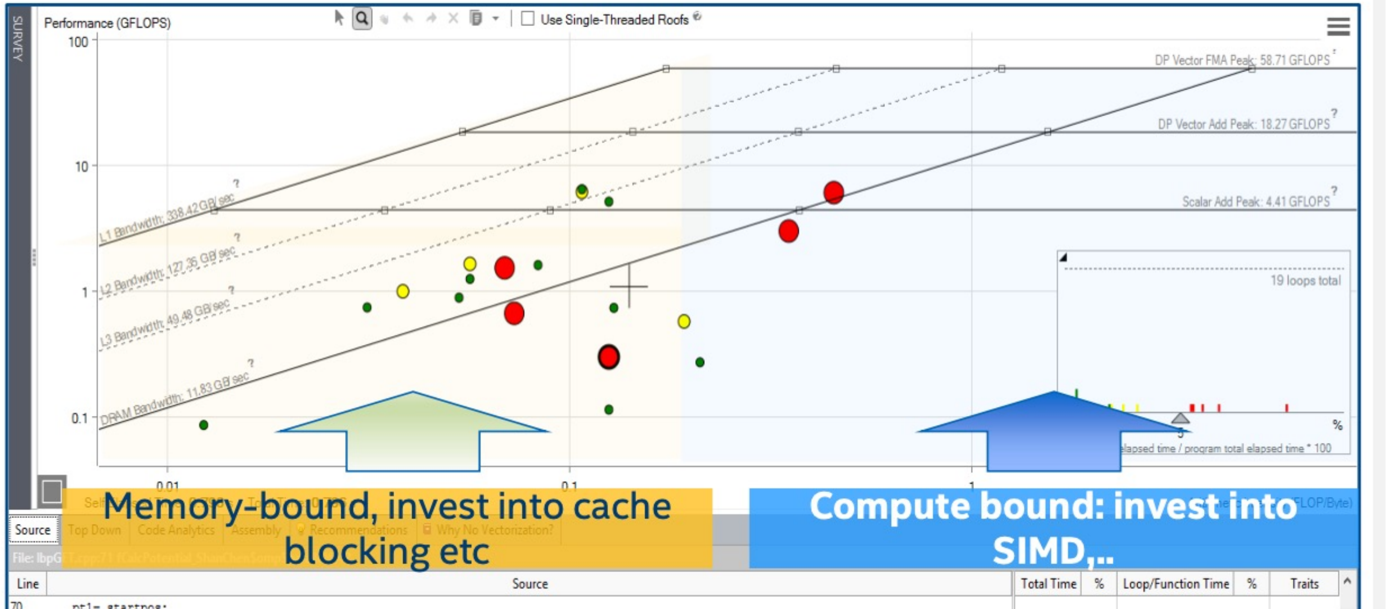
ROOFLINE MODEL

- 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)



INTEL ADVISOR

- Primarily to support vectorization of codes
- Performs dynamic analysis of codes
- Identify data access patterns
- But also computes Operational intensity vs. Performance (flops)
- It helps to identify what loops to focus on (Big red dots first)
- Ideally, during optimisations the dot moves top right



INTEL ADVISOR - EXAMPLE

```
mkdir ~/forge_examples/advisor
```

```
ml Advisor
```

- To analyse MPI application:

```
mpirun -n 2 advixe-cl --collect survey --project-dir advisor/wave_c/ -- ./wave_c 10
```

```
mpirun -n 2 advixe-cl --collect tripcounts --project-dir advisor/wave_c/ --flop --no-trip-counts -- ./wave_c 10
```

```
advixe-gui advisor/wave_c/
```

