VSB TECHNICAL UNIVERSITY OF OSTRAVA | IT4INNOVATIONS NATIONAL SUPERCOMPUTING CENTER

INTRODUCTION TO HIGH PERFORMANCE COMPUTING

PART 5
EFFICIENT SYSTEM USAGE – PROFILING, TUNING, OPTIMIZATION

Radim Vavřík

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

MINISTRY OF EDUCATION, YOUTH AND SPORTS

OUTLINE

- Performance optimization
  - Hardware aspects
  - Development process
  - Best-practices
- Performance metrics
- Profiling methods
- Profiling tools
- Energy efficiency tools
- POP COE

Cray-1 supercomputer (source: wikipedia.org)

TECHNICAL NOTES

- All presented examples/tools can be accessed and reproduced anytime at IT4I clusters
- Ideal case: Login to Barbora cluster and submit an interactive job
- VNC session usually offer better UX For GUI tools than X11
- https://docs.it4i.cz/general/accessing-the-clusters/graphical-user-interface/vnc/
- Most of the presented tools provide remote profiling, e.g., generate output remotely from CLI while analysis can be done locally in GUI
  - Not covered today

EFFICIENT USE OF HPC

- What does it mean?
  - To get the most performance out of your hardware
  - The process is called **Performance Optimization**

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

PERFORMANCE OPTIMIZATION - KEYS

- 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 optimizations?
- 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!**

PERFORMANCE OPTIMIZATION - HARDWARE ASPECTS

- I/O – disk operations
- Network - communication
- CPU sockets – NUMA effect
- CPU cores – threads/processes, affinity, pinning
- Vector registers – vectorization, vector instructions
- Accelerators – GPU/MIC, utilization, data transfers

