SC'13 Tutorial: Hands-on Practical Hybrid Parallel Application Performance Engineering

Markus Geimer
Jülich Supercomputing Centre

Sameer Shende
University of Oregon

Bert Wesarg
Technische Universität Dresden

Brian Wylie
Jülich Supercomputing Centre

Agenda

Time	Topic	Presenter
08:30	Introduction to VI-HPS & parallel performance engineering	Wylie
09:00	VI-HPS Linux Live-ISO and MPI+OpenMP example code	Wylie / all
09:15	Instrumentation & measurement with **Score-P**	Wesarg
10:00	*Break*	
10:30	Profile examination with **CUBE**	Geimer
11:00	Configuration & customization of Score-P measurements	Geimer
11:30	Specialized Score-P measurements & analyses	Wesarg
12:00	*Lunch*	
13:30	Automated trace analysis with **Scalasca**	Geimer
14:15	Interactive trace analysis with **Vampir**	Wesarg
15:00	*Break*	
15:30	Profile examination with **TAU** ParaProf	Shende
16:00	Performance data management with **TAU** PerfExplorer	Shende
16:15	Finding typical parallel performance bottlenecks	Wesarg
16:45	Review & conclusion	Wylie
17:00	*Adjourn*	

Introduction to VI-HPS

Brian Wylie
Jülich Supercomputing Centre

Goal: Improve the quality and accelerate the development process of complex simulation codes running on highly-parallel computer systems

- Start-up funding (2006–2011) by Helmholtz Association of German Research Centres

HELMHOLTZ | ASSOCIATION

- Activities
  - Development and integration of HPC programming tools
    - Correctness checking & performance analysis
  - Training workshops
  - Service
    - Support email lists
    - Application engagement
  - Academic workshops

http://www.vi-hps.org

VI-HPS partners (founders)

Forschungszentrum Jülich

- Jülich Supercomputing Centre

RWTH Aachen University

- Centre for Computing & Communication

Technical University of Dresden

- Centre for Information Services & HPC

University of Tennessee (Knoxville)

- Innovative Computing Laboratory

VI-HPS partners (cont.)

Barcelona Supercomputing Center

- Centro Nacional de Supercomputación

German Research School

- Laboratory of Parallel Programming

Lawrence Livermore National Lab.

- Centre for Applied Scientific Computing

Technical University of Munich

- Chair for Computer Architecture

University of Oregon

- Performance Research Laboratory

University of Stuttgart

- HPC Centre

University of Versailles St-Quentin

- LRC ITACA

Productivity tools

MUST
- MPI usage correctness checking

PAPI
- Interfacing to hardware performance counters

Periscope
- Automatic analysis via an on-line distributed search

Scalasca
- Large-scale parallel performance analysis

TAU
- Integrated parallel performance system

Vampir
- Interactive graphical trace visualization & analysis

Score-P
- Community instrumentation & measurement infrastructure

Productivity tools (cont.)

KCachegrind
- Callgraph-based cache analysis [x86 only]

MAQAO
- Assembly instrumentation & optimization [x86 only]

mpiP/mpiPview
- MPI profiling tool and analysis viewer

Open MPI
- Integrated memory checking

Open|Speedshop
- Integrated parallel performance analysis environment

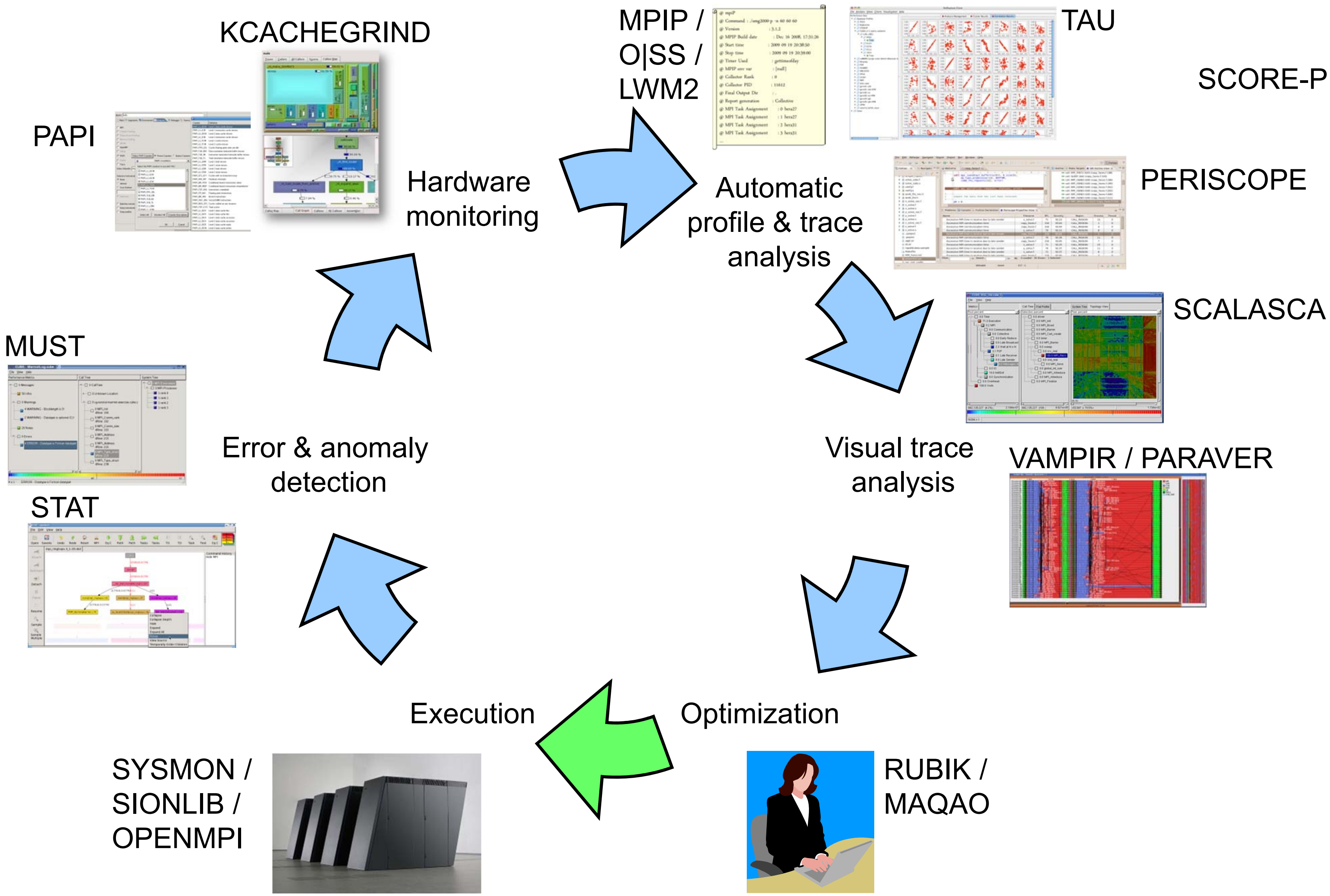
Paraver/Extrae
- Event tracing and graphical trace visualization & analysis

Rubik
- Process mapping generation & optimization [BG only]

SIONlib
- Optimized native parallel file I/O

STAT
- Stack trace analysis tools

Technologies and their integration



Disclaimer

Tools will ***not*** automatically make you, your applications or computer systems more *productive*.

However, they can help you understand ***how*** your parallel code executes and ***when / where*** it's necessary to work on *correctness* and *performance* issues.

VI-HPS training & Tuning Workshops

- Goals
  - Give an overview of the programming tools suite
  - Explain the functionality of individual tools
  - Teach how to use the tools effectively
  - Offer hands-on experience and expert assistance using tools
  - Receive feedback from users to guide future development
- For best results, bring & analyze/tune your own code(s)!

- VI-HPS Hands-on Tutorial series
  - SC'08, ICCS'09, SC'09, Cluster'10, SC'10, SC'11, EuroMPI'12, XSEDE'13 (San Diego), **SC'13 (Denver)**
- VI-HPS Tuning Workshop series
  - 2008 (Aachen & Dresden), 2009 (Jülich & Bremen), 2010 (Garching & Amsterdam/NL), 2011 (Stuttgart & Aachen), 2012 (St-Quentin/F & Garching), 2013 (Saclay/F & Jülich)

Recent & upcoming events

- ISC Extreme Scale Tools Tutorial (16 Jun 2013, Leipzig)
- EuroPar VI-HPS Tools Tutorial (26 Sep 2013, Aachen)
- 12th VI-HPS Tuning Workshop (7-11 Oct 2013, Jülich)
  - Hosted by Jülich Supercomputing Centre, FZJ, Germany
  - Using PRACE Tier-0 *Juqueen* BlueGene/Q system
  - Score-P, Scalasca, Vampir, TAU, Periscope, Paraver, MUST, ...
- Further events to be determined
  - (one-day) tutorials
    - With guided exercises usually using a Live-DVD
  - (multi-day) training workshops
    - With your own applications on actual HPC systems
- Check www.vi-hps.org/training for announced events
- Contact us if you might be interested in hosting an event

- Bootable Linux installation on DVD (or USB memory stick)
- Includes everything needed to try out our parallel tools on an 64-bit x86-architecture notebook computer
  - VI-HPS tools: MUST, PAPI, Score-P, Periscope, Scalasca, TAU, Vampir*
  - Also: Eclipse/PTP, TotalView*, etc.
    - * time/capability-limited evaluation licences provided for commercial products
  - GCC (w/ OpenMP), OpenMPI
  - Manuals/User Guides
  - Tutorial exercises & examples
- Produced by U. Oregon PRL
  - Sameer Shende

VI-HPS Linux Live ISO

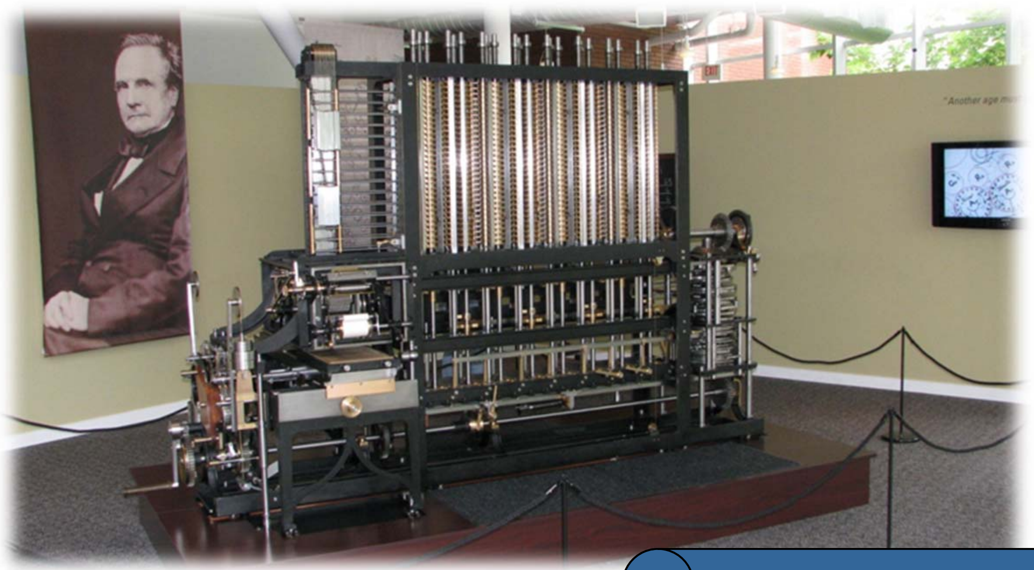
- ISO image approximately 4GB
  - download latest version from website
  - http://www.vi-hps.org/training/livedvd
  - optionally create bootable DVD or USB drive
- Boot directly from disk
  - enables hardware counter access and offers best performance, but no save/resume
- Boot within virtual machine
  - faster boot time and can save/resume state, but may not allow hardware counter access
- Boots into Linux environment for HPC
  - supports building and running provided MPI and/or OpenMP parallel application codes
  - and experimentation with VI-HPS (and third-party) tools

Introduction to Parallel Performance Engineering

Markus Geimer, Brian Wylie
Jülich Supercomputing Centre

(with content used with permission from tutorials by Bernd Mohr/JSC and Luiz DeRose/Cray)

Performance: an old problem



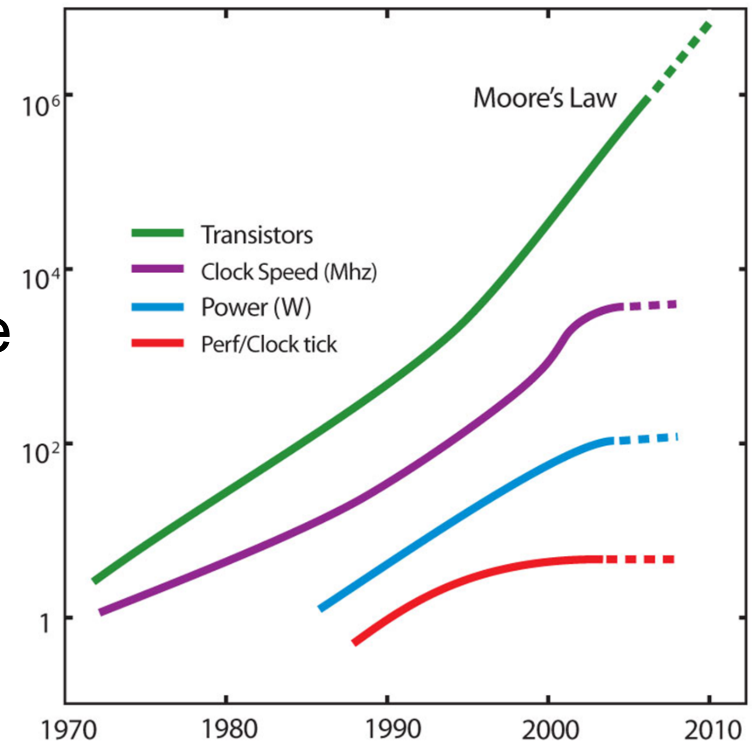
Difference Engine

"The most constant difficulty in contriving the engine has arisen from the desire to reduce the time in which the calculations were executed to the shortest which is possible."

Charles Babbage
1791 – 1871

Today: the "free lunch" is over

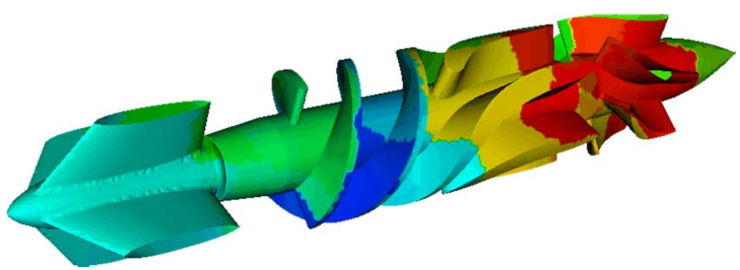
- Moore's law is still in charge, but
  - Clock rates no longer increase
  - Performance gains only through increased parallelism
- Optimizations of applications more difficult
  - Increasing application complexity
    - Multi-physics
    - Multi-scale
  - Increasing machine complexity
    - Hierarchical networks / memory
    - More CPUs / multi-core



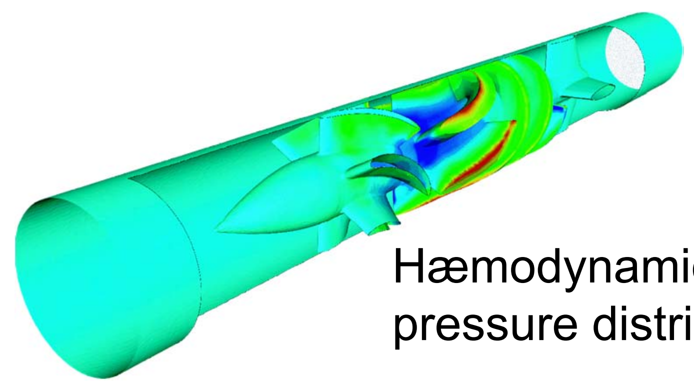
☞ Every doubling of scale reveals a new bottleneck!

