POP CoE performance metrics: a beginner's guide with Score-P, Scalasca & CUBE

16.06.2026|Ilya Zhukov, Marc Schlütter, Jülich Supercomputing Centre

HORIZON-EUROHPC-JU-2023-COE

EuroHPC Joint Undertaking

1 January 2024– 31 December 2026

Grant Agreement No 101143931

POP CoE

- A **Centre of Excellence**
  - On **Performance Optimisation and Productivity**
  - Promoting **best practices in parallel programming**
- Providing **FREE Services**
  - Precise understanding of application execution and system behaviour
  - Suggestion/support on how to refactor code in the most productive way
- **Horizontal**
  - Transversal across application areas, platforms, scales
- **For (EuroHPC) academic AND industrial codes and users !**

Partners

- **Who?**
  - BSC, ES (coordinator)
  - HLRS, DE
  - INESC-ID, PT
  - IT4I, CZ
  - JSC, DE
  - RWTH Aachen, IT Center, DE
  - TERATEC, FR
  - UVSQ, FR

A team with

- Excellence in performance tools and tuning
- Excellence in programming models and practices
- Research and development background AND
  proven commitment in application to real academic and industrial use cases

Motivation

Why?

- Complexity of parallel computers and codes
  - ⇨ Frequent lack of quantified understanding of actual behaviour
  - ⇨ Not clear most productive direction of code refactoring
- Important to maximize efficiency (performance, energy usage) of compute intensive applications and productivity of development efforts

What?

- Parallel programs, mainly MPI & OpenMP
  - Although also OpenACC, OpenCL, CUDA/HIP, …

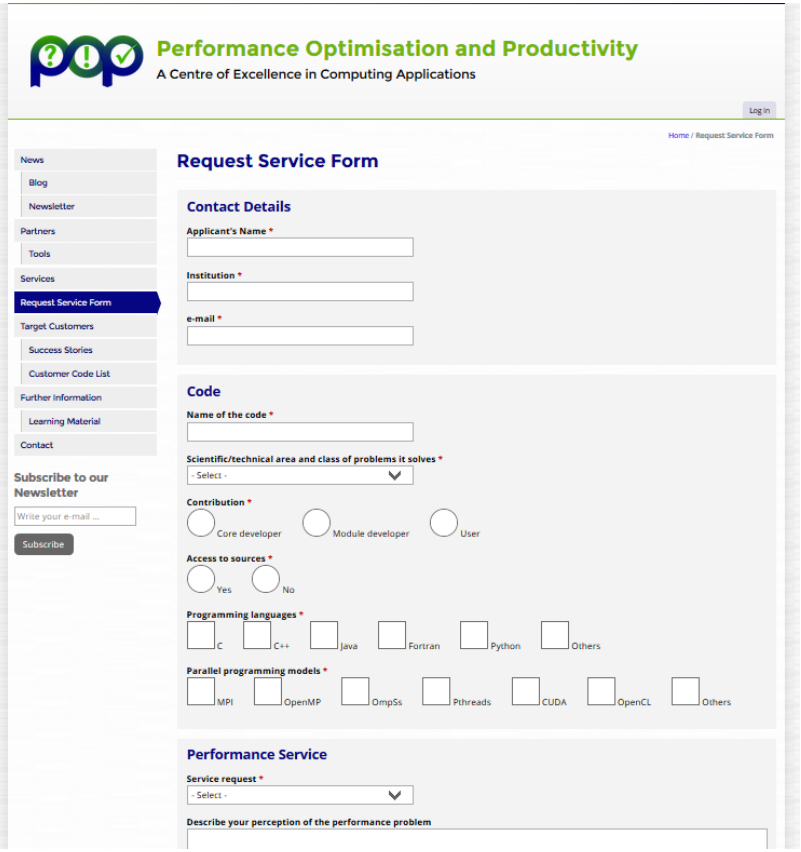
The Process …

When?

January 2024 – December 2026

How?

- Apply
  - Fill in short questionnaire describing application and needs https://pop-coe.eu/request-service-form
  - Questions? Ask pop@bsc.es
- Selection/assignment process
- Install tools @ your production machine (local, EuroHPC, …)
- Interactively: Gather data → Analysis → Report



EuroHPC Joint Undertaking

What are efficiencies?

- POP developed a methodology for analysis of parallel codes to provide a quantitative way of measuring relative impact of the different factors inherent in parallelization
- A hierarchy of metrics
- Each metric reflects a common cause of inefficiency in parallel programs
- Calculated as values between 0 and 1 (the higher the better)
- Values less than 0.8 indicate performance issues (empirical value)

What are efficiencies' benefits?