PERFORMANCE OPTIMIZATION - HARDWARE ASPECTS

- 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

PERFORMANCE OPTIMIZATION - HARDWARE ASPECTS

- Use HTOP tool for interactive jobs
- Configurable (e.g. core id, threads, process tree)
- htop –d 5

```
 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 profiling tools
3. Optimize code 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. Re-measure (verify) optimizations using profiling tools
4. Repeat until job done

PERFORMANCE OPTIMIZATION - TIPS

- Do not optimize 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 optimization 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)
- Speed (FLOPS) and Memory throughput (GBPS)
  - 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 limit
- E.g. Intel® Xeon® Platinum 8280M Processor

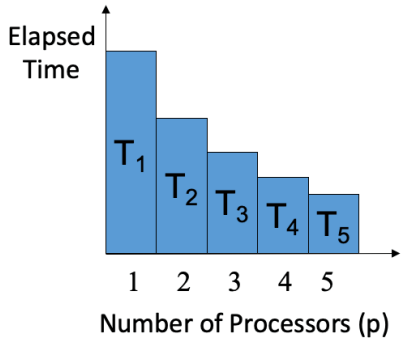
Number of compute nodes (Salomon-size machine)	1000
Number of sockets (CPUs) per node	2
Frequency	2.7 GHz
Number of cores per socket	28
FMA instruction (a * b + c)	2
FMA units per core	2
SIMD (AVX-512) = 8x double precision	8
	4 838 400 Gflop/s
	(4.8 Pflop/s)

SPEEDUP EXAMPLE

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

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

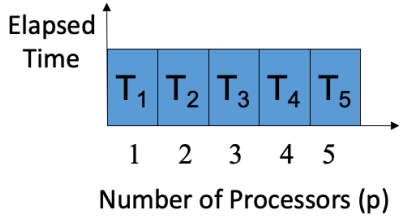
  $E_P = S_P / P$ $\quad$ $E_{16} = S_{16} / 16 = 16 / 16 = 1$



- **Weak scaling**

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

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



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

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

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

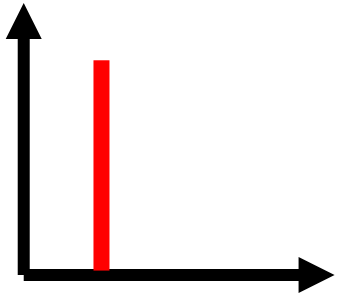
  $S_P = 1 / E_P = T_P / T_1$ $\quad$ $S_{16} = T_{16} / T_1 = 32 / 32 = 1$

PROFILING METHODS

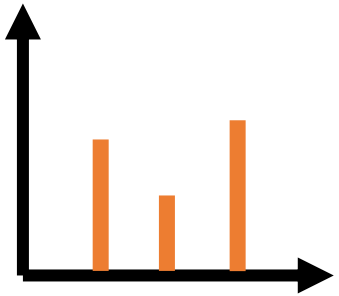
- Reporting
  - General overview of the whole application characteristics
- Tracing
  - Records and timestamps all operations – intrusive
- Instrumenting
  - Add instructions in the source code to collect data - intrusive
- Sampling
  - Automatically collect data per time unit
- Modeling
  - Simulate state, e.g. ideal network, HW failure, etc.

TYPES OF PROFILES

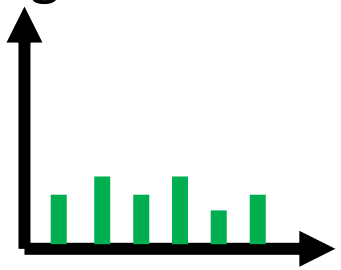
- Hotspot
  - One function corresponds to more 80% of the runtime
  - Large speed-up potential
  - Best optimisation scenario



- 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