Example: XNS

- CFD simulation of unsteady flows
  - Developed by CATS / RWTH Aachen
  - Exploits finite-element techniques, unstructured 3D meshes, iterative solution strategies
- MPI parallel version
  - >40,000 lines of Fortran & C
  - DeBakey blood-pump data set (3,714,611 elements)

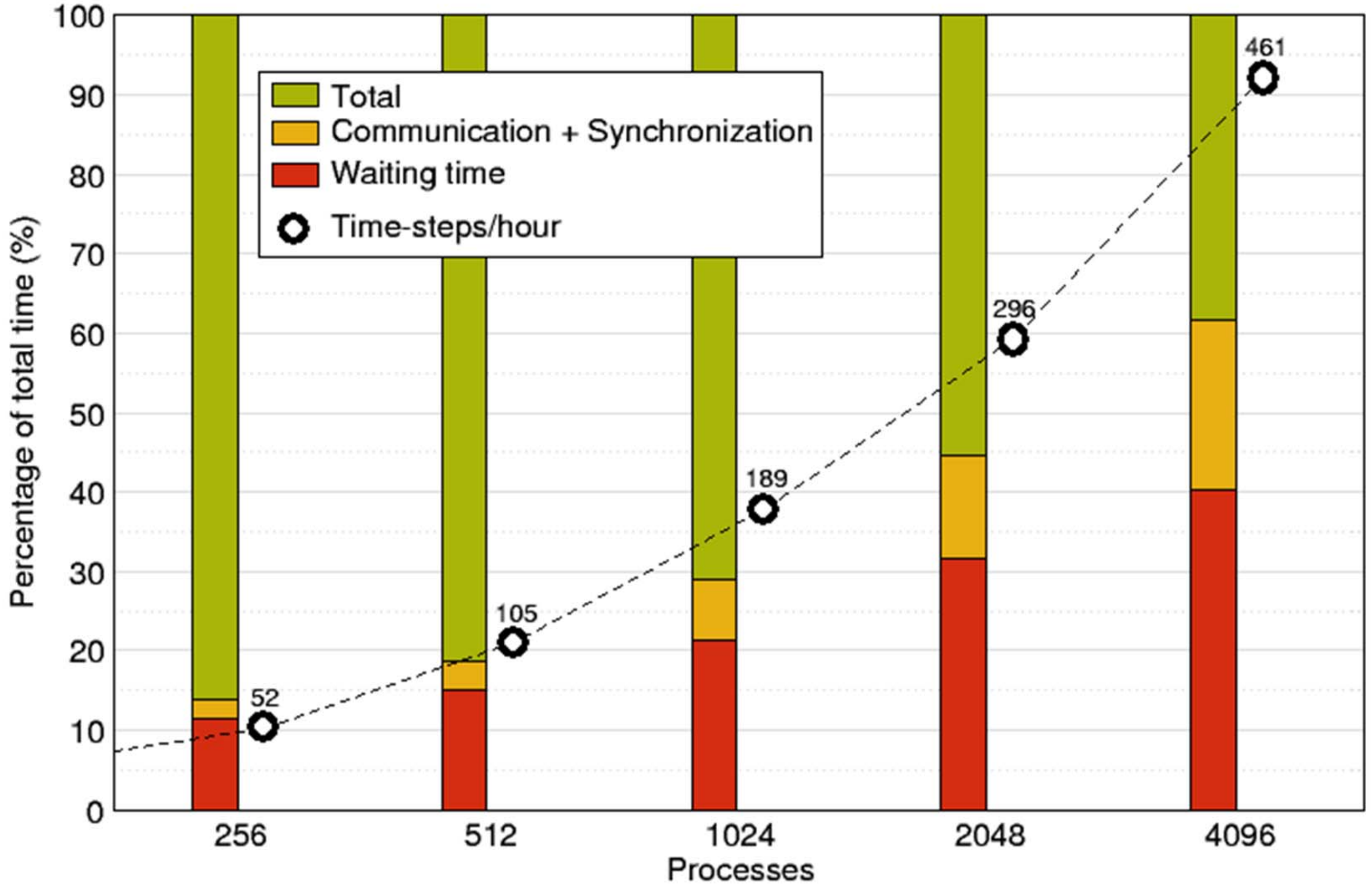


Partitioned finite-element mesh



Hæmodynamic flow pressure distribution

XNS wait-state analysis on BG/L (2007)



Performance factors of parallel applications

- "Sequential" factors
  - Computation
    - ☞Choose right algorithm, use optimizing compiler
  - Cache and memory
    - ☞Tough! Only limited tool support, hope compiler gets it right
  - Input / output
    - ☞Often not given enough attention
- "Parallel" factors
  - Partitioning / decomposition
  - Communication (i.e., message passing)
  - Multithreading
  - Synchronization / locking
    - ☞More or less understood, good tool support

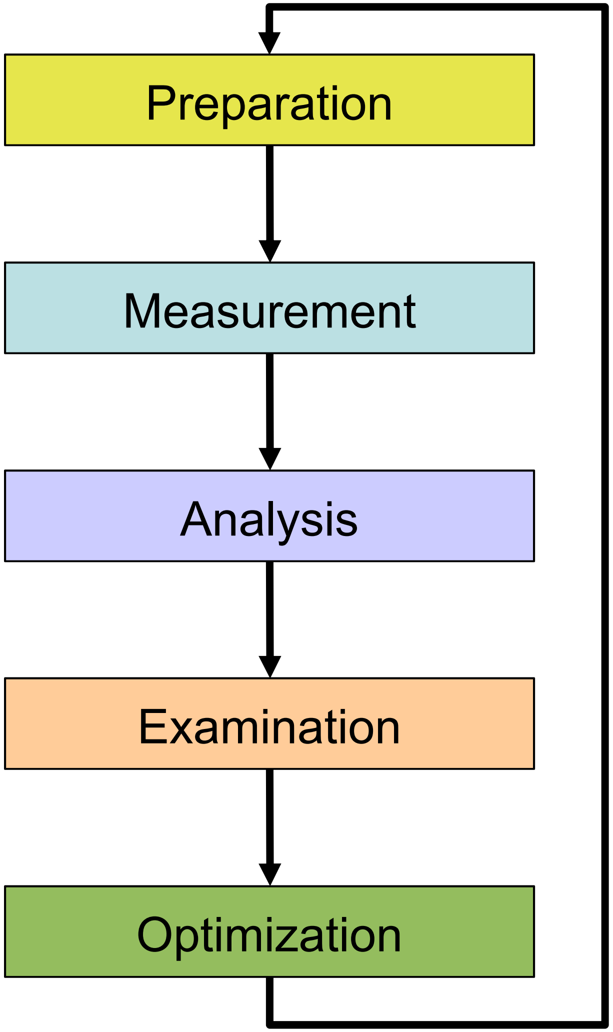
Tuning basics

- Successful engineering is a combination of
  - The right algorithms and libraries
  - Compiler flags and directives
  - Thinking !!!
- Measurement is better than guessing
  - To determine performance bottlenecks
  - To compare alternatives
  - To validate tuning decisions and optimizations
    - ☞After each step!

> "We should forget about small efficiencies, say 97% of the time: premature optimization is the root of all evil."
>
> Charles A. R. Hoare

- It's easier to optimize a slow correct program than to debug a fast incorrect one
  - ☞ *Nobody cares how fast you can compute a wrong answer...*

Performance engineering workflow



- Prepare application (with symbols), insert extra code (probes/hooks)
- Collection of data relevant to execution performance analysis
- Calculation of metrics, identification of performance metrics
- Presentation of results in an intuitive/understandable form
- Modifications intended to eliminate/reduce performance problems

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
  - ☞ *Know when to stop!*
- Don't optimize what does not matter
  - ☞ *Make the common case fast!*

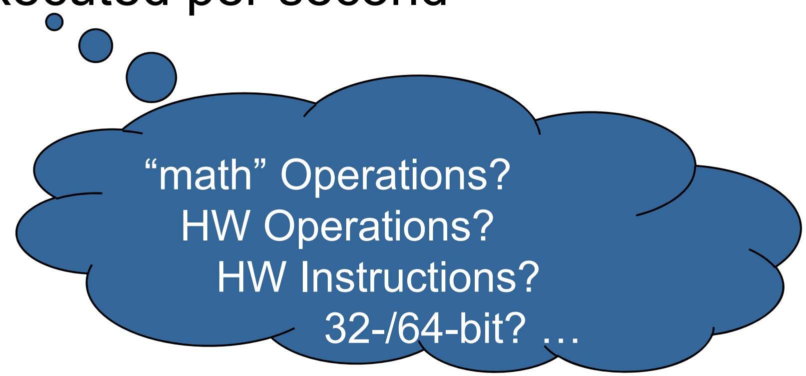
> "If you optimize everything, you will always be unhappy."
>
> Donald E. Knuth

Metrics of performance

- What can be measured?
  - A **count** of how often an event occurs
    - E.g., the number of MPI point-to-point messages sent
  - The **duration** of some interval
    - E.g., the time spent these send calls
  - The **size** of some parameter
    - E.g., the number of bytes transmitted by these calls
- Derived metrics
  - E.g., rates / throughput
  - Needed for normalization

Example metrics

- Execution time
- Number of function calls
- CPI
  - CPU cycles per instruction
- FLOPS
  - Floating-point operations executed per second

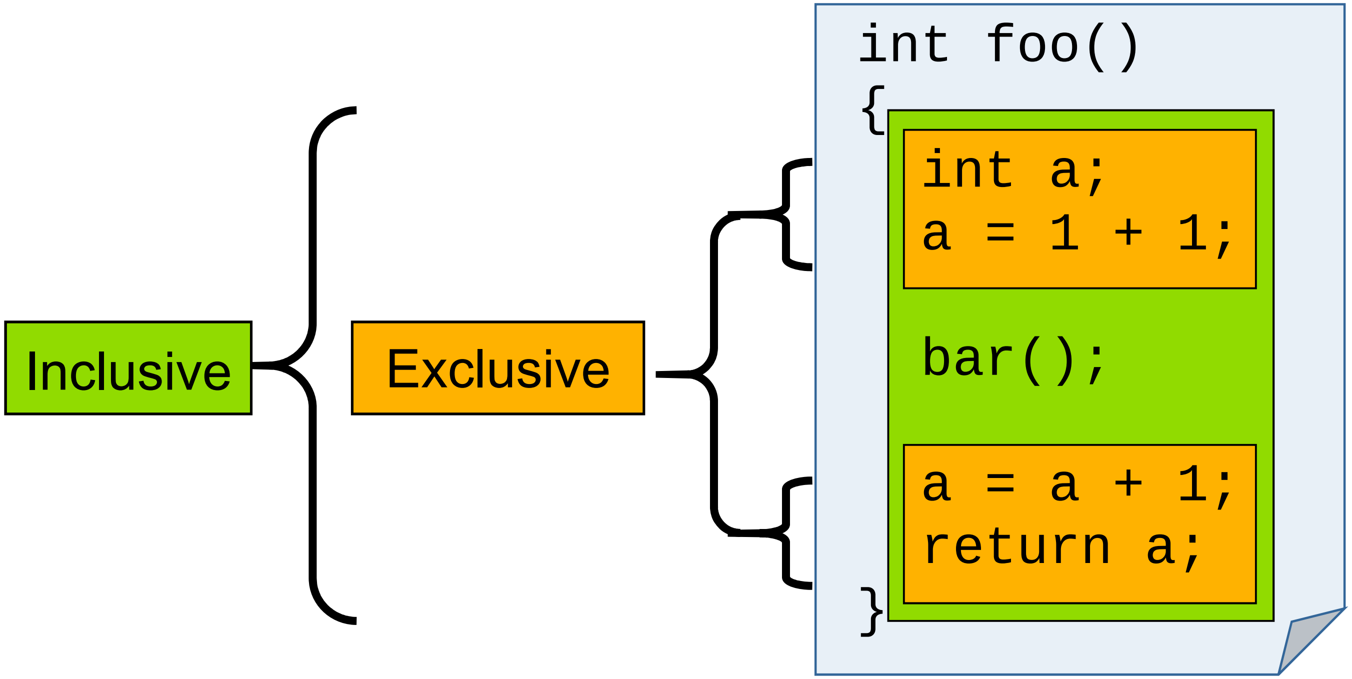


Execution time

- Wall-clock time
  - Includes waiting time: I/O, memory, other system activities
  - In time-sharing environments also the time consumed by other applications
- CPU time
  - Time spent by the CPU to execute the application
  - Does not include time the program was context-switched out
    - Problem: Does not include inherent waiting time (e.g., I/O)
    - Problem: Portability? What is user, what is system time?
- Problem: Execution time is non-deterministic
  - Use mean or minimum of several runs

Inclusive vs. Exclusive values

- Inclusive
  - Information of all sub-elements aggregated into single value
- Exclusive
  - Information cannot be subdivided further

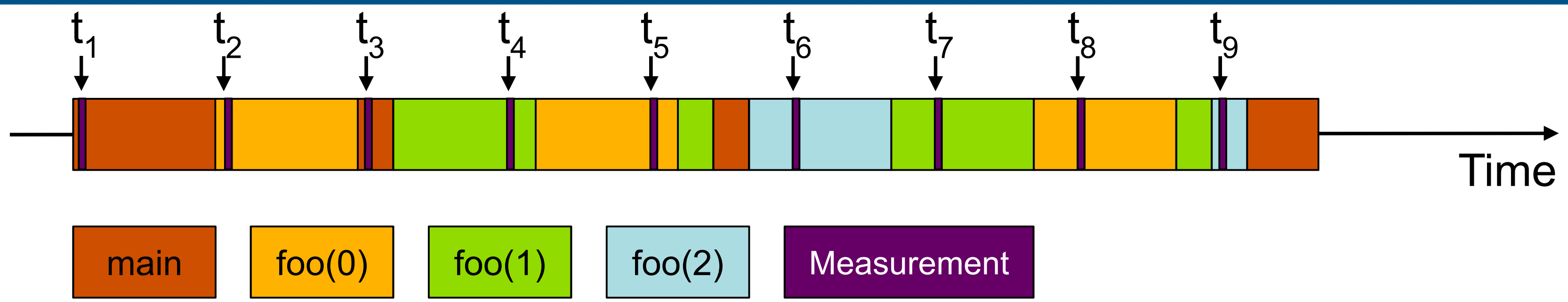


Classification of measurement techniques

- How are performance measurements triggered?
  - Sampling
  - Code instrumentation

- How is performance data recorded?
  - Profiling / Runtime summarization
  - Tracing

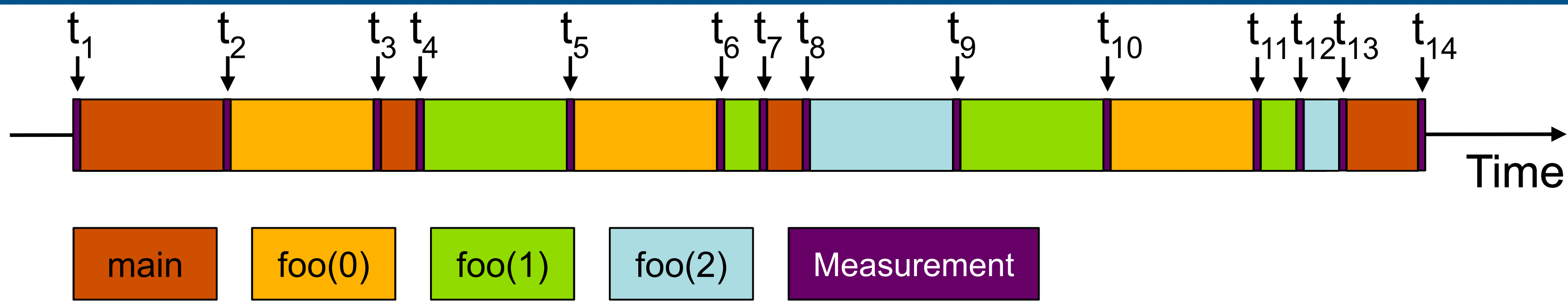
- How is performance data analyzed?
  - Online
  - Post mortem




```
int main()
{
  int i;

  for (i=0; i < 3; i++)
    foo(i);

  return 0;
}

void foo(int i)
{

  if (i > 0)
    foo(i – 1);

}
```

- Running program is periodically interrupted to take measurement
  - Timer interrupt, OS signal, or HWC overflow
  - Service routine examines return-address stack
  - Addresses are mapped to routines using symbol table information
- **Statistical** inference of program behavior
  - Not very detailed information on highly volatile metrics
  - Requires long-running applications
- Works with unmodified executables

Instrumentation




```
int main()
{
  int i;
  Enter("main");
  for (i=0; i < 3; i++)
    foo(i);
  Leave("main");
  return 0;
}

void foo(int i)
{
  Enter("foo");
  if (i > 0)
    foo(i – 1);
  Leave("foo");
}
```

- Measurement code is inserted such that every event of interest is captured **directly**
  - Can be done in various ways
- Advantage:
  - Much more detailed information
- Disadvantage:
  - Processing of source-code / executable necessary
  - Large relative overheads for small functions

Instrumentation techniques

- Static instrumentation
  - Program is instrumented prior to execution
- Dynamic instrumentation
  - Program is instrumented at runtime

- Code is inserted
  - Manually
  - Automatically
    - By a preprocessor / source-to-source translation tool
    - By a compiler
    - By linking against a pre-instrumented library / runtime system
    - By binary-rewrite / dynamic instrumentation tool

Critical issues

- Accuracy
  - Intrusion overhead
    - Measurement itself needs time and thus lowers performance
  - Perturbation
    - Measurement alters program behaviour
    - E.g., memory access pattern
  - Accuracy of timers & counters
- Granularity
  - How many measurements?
  - How much information / processing during each measurement?

