VSB TECHNICAL UNIVERSITY OF OSTRAVA | IT4INNOVATIONS NATIONAL SUPERCOMPUTING CENTER

INTRODUCTION TO PERFORMANCE ANALYSIS, TOOLS, AND POP METHODOLOGY

Tomáš PANOC, Radim VAVŘÍK
Infrastructure Research Lab, IT4I

6. 6. 2023

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

MINISTRY OF EDUCATION, YOUTH AND SPORTS

EURO CZECHIA

PERFORMANCE, METHODS, AND TOOLS

PERFORMANCE ON HPC SYSTEMS I

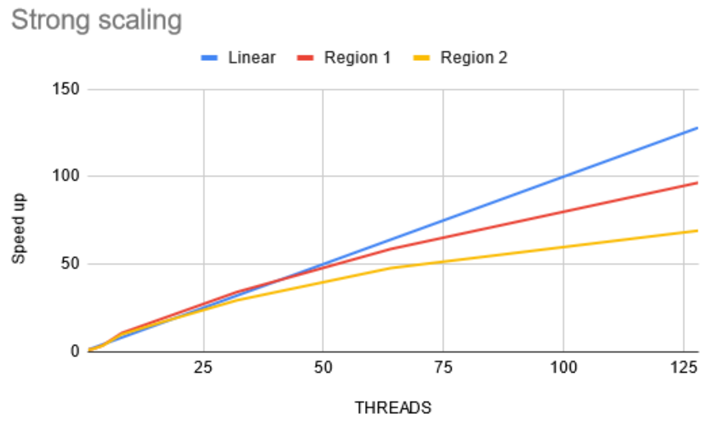
- Why should I be interested in the performance of my code?
- Naturally, one wants results of a computation in the shortest possible time
  - Maximum possible utilization of hardware
  - Large scale of measurements
  - Meeting the deadlines of projects, papers, etc.
- Understanding program behavior better
- Access to HPC machines is usually granted through an open competition
  - Result of the competition is a finite amount of computation time (core/node hours) for an applicants' project
  - An effort to spent the assigned time wisely
- Some HPC centers require a presentation of the code's performance in their application forms

PERFORMANCE ON HPC SYSTEMS II

- What is hidden behind an optimal computational performance?
- Efficient utilization of computational resources, i.e., CPUs, accelerators (GPUs), interconnection, and storage system
- Proper usage of programming models (MPI, OpenMP, CUDA, or a hybrid approach), compilers, and libraries
- Implementation of algorithms which take into account:
  - Hardware features of a single core: memory, caches, vector instructions, NUMA domains, input/output
  - Multicore and distributed environment: work distribution and decomposition, communication, synchronization, multithreading, parallel I/O

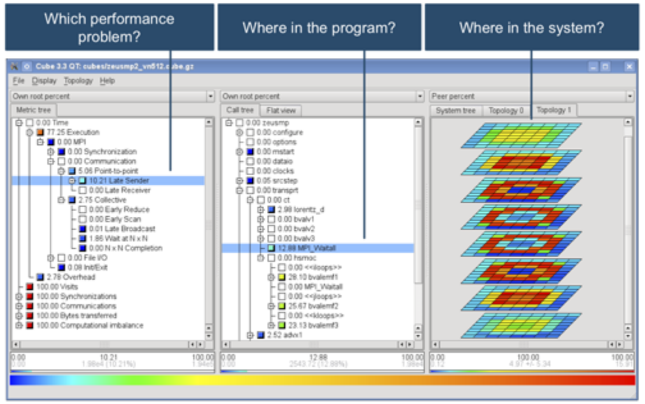
EVALUATING THE PERFORMANCE

- Collecting data and visualizing them
  - Time measurements: full run, specific routines, differences between processes/threads
  - Weak/strong scaling charts
  - Size of data transmitted between processes
  - Frequency of events (how often a routine was called)
  - Performance counters (instructions, cycles, cache misses)



- How to obtain them?
  - Own implementation of an instrumentation layer or a verbose mode
  - Performance tools

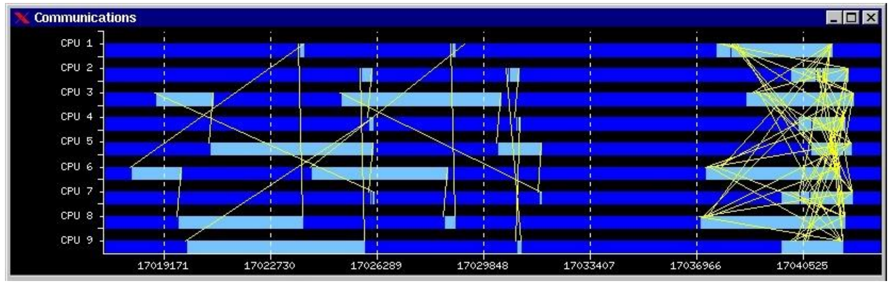
Source: tools.bsc.es



Source: scalasca.org

CLASSIFICATION OF PERFORMANCE TOOLS

- There are various tools differing in a type of information they provide, in a measurement technique, and in ease of use
- Measurement triggering techniques
  - Sampling
  - Code instrumentation
- Form of data gathering and measurement output
  - Profiling and profiles
  - Tracing and traces
- Analysis of the results could be done
  - Online = during a run
  - After a monitored run (post mortem)