- Easy to identify which characteristics of the code contribute to inefficiency
- Allow to compare parallel performance (e.g. over a range of thread/process counts, across different machines, or at different stages of optimisation and tuning)
- Easy to compute (automatic standardized procedure)
- Single run is sufficient to compute basic efficiencies
- Relatively easy to understand
- Applicable to various parallelism paradigms (MPI, OpenMP, MPI+OpenMP, …)

Efficiencies' hierarchy

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

Global Efficiency

- Describes overall quality of parallelisation i.e.
  - Overheads imposed by the parallel nature of a code, quantified with Parallel Efficiency
  - Poor scaling of computation with increasing numbers of processes, quantified with Computation Efficiency

$$GE = PE \cdot CompE$$

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

Parallel Efficiency

- Describes how well the execution of the code in parallel is working, i.e.
  - distribution of computational work across processes, quantified with Load Balance Efficiency
  - communication across processes, quantified with Communication Efficiency

$$PE = LB \cdot CommE$$

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

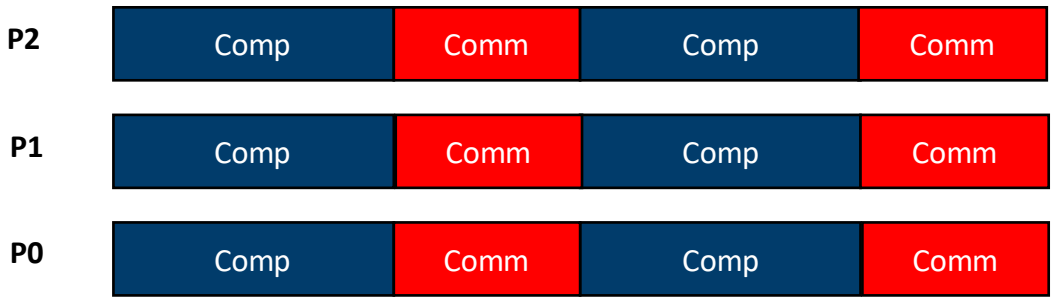
Load Balance Effciency

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

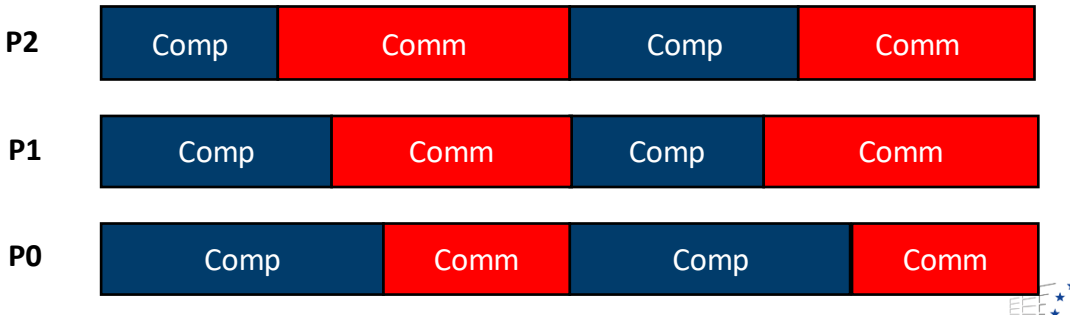
$$LB = \frac{avg(computation\ time)}{\max(computation\ time)}$$

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

Example 1: good load balance (LB = 1.0)



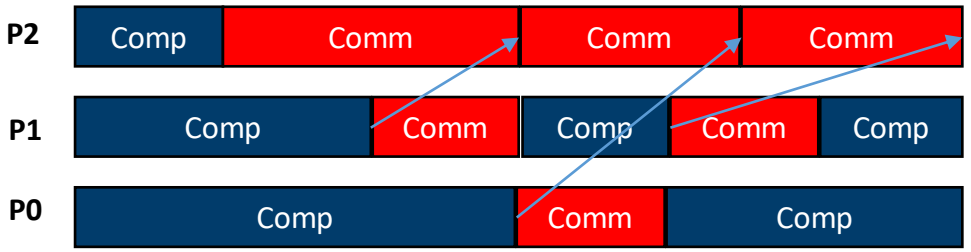
Example 2: bad load balance (LB = 0.77)



Communication Efficiency

- Reflects the loss of efficiency by communication

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



Comp	Comm	CommE
1 s	5 s	1/6
4 s	2 s	4/6
5 s	1 s	5/6

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

CommE = 5/6 = 0.83

There are two possible reasons for low value of Communication Efficiency

- Inefficiency due to waiting time within communications, quantified with Serialization Efficiency
- Inefficient data transfer, quantified with Transfer Efficiency

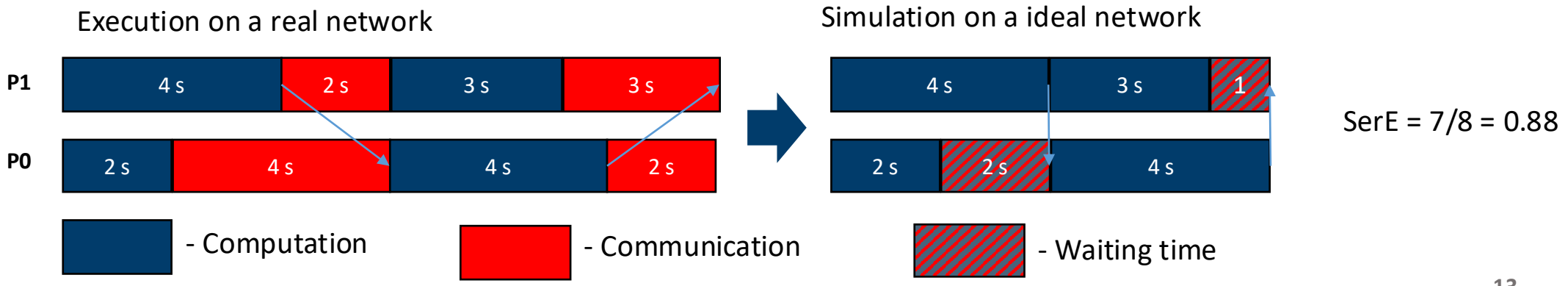
$$CommE = SerE \cdot TE$$

Serialization Efficiency

- Describes loss of efficiency due to dependencies among processes, where dependencies can be observed as waiting time in MPI calls, e.g. Late Sender
- Serialization efficiency computed for idealised network where transmission of data takes zero time