☞ *Tradeoff: Accuracy vs. Expressiveness of data*

Classification of measurement techniques

- How are performance measurements triggered?
  - Sampling
  - Code instrumentation

- How is performance data recorded?
  - Profiling / Runtime summarization
  - Tracing

- How is performance data analyzed?
  - Online
  - Post mortem

Profiling / Runtime summarization

- Recording of aggregated information
  - Total, maximum, minimum, …
- For measurements
  - Time
  - Counts
    - Function calls
    - Bytes transferred
    - Hardware counters
- Over program and system entities
  - Functions, call sites, basic blocks, loops, …
  - Processes, threads

☞ *Profile = summarization of events over execution interval*

Types of profiles

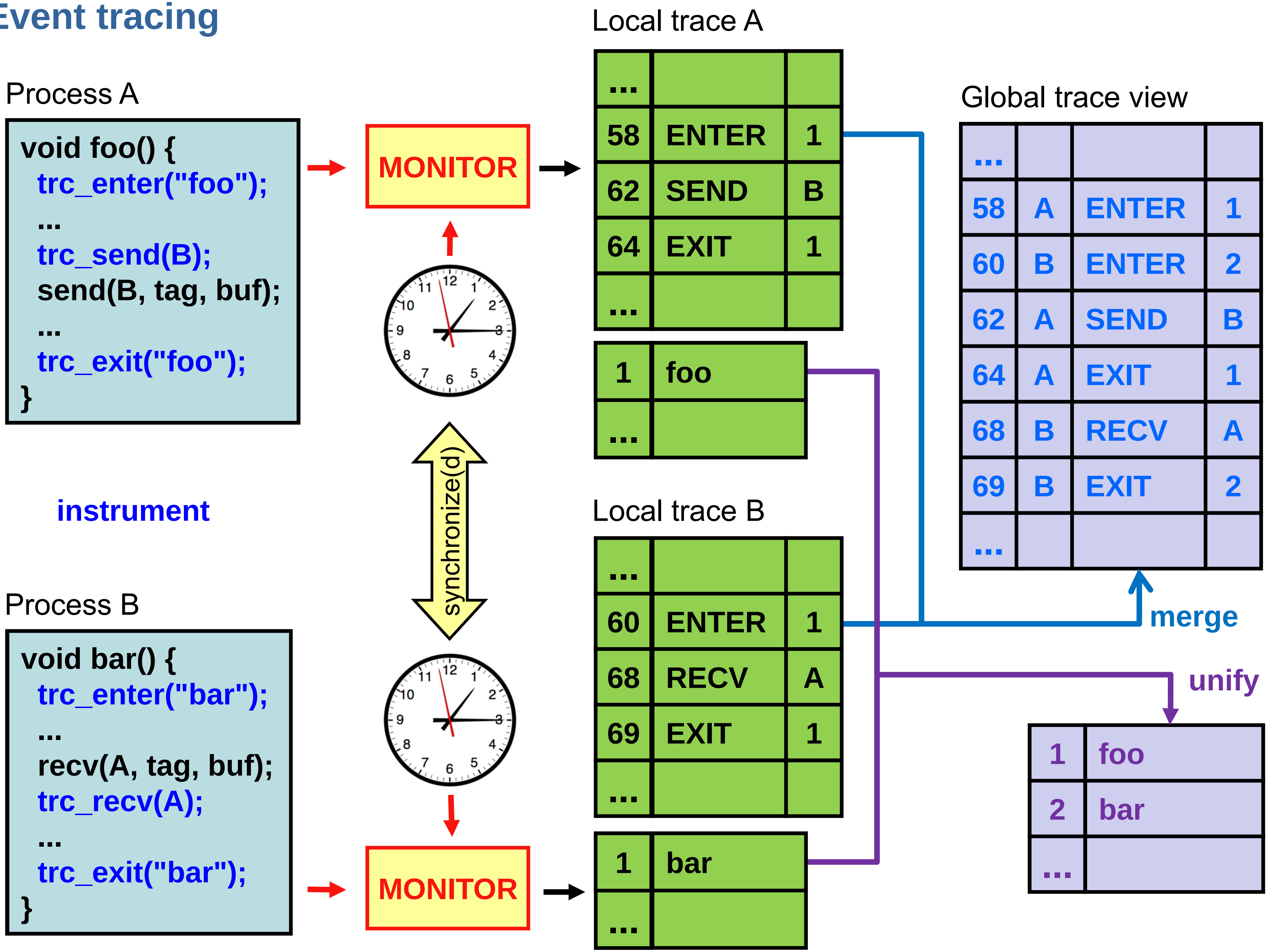
- Flat profile
  - Shows distribution of metrics per routine / instrumented region
  - Calling context is not taken into account
- Call-path profile
  - Shows distribution of metrics per executed call path
  - Sometimes only distinguished by partial calling context (e.g., two levels)
- Special-purpose profiles
  - Focus on specific aspects, e.g., MPI calls or OpenMP constructs
  - Comparing processes/threads

Tracing

- Recording information about significant points (events) during execution of the program
  - Enter / leave of a region (function, loop, …)
  - Send / receive a message, …
- Save information in event record
  - Timestamp, location, event type
  - Plus event-specific information (e.g., communicator, sender / receiver, …)
- Abstract execution model on level of defined events

☞ *Event trace = Chronologically ordered sequence of event records*

Event tracing



Tracing vs. Profiling

- Tracing advantages
  - Event traces preserve the **temporal** and **spatial** relationships among individual events (☞ context)
  - Allows reconstruction of **dynamic** application behaviour on any required level of abstraction
  - Most general measurement technique
    - Profile data can be reconstructed from event traces
- Disadvantages
  - Traces can very quickly become extremely large
  - Writing events to file at runtime causes perturbation
  - Writing tracing software is complicated
    - Event buffering, clock synchronization, ...

Classification of measurement techniques

- How are performance measurements triggered?
  - Sampling
  - Code instrumentation

- How is performance data recorded?
  - Profiling / Runtime summarization
  - Tracing

- How is performance data analyzed?
  - Online
  - Post mortem

Online analysis

- Performance data is processed during measurement run
  - Process-local profile aggregation
  - More sophisticated inter-process analysis using
    - “Piggyback” messages
    - Hierarchical network of analysis agents
- Inter-process analysis often involves application steering to interrupt and re-configure the measurement

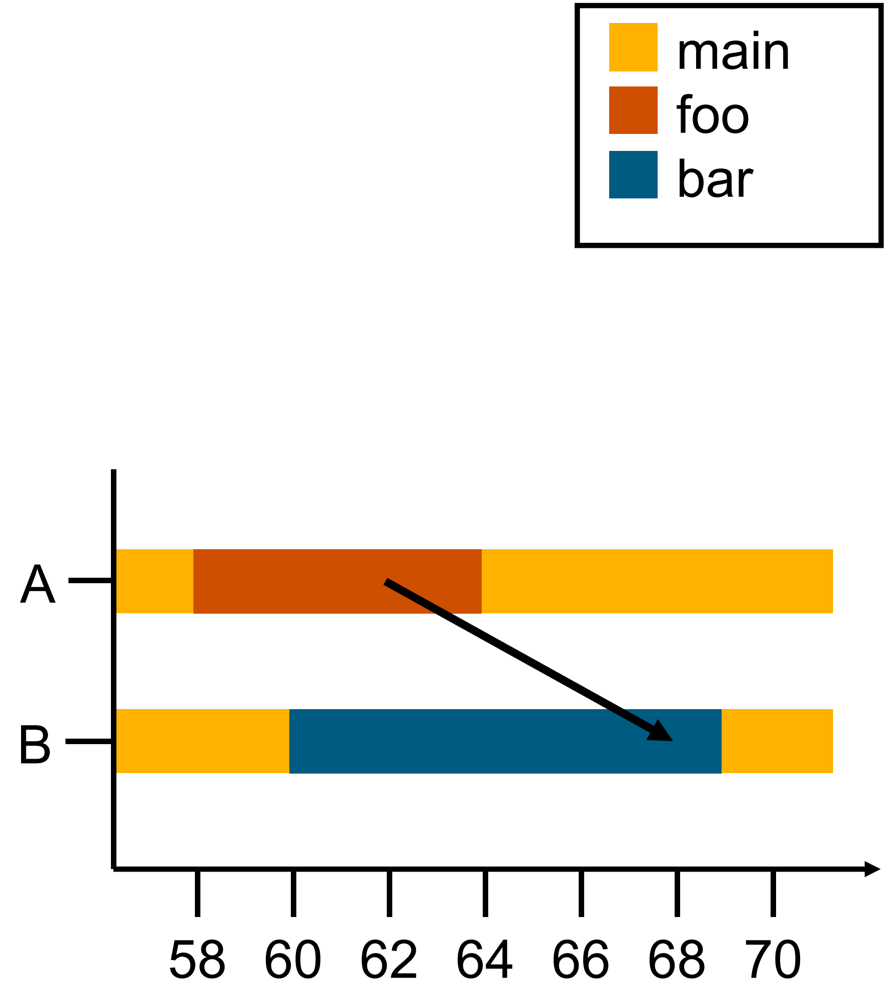
Post-mortem analysis

- Performance data is stored at end of measurement run
- Data analysis is performed afterwards
  - Automatic search for bottlenecks
  - Visual trace analysis
  - Calculation of statistics

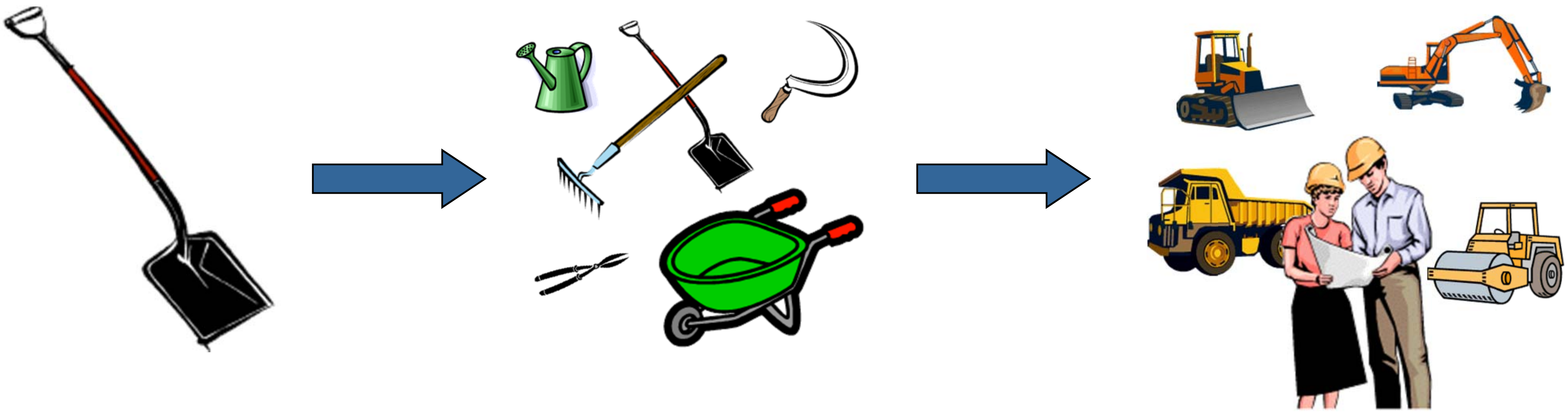
Example: Time-line visualization

1	foo
2	bar
3	...

...			
58	A	ENTER	1
60	B	ENTER	2
62	A	SEND	B
64	A	EXIT	1
68	B	RECV	A
69	B	EXIT	2
...			



No single solution is sufficient!



☞ *A combination of different methods, tools and techniques is typically needed!*

- Analysis
  - Statistics, visualization, automatic analysis, data mining, ...
- Measurement
  - Sampling / instrumentation, profiling / tracing, ...
- Instrumentation
  - Source code / binary, manual / automatic, ...

Typical performance analysis procedure

- Do I have a performance problem at all?
  - Time / speedup / scalability measurements
- What is the key bottleneck (computation / communication)?
  - MPI / OpenMP / flat profiling
- Where is the key bottleneck?
  - Call-path profiling, detailed basic block profiling
- Why is it there?
  - Hardware counter analysis, trace selected parts to keep trace size manageable
- Does the code have scalability problems?
  - Load imbalance analysis, compare profiles at various sizes function-by-function

Hands-on example code: NPB-MZ-MPI / BT (on Live-ISO/DVD)

VI-HPS Team

Performance Analysis Steps

0.0 Reference preparation for validation

1.0 Program instrumentation

1.1 Summary measurement collection

1.2 Summary analysis report examination

2.0 Summary experiment scoring

2.1 Summary measurement collection with filtering

2.2 Filtered summary analysis report examination

3.0 Event trace collection

3.1 Event trace examination & analysis

NPB-MZ-MPI suite

- The NAS Parallel Benchmark suite (MPI+OpenMP version)
  - Available from

    http://www.nas.nasa.gov/Software/NPB
  - 3 benchmarks in Fortran77
  - Configurable for various sizes & classes
- Move into the NPB3.3-MZ-MPI root directory

```
% cd Tutorial; ls
bin/    common/  jobscript/  Makefile  README.install   SP-MZ/
BT-MZ/  config/  LU-MZ/      README    README.tutorial  sys/
```

- Subdirectories contain source code for each benchmark
  - plus additional configuration and common code
- The provided distribution has already been configured for the tutorial, such that it's ready to “make” one or more of the benchmarks and install them into a (tool-specific) “bin” subdirectory

Building an NPB-MZ-MPI benchmark

- Type “make” for instructions

```
% make
  ===========================================
  =      NAS PARALLEL BENCHMARKS 3.3        =
  =      MPI+OpenMP Multi-Zone Versions     =
  =      F77                                =
  ===========================================


  To make a NAS multi-zone benchmark type

         make <benchmark-name> CLASS=<class> NPROCS=<nprocs>

  where <benchmark-name> is “bt-mz”, “lu-mz”, or “sp-mz”
        <class>          is “S”, “W”, “A” through “F”
        <nprocs>         is number of processes

  [...]

 ***************************************************************
 * Custom build configuration is specified in config/make.def  *
 * Suggested tutorial exercise configuration for LiveISO/DVD:  *
 *      make bt-mz CLASS=W NPROCS=4                            *
 ***************************************************************
```

Hint: the recommended build configuration is available via
% make suite

Building an NPB-MZ-MPI benchmark

- Specify the benchmark configuration
  - benchmark name: **bt-mz**, lu-mz, sp-mz
  - the number of MPI processes: NPROCS=**4**
  - the benchmark class (S, W, A, B, C, D, E): CLASS=**W**

```
% make bt-mz CLASS=W NPROCS=4
cd BT-MZ; make CLASS=W NPROCS=4 VERSION=
make: Entering directory 'BT-MZ'
cd ../sys; cc  -o setparams setparams.c
../sys/setparams bt-mz 4 W
mpif77 -c  -O3 -fopenmp bt.f
 [...]
cd ../common;  mpif77 -c  -O3 -fopenmp timers.f
mpif77 –O3 -fopenmp -o ../bin/bt-mz_W.4 \
bt.o initialize.o exact_solution.o exact_rhs.o set_constants.o \
adi.o rhs.o zone_setup.o x_solve.o y_solve.o exch_qbc.o \
solve_subs.o z_solve.o add.o error.o verify.o mpi_setup.o \
../common/print_results.o ../common/timers.o
Built executable ../bin/bt-mz_W.4
make: Leaving directory 'BT-MZ'
```

NPB-MZ-MPI / BT (Block Tridiagonal solver)

- What does it do?
  - Solves a discretized version of unsteady, compressible Navier-Stokes equations in three spatial dimensions
  - Performs 200 time-steps on a regular 3-dimensional grid
- Implemented in 20 or so Fortran77 source modules

- Uses MPI & OpenMP in combination
  - 4 processes with 4 threads each should be reasonable
    - don't expect to see speed-up when run on a laptop!
  - bt-mz_W.4 should run in around 5 to 12 seconds on a laptop
  - bt-mz_B.4 is more suitable for dedicated HPC compute nodes
    - Each class step takes around 10-15x longer

NPB-MZ-MPI / BT reference execution

- Launch as a hybrid MPI+OpenMP application

Alternatively execute script:
`% sh ../jobscript/ISO/run.sh`

```
% cd bin
% OMP_NUM_THREADS=4  mpiexec -np 4 ./bt-mz_W.4
 NAS Parallel Benchmarks (NPB3.3-MZ-MPI) - BT-MZ MPI+OpenMP Benchmark
 Number of zones:   4 x   4
 Iterations:  200   dt:   0.000800
 Number of active processes:     4
 Total number of threads:     16  (  4.0 threads/process)

 Time step    1
 Time step   20
 Time step   40
  [...]
 Time step  160
 Time step  180
 Time step  200
 Verification Successful

 BT-MZ Benchmark Completed.
 Time in seconds = 5.57
```

Hint: save the benchmark output (or note the run time) to be able to refer to it later

Tutorial exercise steps

- The tutorial steps are similar and repeated for each tool
- Edit config/make.def to adjust build configuration
  - Modify specification of compiler/linker: MPIF77
- Make clean and build new tool-specific executable

```
% make clean
% make bt-mz CLASS=W NPROCS=4
Built executable ../bin.$(TOOL)/bt-mz_W.4
```

- Change to the directory containing the new executable before running it with the desired tool configuration

```
% cd bin.$(TOOL)
% export …
% OMP_NUM_THREADS=4  mpiexec -np 4 ./bt-mz_W.4
```

NPB-MZ-MPI / BT: config/make.def

```
#               SITE- AND/OR PLATFORM-SPECIFIC DEFINITIONS
#---------------------------------------------------------------------
# Items in this file may need to be changed for each platform.
#---------------------------------------------------------------------
...
#---------------------------------------------------------------------
# The Fortran compiler used for MPI programs
#---------------------------------------------------------------------
MPIF77 = mpif77

# Alternative variants to perform instrumentation
#MPIF77 = psc_instrument -u user,mpi,omp –s ${PROGRAM}.sir mpif77
#MPIF77 = tau_f90.sh
#MPIF77 = scalasca -instrument mpif77
#MPIF77 = vtf77 –vt:hyb -vt:f77 mpif77
#MPIF77 = scorep --user mpif77

# PREP is a generic preposition macro for instrumentation preparation
#MPIF77 = $(PREP) mpif77

# This links MPI Fortran programs; usually the same as ${MPIF77}
FLINK   = $(MPIF77)
...
```