- Show my results -> Summary -> Survey & Roofline

NVIDIA NSIGHT SYSTEMS

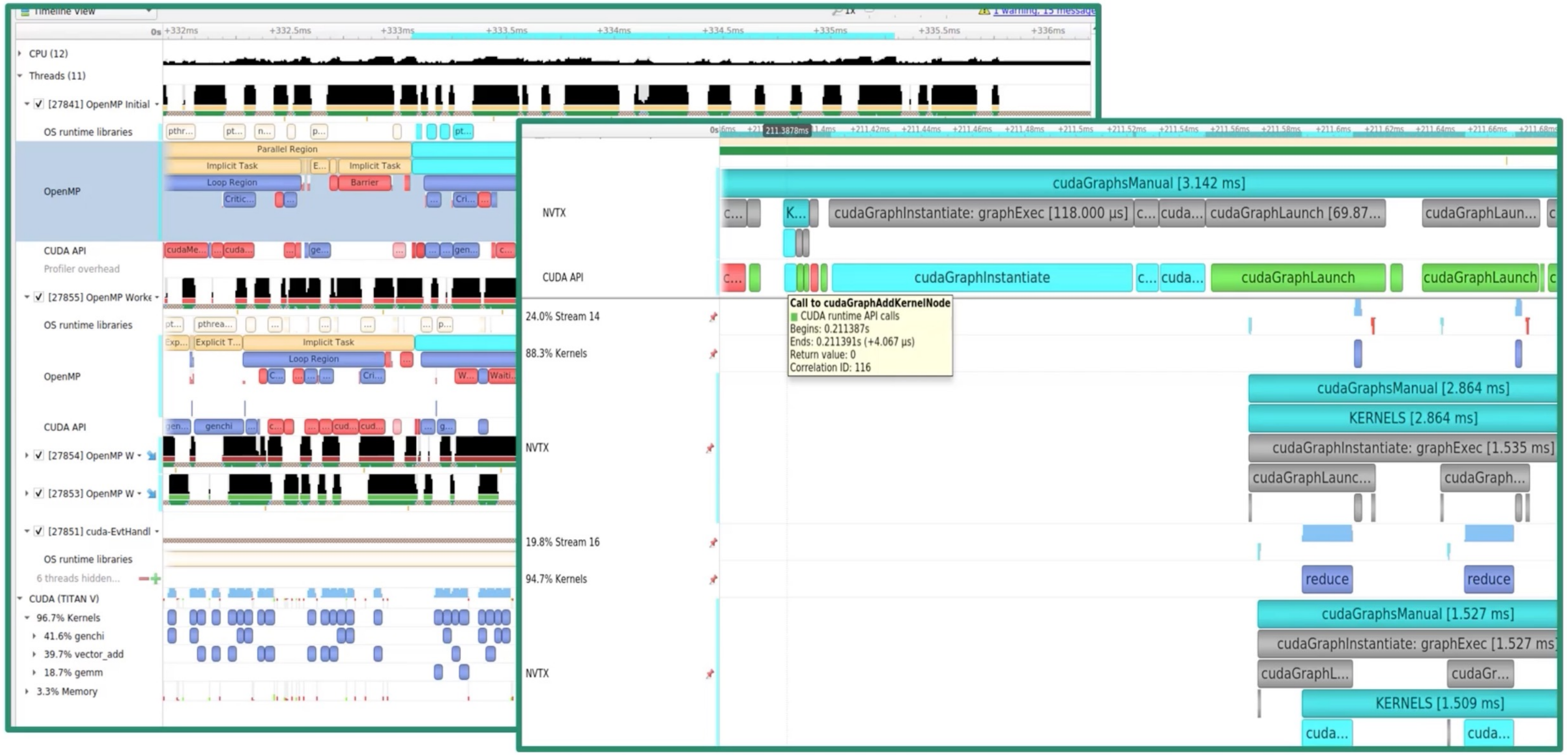
Scalable system-wide performance analysis tool

- Low-overhead multi-node, multi-GPU profiling
- Assess on timeline to narrow down frames/areas of the app to focus
- Locate optimization opportunities
- Determine CPU vs. GPU bottlenecks, idle time
- Visualize millions of events on a very fast GUI timeline
- Or gaps of unused CPU and GPU time
- 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
- Threading
- OS runtime calls
- NVTX
- CUDA API calls
- HtD / DtH data transfers
- CUDA kernels / OpenACC
- CUDA streams
- CUDA libraries (cuBLAS, …), GPU HW metrics, UCX, NIC, …



NVIDIA NSIGHT SYSTEMS

PROFILING WITH NSIGHT SYSTEMS

GUI profiling and analysis

- `nsight-sys`
  - File -> New Project
  - **Select target for profiling… -> acnXX.karolina.it4i.cz** (your allocated GPU node)
  - Enter **Command line** and **Working directory** (absolute path to the binary required)
  - Select tracing modules (CPU, OS, CUDA, GPU, …)
  - Start

Cmd line profiling + GUI analysis

- **`nsys profile -t cuda,osrt --stats=true -o simpleMultiGPU`** `./simpleMultiGPU`
- `nsight-sys`
  - File -> Open -> Select simpleMultiGPU.nsys-rep

NVIDIA NSIGHT SYSTEMS - EXAMPLE

```
git clone https://github.com/NVIDIA/cuda-samples.git
ml CUDAcore/11.6.0 Qt5/5.14.2-GCCcore-10.2.0
cd cuda-samples/Samples/0_Introduction/concurrentKernels/
make SMS=70
```