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

$$SerE = \max_{processes}\left(\frac{computation\ time\ on\ ideal\ network}{total\ runtime\ on\ ideal\ network}\right)$$



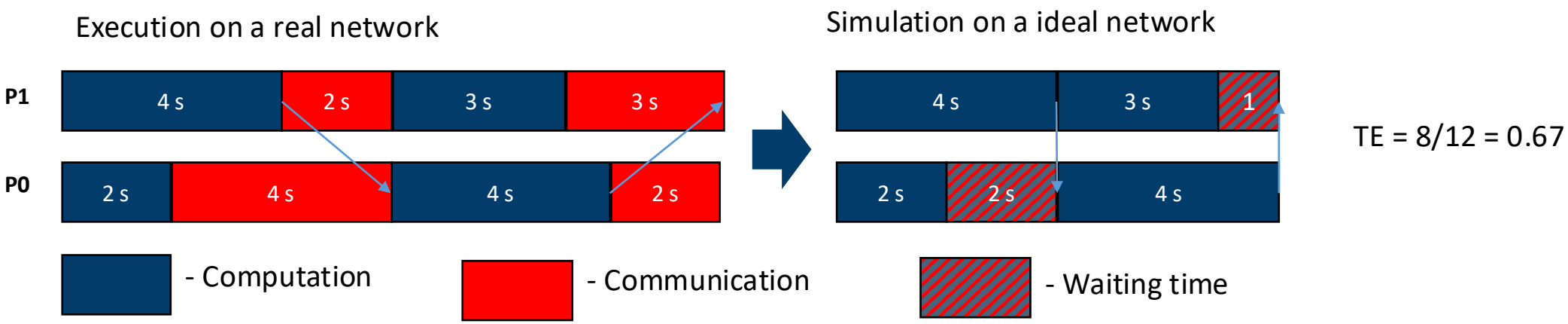
SerE = 7/8 = 0.88

Transfer Efficiency

- Describes loss of efficiency due to actual data transfer

$$TE = \frac{total\ runtime\ on\ ideal\ network}{actual\ total\ runtime}$$

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



TE = 8/12 = 0.67

Computation Efficiency

- Describes efficiency of computational load of application at scale (strong scaling)
- Computation efficiency of a linearly-scaling application strives towards one (can be greater than one, e.g. superlinear speedup)

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

$$CompE = \frac{computation\ time\ on\ 1\ process}{computation\ time\ on\ N\ processes}$$

Instruction and IPC Efficiencies

- There are two possible reasons of low value of Computation Efficiency:
  - Work division over additional processes increases the total computation required, quantified with Instruction Efficiency

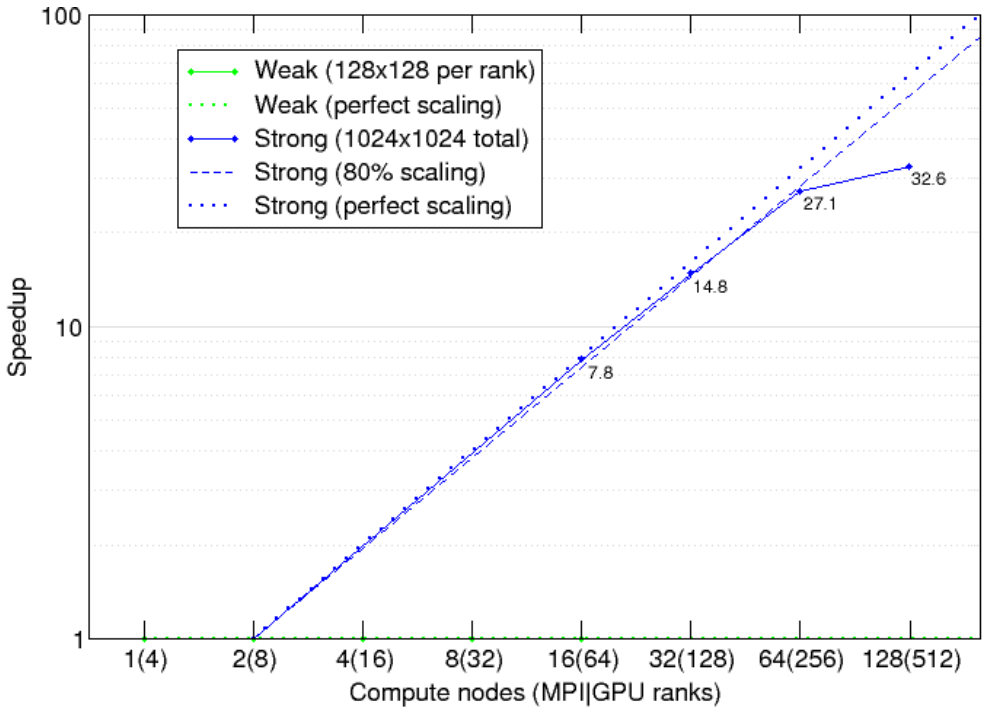
$$InstrE = \frac{amount\ of\ instructions\ on\ 1\ process}{amount\ of instructions\ on\ N\ processes}$$

