Parallel performance measurement & analysis scaling lessons

2012-11-16 | **Brian J. N. Wylie**
Jülich Supercomputing Centre

b.wylie@fz-juelich.de

Overview

Scaling from 2^10 to 2^20 (one thousand to one million)

KOJAK to Scalasca

10 key scaling lessons

Current/future challenges

Conclusions

JSC tools scalability challenge

2003: IBM SP2 p690+ 1312 cores (dual-core POWER4+ processors)

- almost exclusively programmed with MPI
  - *some pure OpenMP with up to 16 threads within SMP nodes*

2006: IBM BlueGene/L 16,384 cores (dual-core PowerPC 440)

2009: IBM BlueGene/P 294,912 cores (quad-core PowerPC 450)

2012: IBM BlueGene/Q 393,216 cores (16-core Power A2)

- hardware support for *1.5 million* threads (64-way SMP nodes)
- most applications combine MPI and OpenMP

Scalasca toolset developed from predecessor KOJAK toolset to support performance analysis of increasingly large-scale parallel applications

What needs to scale?

Techniques that had been established for O(1024) processes/threads needed re-evaluation, re-design & re-engineering each doubling of scale

- Instrumentation of application

- Measurement collection

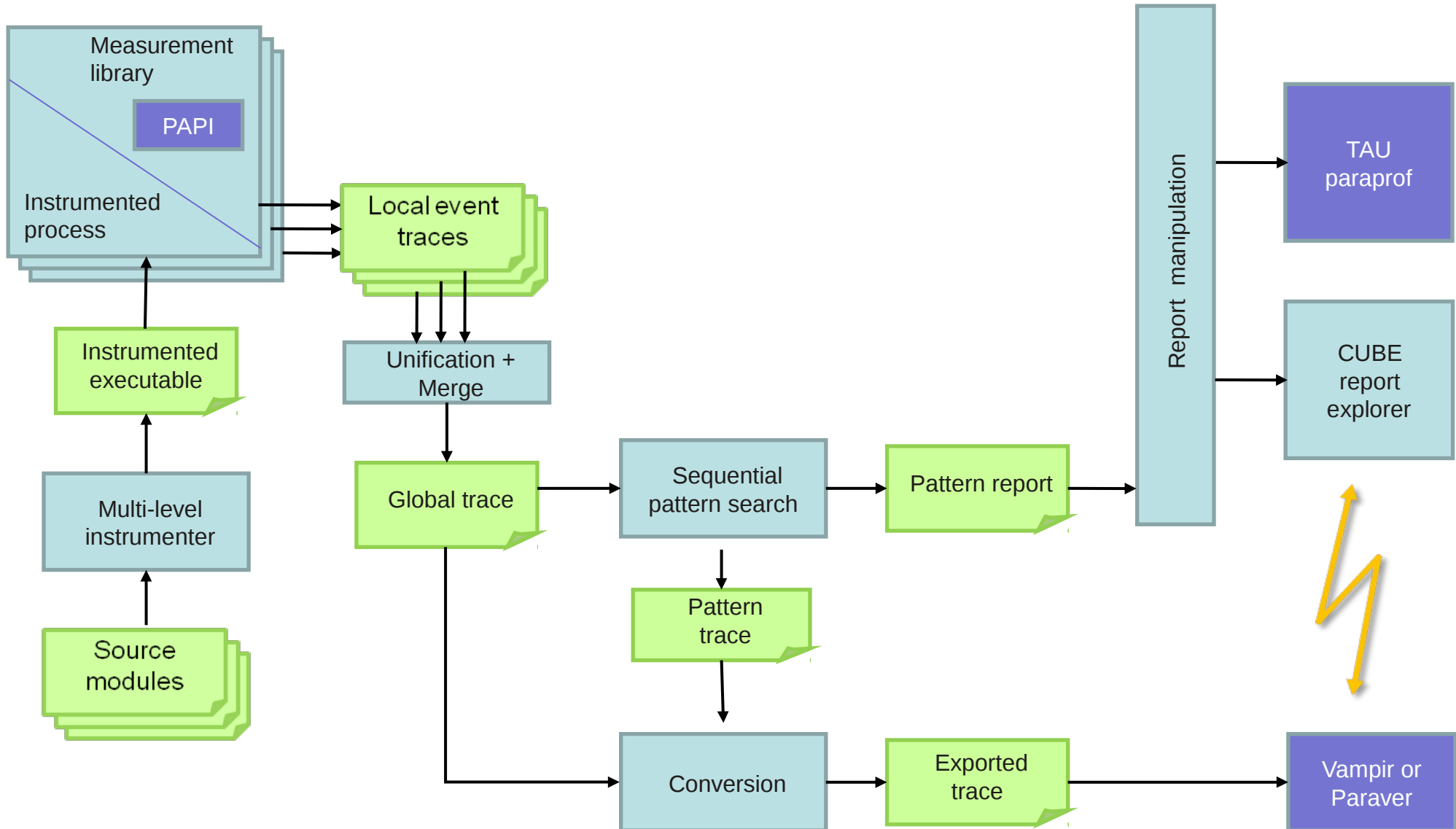- Analysis of execution

- Examination of analysis results

Scalability of the entire process governed by the least scalable part

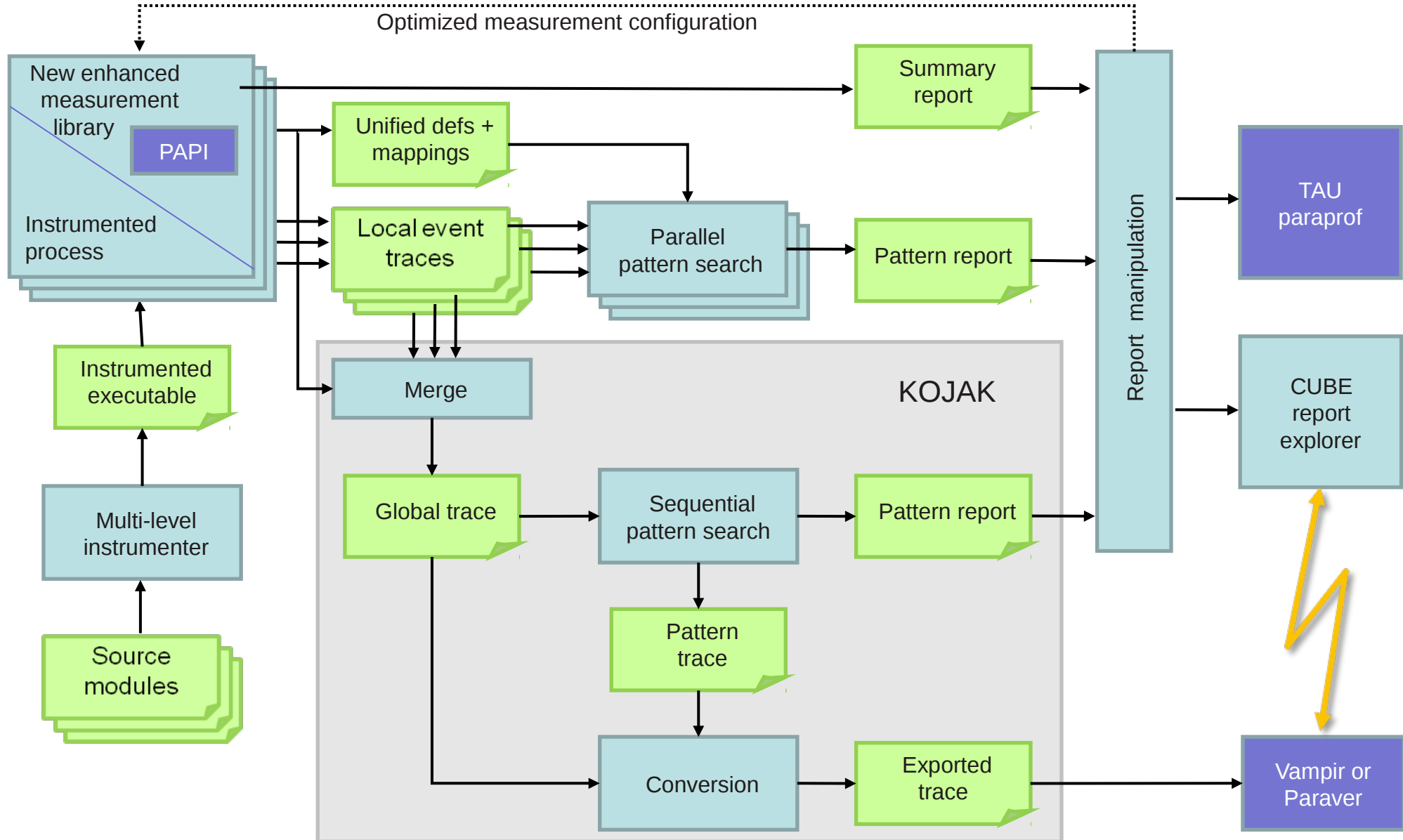- not every application affected by each issue (to the same extent)