PROFILING – CPU TOOLS

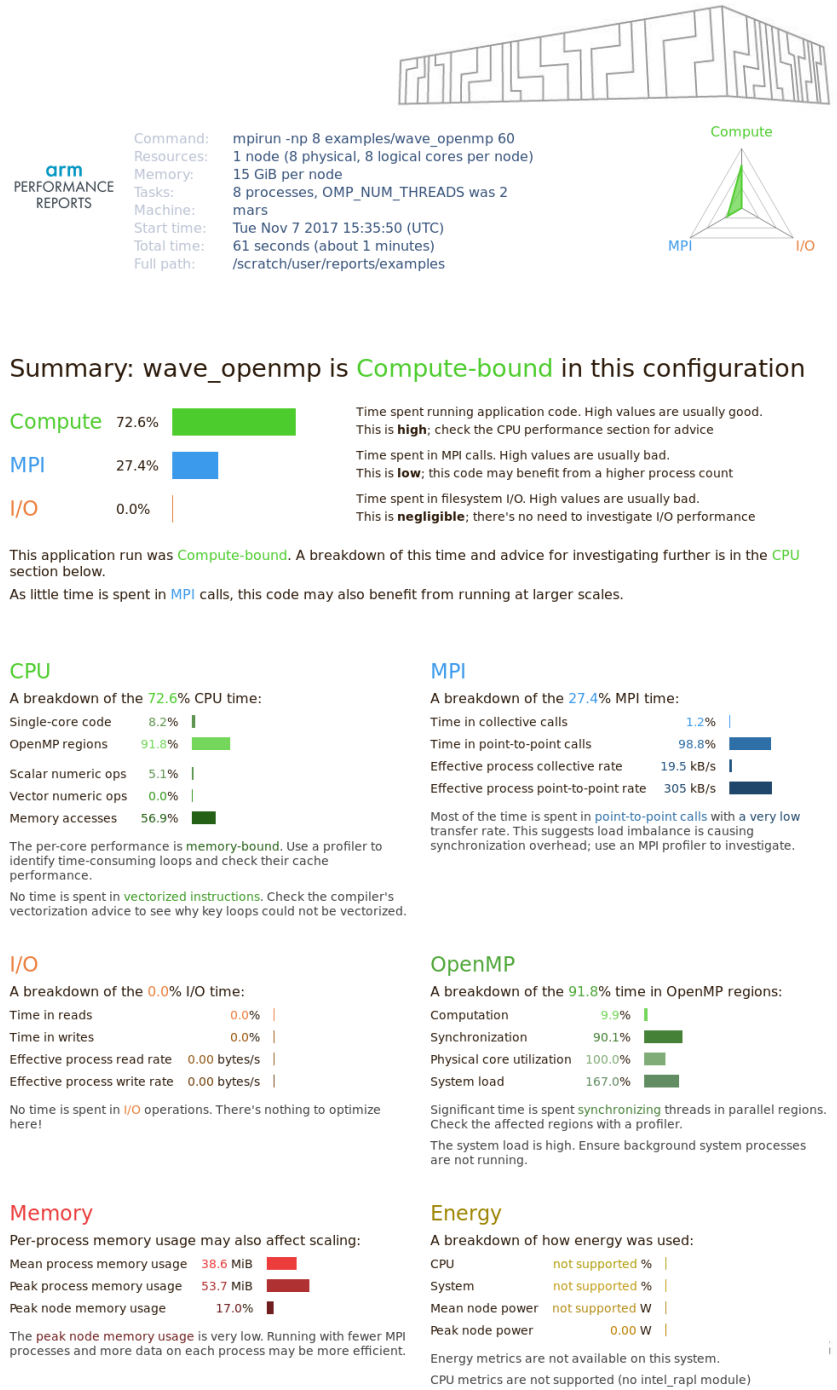
- Single-node, Parallel, Instrumentation, Correctness
- Proprietary tools – usually available on clusters
  - ARM (Allinea) Performance Report
  - ARM (Allinea) MAP
  - Intel Application Performance Snapshot
  - Intel Vtune
  - AMD µProf
  - Vampir
- Open-source tools (VI-HPS)
  - Extrae/Paraver
  - Score-P/Scalasca/Cube
  - MAQAO
  - https://www.vi-hps.org/tools/tools.html

PROFILING – GPU TOOLS

- GUI tools
  - NVIDIA Visual Profiler - deprecated
  - NVIDIA Nsight Systems – application level profiling (also CPU threads, OpenMP, MPI, ...)
  - NVIDIA Nsight Compute – CUDA kernel level profiling
- Command-line tools – useful if you cannot easily use GUI
  - nvprof - deprecated
  - nsys

ARM PERFORMANCE REPORTS

- No source code or recompilation required
- Run: **perf-report mpirun -n 4 app**
- Auto-generated .txt / .html output
- Profile report: **perf-report profile.map**
- Report summary (Compute, MPI, Input/Output)
- CPU, MPI, I/O, OpenMP, Memory, Energy, Accelerator breakdown sections
- Advanced configuration through command line flags possible



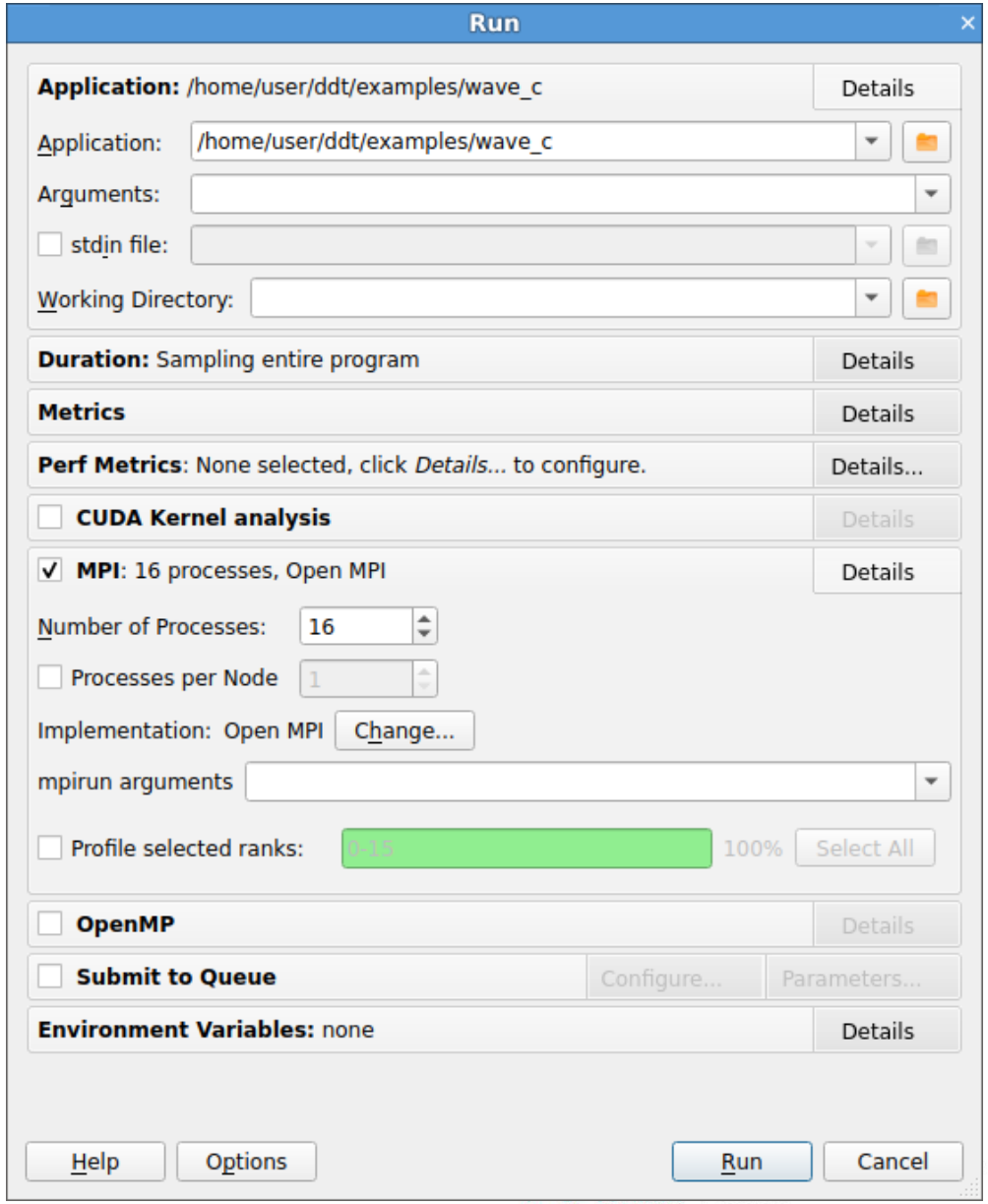
ARM PERFORMANCE REPORTS - EXAMPLE

- ml Forge/20.1.1 impi/2019.7.217-iccifort-2020.1.217
- ml show Forge/20.1.1
- cp –r /apps/all/Forge/20.1.1/examples ~/forge_examples
- cd ~/forge_examples
- make
- mpirun -n 1 ./wave_c 10
- mkdir perf_reports && cd perf_reports
- perf-report mpirun -n 1 ../wave_c 10
- firefox wave_c_1p_1n_1t_YYYY-MM-DD_hh-mm.html
- perf-report mpirun -n 24 ../wave_c 10
- OMP_NUM_THREADS=12 perf-report mpirun -n 2 ../wave_openmp 10

ARM MAP

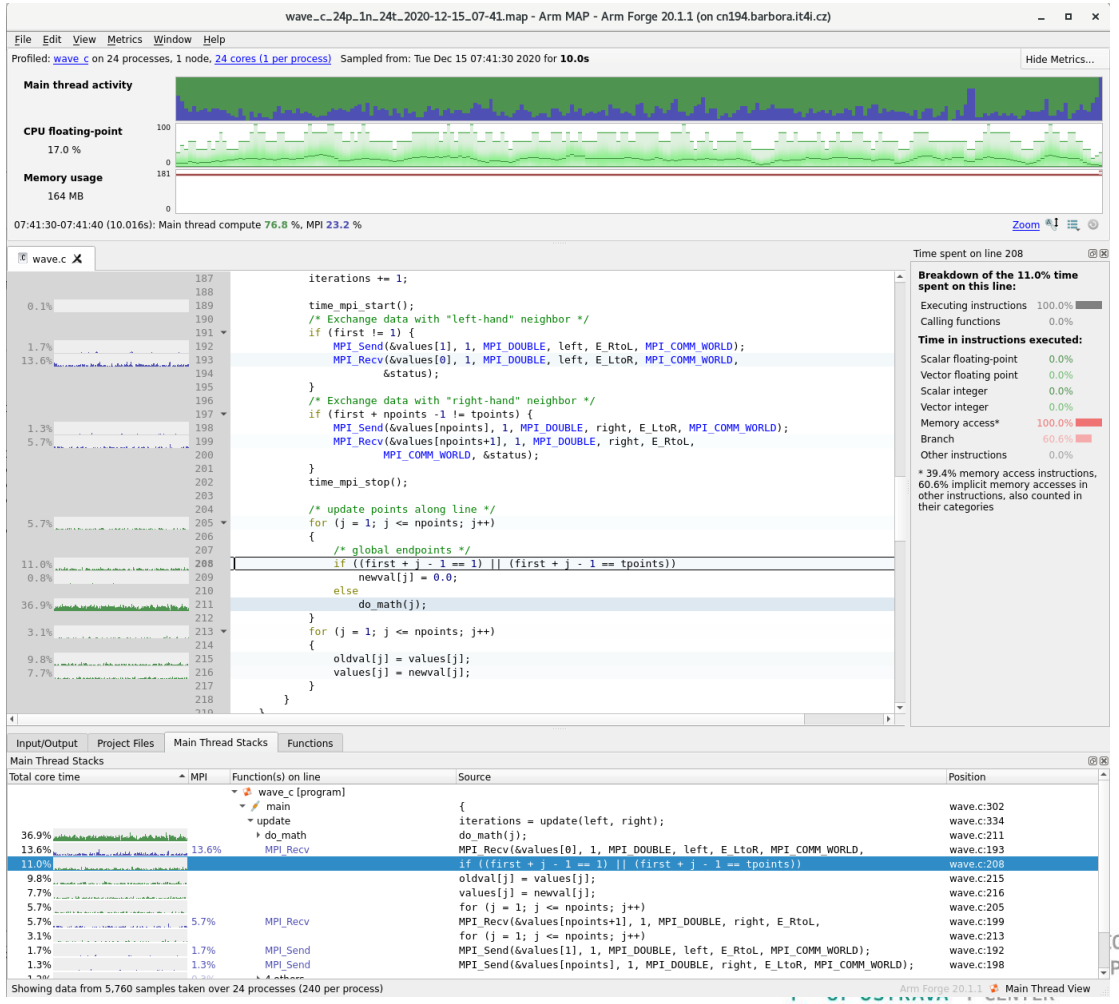
- Low overhead sampling
- No source code or recompilation required
- Only debugging symbols -g / -g1
- Source code viewer
- Bottleneck locations
- CPU, MPI, I/O, Memory, Vectorization
- Run:

map
map program_name [arguments]
map <profile-file>



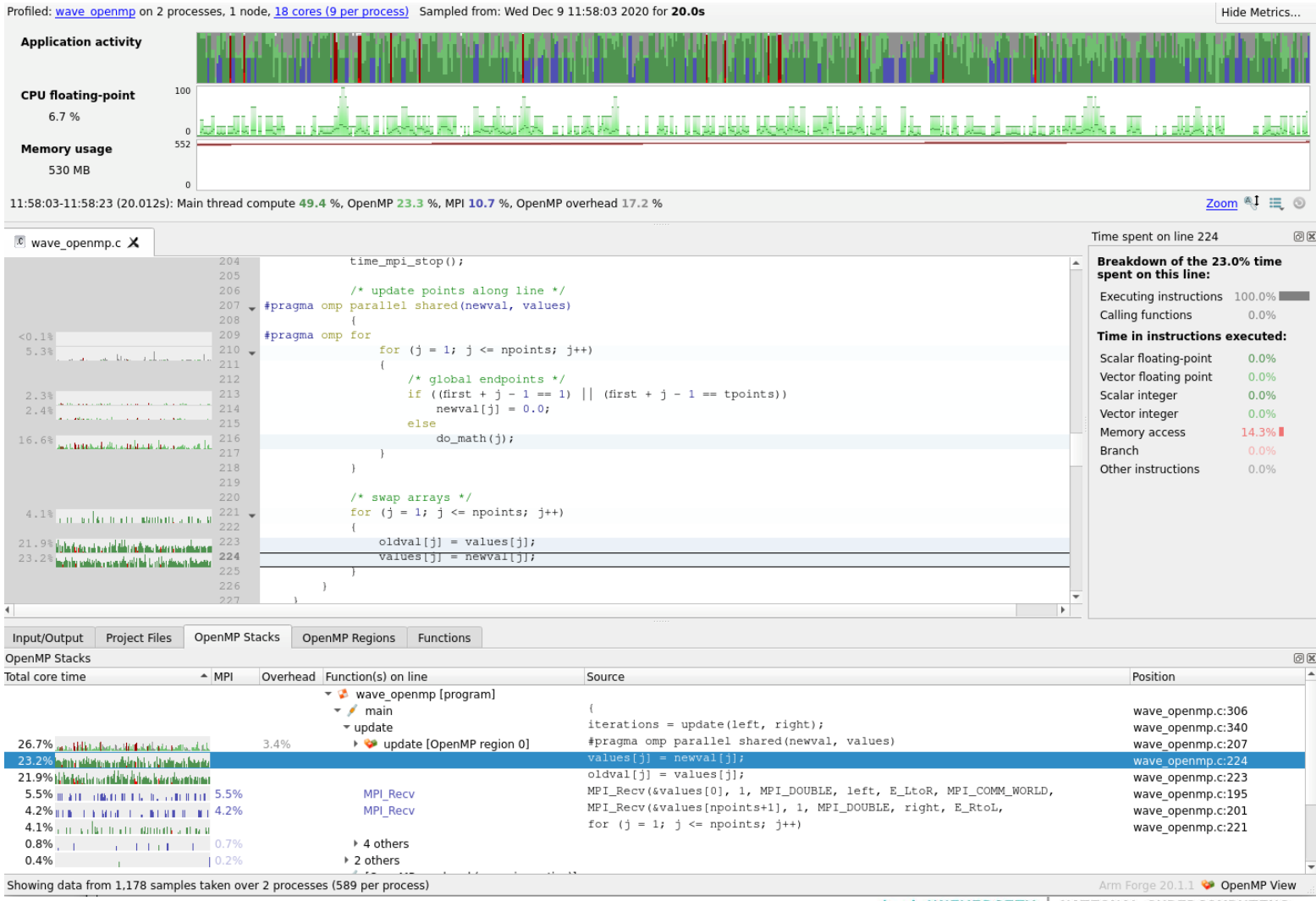
ARM MAP - EXAMPLE

- ml Forge/20.1.1 impi/2019.7.217-iccifort-2020.1.217
- mkdir ~/forge_examples/map && cd ~/forge_examples/map
- map mpirun -n 24 ../wave_c 10
- Optionally limit duration
- Optionally limit metrics
- Click Run



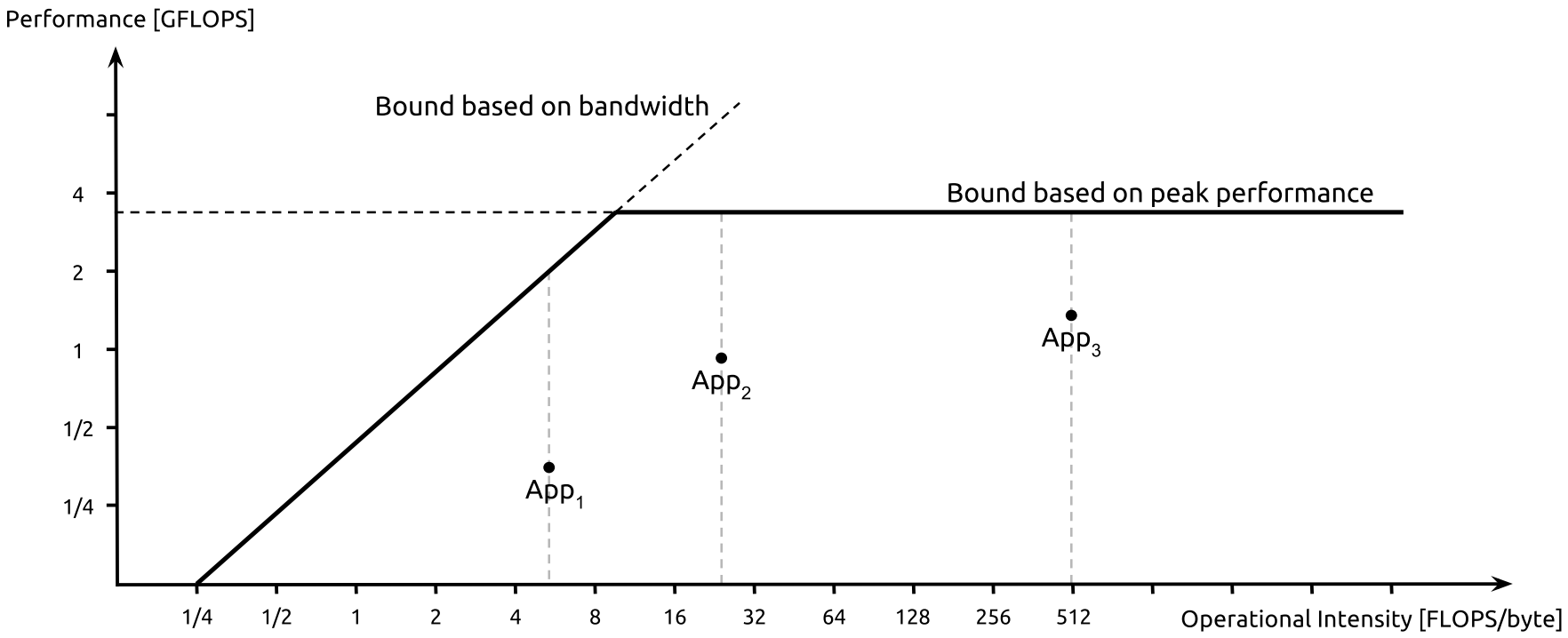
ARM MAP - EXAMPLE

- OMP_NUM_THREADS=12 map mpirun -n 2 ../wave_openmp 10
- All charts timelines
- Horizontal axis time
- Vertical axis processes