  - Using additional processes leads to contention for shared resources, quantified with Instructions Per Cycle (IPC) Efficiency

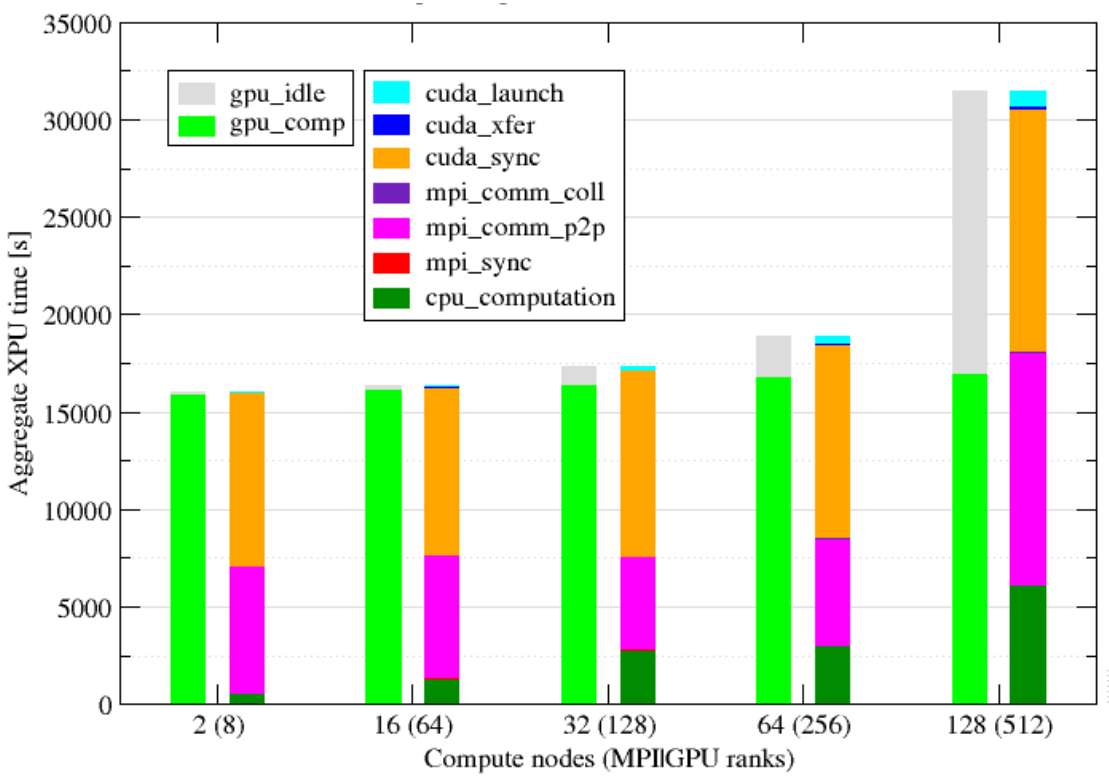
$$IPCE = \frac{amount\ of\ IPC\ on\ N\ processes}{amount\ of\ IPC\ on\ 1\ process}$$

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

SPECFEM3D@Leo-B strong scaling



Problem size	1024x1024	1024x1024	1024x1024	1024x1024	1024x1024
MPI|GPU ranks	**8**	**64**	**128**	**256**	**512**
Wall time [s]	**2001.806**	**255.948**	**135.478**	**73.846**	**61.400**
Global scaling efficiency	0.995	0.973	0.919	0.843	0.507
- Computation time scaling	*1.000*	0.987	0.972	0.950	0.937
- Parallel efficiency	0.995	0.986	0.945	0.887	0.541
- - Load balance efficiency	1.000	0.998	0.996	0.994	0.989
- - Orchestration efficiency	0.995	0.988	0.948	0.892	0.547