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

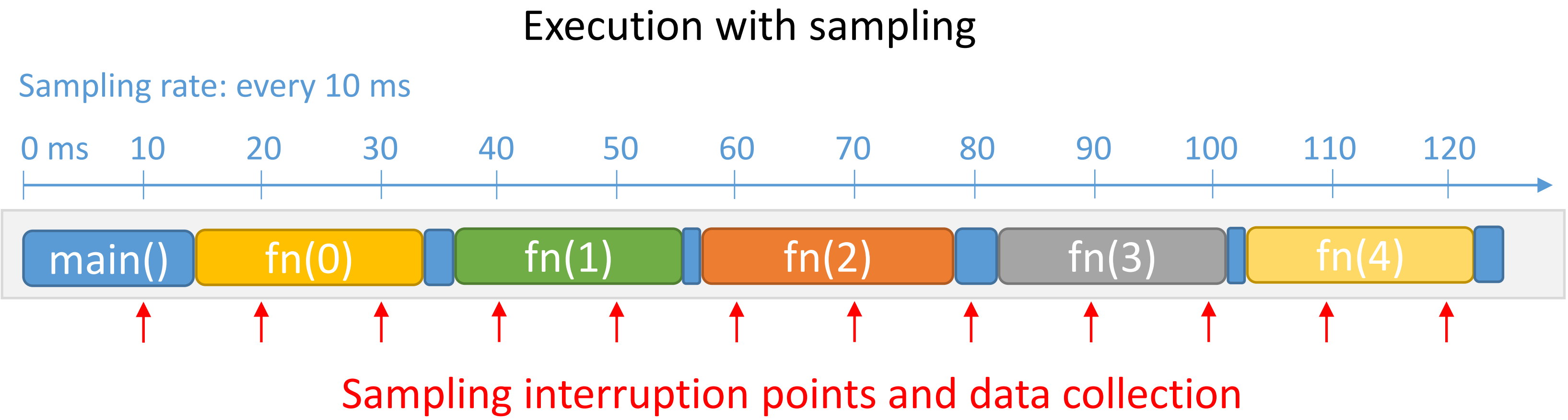
SAMPLING

- Based on interruptions of running application
  - Could be based on HW interrupts, OS timer and signals, or HW counter overflow
- A state of the application is checked when interrupted
  - Call-stack, current line of code, each process/thread, used instructions (e.g., vectorization check), memory consumption
- Advantages
  - Low overhead
  - No modifications to code or executable are necessary
  - Good for statistical performance evaluation
- Disadvantages
  - Less details about the run, lower precision (grows with sampling rate)
  - Requires sufficiently long run to collect enough samples
- Software: Arm MAP, MAQAO LProf, Extrae, Intel VTune, Nsight Systems

SAMPLING EXAMPLE

```
int main() {
  for (int i = 0; i < 10; i++)
  {
    fn(i);
  }
  return 0;
}

void fn(int i) {
  print(i^2);
}
```



CODE INSTRUMENTATION

- An extra code is inserted into regions of interest inside a monitored application
  - Manual instrumentation vs. Automatic instrumentation
  - Static instrumentation vs. Dynamic instrumentation
- Advantages
  - A lot of performance information
  - Can be directed to a specific region by manual instrumentation
- Disadvantages
  - Changes in a source code or an executable are necessary
  - Larger overhead, especially when instrumenting small frequently called functions
- Software: Score-P, Extrae, Nsight Systems (NVTX)