- Useful code green
- MPI blue
- With zoom breakout adapts
- Multiple presets
  - e.g. MPI call duration can show load balance, etc.



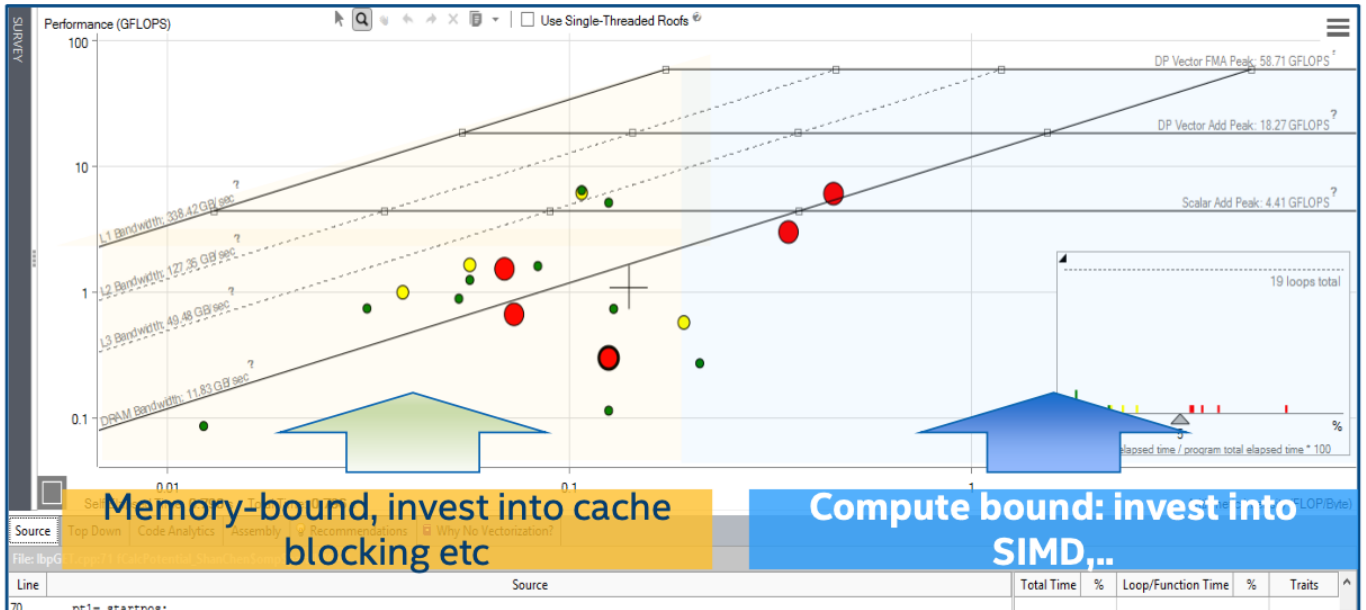
ROOFLINE MODEL

- 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**
- **Operational intensity** is 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 optimizations the dot moves top right



INTEL ADVISOR - EXAMPLE

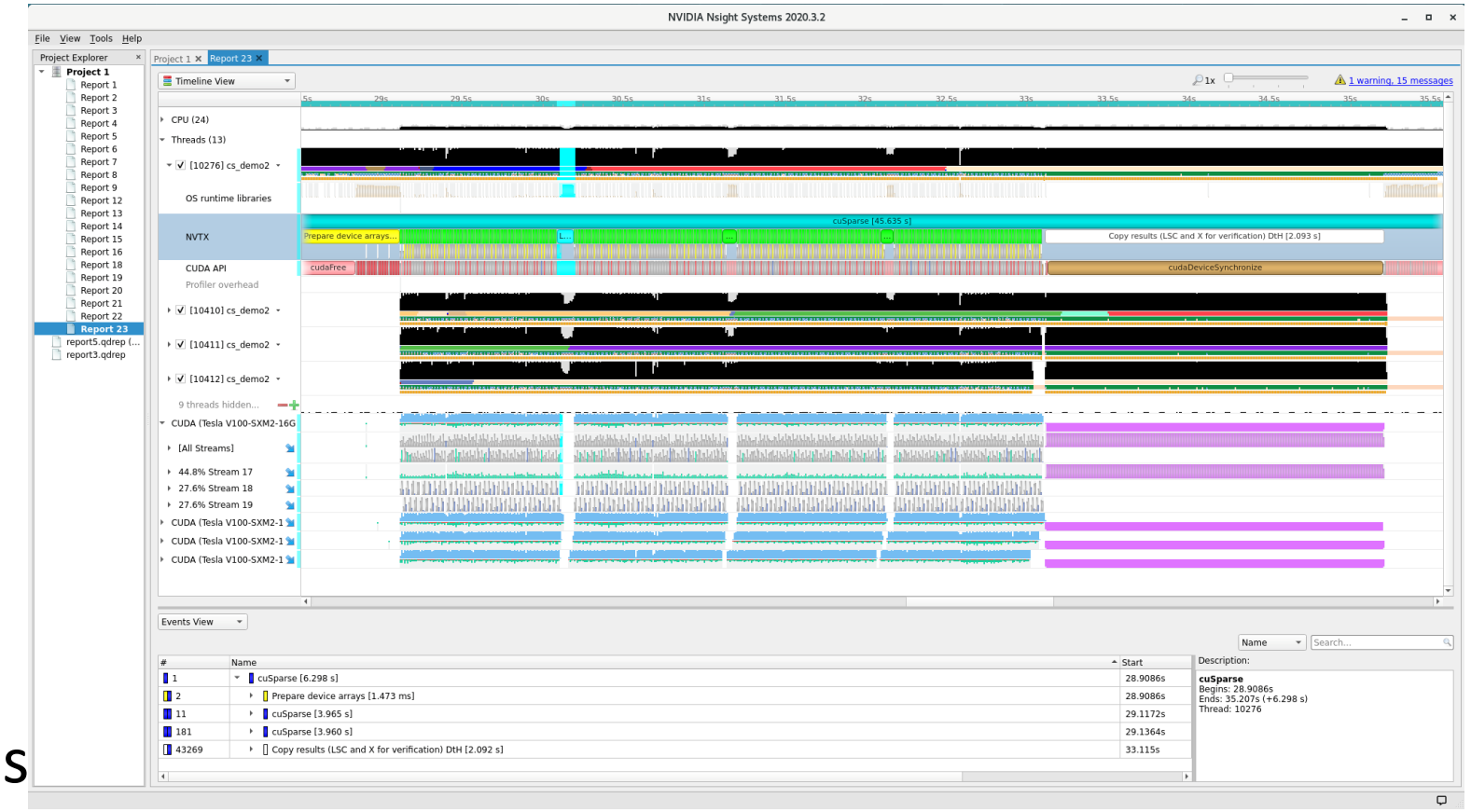
- mkdir ~/forge_examples/advisor
- ml Advisor
- To run MPI application:

1. mpirun -n 2 advixe-cl --collect survey --project-dir advisor/wave_c/ -- ./wave_c 10
2. mpirun -n 2 advixe-cl --collect tripcounts --project-dir advisor/wave_c/ --flop --no-trip-counts -- ./wave_c 10
3. 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
- CPU cores utilization
- MPI calls
- Threading
- OS runtime calls
- NVTX
- CUDA API calls
- HtD / DtH data transfers
- CUDA / OpenACC kernels
- CUDA streams



NVIDIA NSIGHT SYSTEMS - EXAMPLE

- use qnvidia queue to get GPU node
- git clone https://github.com/NVIDIA/cuda-samples.git cuda11-samples
- cd cuda11-samples/Samples/concurrentKernels
- ml CUDA/11.0.2-GCC-9.3.0 Qt5/5.14.1-GCCcore-9.3.0
- make SMS=70
- nsight-sys
- File -> New Project
- Select target for profiling (your GPU node)
- Set command and Working directory to concurrentKernels (absolute paths)
- Tick “Collect CUDA trace”
- Click Start and when no new samples collected click Stop

ADVANCED EFFICIENCY METRICS

- Used in POP methodology
- Multiplicative model, values 0..1
  - Global eff.
    - Parallel eff. (PE)
      - Load Balance (LB)
      - Communication eff. (CommE)
        - Serialization eff. (SerE)
        - Transfer eff. (TE)
    - Computation eff. (CompE)
      - Instruction scaling
      - IPC scaling
- How to evaluate? Best-practice:
  - > 90 % very good
  - 80 – 90 % good
  - < 80 % poor

# Nodes	1	2	4	8	16
Global efficiency [%]	81.35	87.27	95.18	97.92	86.14
- Parallel efficiency [%]	81.35	78.96	76.01	67.10	54.03
- - Load Balance [%]	96	95	91	86	78
- - Communication [%]	84.79	82.95	83.93	78.16	69.37
- Computation scalability [%]	100.00	110.53	125.23	145.94	159.44
Instruction Scaling [%]	100.00	105.77	98.18	96.78	89.91
IPC Scaling [%]	100.00	112.04	129.75	157.31	184.03

GLOBAL EFFICIENCY (GE)

- The Global Efficiency describes how well the parallelization of your application is working
- The Global Efficiency can be split into Parallel Efficiency and Computation Efficiency