Default (no instrumentation)

Hint: uncomment one of these alternative compiler wrappers to perform instrumentation

Score-P – A Joint Performance Measurement Run-Time Infrastructure for Periscope, Scalasca, TAU, and Vampir

Markus Geimer[2)], Bert Wesarg[1)], Brian Wylie[2)]

With contributions from
Andreas Knüpfer[1)] and Christian Rössel[2)]
[1)]ZIH TU Dresden , [2)]FZ Jülich

Fragmentation of Tools Landscape

- Several performance tools co-exist
- Separate measurement systems and output formats
- Complementary features and overlapping functionality
- Redundant effort for development and maintenance
- Limited or expensive interoperability
- Complications for user experience, support, training

Vampir	Scalasca	TAU	Periscope
VampirTrace OTF	EPILOG / CUBE	TAU native formats	Online measurement

SILC Project Idea

- Start a community effort for a common infrastructure
  - Score-P instrumentation and measurement system
  - Common data formats OTF2 and CUBE4
- Developer perspective:
  - Save manpower by sharing development resources
  - Invest in new analysis functionality and scalability
  - Save efforts for maintenance, testing, porting, support, training
- User perspective:
  - Single learning curve
  - Single installation, fewer version updates
  - Interoperability and data exchange
- SILC project funded by BMBF
- Close collaboration PRIMA project funded by DOE

GEFÖRDERT VOM

Bundesministerium für Bildung und Forschung

Partners

- Forschungszentrum Jülich, Germany
- German Research School for Simulation Sciences, Aachen, Germany
- Gesellschaft für numerische Simulation mbH Braunschweig, Germany
- RWTH Aachen, Germany
- Technische Universität Dresden, Germany
- Technische Universität München, Germany
- University of Oregon, Eugene, USA

- Provide typical functionality for HPC performance tools
- Support all fundamental concepts of partner's tools

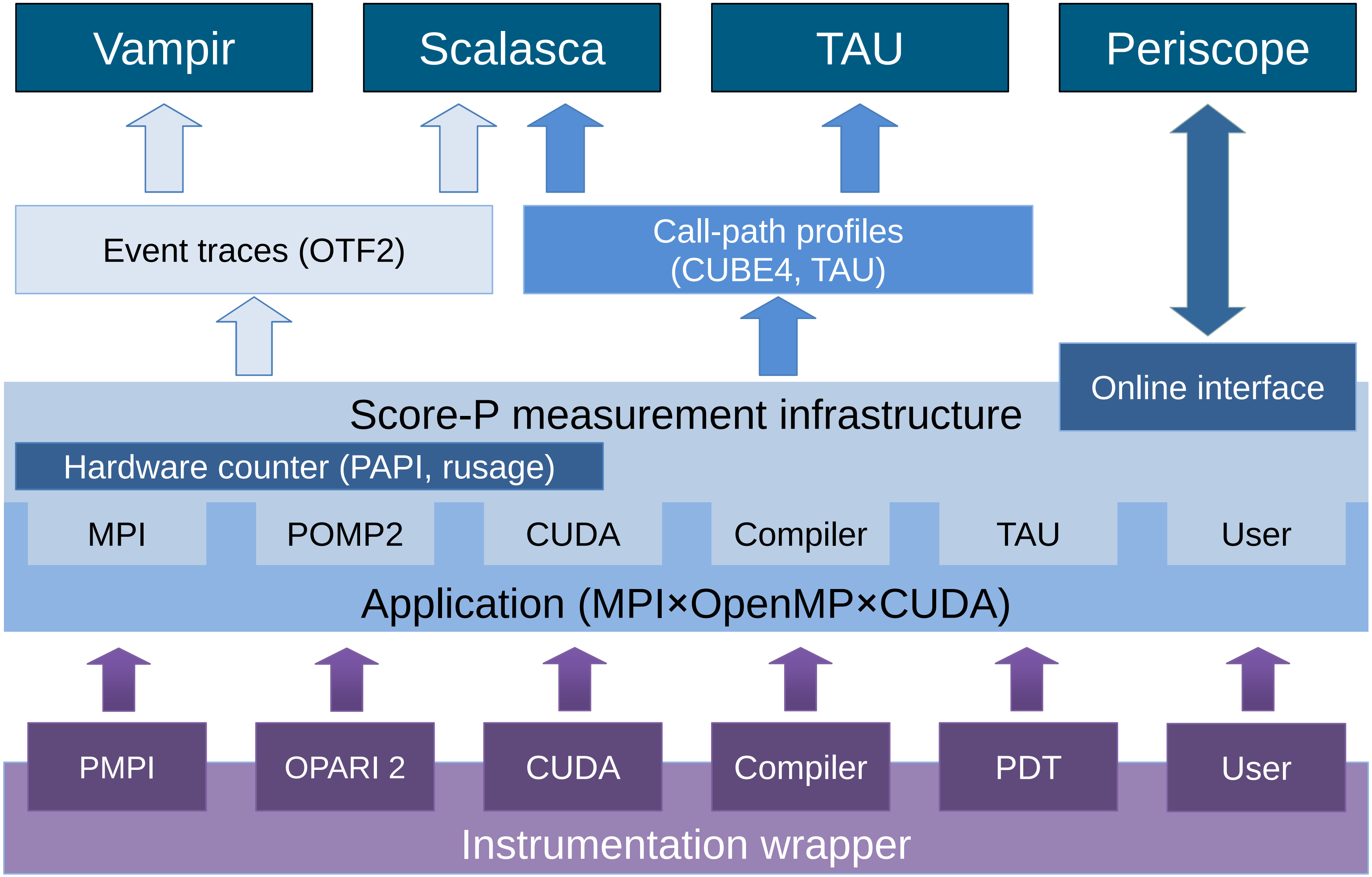
- Instrumentation (various methods)
- Flexible measurement without re-compilation:
  - Basic and advanced profile generation
  - Event trace recording
  - Online access to profiling data

- MPI, OpenMP, and hybrid parallelism (and serial)
- Enhanced functionality (OpenMP 3.0, CUDA, highly scalable I/O)

Design Goals

- Functional requirements
  - Generation of call-path profiles and event traces
  - Using direct instrumentation, later also sampling
  - Recording time, visits, communication data, hardware counters
  - Access and reconfiguration also at runtime
  - Support for MPI, OpenMP, basic CUDA, and all combinations
    - Later also OpenCL/HMPP/PTHREAD/…
- Non-functional requirements
  - Portability: all major HPC platforms
  - Scalability: petascale
  - Low measurement overhead
  - Easy and uniform installation through UNITE framework
  - Robustness
  - Open Source: New BSD License

Score-P Architecture



Future Features and Management

- Scalability to maximum available CPU core count
- Support for OpenCL, HMPP, PTHREAD
- Support for sampling, binary instrumentation
- Support for new programming models, e.g., PGAS
- Support for new architectures

- Ensure a single official release version at all times which will always work with the tools
- Allow experimental versions for new features or research

- Commitment to joint long-term cooperation

Score-P hands-on:
NPB-MZ-MPI / BT

Performance Analysis Steps

0.0 Reference preparation for validation

1.0 Program instrumentation

1.1 Summary measurement collection

1.2 Summary analysis report examination

2.0 Summary experiment scoring

2.1 Summary measurement collection with filtering

2.2 Filtered summary analysis report examination

3.0 Event trace collection

3.1 Event trace examination & analysis

- Change back to directory containing NPB BT-MZ

```
% cd ..
```

- Edit config/make.def to adjust build configuration
  - Modify specification of compiler/linker: MPIF77

```
...
#---------------------------------------------------------------------
# The Fortran compiler used for MPI programs
#---------------------------------------------------------------------
#MPIF77 = mpif77

# Alternative variants to perform instrumentation
...
MPIF77 = scorep mpif77

# This links MPI Fortran programs; usually the same as ${MPIF77}
FLINK   = $(MPIF77)
...
```

Uncomment the Score-P compiler wrapper specification

- Return to root directory and clean-up

```
% make clean
```

- Re-build executable using Score-P instrumenter

```
% make bt-mz CLASS=W NPROCS=4
cd BT-MZ; make CLASS=W NPROCS=4 VERSION=
make: Entering directory 'BT-MZ'
cd ../sys; cc  -o setparams setparams.c -lm
../sys/setparams bt-mz 4 W
scorep mpif77 -c  -O3 -fopenmp bt.f
 [...]
cd ../common;  scorep mpif77 -c  -O3 -fopenmp timers.f
scorep mpif77 –O3 -fopenmp -o ../bin.scorep/bt-mz_W.4 \
bt.o initialize.o exact_solution.o exact_rhs.o set_constants.o \
adi.o rhs.o zone_setup.o x_solve.o y_solve.o exch_qbc.o \
solve_subs.o z_solve.o add.o error.o verify.o mpi_setup.o \
../common/print_results.o ../common/timers.o
Built executable ../bin.scorep/bt-mz_W.4
make: Leaving directory 'BT-MZ'
```

Measurement Configuration: scorep-info

- Score-P measurements are configured via environment variables:

```
% scorep-info config-vars --full
SCOREP_ENABLE_PROFILING
   Description: Enable profiling
  [...]
SCOREP_ENABLE_TRACING
   Description: Enable tracing
  [...]
SCOREP_TOTAL_MEMORY
   Description: Total memory in bytes for the measurement system
  [...]
SCOREP_EXPERIMENT_DIRECTORY
   Description: Name of the experiment directory
  [...]
SCOREP_FILTERING_FILE
   Description: A file name which contain the filter rules
  [...]
SCOREP_METRIC_PAPI
   Description: PAPI metric names to measure
  [...]
SCOREP_METRIC_RUSAGE
   Description: Resource usage metric names to measure
  [... More configuration variables ...]
```

Summary Measurement Collection

- Change to the directory containing the new executable adjust configuration and run application

```
% cd bin.scorep
% export SCOREP_EXPERIMENT_DIRECTORY=scorep_bt-mz_W_4x4_sum
% OMP_NUM_THREADS=4  mpiexec -np 4 ./bt-mz_W.4
 NAS Parallel Benchmarks (NPB3.3-MZ-MPI) - BT-MZ MPI+OpenMP Benchmark
 Number of zones:   4 x   4
 Iterations:  200   dt:   0.000800
 Number of active processes:     4
 Use the default load factors with threads
 Total number of threads:     16  (  4.0 threads/process)
 Use the default load factors with threads

 Time step    1
 Time step   20
  [...]
 Time step  180
 Time step  200
 Verification Successful

 BT-MZ Benchmark Completed.
 Time in seconds = 54.39
```

BT-MZ Summary Analysis Report Examination

- Creates experiment directory ./scorep_bt-mz_W_4x4_sum containing
  - a record of the measurement configuration (scorep.cfg)
  - the analysis report that was collated after measurement (profile.cubex)

```
% ls
...  scorep_bt-mz_W_4x4_sum
% ls scorep_bt-mz_W_4x4_sum
profile.cubex  scorep.cfg
```

- Interactive exploration with CUBE / ParaProf

```
% cube scorep_bt-mz_W_4x4_sum/profile.cubex

             [CUBE GUI showing summary analysis report]

% paraprof scorep_bt-mz_W_4x4_sum/profile.cubex

         [TAU ParaProf GUI showing summary analysis report]
```

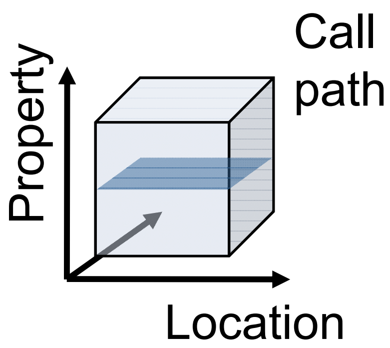
Analysis report examination with CUBE

Markus Geimer
Jülich Supercomputing Centre

- Parallel program analysis report exploration tools
  - Libraries for XML report reading & writing
  - Algebra utilities for report processing
  - GUI for interactive analysis exploration
    - requires Qt4
- Originally developed as part of Scalasca toolset
- Now available as a separate component
  - Can be installed independently of Score-P, e.g., on laptop or desktop
  - Latest release: CUBE 4.2 (August 2013)

Analysis presentation and exploration

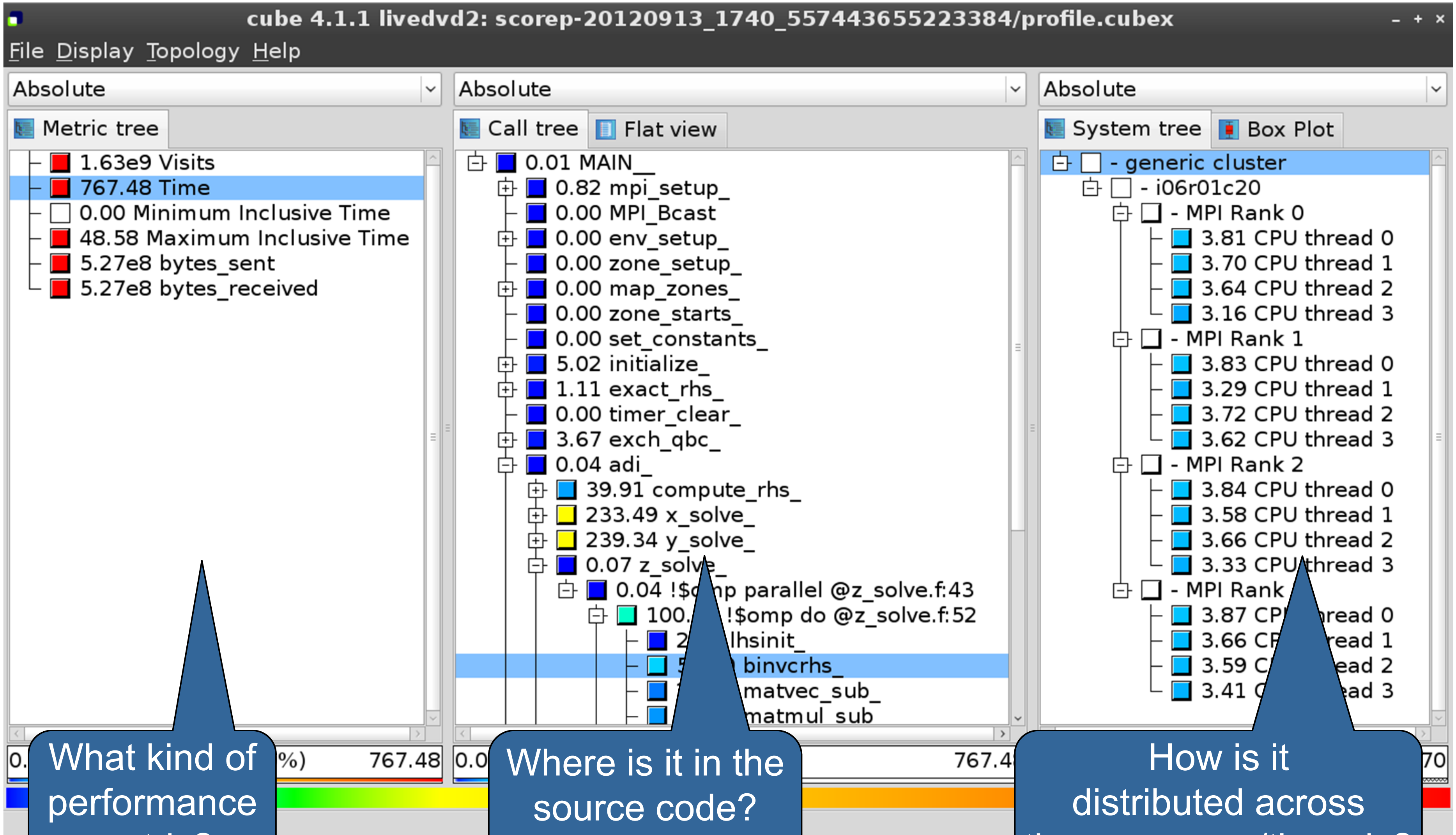
- Representation of values (severity matrix) on three hierarchical axes
  - Performance property (metric)
  - Call path (program location)
  - System location (process/thread)



- Three coupled tree browsers

- CUBE displays severities
  - As value: for precise comparison
  - As colour: for easy identification of hotspots
  - Inclusive value when closed & exclusive value when expanded
  - Customizable via display modes