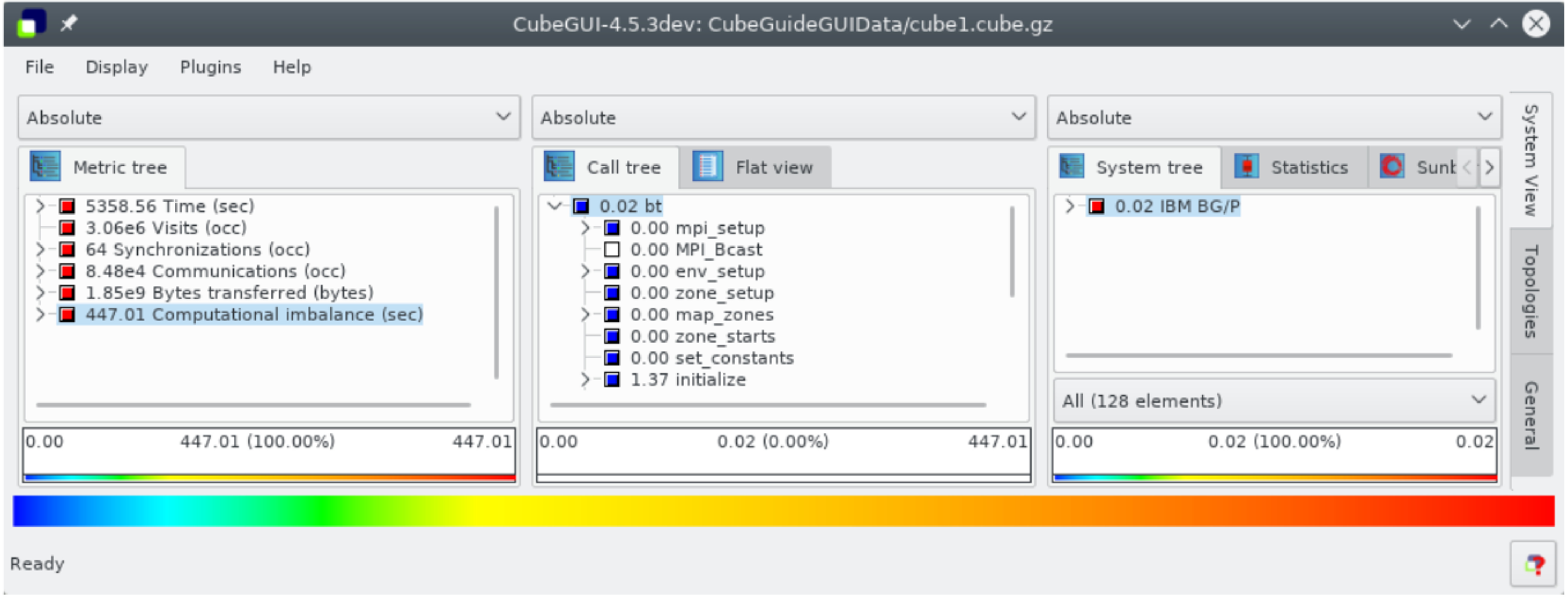
```
int main() {
  BEGIN(“main”);
  for (int i = 0; i < 10; i++)
  {
    fn(i);
  }
  END(“main”);
  return 0;
}

void fn(int i) {
  BEGIN(“fn”);
  print(i^2);
  END(“fn”);
}
```

PROFILING

- Description of a run with numbers and metrics
  - Time spent in a routine, number of visits, performance counter data, transferred bytes
  - Statistical information (max, min, mean,…)
- Provides information about program entities
  - Functions, blocks, loops, API calls (MPI, OpenMP)
  - Processes, threads, GPU kernels
  - The measured metrics are matched with corresponding entities
- Flat profile – a list of called entities without a calling context
- Call-path profile – a call-tree with a hierarchy of the program entities
- Software: Score-P + Cube (+ Scalasca), Extrae + Paraver, Intel VTune, Nsight Systems, Arm MAP, MAQAO