$$GE = PE * CompE$$

- Global Efficiency (GE)
  - Parallel Efficiency (PE)
    - Load Balance Efficiency (LB)
    - Communication Efficiency (CommE)
      - Serialization Efficiency (SerE)
      - Transfer Efficiency (TE)
  - Computation Efficiency (CompE)
    - IPC Scaling
    - Instruction Scaling

PARALLEL EFFICIENCY (PE)

- The Parallel Efficiency describes how well the execution of the code in parallel is working
- The Parallel Efficiency can be split into Load Balance Efficiency and Communication Efficiency

$$PE = LB * CommE$$

- Global Efficiency (GE)
  - Parallel Efficiency (PE)
    - Load Balance Efficiency (LB)
    - Communication Efficiency (CommE)
      - Serialization Efficiency (SerE)
      - Transfer Efficiency (TE)
  - Computation Efficiency (CompE)
    - IPC Scaling
    - Instruction Scaling

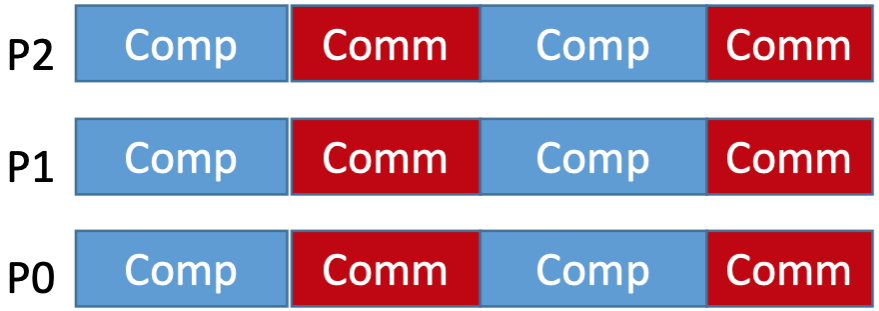
LOAD BALANCE EFFICIENCY (LB)

- Global Efficiency (GE)
  - Parallel Efficiency (PE)
    - Load Balance Efficiency (LB)
    - Communication Efficiency (CommE)
      - Serialization Efficiency (SerE)
      - Transfer Efficiency (TE)
  - Computation Efficiency (CompE)
    - IPC Scaling
    - Instruction Scaling

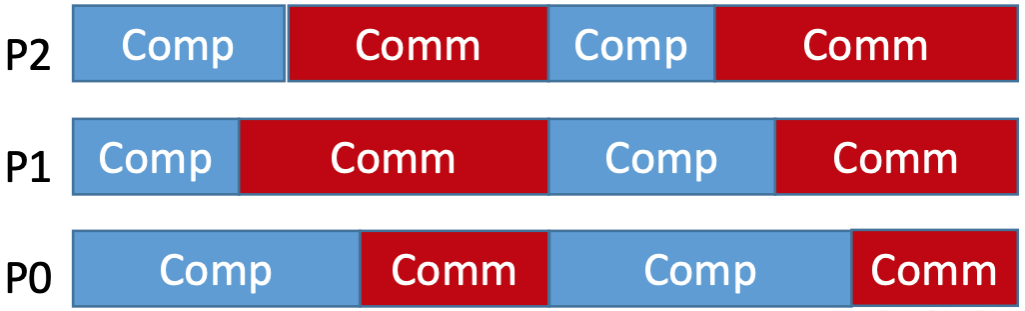
- The Load Balance Efficiency reflects how well the distribution of work to processes of threads is done in the application
- The Load Balance Efficiency is the ratio between the average time of a process spend in computation and the maximum time a process spends in computation

$$LB = avg(tcomp) / \max(tcomp)$$

Example 1: good load balance (LB = 100%)



Example 2: bad load balance (LB = 77%)



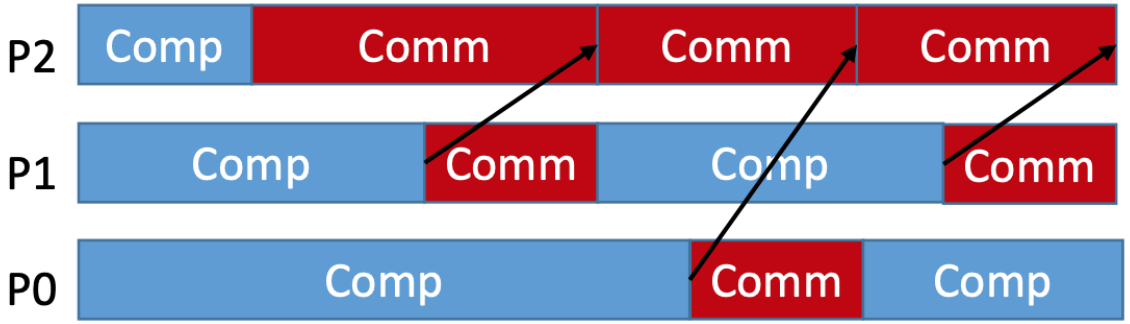
COMMUNICATION EFFICIENCY (CommE)

- Global Efficiency (GE)
  - Parallel Efficiency (PE)
    - Load Balance Efficiency (LB)
    - Communication Efficiency (CommE)
      - Serialization Efficiency (SerE)
      - Transfer Efficiency (TE)
  - Computation Efficiency (CompE)
    - IPC Scaling
    - Instruction Scaling

- The Communication Efficiency reflects the loss of efficiency by communication

$$\text{CommE} = \max_{processes} \left(\frac{computation\ time}{total\ runtime} \right)$$

Example:



CommE = 5/6 = 83%

Compute	Communication	Efficiency
1 sec.	5 sec.	$^1/_6$
4 sec.	2 sec.	$^4/_6$
5 sec.	1 sec.	$^5/_6$

- The Communication Efficiency can be split further into Serialization Efficiency and Transfer Efficiency.