Analysis presentation

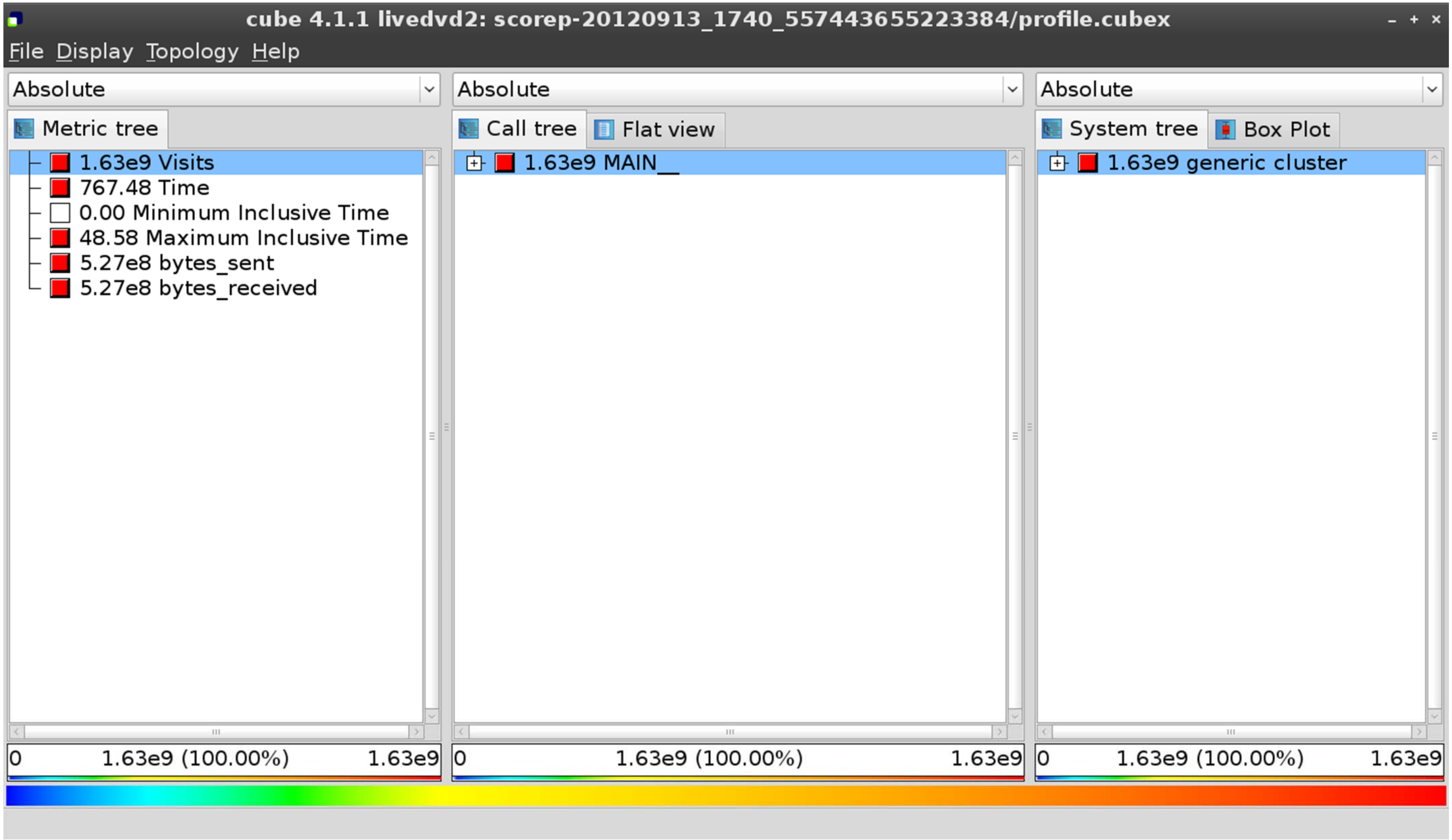


What kind of performance metric?

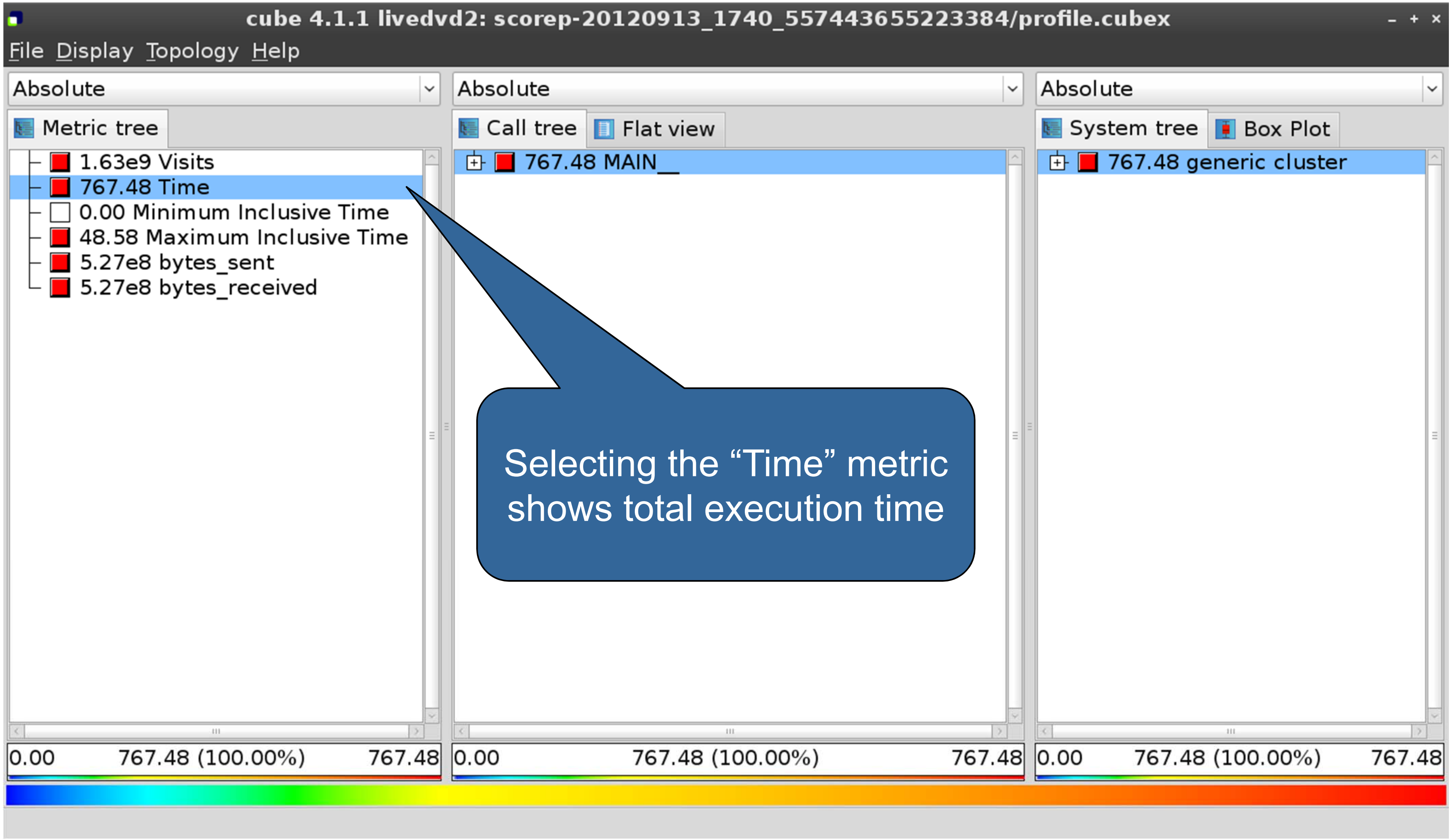
Where is it in the source code? In what context?

How is it distributed across the processes/threads?

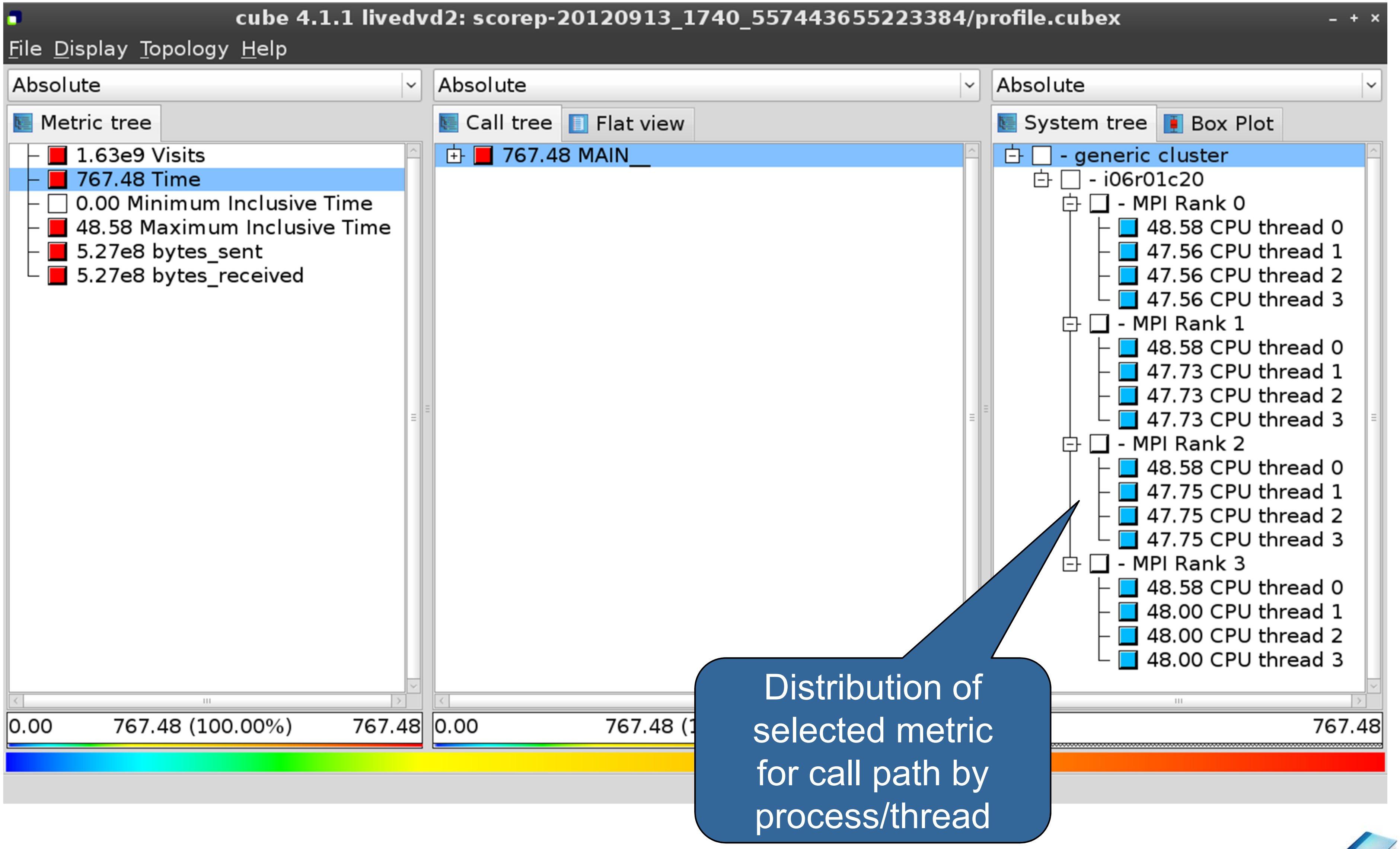
Analysis report exploration (opening view)