PROFILE EXAMPLE: CUBE

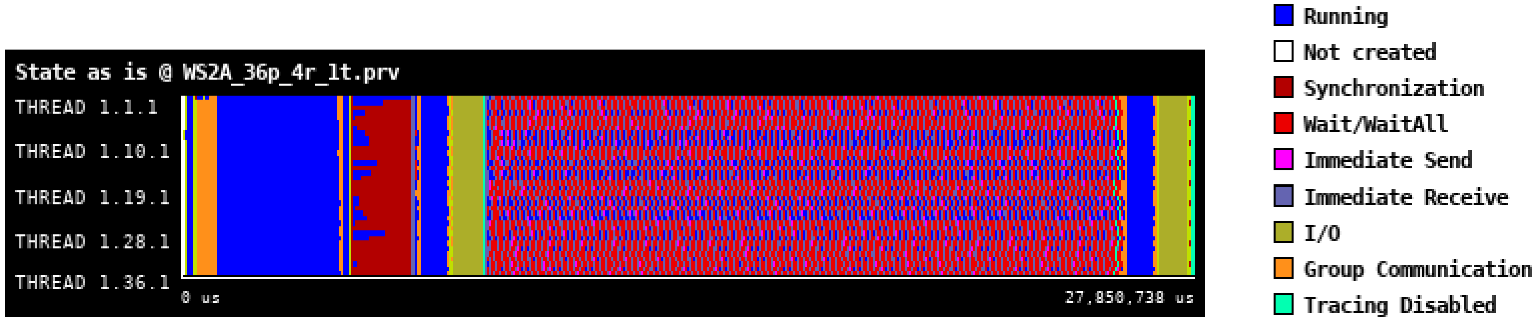


Source: scalasca.org

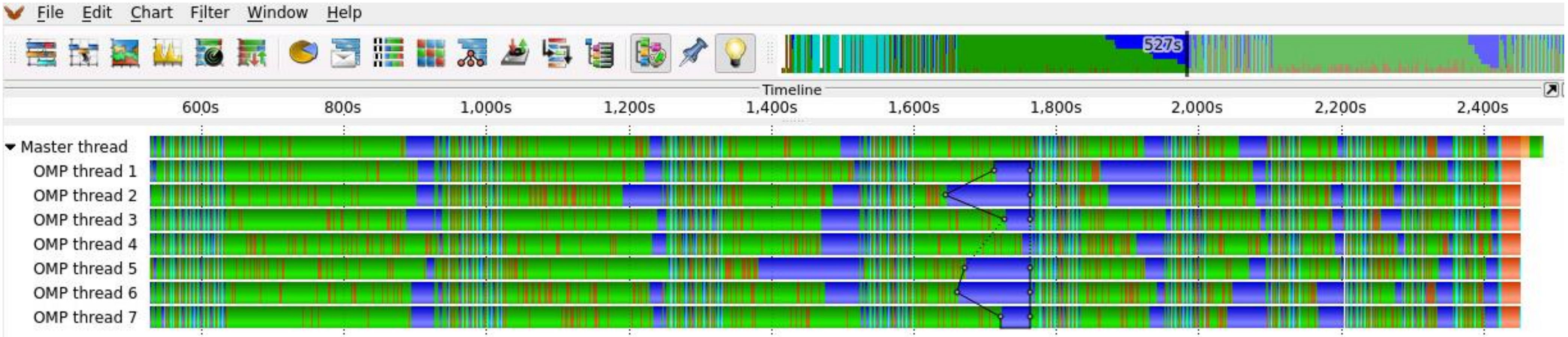
TRACING

- A complete run record
  - Complete timeline with recorded events for each involved computational resource (process, thread, GPU stream) in original order
  - Most general and detailed measurement method (a profile can be constructed from a trace, e.g. Scalasca)
  - Traces may become large (many processes, long run, too many monitored regions) and writing the events to a file may produce an overhead
- An event
  - Typical examples: enter/leave of a region (function, loop, API call,…) or send/receive of a message
  - An event record contains information including timestamp, location, event type, and other specific data related to the event type
- Software: Extrae + Paraver, Score-P + Vampir, Nsight Systems

TRACE EXAMPLE: PARAVER, VAMPIR



Paraver



Vampir

WHICH WAY TO CHOOSE?

- Technical aspects
  - The performance tools usually support only a limited group of programming languages, models, and hardware architectures
  - Look for what is available for your code
- Practical aspects
  - Tools based on code instrumentation usually require more effort at the beginning than the samplers which may be a good choice to start with
  - If you think you know what might be your performance bottleneck or you are interested in a specific analysis, you can go straight for a particular tool
    - A good overview of performance tools: https://www.vi-hps.org/cms/upload/material/general/ToolsGuide.pdf
  - A good strategy for the performance evaluation would include:
    - Repeated measurements with one input data set to check invariability
    - Testing several input data sets, if possible, to check the impact of different data on the behavior
    - Evaluation of strong/weak scaling
    - Focusing on regions which do matter (i.e., time spent in them is not negligible)
    - Employment of one or more performance tools (measurement + visualization)

POP COE AND METHODOLOGY

WHAT IS POP COE?

- = Performance Optimisation and Productivity Centre of Excellence in HPC
  - https://www.pop-coe.eu
- Free-of-charge service offering performance analysis of HPC codes for academic and industrial entities from European Union
- Apart from the customer service, POP team
  - works on a complete methodology for performance assessments,
  - takes part in the development of tools and provides training events and webinars,
  - continuously prepares a co-design database with patterns and best practices for HPC code development (https://co-design.pop-coe.eu).
- Project partners: BSC (ES), JSC (GE), HLRS (GE), IT4I (CZ), NAG (UK), RWTH (GE), TERATEC (FR), UVSQ (FR), INESC-ID (PT)
- POP1 (10/2015 – 5/2018), POP2 (12/2018 – 5/2022)
  - POP3 in preparation, 1/2024 – 12/2027 if accepted