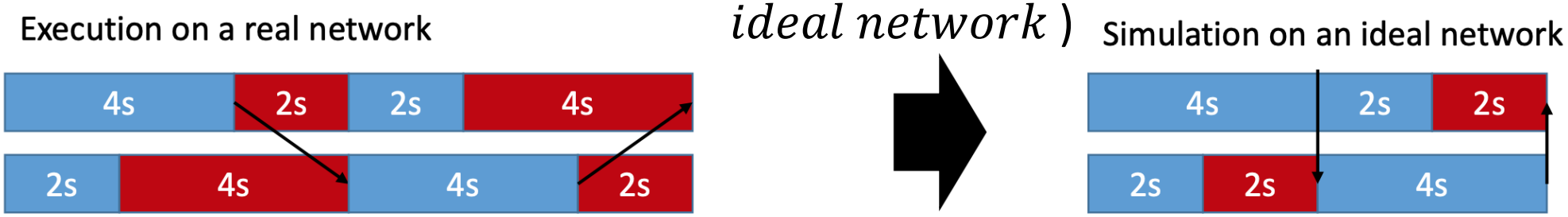
$$\text{CommE} = \text{SerE} * \text{TE}$$

SERIALIZATION EFFICIENCY (SerE)

- Global Efficiency (GE)
  - Parallel Efficiency (PE)
    - Load Balance Efficiency (LB)
    - Communication Efficiency (CommE)
      - Serialization Efficiency (SerE)
      - Transfer Efficiency (TE)
  - Computation Efficiency (CompE)
    - IPC Scaling
    - Instruction Scaling

- The Serialization Efficiency describes loss of efficiency due to dependencies between processes
- Dependencies can be observed as waiting time in MPI calls where no data is transferred, because one required process did not arrive at the communication call yet
- On an ideal network with instantaneous data transfer these inefficiencies are still present, as no real data transfer happens

SerE = max $processes$ ($computation\ time\ on\ ideal\ network$ / $total\ runtime\ on\ ideal\ network$)



SerE = 6/8 = 75%

= Computation

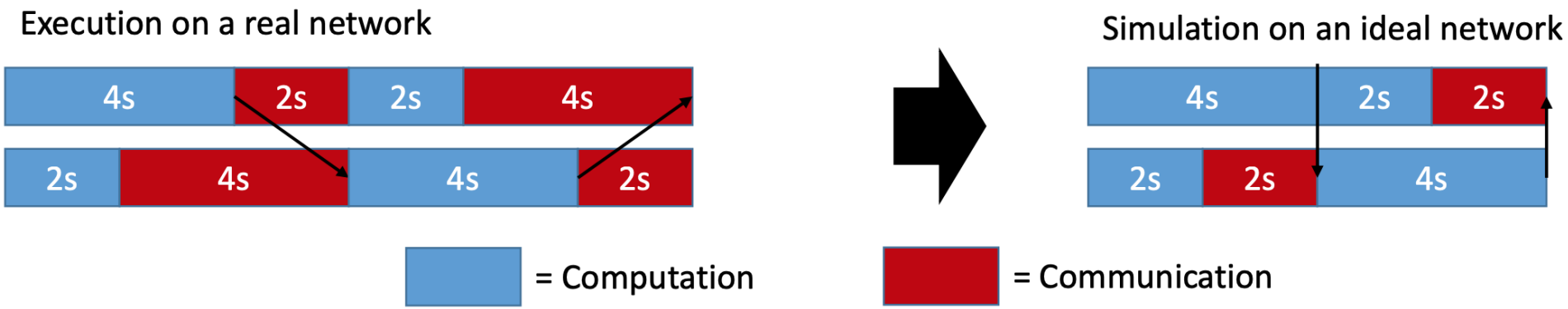
= Communication

TRANSFER EFFICIENCY (TE)

- The Transfer Efficiency describes loss of efficiency due to actual data transfer

$$TE = \textit{total runtime on ideal network} \; / \; \textit{total measured runtime}$$

- Global Efficiency (GE)
  - Parallel Efficiency (PE)
    - Load Balance Efficiency (LB)
    - Communication Efficiency (CommE)
      - Serialization Efficiency (SerE)
      - Transfer Efficiency (TE)
  - Computation Efficiency (CompE)
    - IPC Scaling
    - Instruction Scaling



TE = 8 / 12 = 66.6 %

COMPUTATION EFFICIENCY (CompE)

- The Computation Efficiency describes how well the computational load of an application scales with the number of processes
- The Computation Efficiency is computed by comparing the total time spend in computation for a different number of threads/processes
- For a linearly-scaling application the total time spend in computation is constant and thus the Computation efficiency is one

- Global Efficiency (GE)
  - Parallel Efficiency (PE)
    - Load Balance Efficiency (LB)
    - Communication Efficiency (CommE)
      - Serialization Efficiency (SerE)
      - Transfer Efficiency (TE)
  - Computation Efficiency (CompE)
    - IPC Scaling
    - Instruction Scaling

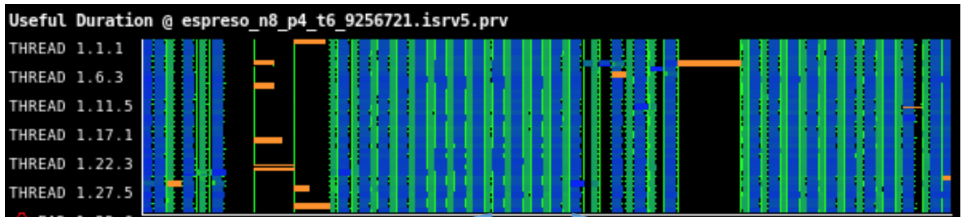
IPC SCALING / INSTRUCTION SCALING

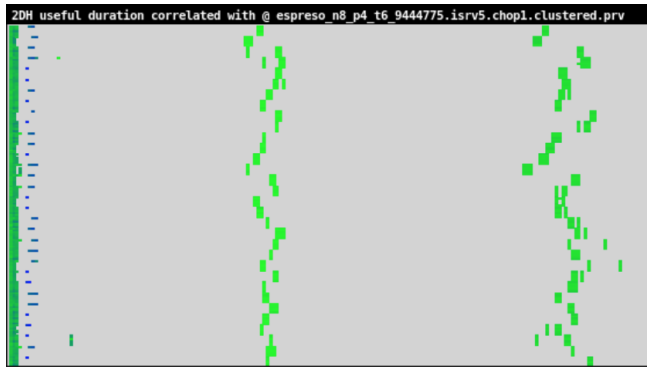
- A low computation efficiency can have two reasons:

1. With more processes more instructions are executed, e.g. some extra computation for the domain decomposition is needed. Instruction Scaling compares the total number of instructions executed for a different number of threads/processes
2. The same number of instructions is computed but the computation takes more time, this can happen e.g. due to shared recourses like memory channels. IPC Scaling compares how many instructions per cycle are executed for a different number of threads/processes

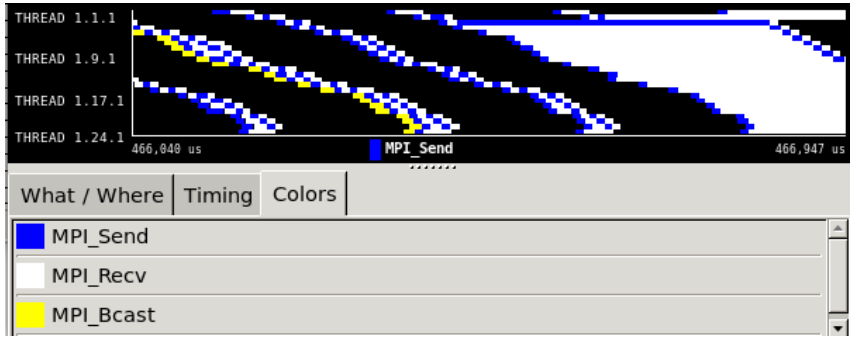
- Global Efficiency (GE)
  - Parallel Efficiency (PE)
    - Load Balance Efficiency (LB)
    - Communication Efficiency (CommE)
      - Serialization Efficiency (SerE)
      - Transfer Efficiency (TE)
  - Computation Efficiency (CompE)
    - IPC Scaling
    - Instruction Scaling

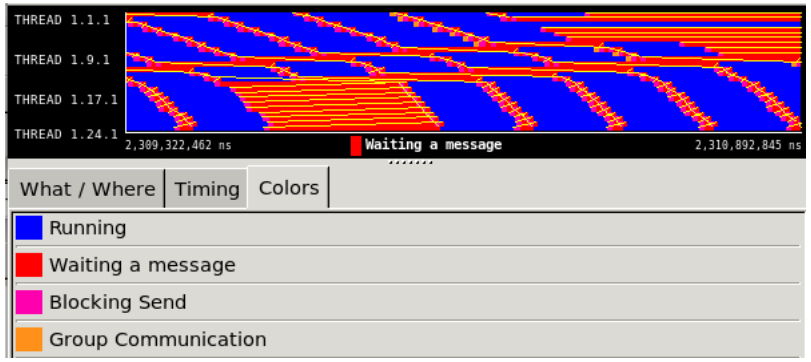
BSC TOOLS





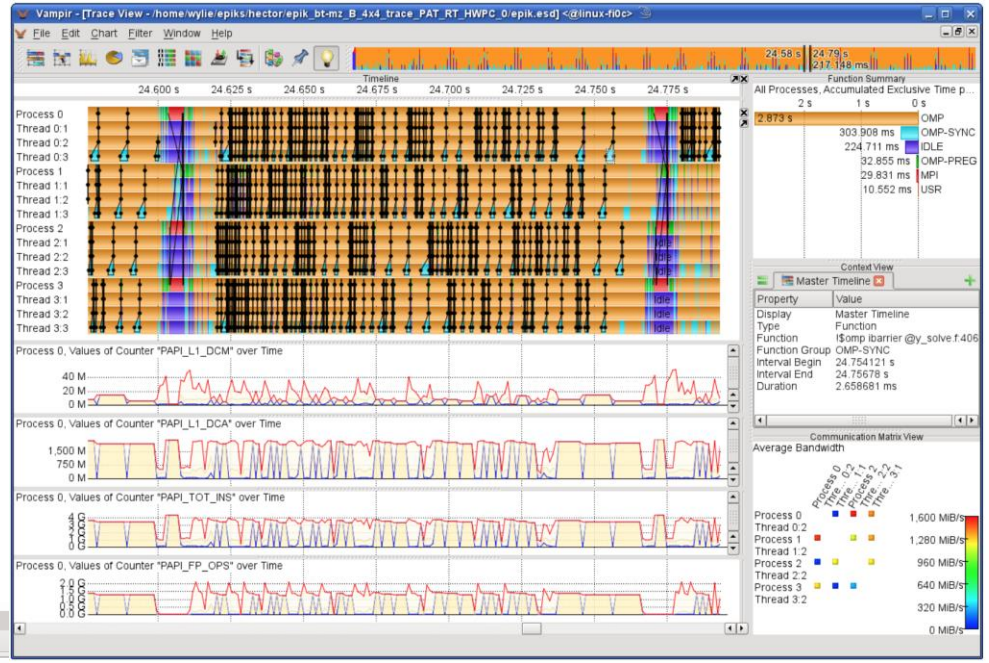
- **Extrae** tracing
  - No recompilation
  - Highly scalable
  - Parallel programming models
    - MPI, OpenMP, pthreads, OmpSs, CUDA, OpenCL, Java, Python, …
  - Platforms
    - Intel, Cray, BlueGene, MIC, ARM, Android, Fujitsu Spark, …
  - PAPI counters
  - Link to source code
- **Paraver** visualization
  - Tons of metrics
  - Statistics
  - Tutorials included
- Part of Vi-HPS package

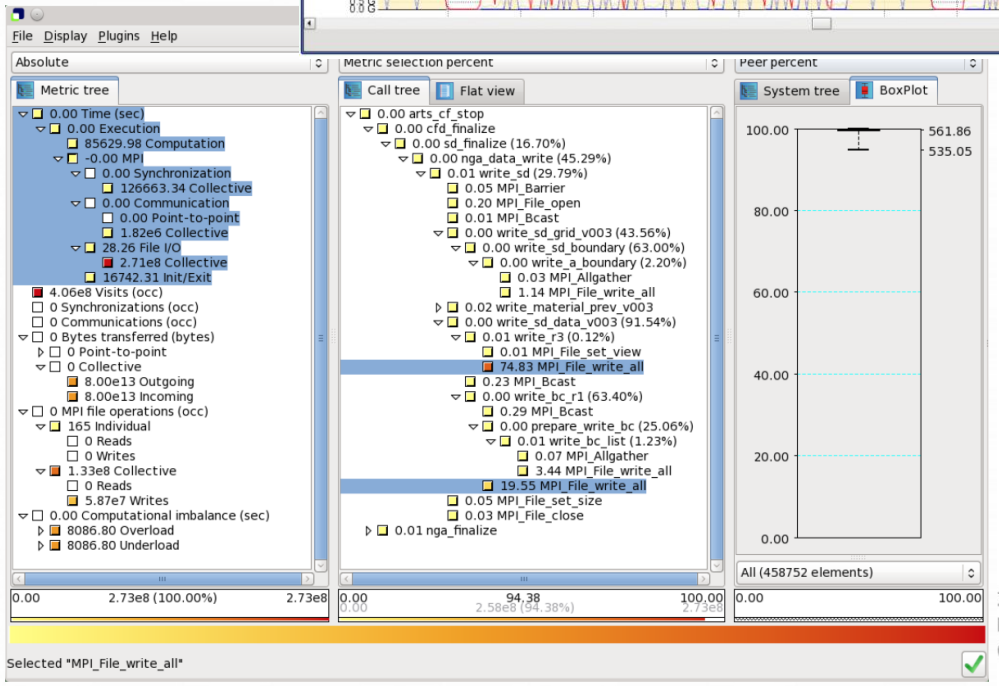




JSC TOOLS

- **SCORE-P** instrumentation
- **SCALASCA** automatic trace analyzer
- **CUBE / Vampir** visualization
- Highly scalable
- Parallel programming models
  - MPI, SHMEM, OpenMP, Pthreads, CUDA, OpenCL, OpenACC and their valid combinations
- All major platforms
- Robust ecosystem
- PAPI counters
- Part of Vi-HPS package



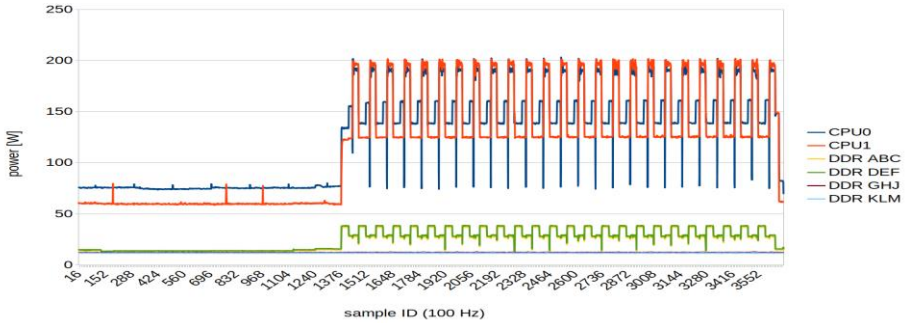


PERFORMANCE TUNING - RECAP

- General rules:
  - File I/O, disk operations
  - Network communication, MPI
  - Data locality, caches
  - Data transfers between CPUs and accelerators

ENERGY EFFICIENCY

- Huge supercomputers have requirements in energy and power consumption management
- MERIC runtime system
  - Developed under H2020 READEX project
  - C/C++ lightweight library for an MPI/OpenMP applications behavior analysis and tuning from energy consumption point of view
  - Dynamic CPU core and uncore frequency, CPU+GPU power limits, number of active threads (concurrency throttling), GPU memory and SM frequency tuning
  - Wide support of power monitoring systems