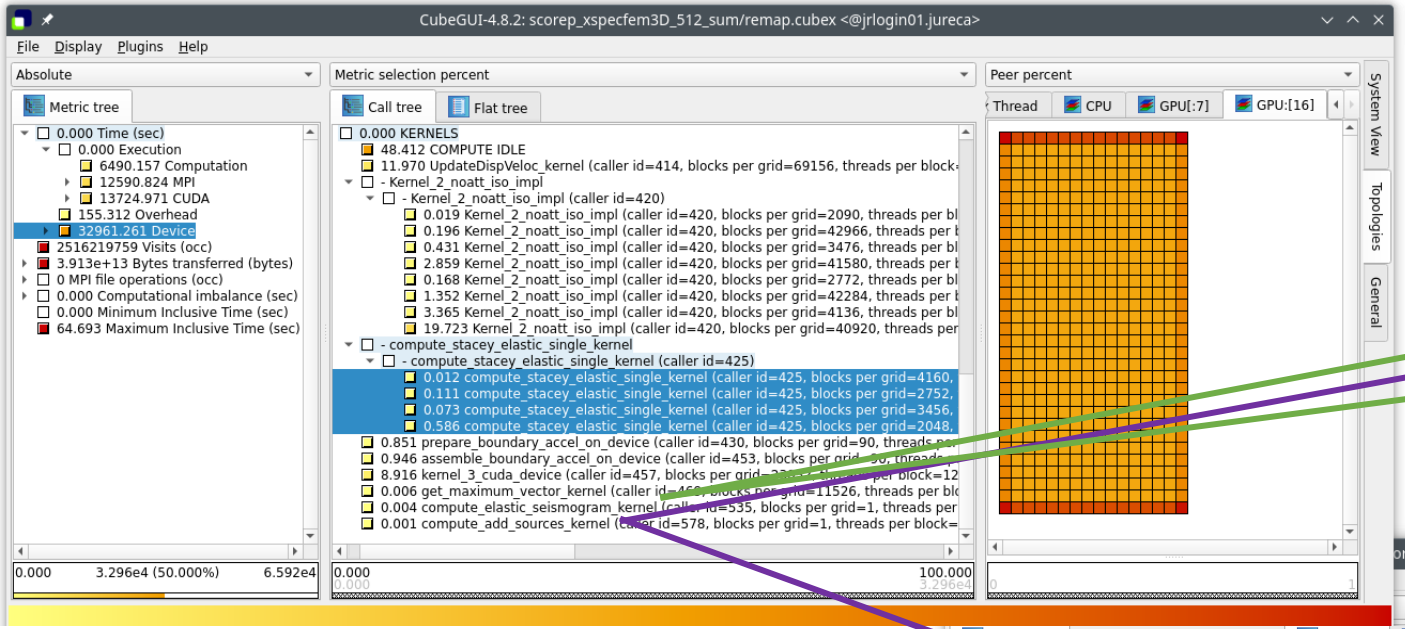
ChEESE

POP3_AR_002

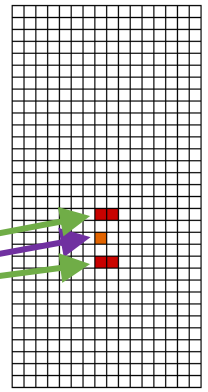
- Fortran90 parallelised with MPI+CUDA (1 rank/GPU)
- *iterate_time* (solver) chosen as Focus of Analysis
- Good strong scaling up to 256 GPUs
- With 512 GPUs no longer able to sufficiently overlap MPI communication with CUDA kernels

DOI 10.5281/zenodo.13643996

SPECFEM3D@Leo-B (512 GPUs)

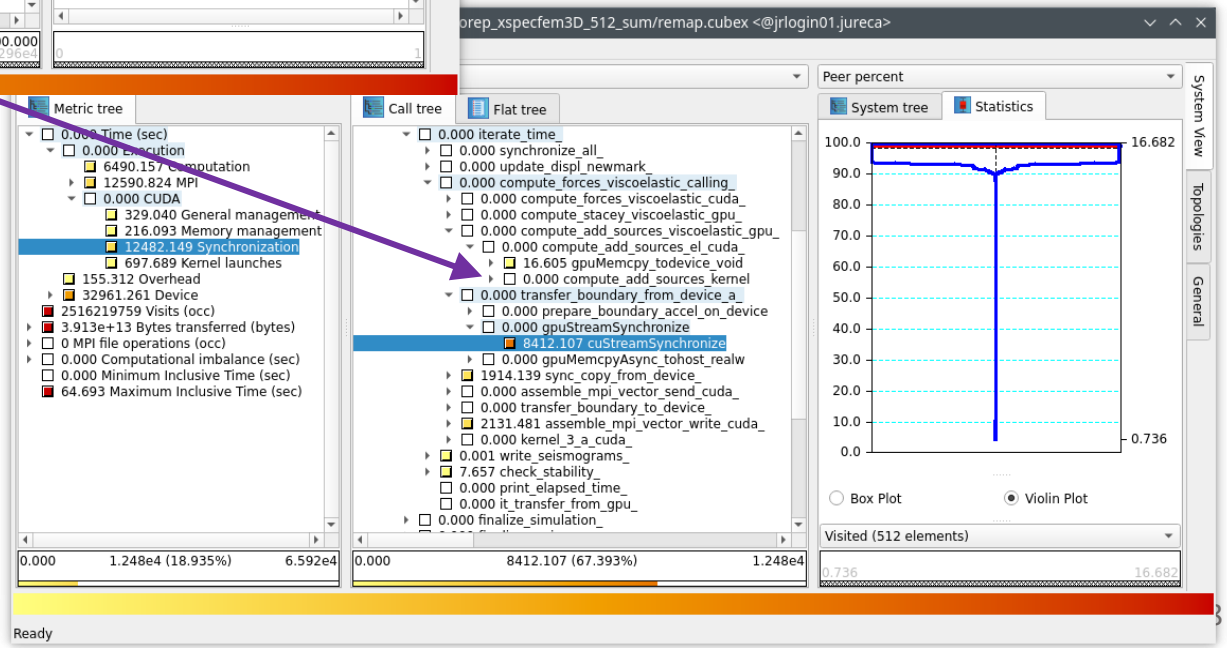


16x32 grid

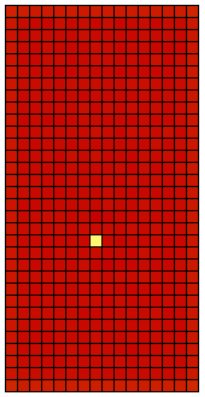


2D domain decomposition on GPUs

- many kernels are well balanced
- some kernels execute much faster for interior compared to edges
- only four GPUs handle seismogram receivers
- only one GPU (#243) executes *compute_add_sources_kernel*