POP SERVICES

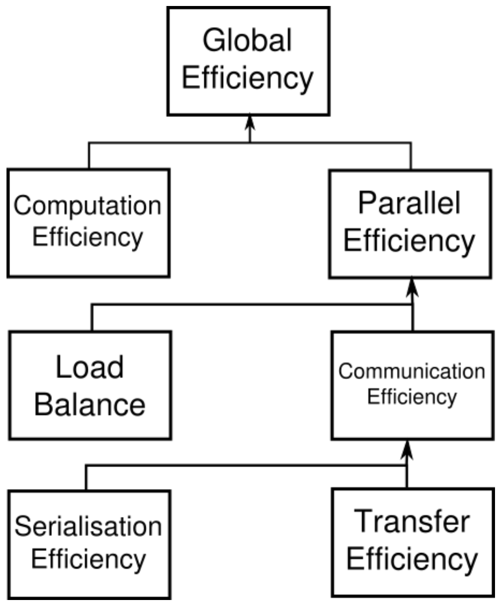
- Performance assessment / audit (PA)
  - Primary service
  - Identifies performance issues of customer’s code
  - Offers recommendations for fixing the found issues
  - Helps customers to better understand application behavior
  - Usually takes 1 – 3 months to complete
- Follow-on study
  - Repeated performance audit with a code that was analyzed already but customer applied changes (e.g., based on findings in the previous audit) or has a different code version
- Proof-of-Concept (PoC)
  - Follows the assessment
  - Fixes suggested during the audit are applied by a POP analyst
    - They can be directly applied to the code or demonstrated with an extracted kernel/mini-app
  - Usually takes 3 – 6 months to complete

PERFORMANCE ASSESSMENT SCENARIO

1. Preparation of environment - installation of all dependencies
2. Instrumentation and test run
3. Measurement with various number of computational resources (scaling test) and input cases
4. Analysis of profiles and traces – identifying focus of analysis
5. Computation of POP efficiency metrics and further analysis
6. Additional measurement and analysis if necessary
7. Summary of findings and recommendations for the customer
8. Output: presentation slides reporting on everything important found during the assessment

POP EFFICIENCY METRICS

- Numbers describing behavior of a selected program region
- Basic set of metrics for MPI/OpenMP codes:
  - Global efficiency (GE)
    - Parallel efficiency (PE)
    - Load balance efficiency (LB)
    - Communication efficiency (CE)
      - Serialization efficiency (SE)
      - Transfer efficiency (TE)
    - Computation efficiency (CompE)
      - Instruction scaling efficiency
      - IPC scaling efficiency
      - Frequency scaling efficiency
- There exist metrics for hybrid (MPI+OpenMP) codes too (pop-coe.eu)

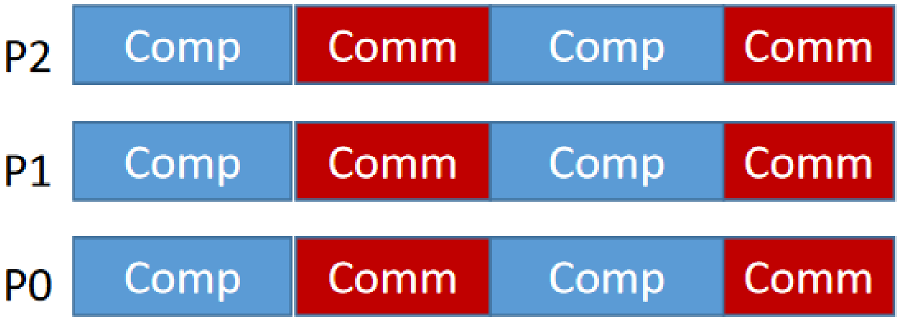


Source: pop-coe.eu

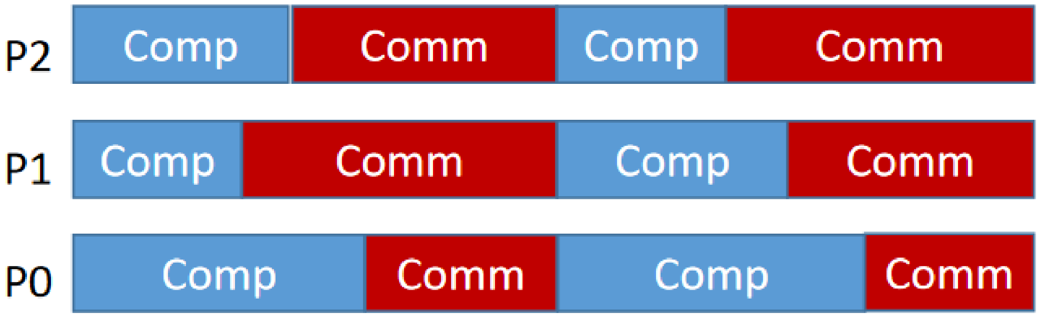
LOAD BALANCE EFFICIENCY

- Indicates how well the distribution of work between processes/threads is done
- It is a ratio of average time spent only in computation in processes/threads and maximum time a process spent only in computation
- $LB = \frac{average(computation\ time)}{\max(computation\ time)}$