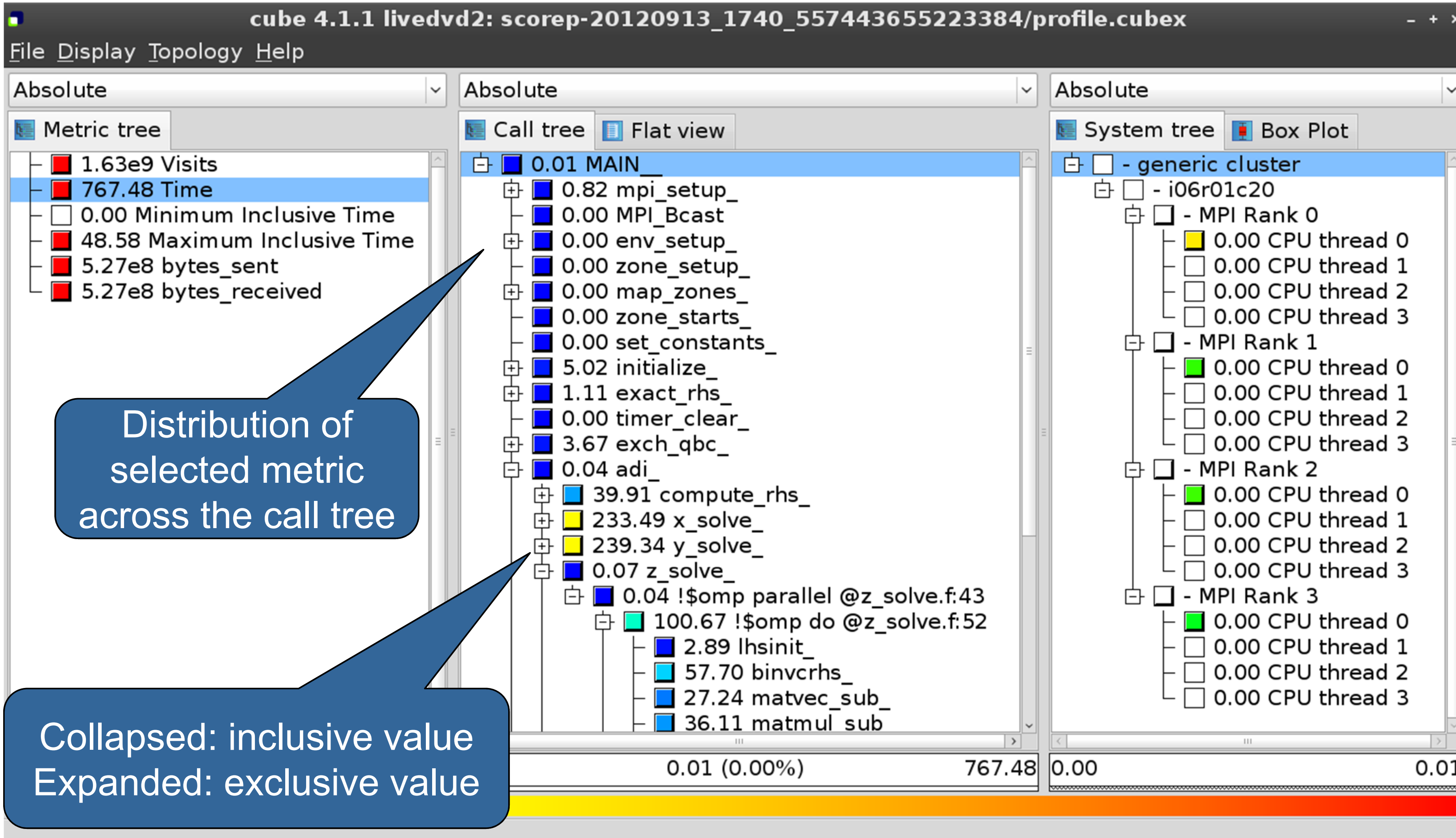
Metric selection



Expanding the system tree

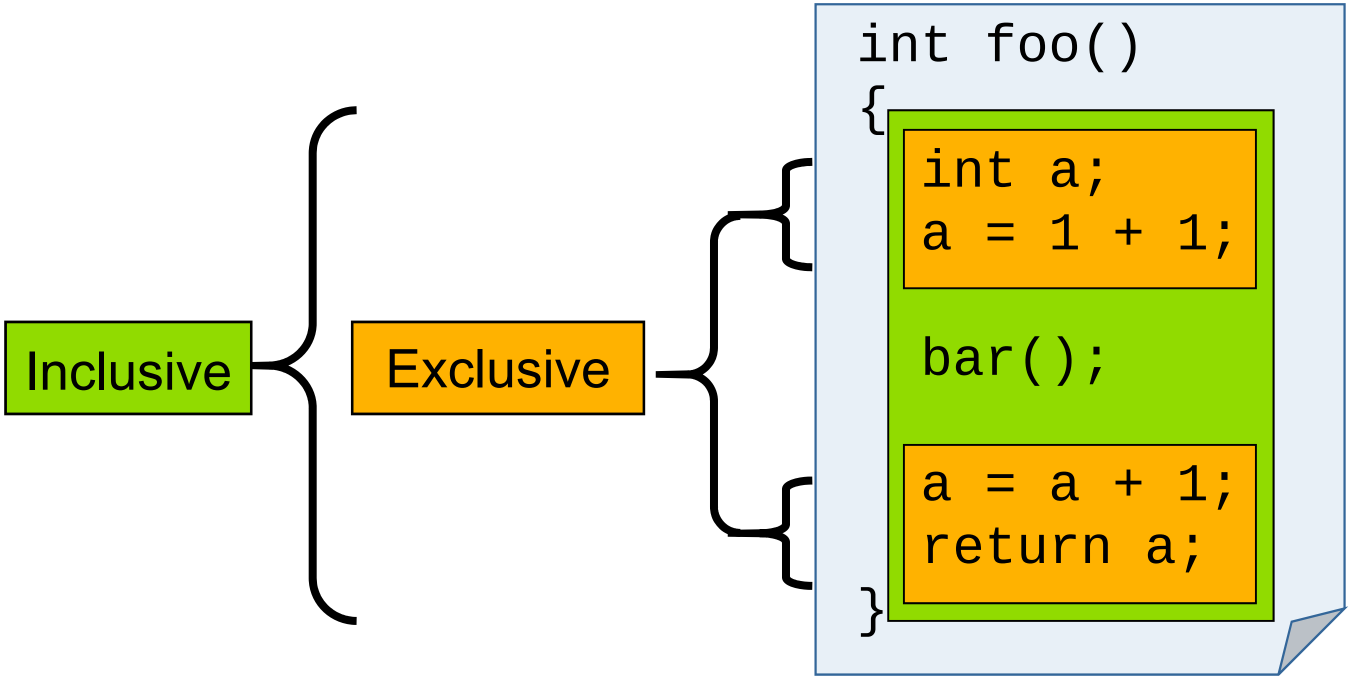


Expanding the call tree

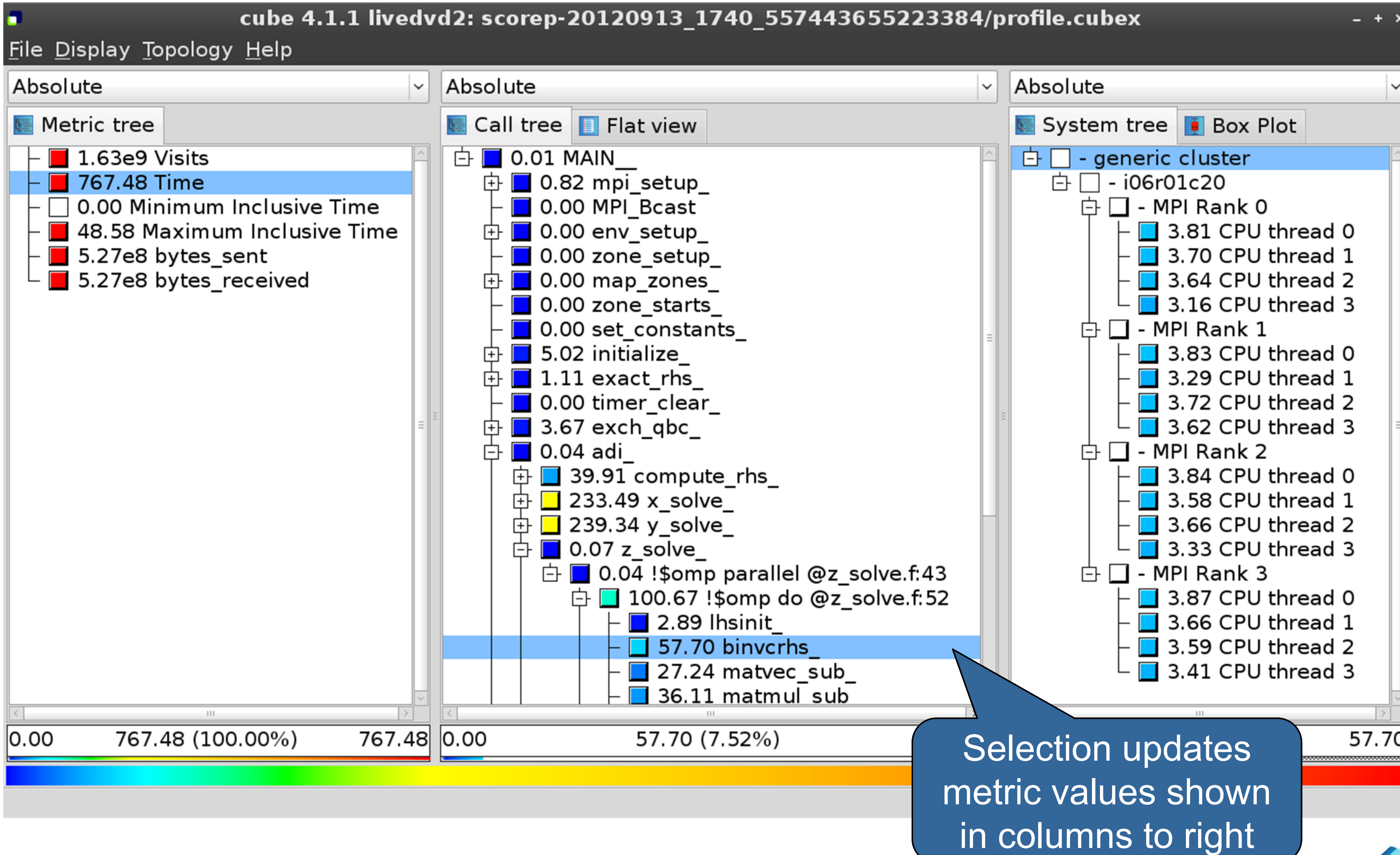


Inclusive vs. Exclusive values

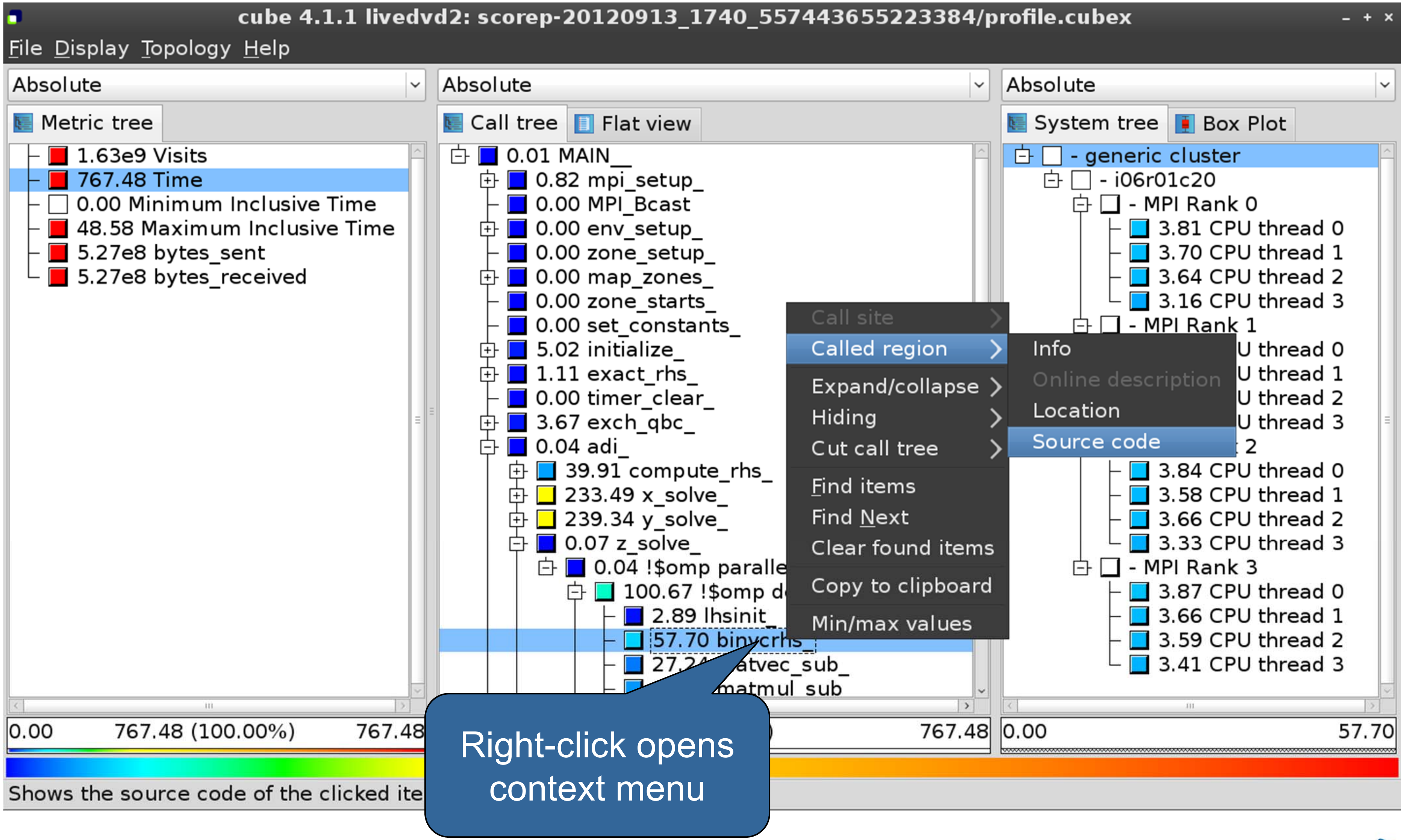
- Inclusive
  - Information of all sub-elements aggregated into single value
- Exclusive
  - Information cannot be subdivided further



Selecting a call path



Source-code view via context menu



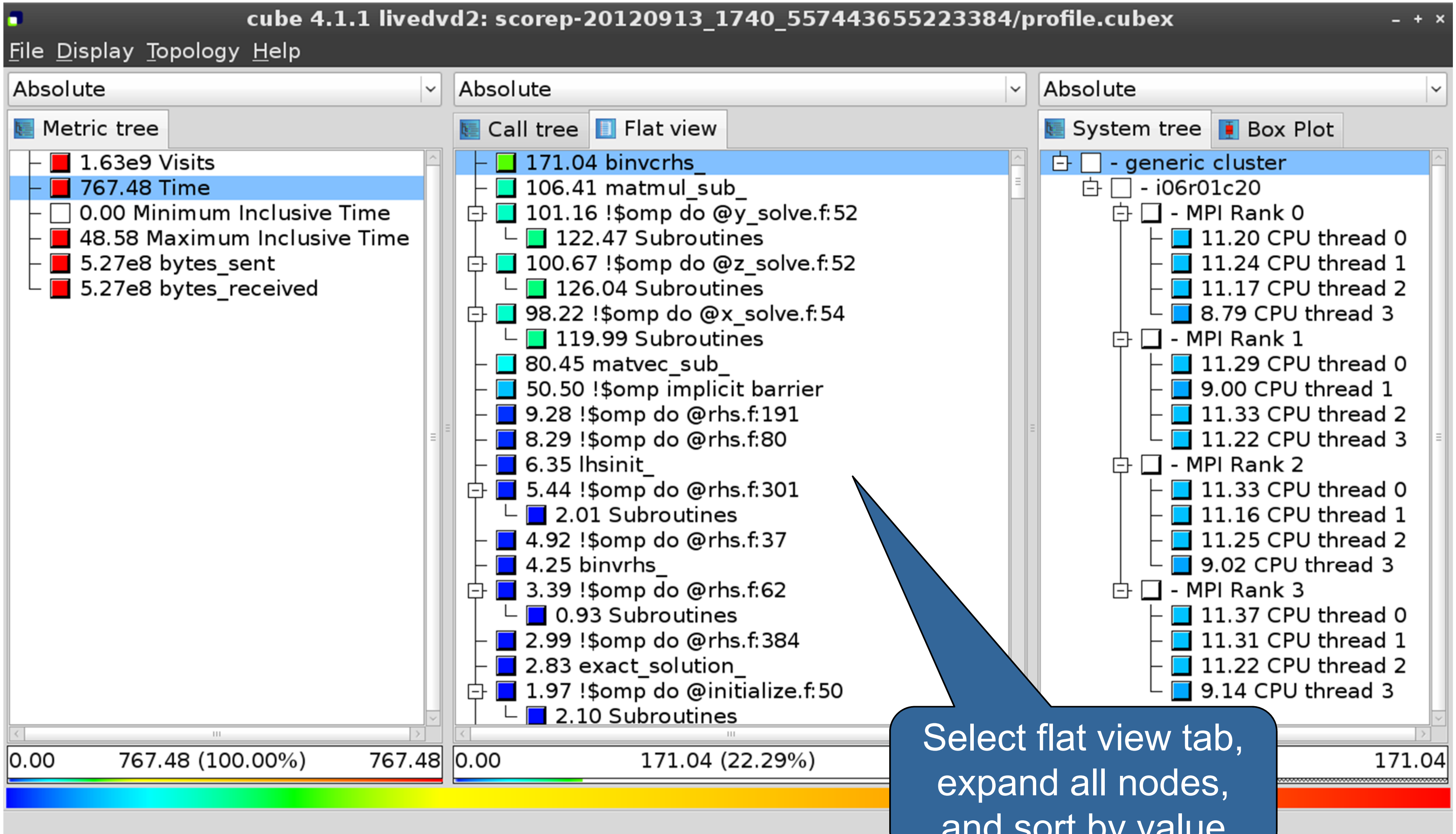
Source-code view

/home/geimer/Projects/Tests/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f

```
      subroutine binvcrhs( lhs,c,r )

c---------------------------------------------------------------------
c---------------------------------------------------------------------

c---------------------------------------------------------------------
c
c---------------------------------------------------------------------

      implicit none

      double precision pivot, coeff, lhs
      dimension lhs(5,5)
      double precision c(5,5), r(5)

c---------------------------------------------------------------------
c
c---------------------------------------------------------------------

      pivot = 1.00d0/lhs(1,1)
      lhs(1,2) = lhs(1,2)*pivot
      lhs(1,3) = lhs(1,3)*pivot
      lhs(1,4) = lhs(1,4)*pivot
      lhs(1,5) = lhs(1,5)*pivot
      c(1,1) = c(1,1)*pivot
      c(1,2) = c(1,2)*pivot
      c(1,3) = c(1,3)*pivot
      c(1,4) = c(1,4)*pivot
```

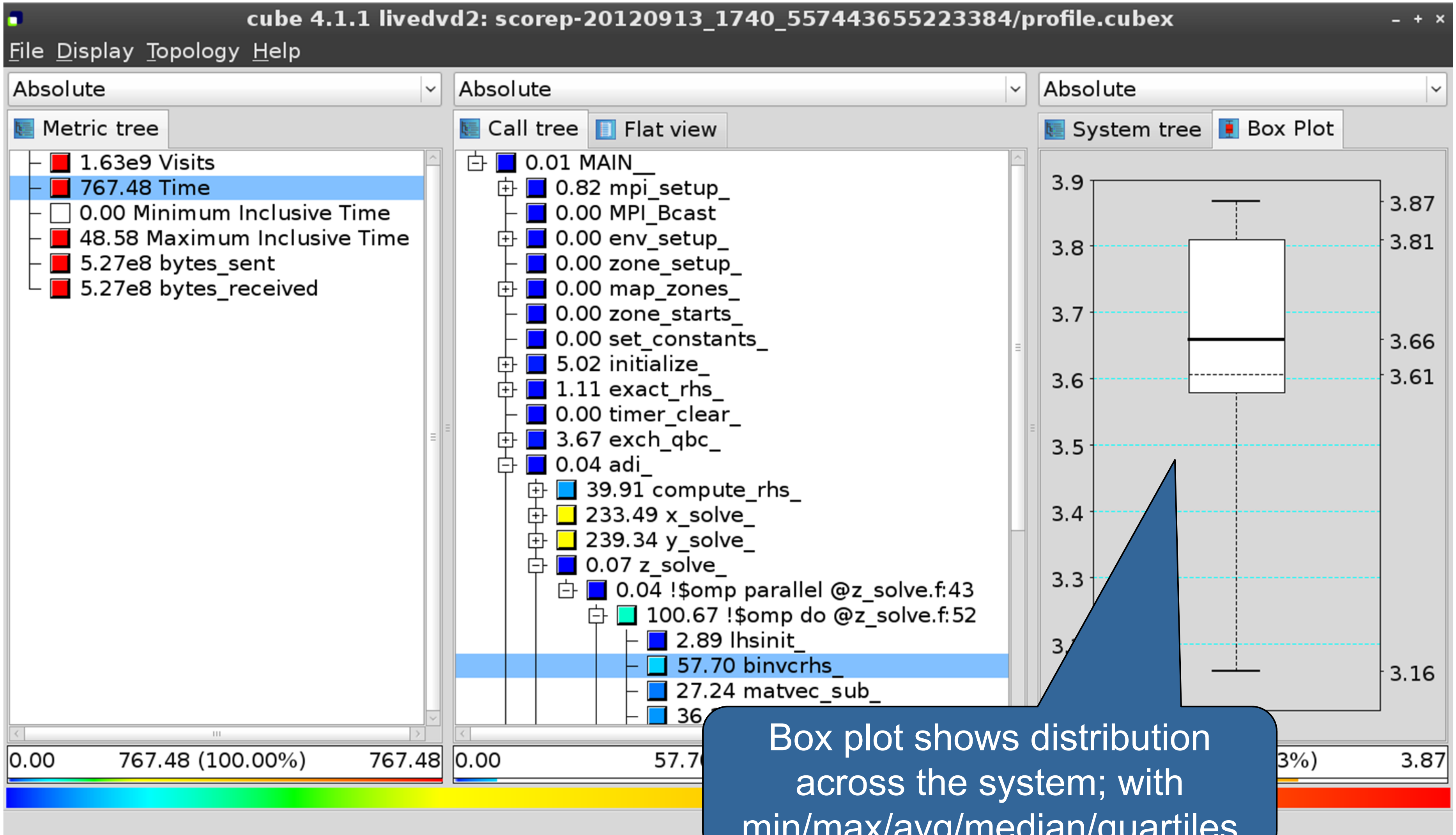
Read only | Save | Save as | Font... | Close