- Perform profiling of **concurrentKernels** example with:
  - CPU context switch
  - OS runtime libraries
  - CUDA
  - GPU metrics

- An extra example:

```
cd cuda-samples/Samples/0_Introduction/simpleMultiGPU/
simpleMultiGPU                                   # at least 2 GPUs required
make SMS=70
```

POP COE

An EU H2020 **Centre of Excellence** (CoE)

- On **Performance Optimisation and Productivity**
- Promoting **best practices** in parallel programming

Providing **FREE Services**

- Precise understanding of application and system behaviour
- Suggestion/support on how to refactor code in the most productive way

Horizontal

- Transversal across application areas, platforms, scales

For EU **academic AND industrial codes and users**

www.pop-coe.eu pop@bsc.es @POP_HPC youtube.com/POPHPC

POP COE

Performance Assessment

- Primary service
- Identifies performance issues of customer code
- If needed, identifies the root causes of the issues found and qualifies and quantifies approaches to address them (recommendations)
- Medium effort (1-3 months)
- Performance report

Proof-of-Concept

- Follow-up service
- Experiments and mock-up tests for customer codes
- Kernel extraction, parallelisation, mini-apps experiments to show effect of proposed optimisations
- Larger effort (3-6 months)

Note: Effort shared between our analysts and customer

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

Intel performance tools: VTune and Advisor

- Running VTune on IT4I systems requires loading of special kernel modules, see the docs

Nvidia tools for GPUs: Nsight Systems and Nsight Compute

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

Further reading:

- https://software.intel.com/content/www/us/en/develop/articles/predicting-and-measuring-parallel-performance.html
- https://developer.arm.com/documentation/101136/2020/Performance-Reports?lang=en
- https://developer.arm.com/documentation/101136/2020/MAP?lang=en
- https://software.intel.com/content/www/us/en/develop/articles/intel-advisor-roofline.html
- https://scc.ustc.edu.cn/zlsc/tc4600/intel/2018.1.163/advisor/welcomepage/get_started.htm
- https://llvm.org/docs/Benchmarking.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

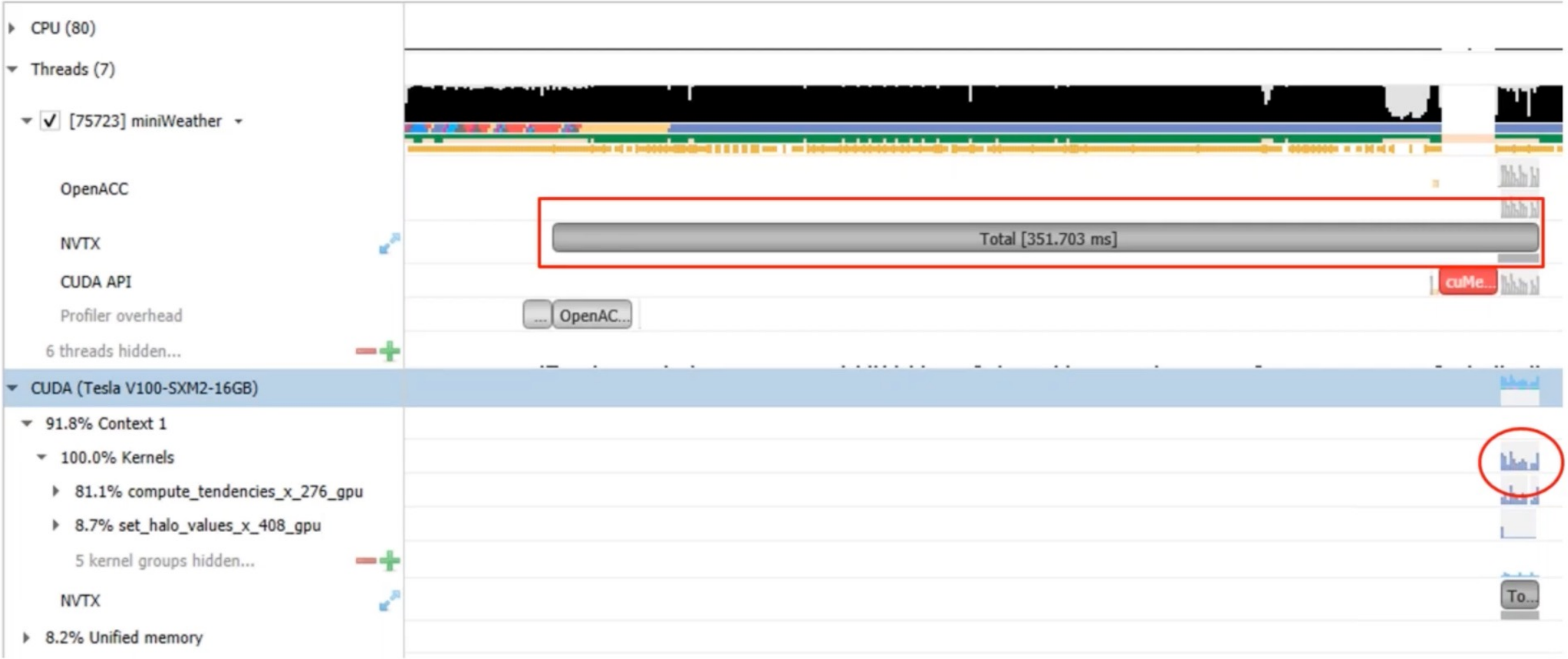
VSB TECHNICAL UNIVERSITY OF OSTRAVA | IT4INNOVATIONS NATIONAL SUPERCOMPUTING CENTER