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



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

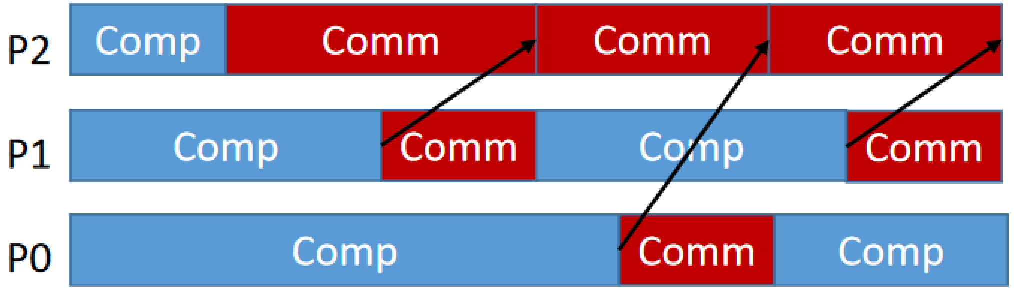


Source: pop-coe.eu

COMMUNICATION EFFICIENCY

- Shows the loss of efficiency caused by communication
- Can be computed directly: $\text{CE} = \max_{processes}\left(\frac{computation\ time}{total\ time}\right)$
- However, it can be split into two components: Serialization and Transfer efficiencies
- Then, $CE = SE * TE$

Example:



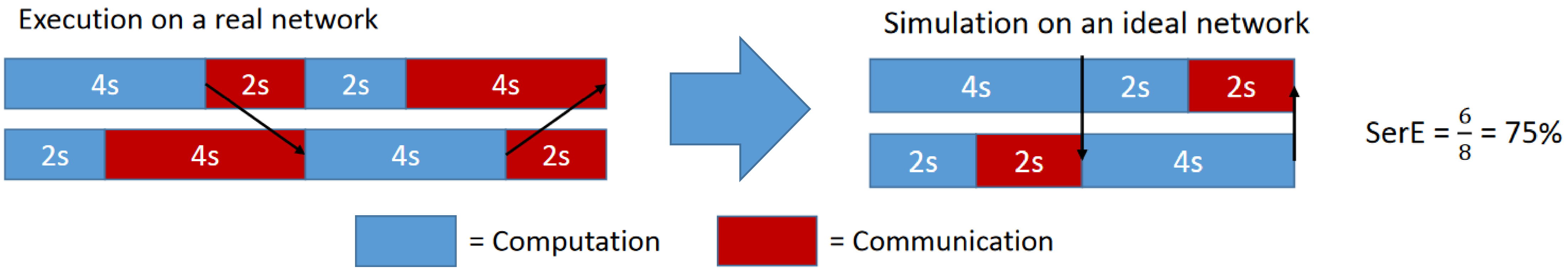
Compute	Communication	Efficiency
1 sec.	5 sec.	$^1/_6$
4 sec.	2 sec.	$^4/_6$
5 sec.	1 sec.	$^5/_6$

CommE = $^5/_6$ = 83%

Source: pop-coe.eu

SERIALIZATION EFFICIENCY

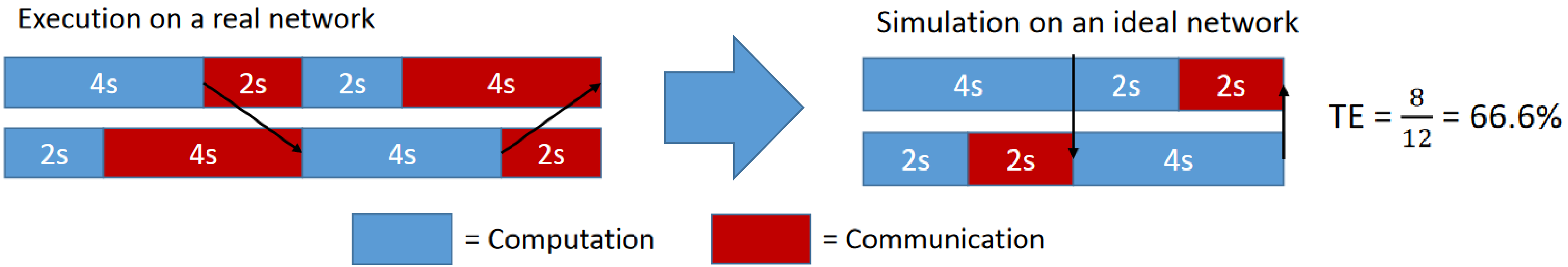
- Dependencies between processes can cause loss that affects serialization efficiency
- In practice, this happens when one process stays in a MPI call waiting for another process which did not get to the corresponding communication call
- Therefore, this problem would persist with an ideal network and instant data transfers
- $SE = \max_{processes}\left(\frac{computation\ time\ on\ ideal\ network}{total\ time\ on\ ideal\ network}\right)$