Applications themselves faced the same scalability challenges and needed similar re-engineering

KOJAK workflow

= Third-party component

Measurement library

PAPI

Instrumented process

Local event traces

Instrumented executable

Unification + Merge

Multi-level instrumenter

Global trace

Sequential pattern search

Pattern report

Source modules

Pattern trace

Report manipulation

TAU paraprof

CUBE report explorer

Conversion

Exported trace

Vampir or Paraver

Scalasca workflow

10 key lessons

11. Collect and analyse measurements in memory

12. Analyse event traces in parallel

13. Avoid re-writing/merging event trace files

14. Avoid creating too many files

15. Manage MPI communicators

16. Unify metadata hierarchically

17. Summarize measurements during collection

18. Present analysis results associated to application/machine topologies

19. Provide statistical summaries

20. Load analysis results on-demand/incrementally

Collect and analyse measurements in memory

Storage required for measurement collection and analysis

- memory buffers for traced events of each thread
- full buffers flushed (asynchronously) to trace files on disk

However

- flushing disturbs measurement
  - *communication partners must wait for flush to complete*
- trace files too large to fit in memory may not be analysable
  - *analysis may require memory several times trace size on disk*

Therefore, specify trace buffer sizes and measurement intervals (with associated instrumentation/filtering) to avoid intermediate buffer flushes

Analyse event traces in parallel

Memory and time for serial trace analysis

- grow with number of processes/threads in measured application

However

- processors and memory available for execution analysis are identical to that for the subject parallel application execution itself
- event records contain the necessary attributes for a parallel replay

Therefore

- re-use allocated machine partition after measurement complete
- use pt2pt/collective operations to communicate partner data
  - *communication/synchronization replay time similar to original*
- [EuroPVM/MPI'06, PARA'06]

Avoid re-writing files

Merging events from separate trace files for each process and thread

- allowed traces to be written independently
- produced a single file and event stream for convenient analysis

However

- the single file becomes extremely large and unmanagable
- only a limited number of files can be opened simultaneously
- write/read/re-write becomes increasingly burdensome
  - *especially slow when using a single filesystem*
- parallel analysis ends up splitting stream again

Therefore write files in a form convenient for (parallel) reading

- [EuroPVM/MPI'06]

Avoid creating too many files

Separate trace files for each process and thread

- allowed traces to be written independently
- and read independently during parallel analysis

However

- creating the files burdens the filesystem
  - *locking required to ensure directory metadata consistency*
- simultaneous creation typically slower than serialized
- listing/archiving/deleting directories becomes painful

Therefore write filesystem blocks offset in a few multifiles

- [SIONlib, SC'09]

Trace analysis scaling (Sweep3D on BG/P)
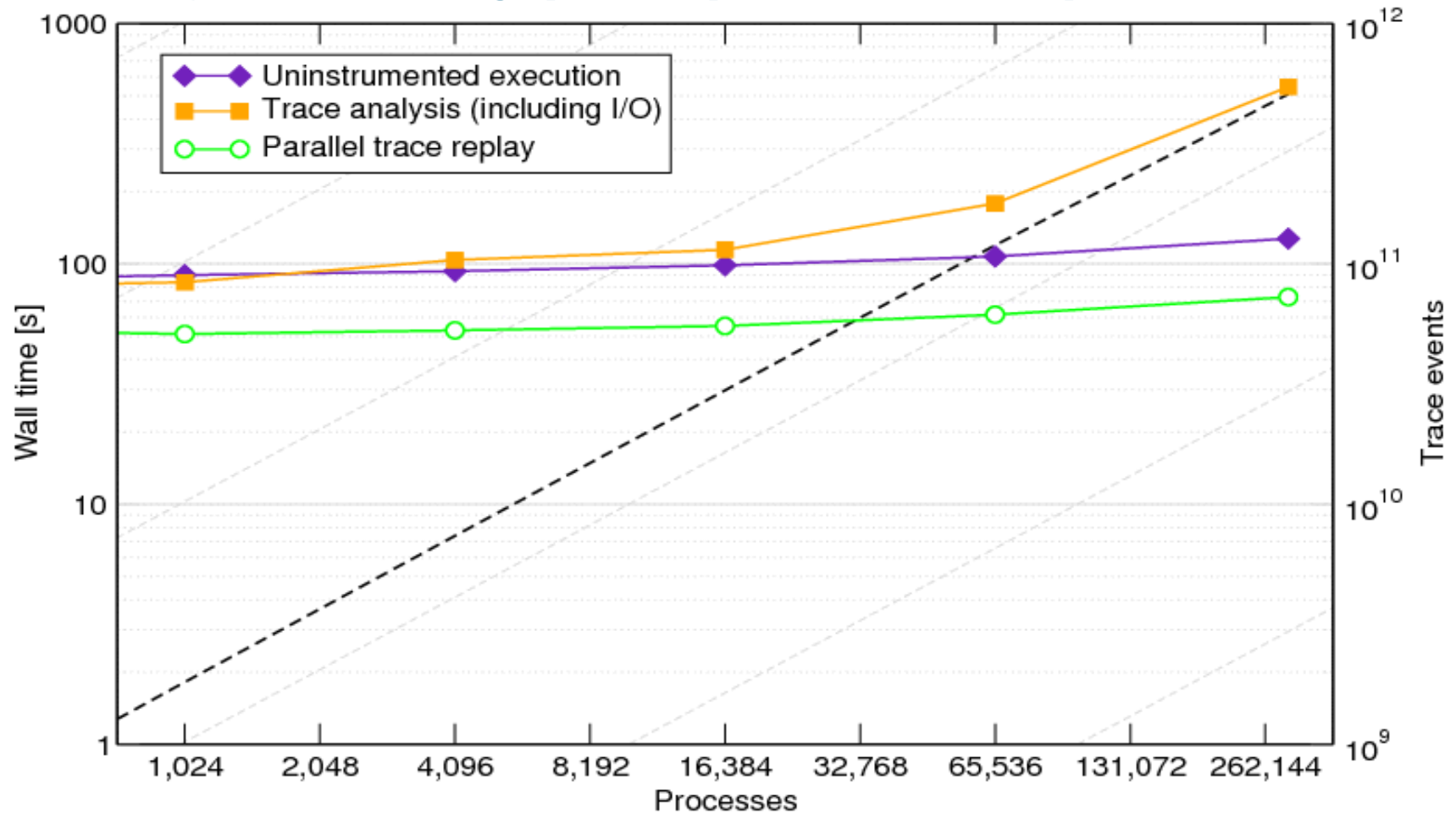
- Total trace size (---) increases to 7.6TB for 510G events
- Parallel analysis replay time scales with application execution time