16x32 grid



compute_add_sources_kernel executed on single GPU (#243) is rather short, however, results in all other GPUs having very long synchronization times in following *transfer_boundary_from_device_a*

- over two-thirds of CUDA synch time and over 30% of total CPU time

DOI 10.5281/zenodo.13643996

EuroHPC Joint Undertaking

How to get efficiencies: With Score-P

- Available metrics depend on data in Cube file/measurement:
  - Basic set almost always available
  - Counters required for IPC and instruction information
  - Tracing required for MPI Serialization and Transfer Efficiencies
  - GPU device side activities for GPU metrics

- Procedure:
  - Measure required/intended data
    - Merge data if necessary
  - Use Cube to calculate metrics:
    - Cmd line: cube_pop_metrics
    - GUI: POP Advisor panel

Scalasca multi-run for convenience

Scalasca Multi-run

- Multiple runs through configurations, repetitions or a combination of both
  - Aggregation into singular Cube result via *square*
- Configurations:
  - Set of environment variables for a run; User and Score-P/Scalasca variables
  - For consistency, used Scalasca and Score-P variables have to be stated in the config file
  - **Use case**: different application/measurement settings, e.g., varying hardware counters, input sets
  - **Note**: Runs with too much variance in application behaviour might prevent combination
- Runs per configuration:
  - Number of runs with the same configuration (default environment or a specific run configuration)
  - **Use case**: statistical stability/analysis

- **Preset: Predefined multi-run configuration – current use case pop metrics**

Scalasca Multi-run: POP preset

```
% scan
Scalasca 2.6.2: measurement collection & analysis nexus
usage: scan {options} [launchcmd [launchargs]] target [targetargs]
       where {options} may include:
...
  -R #runs          : Specify the number of measurement runs per config.
  -M cfgfile        : Specify a config file for a multi-run measurement.
  -P preset         : Specify a preset for a multi-run measurement, e.g., 'pop'.
  -L                : List available multi-run presets.
  -D cfgfile        : Check a multi-run config file for validity and dump
                    : the processed configuration for comparison.
```

```
% square
Scalasca 2.6.2: analysis report explorer
usage: square [OPTIONS] <experiment archive | cube file>
...
  -S <mean | merge>          : Aggregation method for summarization results of
                               each configuration (default: merge)
  -T <mean | merge>          : Aggregation method for trace analysis results of
                               each configuration (default: merge)
  -A                         : Post-process every step of a multi-run experiment
  -I                         : Ignore structural sanity checks and force aggregation
                               of measurements in a multi-run experiment
```

Scalasca Multi-run: Config file format

Global Section '--'
First one, max. one

Run Section '-'
Everything after '-' is comment

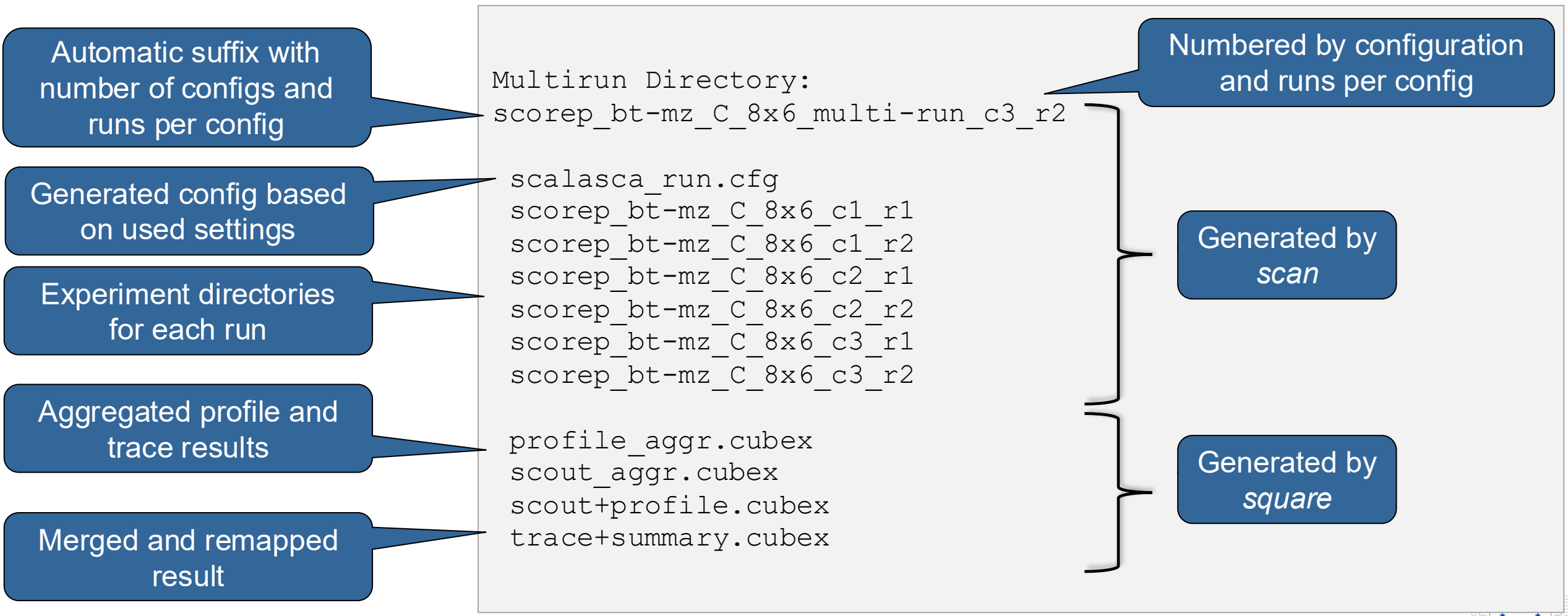
KEY=VALUE Pair
Everything after first = is taken as value