Flat profile view



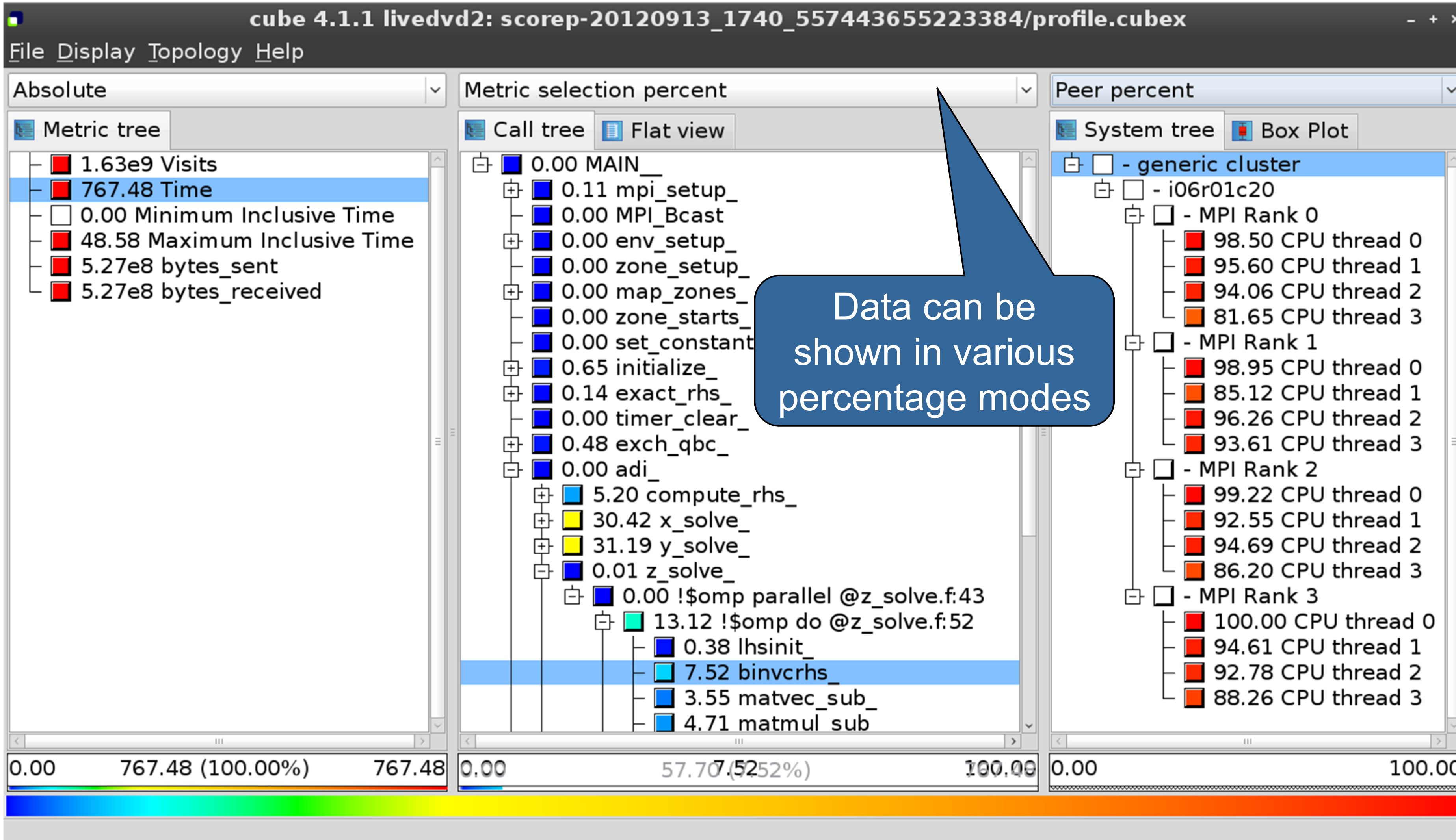
Select flat view tab, expand all nodes, and sort by value

Box plot view



Box plot shows distribution across the system; with min/max/avg/median/quartiles

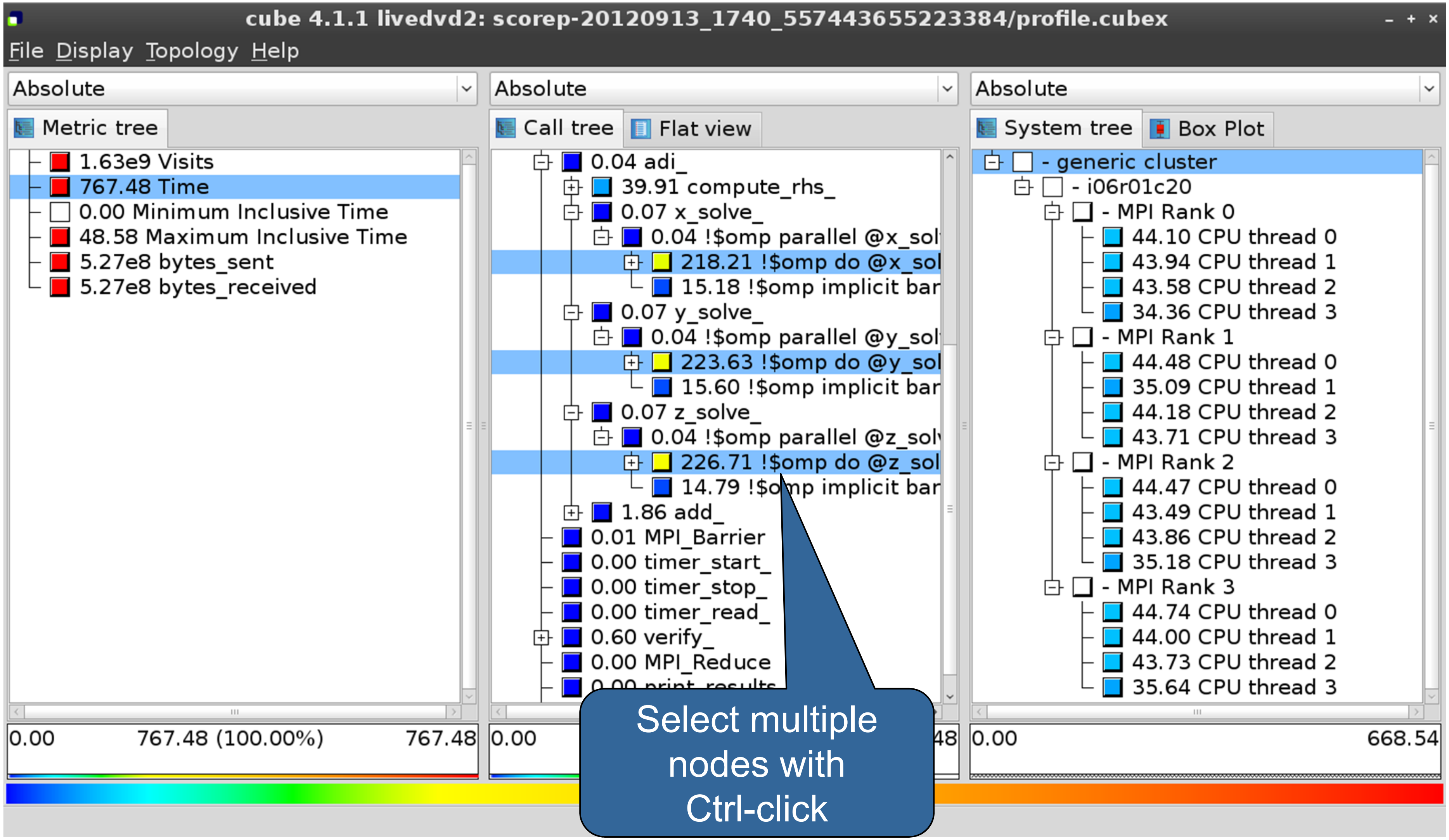
Alternative display modes



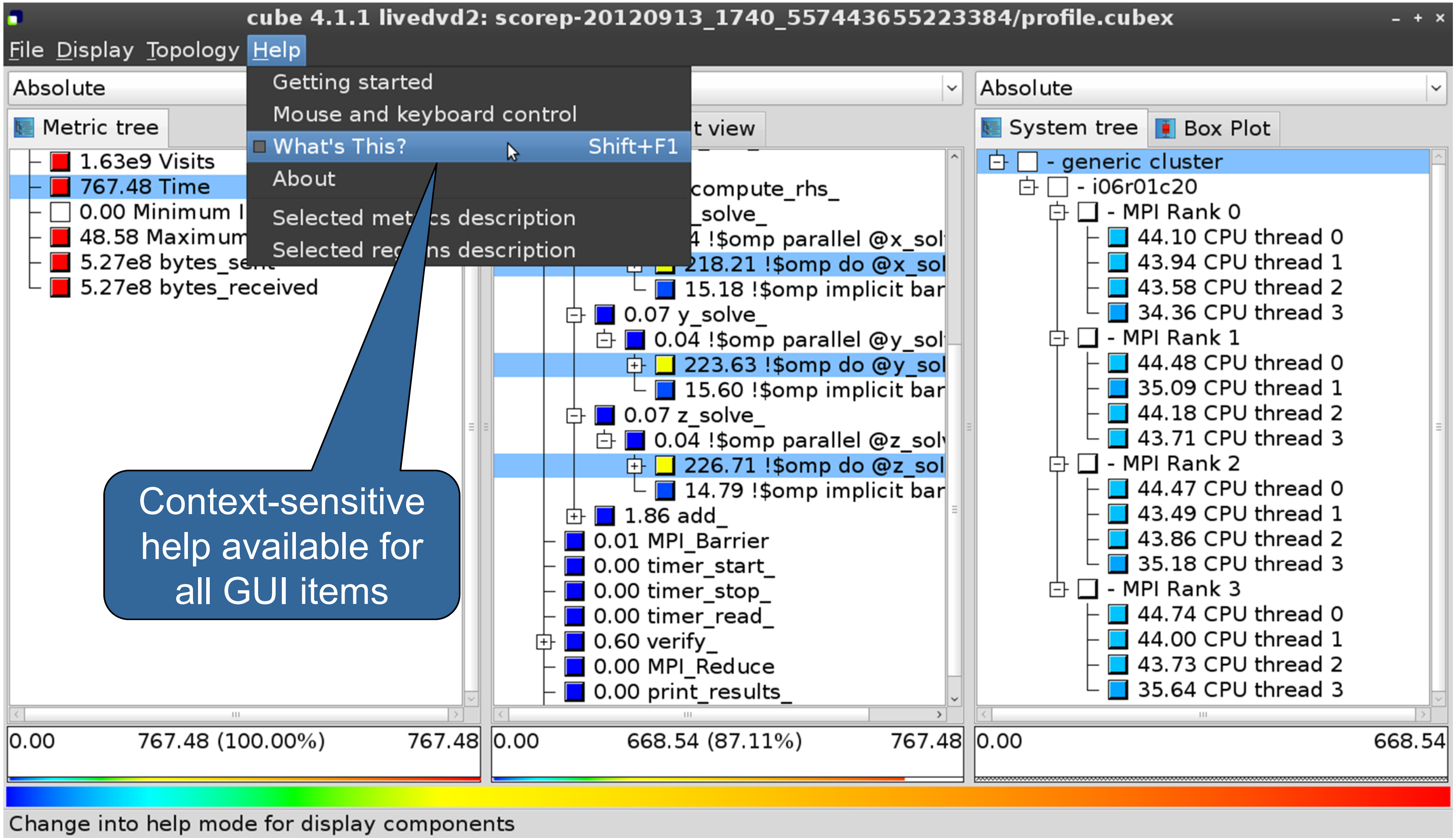
Important display modes

- Absolute
  - Absolute value shown in seconds/bytes/counts
- Selection percent
  - Value shown as percentage w.r.t. the selected node “on the left“ (metric/call path)
- Peer percent (system tree only)
  - Value shown as percentage relative to the maximum peer value

Multiple selection



Context-sensitive help



CUBE algebra utilities

- Extracting solver sub-tree from analysis report

```
% cube_cut  -r '<<ITERATION>>'  scorep_bt-mz_W_4x4_sum/profile.cubex
Writing cut.cubex... done.
```

- Calculating difference of two reports

```
% cube_diff  scorep_bt-mz_W_4x4_sum/profile.cubex  cut.cubex
Writing diff.cubex... done.
```

- Additional utilities for merging, calculating mean, etc.
  - Default output of cube_*utility* is a new report *utility*.cubex
- Further utilities for report scoring & statistics
- Run utility with “-h” (or no arguments) for brief usage info

Further information

- CUBE
  - Parallel program analysis report exploration tools
    - Libraries for XML report reading & writing
    - Algebra utilities for report processing
    - GUI for interactive analysis exploration
  - Available under New BSD open-source license
  - Documentation & sources:
    - http://www.scalasca.org
  - User guide also part of installation:
    - `cube-config --cube-dir`/share/doc/CubeGuide.pdf
  - Contact:
    - mailto: scalasca@fz-juelich.de

Score-P hands-on:
NPB-MZ-MPI / BT (filtered)

Performance Analysis Steps

0.0 Reference preparation for validation

1.0 Program instrumentation

1.1 Summary measurement collection

1.2 Summary analysis report examination

2.0 Summary experiment scoring

2.1 Summary measurement collection with filtering

2.2 Filtered summary analysis report examination

3.0 Event trace collection

3.1 Event trace examination & analysis

Congratulations!?

- If you made it this far, you successfully used Score-P to
  - instrument the application
  - analyze its execution with a summary measurement, and
  - examine it with one the interactive analysis report explorer GUIs
- ... revealing the call-path profile annotated with
  - the “Time” metric
  - Visit counts
  - MPI message statistics (bytes sent/received)
- ... but how ***good*** was the measurement?
  - The measured execution produced the desired valid result
  - however, the execution took rather longer than expected!
    - even when ignoring measurement start-up/completion, therefore
    - it was probably dilated by instrumentation/measurement overhead

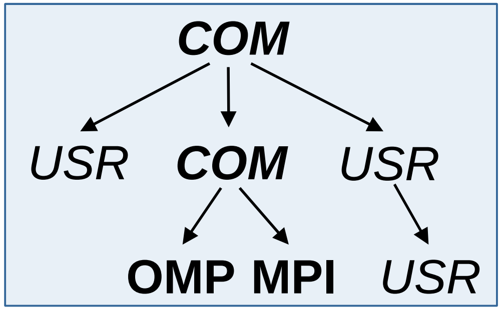
BT-MZ Summary Analysis Result Scoring

- Report scoring as textual output

```
% scorep-score scorep_bt-mz_W_4x4_sum/profile.cubex
Estimated aggregate size of event trace:                   909.683.150 bytes
Estimated requirements for largest trace buffer (max_tbc): 235.123.450 bytes
(hint: When tracing set SCOREP_TOTAL_MEMORY > max_tbc to avoid intermediate flushes
 or reduce requirements using file listing names of USR regions to be filtered.)

flt type         max_tbc         time      % region
     ALL       235123450       683.87  100.0 ALL
     USR       232516724       113.57   16.6 USR
     OMP         5973040       475.03   69.5 OMP
     COM          314732        66.30    9.7 COM
     MPI           88898        28.96    4.2 MPI
```

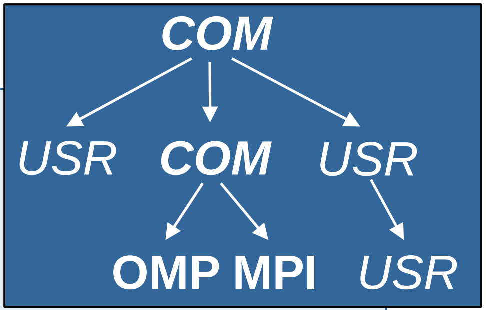
909 MB total memory
235 MB per rank!