Manage MPI communicators

MPI communicators organise process communication & synchronization

- describe process group membership and ranking for MPI events
  - *MPI_COMM_SELF & MPI_COMM_WORLD are special*
- required for event replay

However

- array representation grows with total number of processes
- cost of translation of local to global rank increases too
  - *MPI_Group_translate_ranks also varies with rank to translate*

Therefore define communicator creation relationship (with special handling of MPI_COMM_SELF) and record events with local ranks (translated when required by analysis)

- [EuroMPI'11]

Unify metadata hierarchically

Merging of individual process definitions and generation of mappings

- allowed event data for traces to be written independently

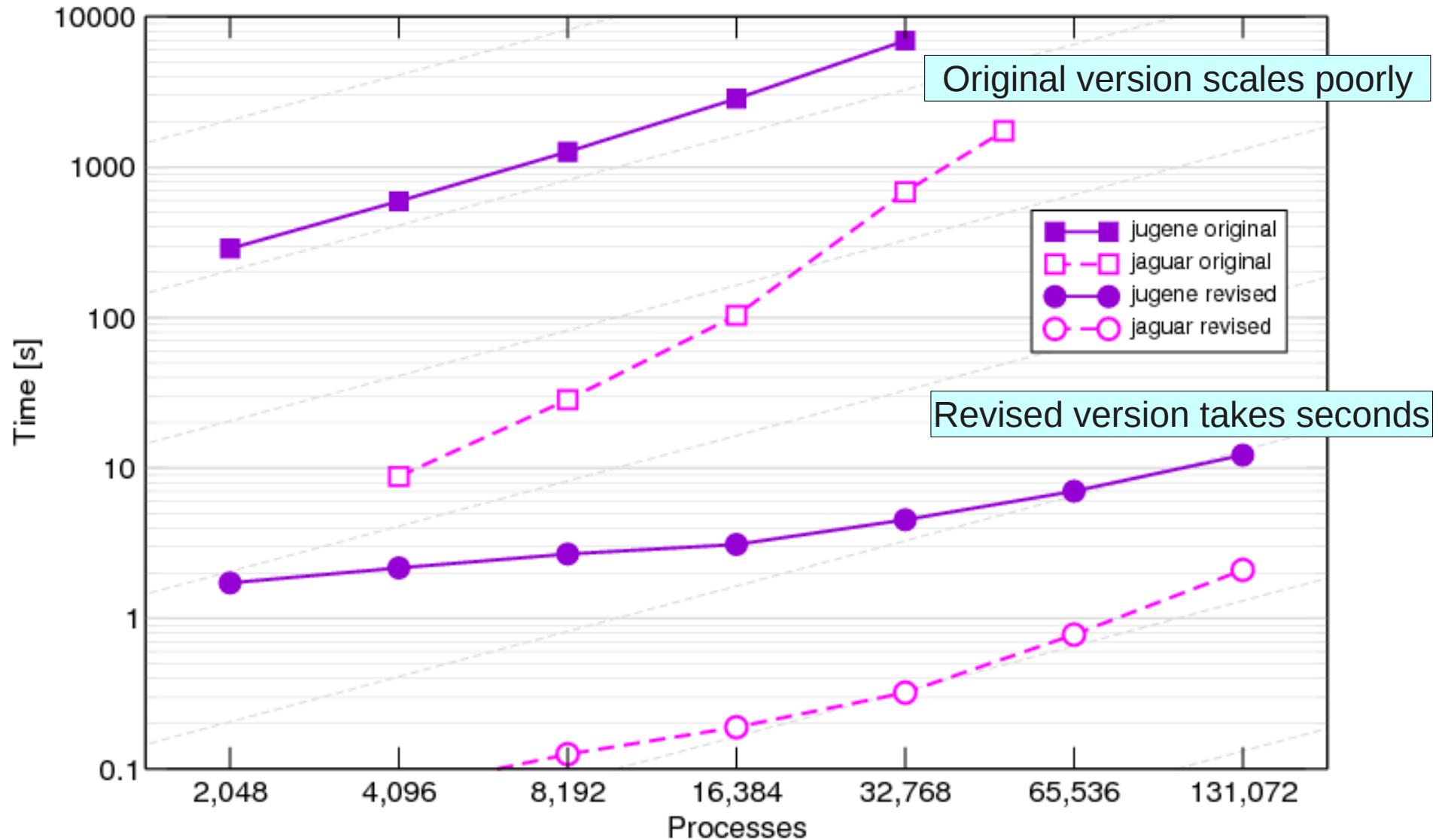- provides a consistent unified view of the set

However

- time increases linearly with number of processes if serialized

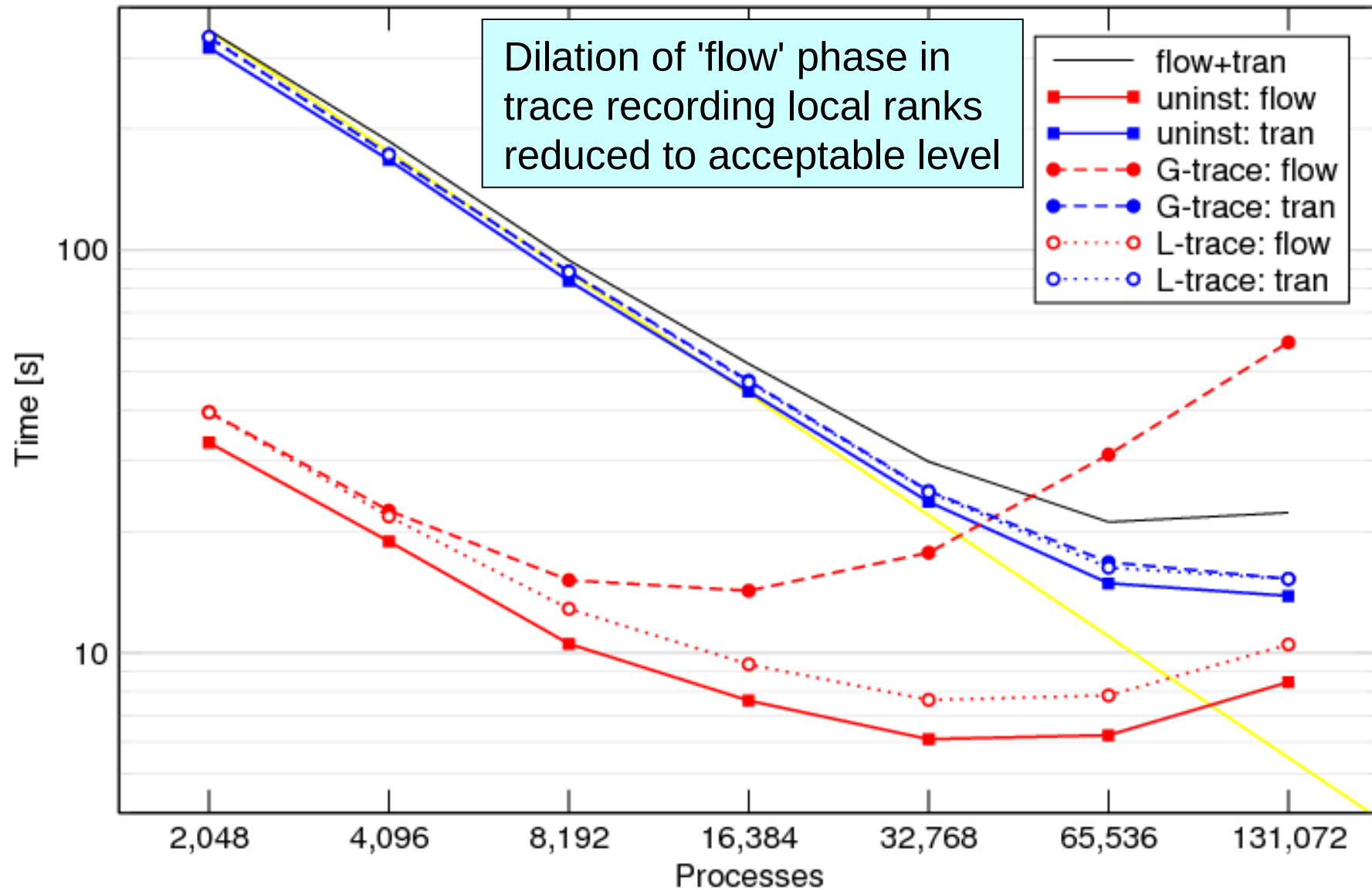- or a reduction/multicast infrastructure needs to be overlaid