Comments '#' and blank lines are ignored

Run Settings supersede Global settings

```
-- Global variable section marked by '--', comment optional
SCOREP_ENABLE_PROFILING=true
SCOREP_ENABLE_TRACING=false
# for all runs, not doing so will result in un-mergable results
SCOREP_FILTERING_FILE=../config/scorep.filt

# Begin of the per run settings each marked with a '-'
- Profiling run with PAPI counters
SCOREP_METRIC_PAPI=PAPI_TOT_INS,PAPI_TOT_CYC,PAPI_RES_STL

- Profiling without counters to get a low overhead measurement
# no additional Score-P variables needed

- Scalasca tracing run
SCOREP_TOTAL_MEMORY=80M
SCOREP_ENABLE_PROFILING=false
SCOREP_ENABLE_TRACING=true
SCAN_ANALYZE_OPTS=--time-correct
```

Scalasca Multi-run: Results



SCALASCA MULTI-RUN DEMO

Demo: POP Preset (shortened)

```
-- POP preset for Cube Advisor: hardware counter measurement and trace collection/analysis

# Variables that interfere with the preset when set in the environment
SCOREP_CUDA_ENABLE=no
SCOREP_METRIC_PAPI_SEP=','
SCOREP_METRIC_PERF_PER_PROCESS=''
...

# Variables that may adversely impact measurement overhead
SCOREP_ENABLE_UNWINDING=false
SCOREP_MEMORY_RECORDING=false
...
SCOREP_VERBOSE=false


# timing information is used from tracing run
- Scalasca tracing run
SCOREP_ENABLE_PROFILING=false
SCOREP_ENABLE_TRACING=true

- Profiling run with PAPI counters
SCOREP_ENABLE_PROFILING=true
SCOREP_ENABLE_TRACING=false
SCOREP_METRIC_PAPI=PAPI_TOT_INS,PAPI_TOT_CYC
```

Global Settings

Per Run Settings

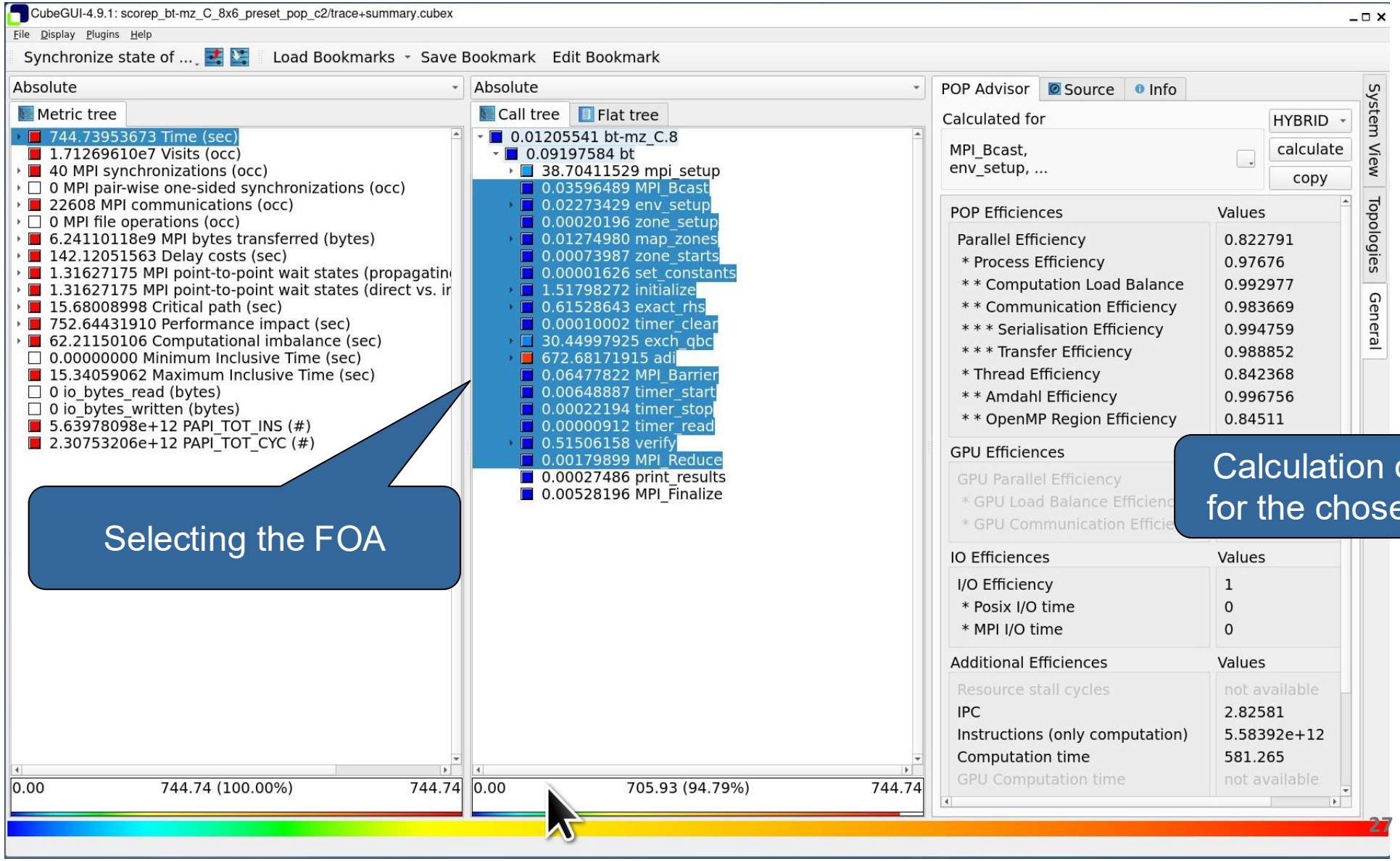
Demo: POP Metrics (shortened)

```
% cube_pop_metrics -a hybrid-mult scorep_bt-mz_C_8x6_preset_pop_c2/trace+summary.cubex
Reading scorep_bt-mz_C_8x6_preset_pop_c2/trace+summary.cubex ...  done.
Calculating
...

POP Metric                                          Profile 0
------------------------------------
Parallel Efficiency                                 0.779948
 * Process Efficiency                               0.926021
 * * Computation Load Balance                       0.992994
 * * Communication Efficiency                       0.932554
 * * * Serialisation Efficienc                      0.994759
 * * * Transfer Efficiency                          0.937467

...

Additional Metrics
------------------------------------
Resource stall cycles                               nan
IPC                                                 2.825757
Instructions (only computation                      5583968698416.000000
Computation time                                    581.284058
GPU Computation time                                nan
------------------------------------

FOA Quality Control Metrics
------------------------------------
Wall-clock time; min                                15.360373, 0.999228%
                 avg                                15.515407
                 max                                15.670707, 1.000942%
```