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

MINISTRY OF EDUCATION, YOUTH AND SPORTS

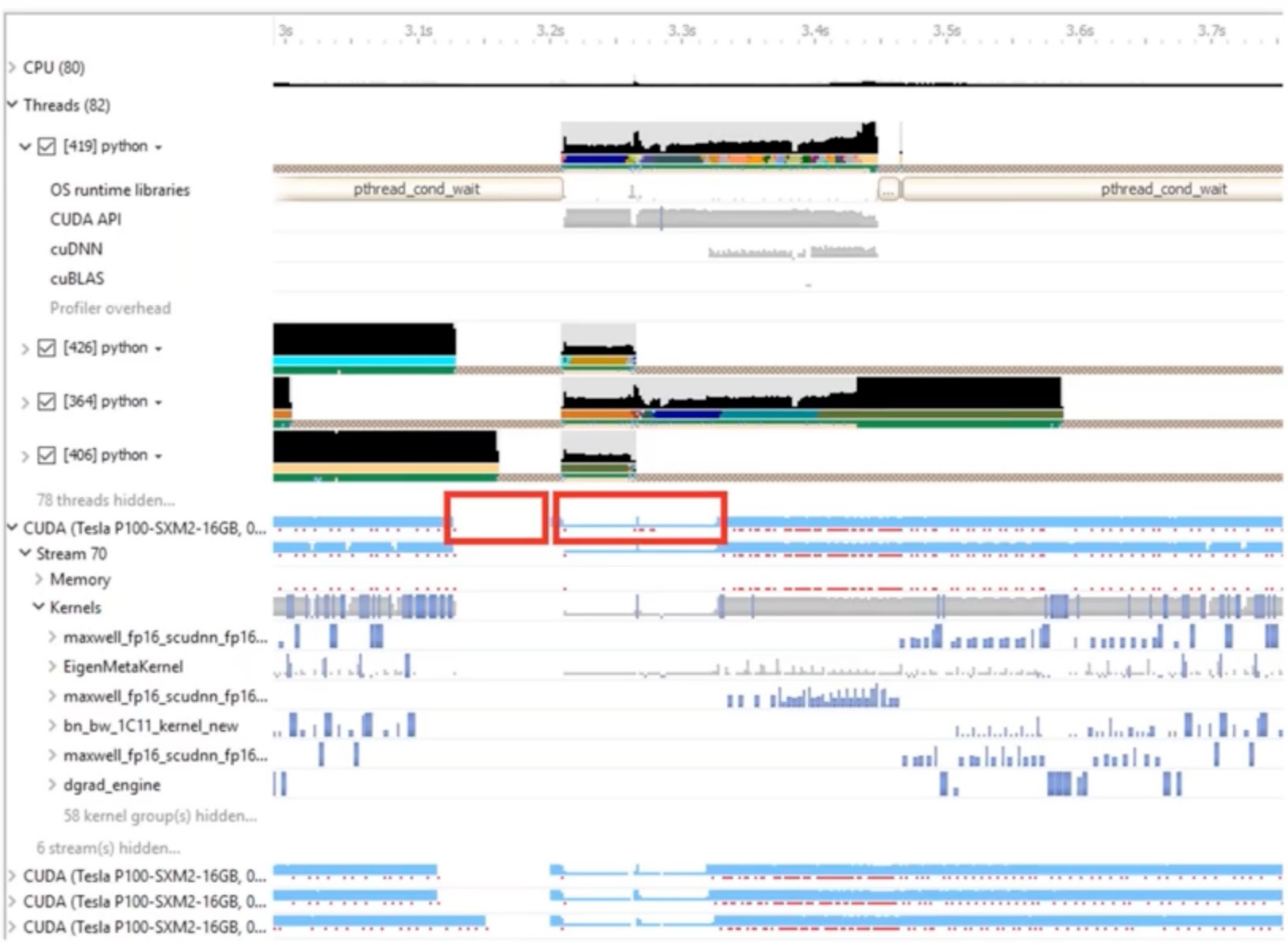
ANALYSIS WITH NSIGHT SYSTEMS

Only small portion of application accelerated



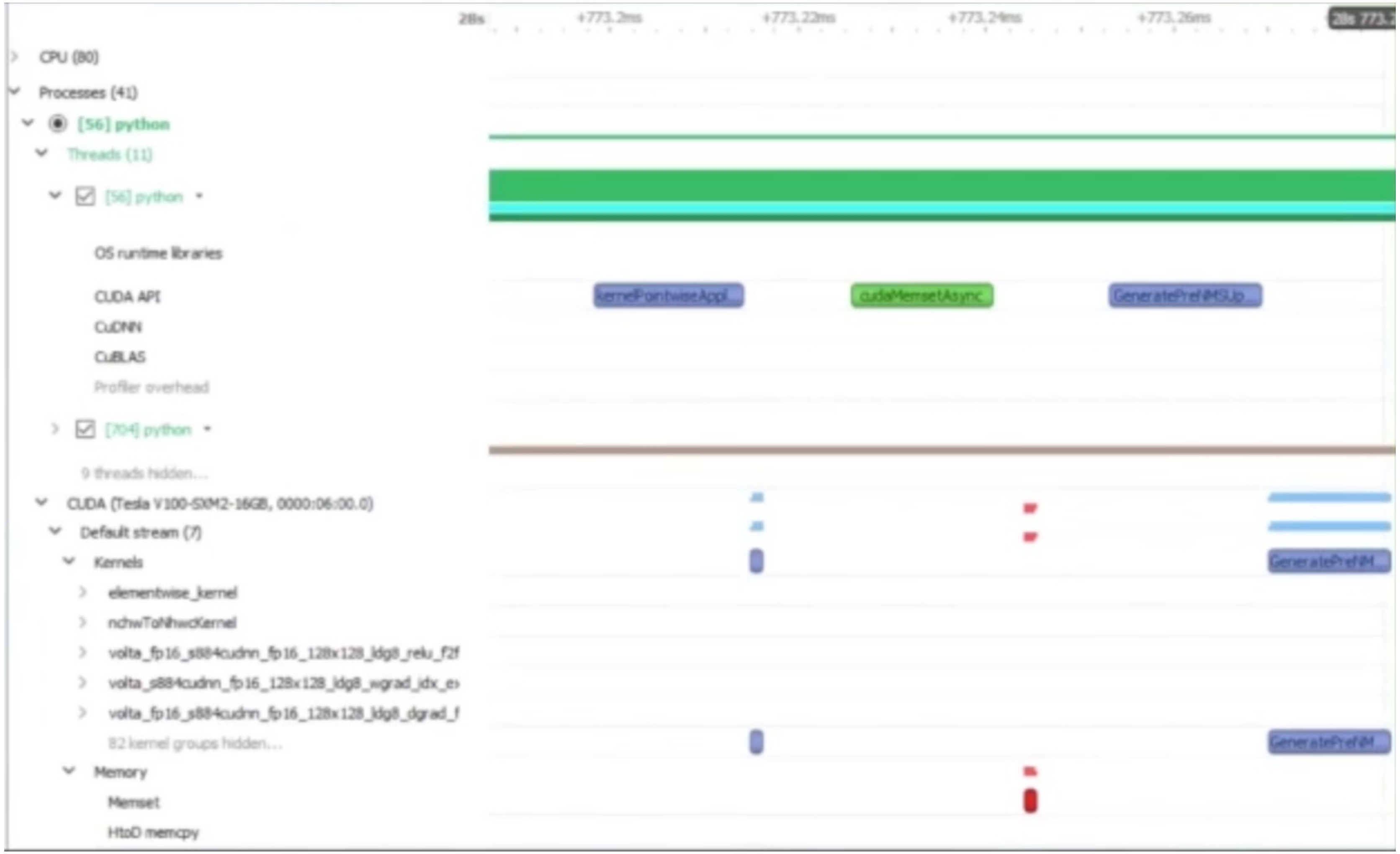
ANALYSIS WITH NSIGHT SYSTEMS

GPU idle or low utilization level of details (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 synchronous – DtH pageable memory -> Mitigate with pinned memory

ANALYSIS WITH NSIGHT SYSTEMS

GPU idle caused by stream synchronization