Therefore employ a hierarchical unification scheme during finalization

- [PARA'10, EuroMPI'11]

Improved unification of identifiers (PFLOTRAN)

Original version scales poorly

Revised version takes seconds

jugene original
jaguar original
jugene revised
jaguar revised

Reduction of trace measurement dilation (PFLOTRAN)

Dilation of 'flow' phase in trace recording local ranks reduced to acceptable level

Legend:
- flow+tran
- uninst: flow
- uninst: tran
- G-trace: flow
- G-trace: tran
- L-trace: flow
- L-trace: tran

Y-axis: Time [s]
X-axis: Processes (2,048, 4,096, 8,192, 16,384, 32,768, 65,536, 131,072)

Summarize measurements during collection

Event trace size grows with duration and level of detail, per thread

- not always practical or productive to record every detail
- overhead for frequent short events particularly counter-productive
  - *may distort timing measurements of interest*

Therefore

- start with per-thread runtime summarization of events
  - *ideal for hardware counter measurements*
- produce aggregated execution profiles to identify events and execution intervals with(out) sufficient value for tracing
  - *filter and pause measurement*
  - *determine buffer/disk storage requirements*
- [PARA'06]

Present analysis results associated with topology

Process and thread ranks are only one aspect of application execution

- presentation is natural but not particularly scalable
- complemented with application and machine topologies
  - *often make execution performance metrics more accessible*

Therefore

- record topologies as an integral part of measurements
- allow additional topologies (and mappings) to be manually defined
- allow topologies to be interactively adjusted
  - *slicing and folding of high-dimensional topologies*

Example: Sweep3D, PFLOTRAN, COSMO, WRF, ...

Application topologies

19
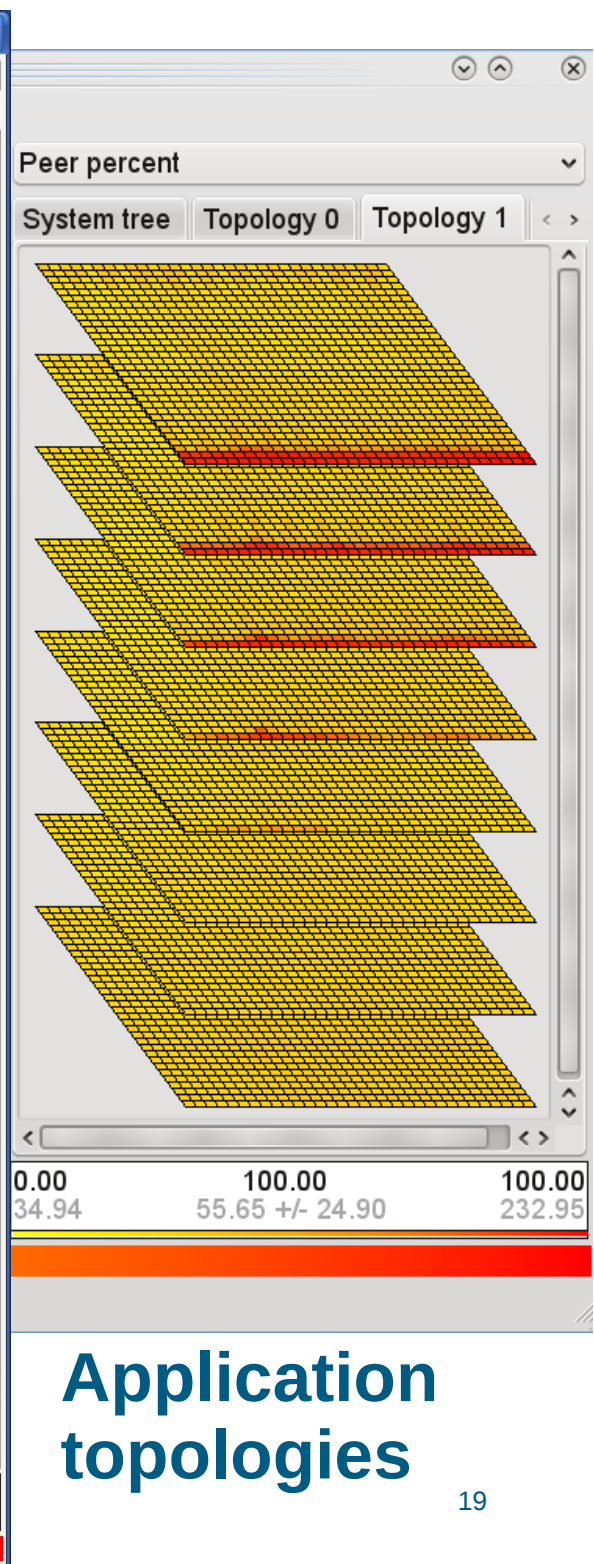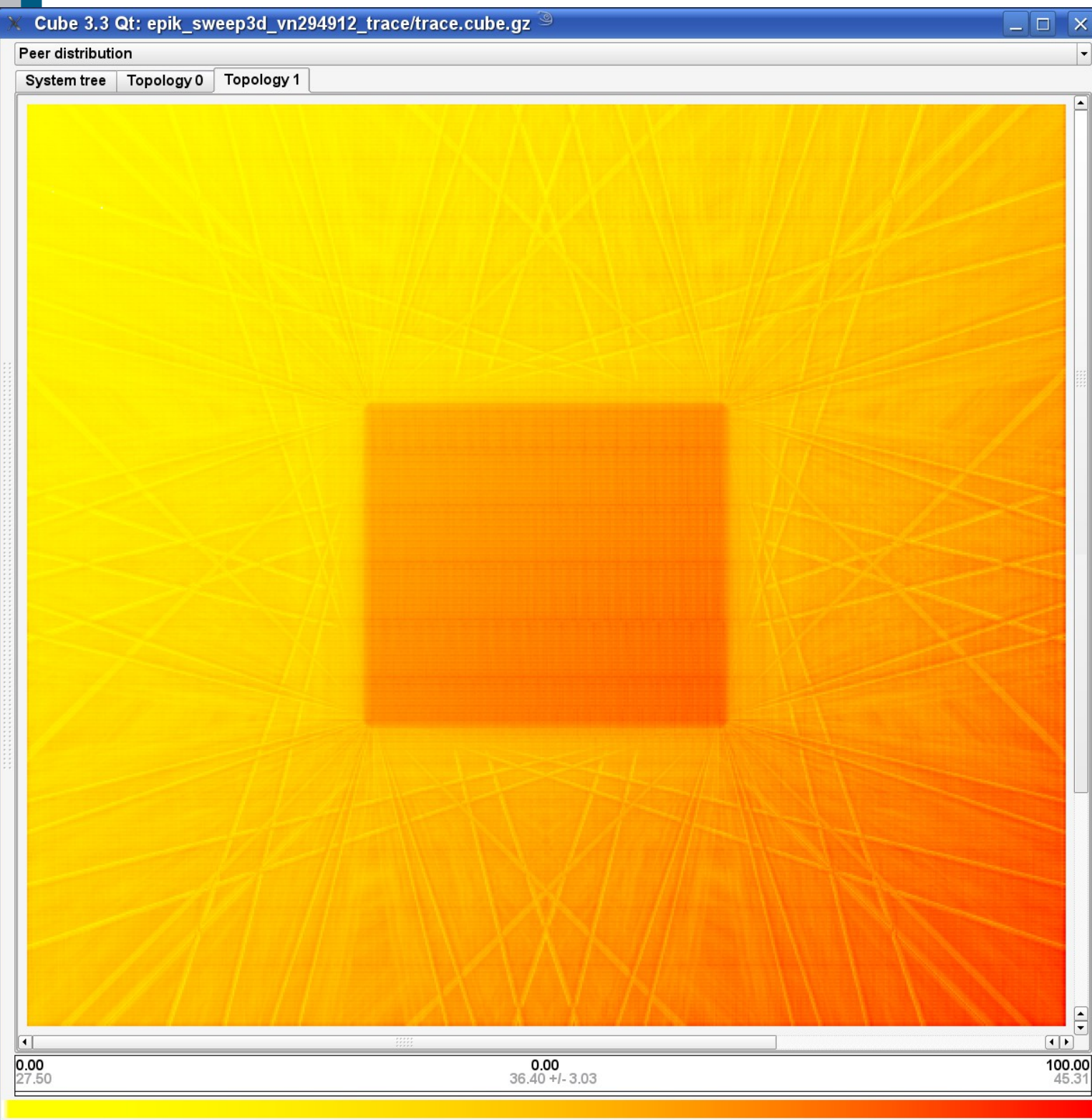
Provide statistical summaries