Produced by Scalasca/Tracing

Produced by Profiling with PAPI counters

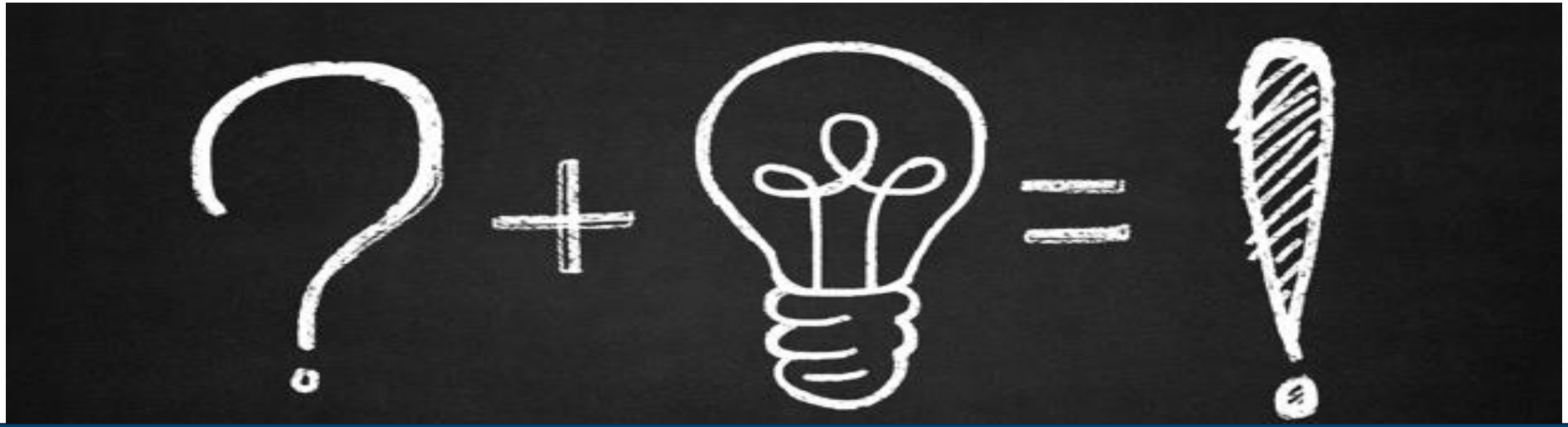
Demo: POP Metrics GUI



Wider context of POP Metrics & Tools

- Wide range of tools to calculate POP metrics:
  - Score-P/Scalasca/Cube tool environment: score-p.org scalasca.org
  - BSC Tools: Basic Analysis/TALP: https://tools.bsc.es
  - RWTH Aachen: OTF-CPT: https://github.com/RWTH-HPC/OTF-CPT
  - MAQAO: https://maqao.org

- Further Information, Learning Material and Webinars on:
  - https://pop-coe.eu



QUESTIONS