Source: [pop-coe.eu](pop-coe.eu)

TRANSFER EFFICIENCY

- Represents efficiency loss caused by data transfers
- $TE = \frac{total\ time\ on\ ideal\ network}{total\ measured\ time}$



Source: [pop-coe.eu](pop-coe.eu)

TOP LEVEL EFFICIENCIES

- **Parallel efficiency** describes how well the execution of code in parallel is working
  - $PE = LB * CE$
- **Computation efficiency** describes how well the computational load of an application scales with the number of processes/threads
  - It is computed by comparing the total time spent only in computation for a different number of processes/threads
  - If a program scales linearly the time spent in computation does not change and computation efficiency remains constant (equal to 1)
- **Global efficiency** describes how the parallelization of your code works in general
  - $GE = PE * CompE$

INSTRUCTION SCALING EFFICIENCY

- Computation efficiency can be influenced by instruction scaling, IPC scaling, and frequency scaling efficiencies
  - Also, it can be computed as product of these efficiencies
- Number of instructions can be obtained from HW performance counters
  - PAPI is used primarily for reading the data (PAPI_TOT_INS)
  - Only instructions involved in computation (i.e., useful instructions) are taken into consideration
  - Sum of instructions from all processes/threads is used
- Instruction scaling efficiency is computed by comparing instruction counts of runs with different number of processes/threads
  - It could happen that with more processes more instructions are executed
  - This might be caused for example by a work distribution algorithm which requires more effort to split the job and distribute it among more processes

IPC AND IPC SCALING EFFICIENCY

- IPC – instructions per cycle
  - $IPC = \frac{total\ number\ of\ instructions}{total\ number\ of\ cycles}$
  - Uses instructions and cycles spent in computation only (i.e., useful instructions and cycles)
  - Uses data from all processes/threads
  - HW performance counters need to be recorded (PAPI is used primarily)

- IPC scaling efficiency is computed by comparing IPCs of runs with different number of processes/threads
  - Low IPC may indicate long memory waits (memory bound problem) when number of instructions is almost constant but cycles increase

FREQUENCY SCALING EFFICIENCY

- CPU frequency (Hz)
  - $CPU\ frequency = \frac{total\ number\ of\ cycles}{total\ computation\ time}$
  - Uses cycles and time spent in computation only (i.e., useful cycles and time)
  - Uses data from all processes/threads
  - HW performance counters need to be recorded (PAPI is used primarily, PAPI_TOT_CYC)

- Frequency scaling efficiency is computed by comparing frequencies of runs with different number of processes/threads
  - Frequency might change when vector instructions are used (AVX, AVX2, AVX512)
  - They cause decrease of frequency

USEFUL WEBSITES AND DOCUMENTS

- Performance and tools
  - 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
- POP COE
  - Application form for an analysis (PA/PoC): pop-coe.eu
  - Database of analyzed codes, patterns, and best practices for particular parallel programming situations: co-design.pop-coe.eu
  - Materials for learning (POP methodology) including a guide for creating an assessment on your own: https://pop-coe.eu/further-information/learning-material
  - Webinars including tutorials for the tools and the methodology, and presentations of successful assessments: https://pop-coe.eu/further-information/webinars