Presentation of metric values for all processes/threads individually

- provides a good overview to identify distribution and imbalance
- allows localization of extreme values

However

- requires display resolution which is not always available
  - *may have less than a pixel for each process/thread*
  - *topological presentation may obscure some values*
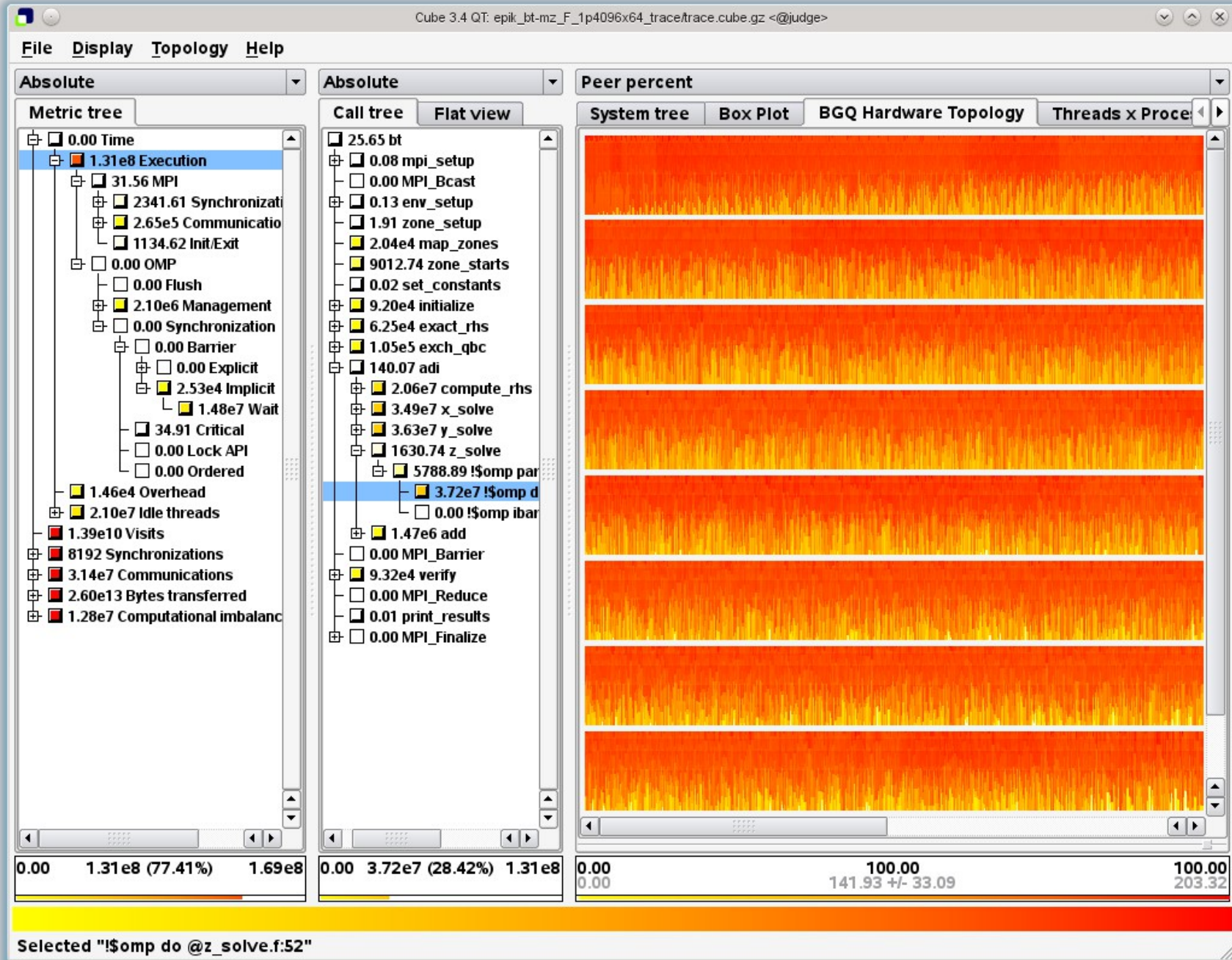- not straightforward to quantify/compare