- RADAR provides interactive visualization of MERIC's measurements



Uncore freq [GHz] / Core freq [GHz]	1.2	1.4	1.6	1.8	2.0	2.2	2.4	2.6	2.8	3.0
1.2	13,200.02	12,717.1	12,621.78	12,410.62	12,380.68	12,507.38	12,774.16	13,108.6	13,604.2	14,040.8
1.4	13,161.9	12,597.78	12,125.18	12,065.52	12,074.54	12,173.36	12,312.24	12,802.26	13,095.84	13,450.8
1.6	13,320.66	12,640.76	12,256.22	12,033.62	11,966.36	11,992.7	12,372.04	12,579.22	13,126.44	13,370.24
1.8	13,878.04	13,082.66	12,700.92	12,457.08	12,373.86	12,445.98	12,574.6	12,831.82	13,081.62	13,296.04
2	14,218.58	13,327.12	12,902.62	12,544.82	12,456.82	12,494.8	12,680.32	13,038.86	13,207.38	13,474.8
2.2	14,625.62	13,849.58	13,240.14	12,851	12,760.98	12,802.24	12,993.44	13,260.38	13,497.6	13,767.62
2.4	15,083.2	14,412.62	13,568.68	13,447.18	12,973.38	13,238.6	13,332.7	13,388.7	13,777.68	14,030.66
2.5	15,554.96	14,465.2	13,991	13,553.84	13,300.24	13,354.46	13,472.36	14,179.16	14,083.06	14,231.3

POP COE

- A Centre of Excellence
  - 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!
- https://www.pop-coe.eu/

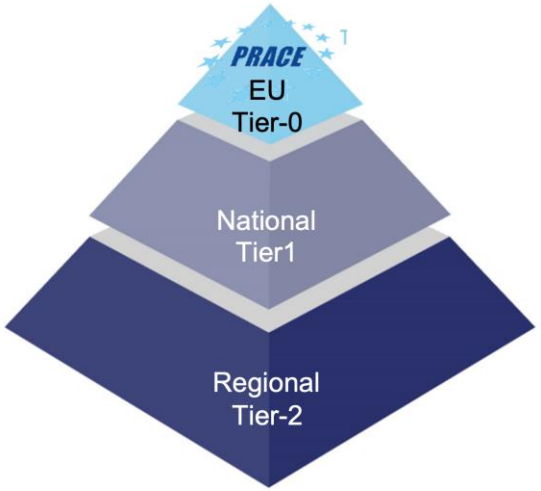
POP COE

- Parallel Application Performance Assessment
  - Primary service
  - Identifies performance issues of customer code (at customer site)
  - If needed, identifies the root causes of the issues found and qualifies and quantifies approaches to address them (recommendations)
  - Combines former Performance Audit (?) and Plan (!)
  - Medium effort (1-3 months)
- 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 experts and customer!

PRACE ENHANCING HLST

- High Level Support Teams
- The project aims for HPC users with large amount of CPU hours consumption (order of millions)
- Its goal is to help these users with transition from Tier-1 system (IT4I) to Tier-0 systems (Any PRACE Tier-0 machine)
- Help them port their applications to targeted Tier-0 system architecture
- Optimize their applications in this process to achieve better performance
- Finally migrate the user to this Tier-0 system to release computational capacity on Tier-1 system
- https://prace-ri.eu/training-support/high-level-support-teams/



Europe's HPC Provisioning Pyrmid
(source: prace-ri.eu)

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://www.vi-hps.org/
- https://docs.it4i.cz/general/accessing-the-clusters/graphical-user-interface/vnc/
- 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