THANK YOU FOR YOUR ATTENTION

ACKNOWLEDGMENTS

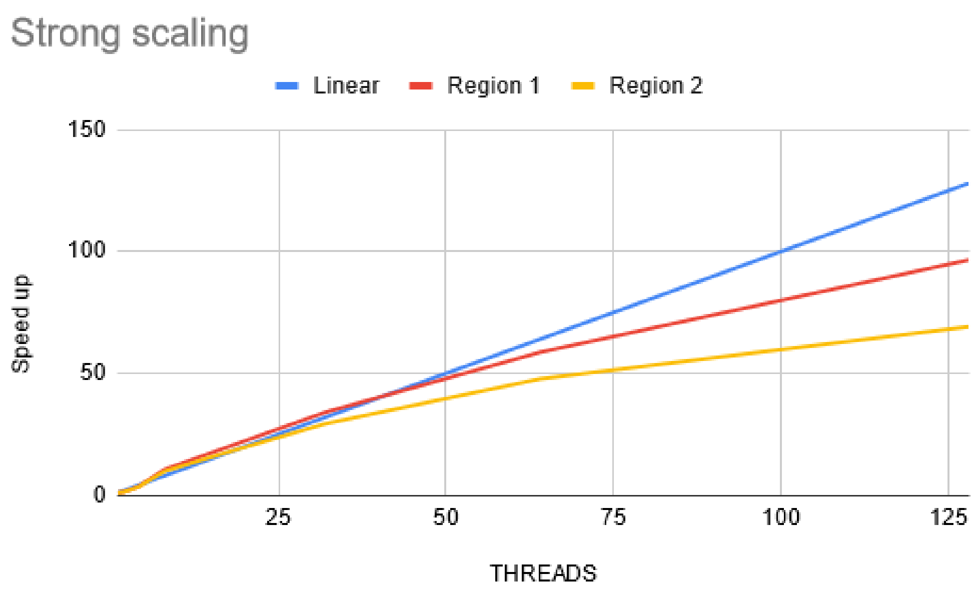
Funded by the European Union. This project has received funding from the European High Performance Computing Joint Undertaking (JU) and Germany, Bulgaria, Austria, Croatia, Cyprus, the Czech Republic, Denmark, Estonia, Finland, Greece, Hungary, Ireland, Italy, Lithuania, Latvia, Poland, Portugal, Romania, Slovenia, Spain, Sweden, France, the Netherlands, Belgium, Luxembourg, Slovakia, Norway, Türkiye, Republic of North Macedonia, Iceland, Montenegro, Serbia under grant agreement No 101101903. This project has received funding from the Ministry of Education, Youth and Sports of the Czech Republic.

APPENDIX

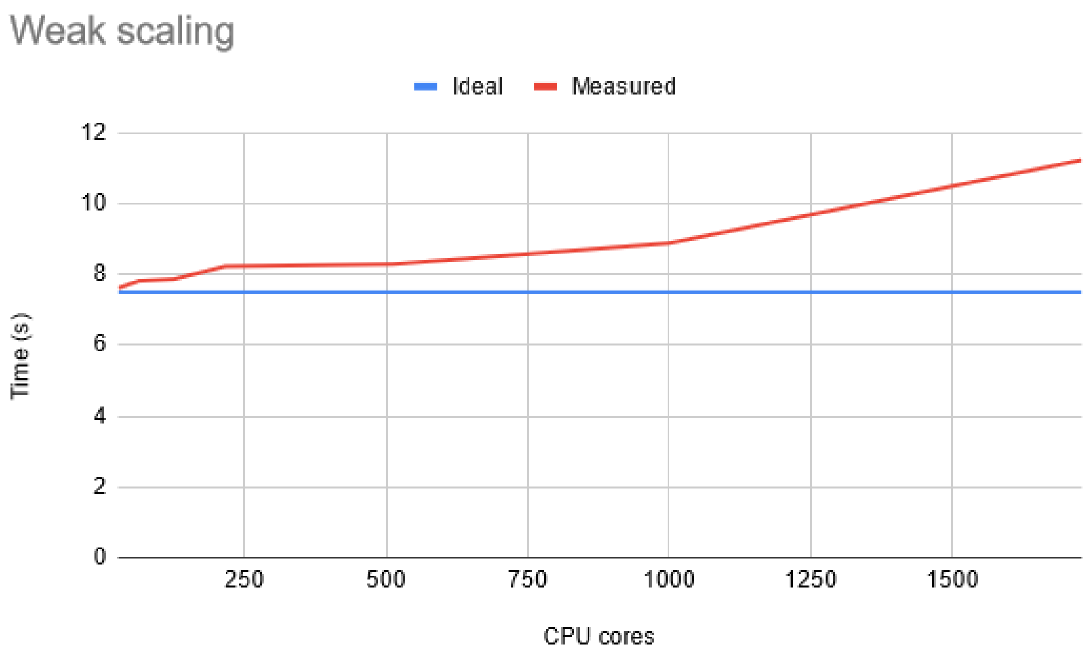
SCALABILITY / SCALING I

- Ability of program to achieve greater performance when number of computational resources is increased
- **Strong scaling** tests how the computation time differs with the number of computational resources (CPUs) for a fixed problem size
  - Ideal (linear) scaling would mean that if we double the number of resources, we achieve two times faster execution
  - If the achieved speed up is greater than linear, it is called superlinear scaling
- **Weak scaling** tests how the computation time differs with the number of computational resources (CPUs) for a fixed problem size per a computation unit (a CPU core)
  - Ideal scaling would mean that if we double the number of resources and we double the problem size too, the total run time does not change (remains constant)

SCALABILITY / SCALING II



Example of strong scaling of an application where two regions were monitored. Region 1 achieves superlinear scaling at the beginning.



Example of weak scaling