Therefore, include simple distribution statistics (min/mean/max, quartiles)
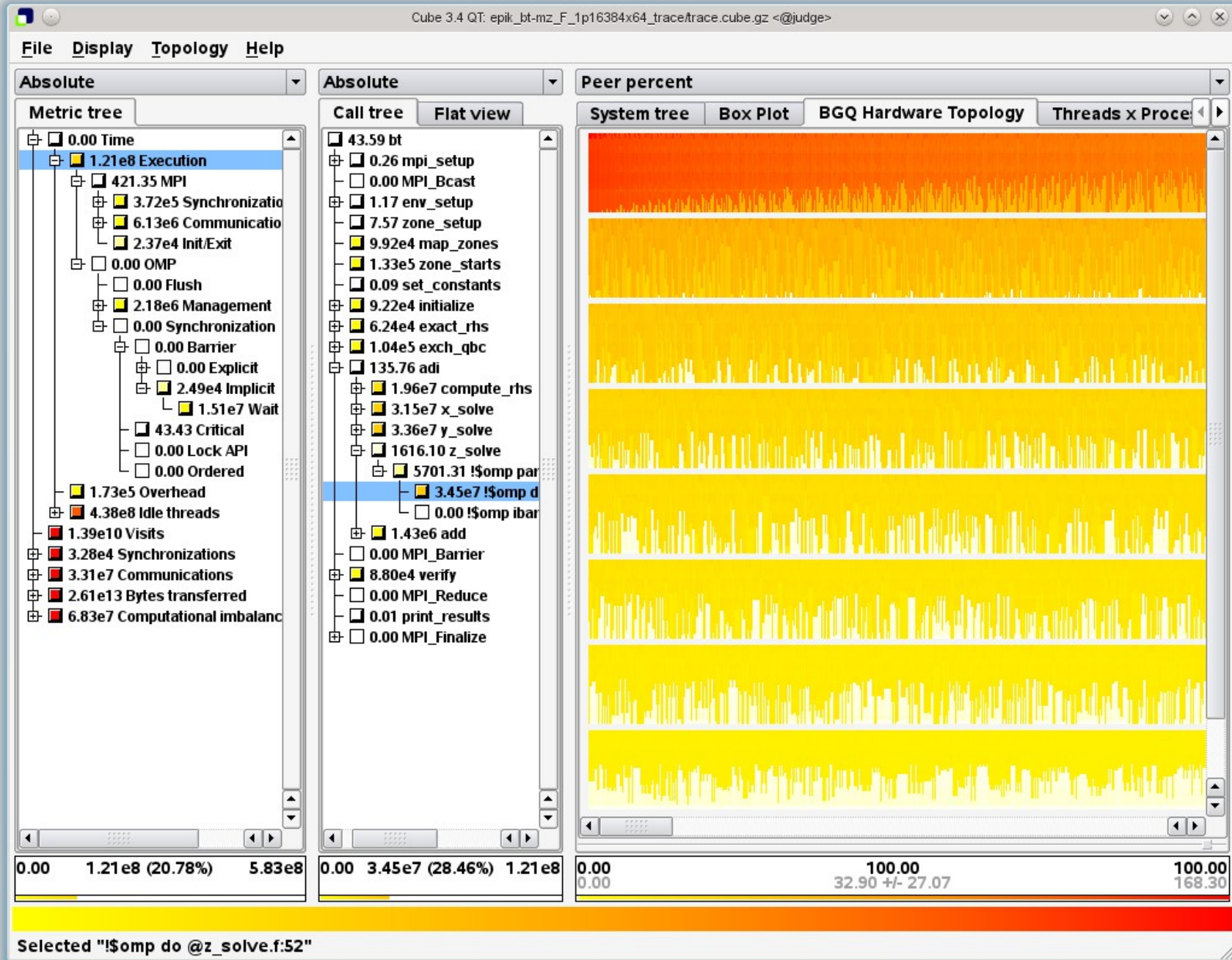
- Example: BT-MZ with 1M threads

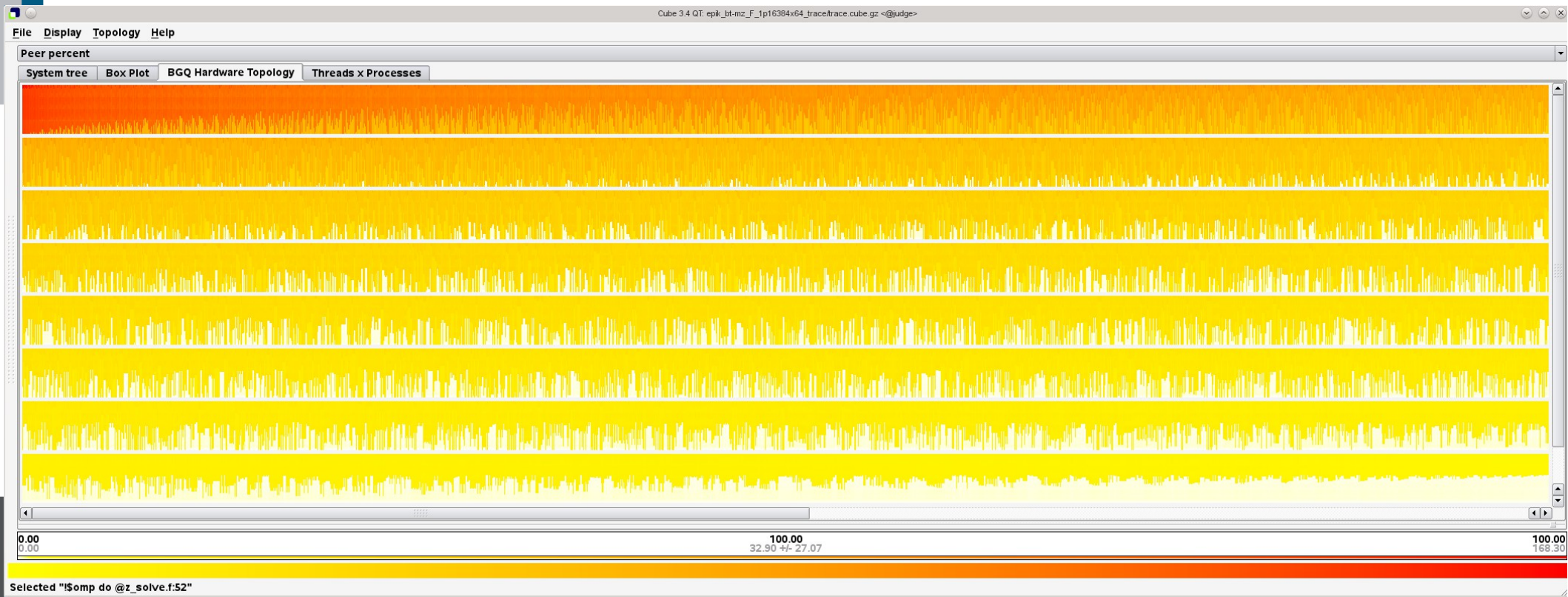# BT-MZ.F 4096x64 *z_solve* wait at implicit barrier
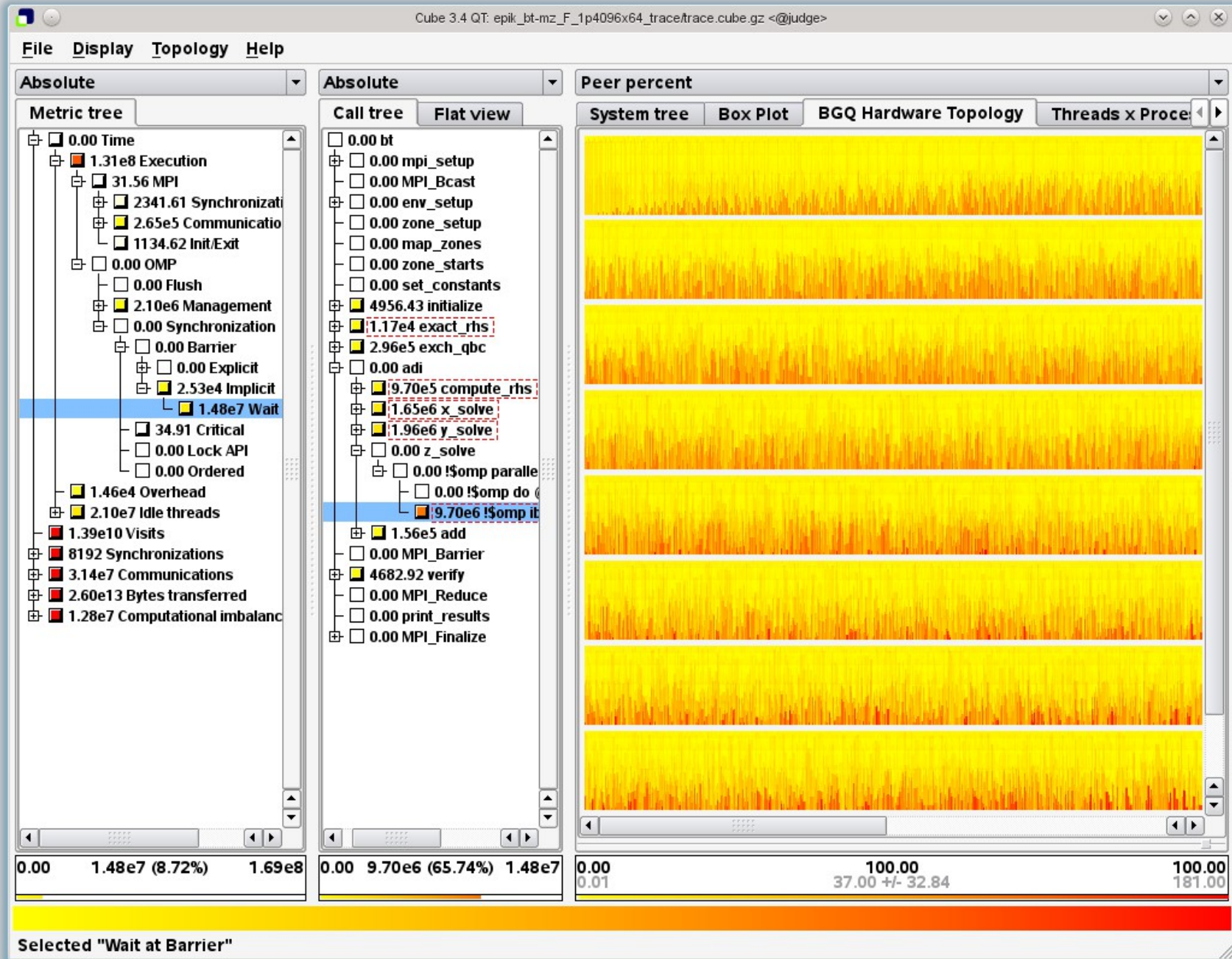
BT-MZ.F 4096x64 *z_solve* execution imbalance

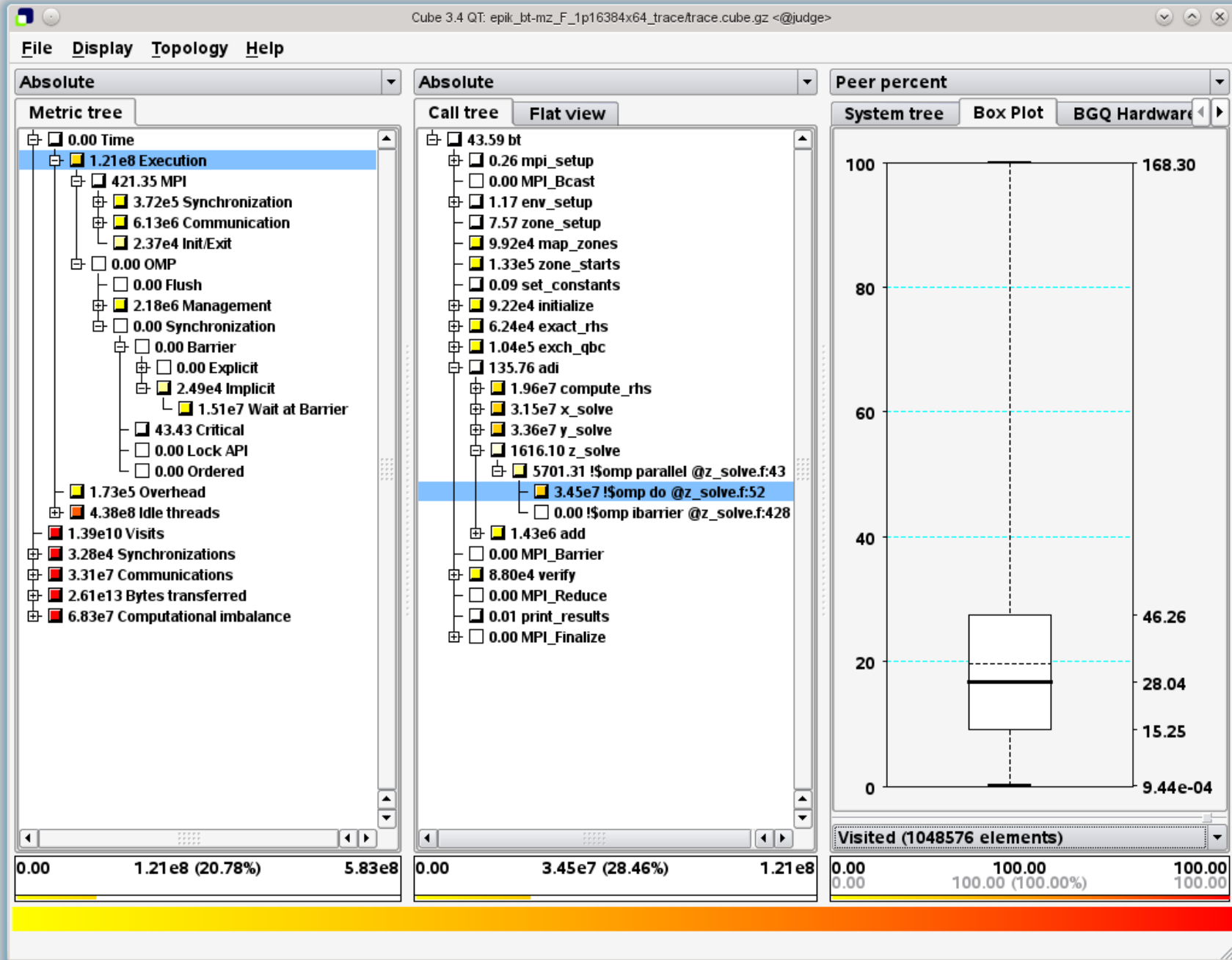# BT-MZ.F 16384x64 *z_solve* execution imbalance

BT-MZ.F 16384x64 *z_solve* execution imbalance

BT-MZ.F 16384x64 *z_solve* execution imbalance

Load analysis results on-demand/incrementally

Loading entire analysis reports into memory

- convenient for interactive exploration

However

- loading time and memory required grow with the size of the report
    - *proportional to numbers of metrics, callpaths, and threads*
    - *only a small subset can be shown at any time*
- inclusive metric values must be aggregated from exclusive ones

Therefore, store inclusive values in reports for incremental retrieval when required for presentation (or calculating exclusive metric values)

- [PARA'10]

Current/future challenges

Analysis report size & collation time (proportional to threads)

More processes and threads

More dynamic behaviour

- dynamically created processes and threads, tasks
- varying clock speed

More heterogeneous systems

- accelerators, combined programming models

More detailed measurements and analyses

- iterations, counters (at different levels)

More irregular behaviour (e.g., sampled events)

Conclusions

Complex large-scale applications provide significant challenges for performance analysis tools

Scalasca offers a range of instrumentation, measurement & analysis capabilities, with a simple GUI for interactive analysis report exploration

- works across BlueGene, Cray, K & many other HPC systems

- analysis reports and event traces can also be examined with complementary third-party tools such as TAU/ParaProf & Vampir

- convenient automatic instrumentation of applications and libraries must be moderated with selective measurement filtering

Scalasca is continually improved in response to the evolving requirements of application developers and analysts

Scalable performance **a**nalysis of **la**rge-**sc**ale parallel **a**pplications

- portable toolset for scalable performance measurement & analysis of MPI, OpenMP & hybrid OpenMP+MPI parallel applications

- supporting most popular HPC computer systems

- available under New BSD open-source license

- ready to run from VI-HPS HPC Linux Live DVD/ISO/OVA

- sources, documentation & publications:
    - *http://www.scalasca.org*
    - *mailto: scalasca@fz-juelich.de*

Scalasca project

Overview

- Headed by Bernd Mohr (JSC) & Felix Wolf (GRS-Sim)

  - *Helmholtz Initiative & Networking Fund project started in 2006*

- Follow-up to pioneering KOJAK project (started 1998)

  - *Automatic pattern-based trace analysis*

Objective

- Development of a **<u>scalable</u> performance analysis** toolset

- Specifically targeting **<u>large-scale</u> parallel applications**

Status

- Scalasca v1.4.2 released in July 2012

- Available for download from www.scalasca.org

Scalasca features

Open source, New BSD license

Portable

- Cray XT/XE/XK, IBM BlueGene L/P/Q, IBM SP & blade clusters, K/Fujitsu, NEC SX, SGI Altix, Linux cluster® (SPARC, x86-64), ...

Supports typical HPC languages & parallel programming paradigms

- Fortran, C, C++
- MPI, OpenMP & hybrid MPI+OpenMP

Integrated instrumentation, measurement & analysis toolset

- Customizable automatic/manual instrumentation
- Runtime summarization (*aka* profiling)
- Automatic event trace analysis

Scalasca components

- Automatic program instrumenter creates instrumented executable

- Unified measurement library supports both
  - *runtime summarization*
  - *trace file generation*

- Parallel, replay-based event trace analyzer invoked automatically on set of traces

- Common analysis report explorer & examination/processing tools
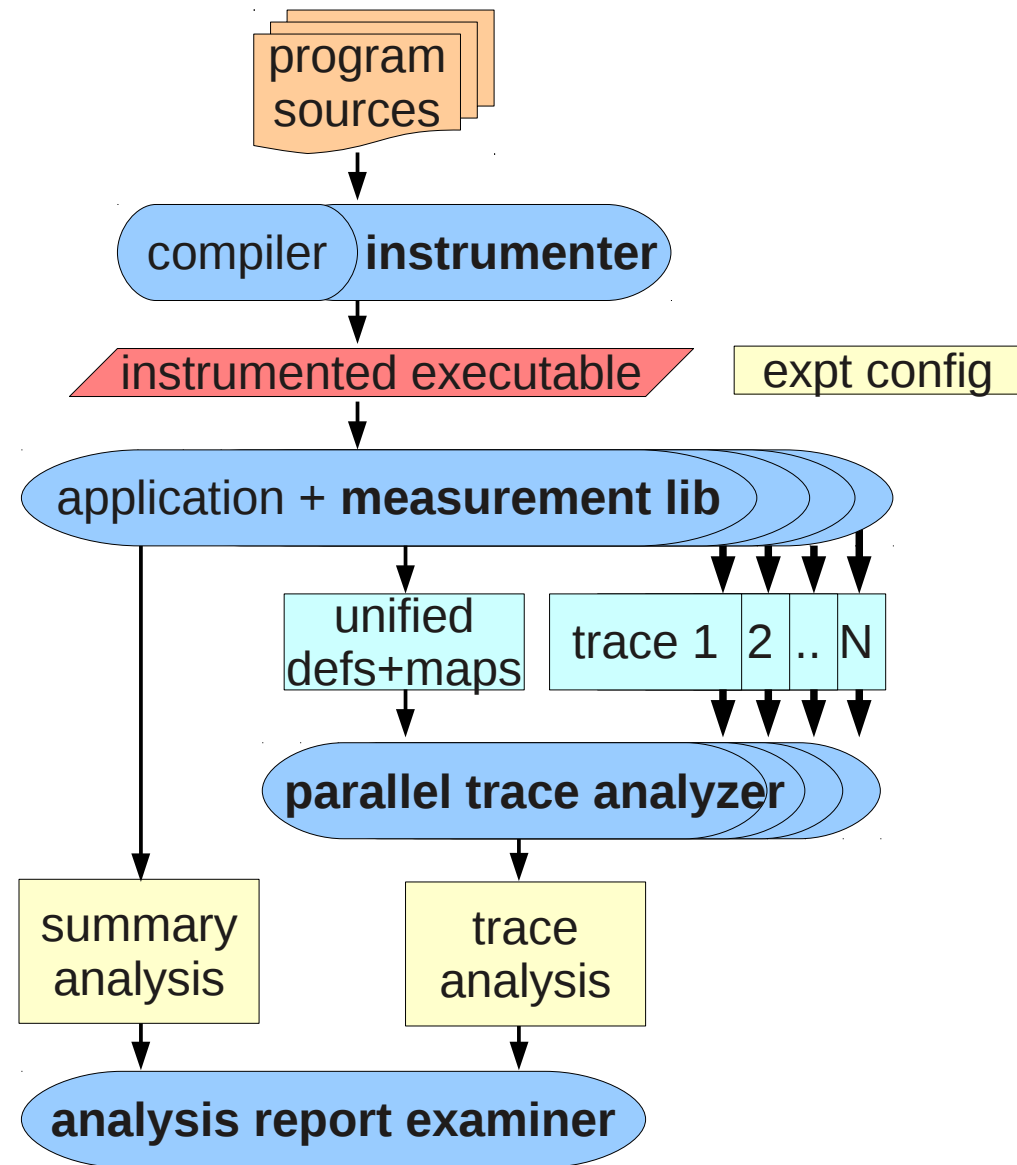
Scalasca usage (commands)

1. Prepare application objects and executable for measurement:
   - **scalasca -instrument** `mpicc -fopenmp -O3 -c …`
   - **scalasca -instrument** `mpif77 -fopenmp -O3 -o bt-mz.exe …`
     - *instrumented executable bt-mz.exe produced*

2. Run application under control of measurement & analysis nexus:
   - **scalasca -analyze** `mpiexec -np 16384 bt-mz.exe …`
     - *epik_bt-mz_16384x64_sum* experiment produced
   - **scalasca -analyze -t** `mpiexec -np 16384 bt-mz.exe …`
     - *epik_bt-mz_16384x64_trace* experiment produced

   BATCH JOB

3. Interactively explore experiment analysis report:
   - **scalasca -examine** `epik_bt-mz_16384x64_trace`
     - *epik_bt-mz_16384x64_trace/trace.cube.gz* presented

Acknowledgments

The application and benchmark developers who generously provided their codes and/or measurement archives

The facilities who made their HPC resources available and associated support staff who helped us use them effectively

- ALCF, BSC, CEA, CSC, CSCS, CINECA, DKRZ, EPCC, HLRN, HLRS, ICL, ICM, IMAG, JSC, KAUST, KTH, LRZ, NCAR, NCCS, NICS, NLHPC, RWTH, RZG, SARA, TACC, ZIH

- Access & usage supported by European Union, German and other national funding organizations

Scalasca users who have provided valuable feedback and suggestions for improvements