- Region/callpath classification
  - MPI (pure MPI library functions)
  - OMP (pure OpenMP functions/regions)
  - USR (user-level source local computation)
  - COM ("combined" USR + OpenMP/MPI)
  - ANY/ALL (aggregate of all region types)



BT-MZ Summary Analysis Report Breakdown

- Score report breakdown by region




```
% scorep-score -r scorep_bt-mz_W_4x4_sum/profile.cubex
   [...]
flt type          max_tbc         time      % region
      ALL       235123450       683.87  100.0 ALL
      USR       232516724       113.57   16.6 USR
      OMP         5973040       475.03   69.5 OMP
      COM          314732        66.30    9.7 COM
                    88898        28.96    4.2 MPI

                 72578286        33.02    4.8 matvec_sub_
                 72578286        37.37    5.5 binvcrhs_
      USR        72578286        34.81    5.1 matmul_sub_
      USR         6747972         2.72    0.4 binvrhs_
      USR         6747972         3.41    0.5 lhsinit_
      USR         2939464         2.24    0.3 exact_solution_
      OMP          369840         0.05    0.0 !$omp parallel @exch…
      OMP          369840         0.06    0.0 !$omp parallel @exch…
      OMP          369840         0.06    0.0 !$omp parallel @exch…
      OMP          369840         0.06    0.0 !$omp parallel @exch…
   [...]
```

More than 232 MB just for these 6 regions

BT-MZ Summary Analysis Score

- Summary measurement analysis score reveals
  - Total size of event trace would be ~900 MB
  - Maximum trace buffer size would be ~235 MB per rank
    - smaller buffer would require flushes to disk during measurement resulting in substantial perturbation
  - 99.8% of the trace requirements are for USR regions
    - purely computational routines never found on COM call-paths common to communication routines or OpenMP parallel regions
  - These USR regions contribute around 32% of total time
    - however, much of that is very likely to be measurement overhead for frequently-executed small routines
- Advisable to tune measurement configuration
  - Specify an adequate trace buffer size
  - Specify a filter file listing (USR) regions not to be measured

BT-MZ Summary Analysis Report Filtering

- Report scoring with prospective filter listing
  6 USR regions

```
% cat ../config/scorep.filt
SCOREP_REGION_NAMES_BEGIN EXCLUDE
binvcrhs*
matmul_sub*
matvec_sub*
exact_solution*
binvrhs*
lhs*init*
timer_*

% scorep-score -f ../config/scorep.filt scorep_bt-mz_W_4x4_sum/profile.cubex
Estimated aggregate size of event trace:                   20.482.486 bytes
Estimated requirements for largest trace buffer (max_tbc):  6.377.264 bytes
(hint: When tracing set SCOREP_TOTAL_MEMORY > max_tbc to avoid intermediate flushes
 or reduce requirements using file listing names of USR regions to be filtered.)
```

20.5 MB of memory in total,
6.4 MB per rank!

BT-MZ Summary Analysis Report Filtering

- Score report breakdown by region

```
% scorep-score -r -f ../config/scorep.filt \
> scorep_bt-mz_W_4x4_sum/profile.cubex
flt type         max_tbc         time      % region
 *   ALL         6377264       570.30   83.4 ALL-FLT
 +   FLT       232516108       113.57   16.6 FLT
 -   OMP         5973040       475.03   69.5 OMP-FLT
 *   COM          314732        66.30    9.7 COM-FLT
 -   MPI           88898        28.96    4.2 MPI-FLT
 *   USR             616         0.00    0.0 USR-FLT

 +   USR        72578286        33.02    4.8 matvec_sub_
 +   USR        72578286        37.37    5.5 binvcrhs_
 +   USR        72578286        34.81    5.1 matmul_sub_
 +   USR         6747972         2.72    0.4 binvrhs_
 +   USR         6747972         3.41    0.5 lhsinit_
 +   USR         2939464         2.24    0.3 exact_solution_
 -   OMP          369840         0.05    0.0 !$omp parallel @exch…
 -   OMP          369840         0.06    0.0 !$omp parallel @exch…
 -   OMP          369840         0.06    0.0 !$omp parallel @exch…
 [...]
```

Filtered routines marked with '+'

BT-MZ Filtered Summary Measurement

- Set new experiment directory and re-run measurement with new filter configuration

```
% export SCOREP_EXPERIMENT_DIRECTORY=scorep_bt-mz_W_4x4_sum_filtered
% export SCOREP_FILTERING_FILE=../config/scorep.filt
% OMP_NUM_THREADS=4  mpiexec -np 4 ./bt-mz_W.4
 NAS Parallel Benchmarks (NPB3.3-MZ-MPI) - BT-MZ MPI+OpenMP Benchmark
 Number of zones:   4 x   4
 Iterations:  200   dt:   0.000800
 Number of active processes:     4
 Use the default load factors with threads
 Total number of threads:     16  (  4.0 threads/process)
 Use the default load factors with threads

 Time step    1
 Time step   20
  [...]
 Time step  180
 Time step  200
 Verification Successful

 BT-MZ Benchmark Completed.
 Time in seconds = 8.11
```

BT-MZ Tuned Summary Analysis Report Score

- Scoring of new analysis report as textual output

```
% scorep-score scorep_bt-mz_W_4x4_sum_filtered/profile.cubex
Estimated aggregate size of event trace:                   20.482.486 bytes
Estimated requirements for largest trace buffer (max_tbc):  6.377.264 bytes
(hint: When tracing set SCOREP_TOTAL_MEMORY > max_tbc to avoid intermediate flushes
 or reduce requirements using file listing names of USR regions to be filtered.)

flt type         max_tbc         time      % region
     ALL         6377264        74.16  100.0 ALL
     OMP         5973040        45.45   61.3 OMP
     COM          314732         9.77   13.2 COM
     MPI           88898        18.94   25.5 MPI
     USR             616         0.00    0.0 USR
```

- Significant reduction in runtime (measurement overhead)
  - Not only reduced time for USR regions, but MPI/OMP reduced too!
- Further measurement tuning (filtering) may be appropriate
  - e.g., use “timer_*” to filter timer_start_, timer_read_, etc.

Hardware performance/soft counter measurements hands-on

VI-HPS Team

Advanced Measurement Configuration: Metrics

- If Score-P has been built with performance metric support it is capable of recording performance counter information
- Requested counters will be recorded with every enter/exit event
- Supported metric sources
  - PAPI
  - Resource usage statistics

Note: Additional memory is needed to store metric values. Therefore, you may have to adjust SCOREP_TOTAL_MEMORY, for example as reported using “scorep-score -c”

Advanced Measurement Configuration: Metrics

- Recording hardware counters via PAPI

```
% export SCOREP_METRIC_PAPI=PAPI_TOT_INS,PAPI_FP_INS
% OMP_NUM_THREADS=4 mpiexec –n 4 ./bt-mz_W.4

 NAS Parallel Benchmarks (NPB3.3-MZ-MPI) - BT-MZ MPI+OpenMP Benchmark

 [... More application output ...]
```

- Also possible to record them only per rank

```
% export SCOREP_METRIC_PAPI_PER_PROCESS=PAPI_L3_DCM
% OMP_NUM_THREADS=4 mpiexec –n 4 ./bt-mz_W.4

 NAS Parallel Benchmarks (NPB3.3-MZ-MPI) - BT-MZ MPI+OpenMP Benchmark

 [... More application output ...]
```

- Available PAPI metrics
  - Preset events: common set of events deemed relevant and useful for application performance tuning
    - Abstraction from specific hardware performance counters, mapping onto available events done by PAPI internally

```
% papi_avail
```

  - Native events: set of all events that are available on the CPU (**platform dependent**)

```
% papi_native_avail
```

Note:
Due to hardware restrictions
- number of concurrently measured events is limited
- there may be unsupported combinations of concurrent events
- Use `papi_event_chooser` tool to test event combinations

Advanced Measurement Configuration: Metrics

- Recording operating system resource usage

```
% export SCOREP_METRIC_RUSAGE=ru_stime
% OMP_NUM_THREADS=4 mpiexec –n 4 ./bt-mz_W.4

 NAS Parallel Benchmarks (NPB3.3-MZ-MPI) - BT-MZ MPI+OpenMP Benchmark

 [... More application output ...]
```

- Also possible to record them only per rank

```
% export SCOREP_METRIC_RUSAGE_PER_PROCESS=ru_maxrss
% OMP_NUM_THREADS=4 mpiexec –n 4 ./bt-mz_W.4

 NAS Parallel Benchmarks (NPB3.3-MZ-MPI) - BT-MZ MPI+OpenMP Benchmark

 [... More application output ...]
```

- Available resource usage metrics

```
% man getrusage
 [... Output ...]

struct rusage {
    struct timeval ru_utime; /* user CPU time used */
    struct timeval ru_stime; /* system CPU time used */
    long   ru_maxrss;        /* maximum resident set size */
    long   ru_ixrss;         /* integral shared memory size */
    long   ru_idrss;         /* integral unshared data size */
    long   ru_isrss;         /* integral unshared stack size */
    long   ru_minflt;        /* page reclaims (soft page faults) */
    long   ru_majflt;        /* page faults (hard page faults) */
    long   ru_nswap;         /* swaps */
    long   ru_inblock;       /* block input operations */
    long   ru_oublock;       /* block output operations */
    long   ru_msgsnd;        /* IPC messages sent */
    long   ru_msgrcv;        /* IPC messages received */
    long   ru_nsignals;      /* signals received */
    long   ru_nvcsw;         /* voluntary context switches */
    long   ru_nivcsw;        /* involuntary context switches */
};

 [... More output ...]
```

Note:
(1) Not all fields are maintained on each platform.
(2) Check scope of metrics (per process vs. per thread)

Score-P hands-on

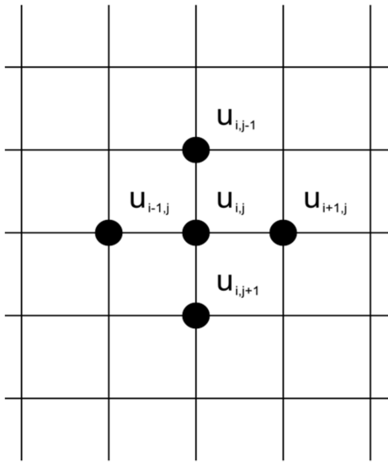
CUDA: Jacobi example

VI-HPS Team

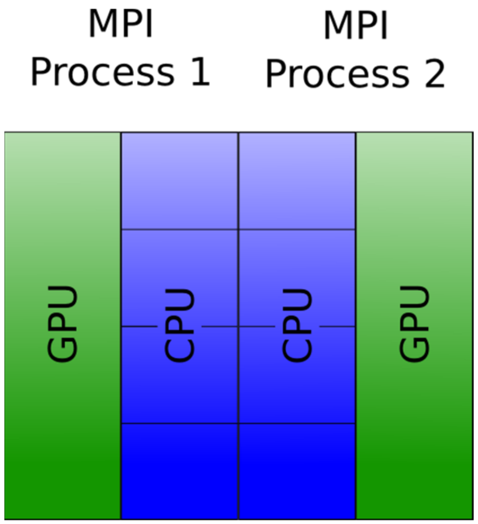
- Jacobi Example
  - Iterative solver for system of equations

$$U_{old} = U$$
$$u_{i,j} = bu_{old,i,j} + a_x(u_{old,i-1,j} + u_{old,i+1,j}) + a_y(u_{old,i,j-1} + u_{old,i,j+1}) - rHs/b$$

  - Code uses OpenMP, CUDA and MPI for parallelization



- Domain decomposition
  - Halo exchange at boundaries:
    - Via MPI between processes
    - Via CUDA between hosts and accelerators



Jacobi Without Instrumentation

```
# Compile host code
%        mpicc -O3 -fopenmp -DUSE_MPI –I<path_to_cuda_header>
          -c jacobi_cuda.c -o jacobi_mpi+cuda.o

# Compile CUDA kernel
%        nvcc -O3 -c jacobi_cuda_kernel.cu
          -o jacobi_cuda_kernel.o

# Link executable
%        mpicc -fopenmp -lm –L<path_tocuda_libs> -lcudart
          jacobi_mpi+cuda.o jacobi_cuda_kernel.o -o ./jacobi_mpi+cuda
```

Instrumentation with Score-P

```
# Compile host code
% scorep mpicc -O3 -fopenmp -DUSE_MPI –I<path_to_cuda_header>
         -c jacobi_cuda.c -o jacobi_mpi+cuda.o

# Compile CUDA kernel
% scorep nvcc -O3 -c jacobi_cuda_kernel.cu
         -o jacobi_cuda_kernel.o

# Link executable
% scorep mpicc -fopenmp -lm –L<path_tocuda_libs> -lcudart
         jacobi_mpi+cuda.o jacobi_cuda_kernel.o -o ./jacobi_mpi+cuda
```

CUDA Advanced Measurement Configuration

- Enable recording of CUDA events with the CUPTI interface via environment variable **`SCOREP_CUDA_ENABLE`**

- Provide a list of recording types, e.g.

```
% export SCOREP_CUDA_ENABLE=runtime,driver,gpu,kernel,idle
```

- Start with using the default configuration

```
% export SCOREP_CUDA_ENABLE=yes
```

- Adjust CUPTI buffer size (in bytes) as needed

```
% export SCOREP_CUDA_BUFFER=100000
```

SCOREP_CUDA_ENABLE: Recording Types

Recording type	Remark
yes/DEFAULT/1	"runtime, kernel, concurrent, memcpy"
no	Disable CUDA measurement (same as unset SCOREP_CUDA_ENABLE)
runtime	CUDA runtime API
driver	CUDA driver API
kernel	CUDA kernels
kernel_counter	Fixed CUDA kernel metrics
concurrent	Concurrent kernel recording
idle	GPU compute idle time
pure_idle	GPU idle time (memory copies are not idle)
memcpy	CUDA memory copies
sync	Record implicit and explicit CUDA synchronization
gpumemusage	Record CUDA memory (de)allocations as a counter
stream_reuse	Reuse destroyed/closed CUDA streams
device_reuse	Reuse destroyed/closed CUDA devices

Measurement (Profiling)

```
% export OMP_NUM_THREADS=6
% export SCOREP_CUDA_ENABLE=yes
% export SCOREP_CUDA_BUFFER=500000
% export SCOREP_EXPERIMENT_DIRECTORY=jacobi_cuda_profile

% mpirun -n 2 ./jacobi_mpi+cuda 4096 4096 0.15

Jacobi relaxation Calculation: 4096 x 4096 mesh with
 2 processes and 6 threads + one Tesla T10 Processor for each process.
 307 of 2049 local rows are calculated on the CPU to balance the load
 between the CPU and the GPU.
    0, 0.113429
  … … … … … …
  900, 0.000101
 total: 12.83581
```

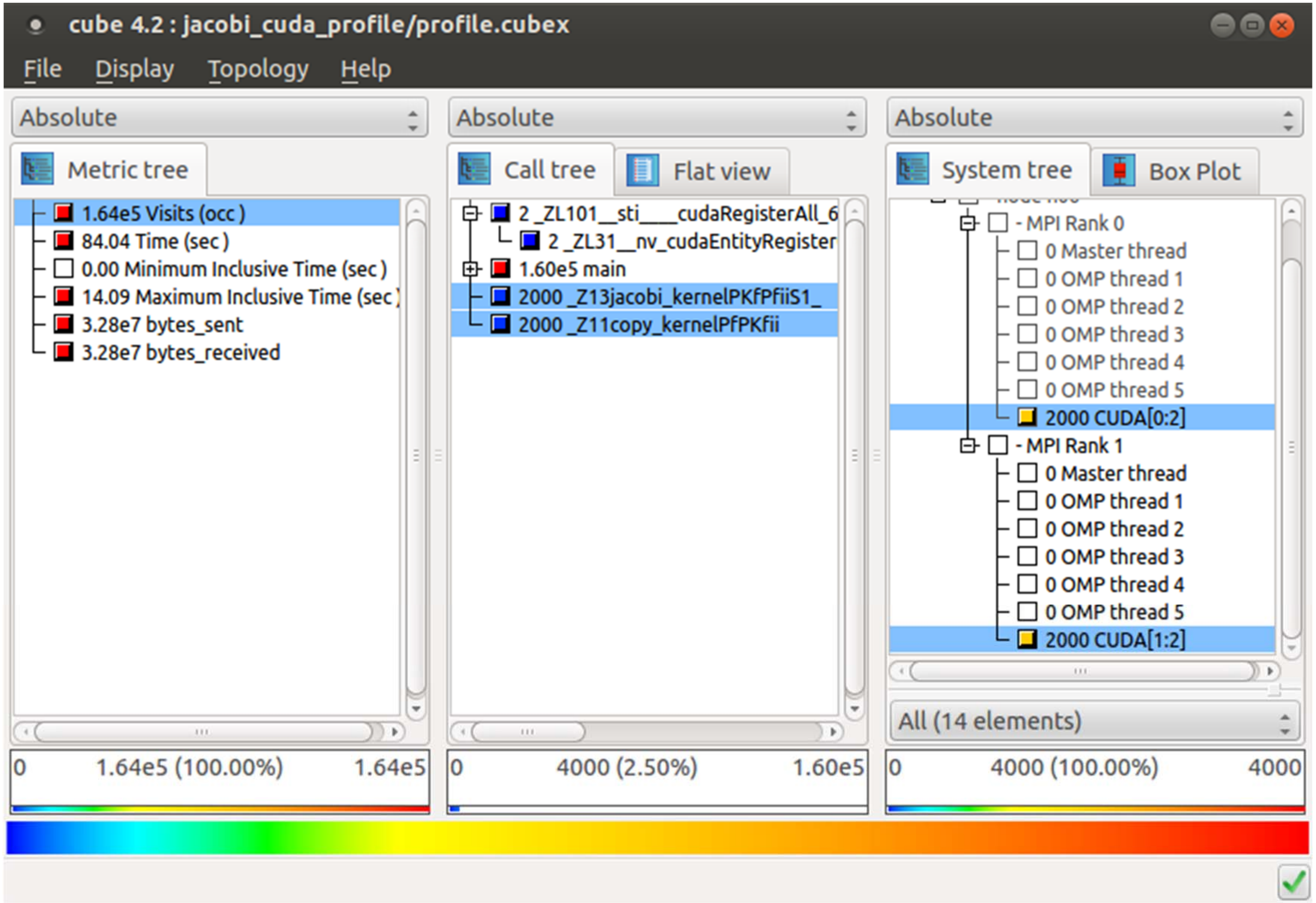
Problem size
(x dimension)

Problem size
(y dimension)

Load balancing factor
(in this example 15% of the computations are calculated on the CPU)

CUBE4 Analysis

```
% cube jacobi_cuda_profile/profile.cubex
```



Scoring

- Do we need to filter? (Overhead and memory footprint)

```
% scorep-score jacobi_cuda_profile/profile.cubex
Estimated aggregate size of event trace (total_tbc):       3.875.472 bytes
Estimated requirements for largest trace buffer (max_tbc): 1.937.936 bytes
(hint: When tracing set SCOREP_TOTAL_MEMORY > max_tbc to avoid
       intermediate flushes or reduce requirements using file listing
       names of USR regions to be filtered.)

flt type         max_tbc         time      % region
     ALL         1937936        24.97  100.0 ALL
     OMP         1154110        18.78   75.2 OMP
     USR          667480         5.95   23.8 USR
     MPI          116192         0.14    0.5 MPI
     COM             154         0.10    0.4 COM
```

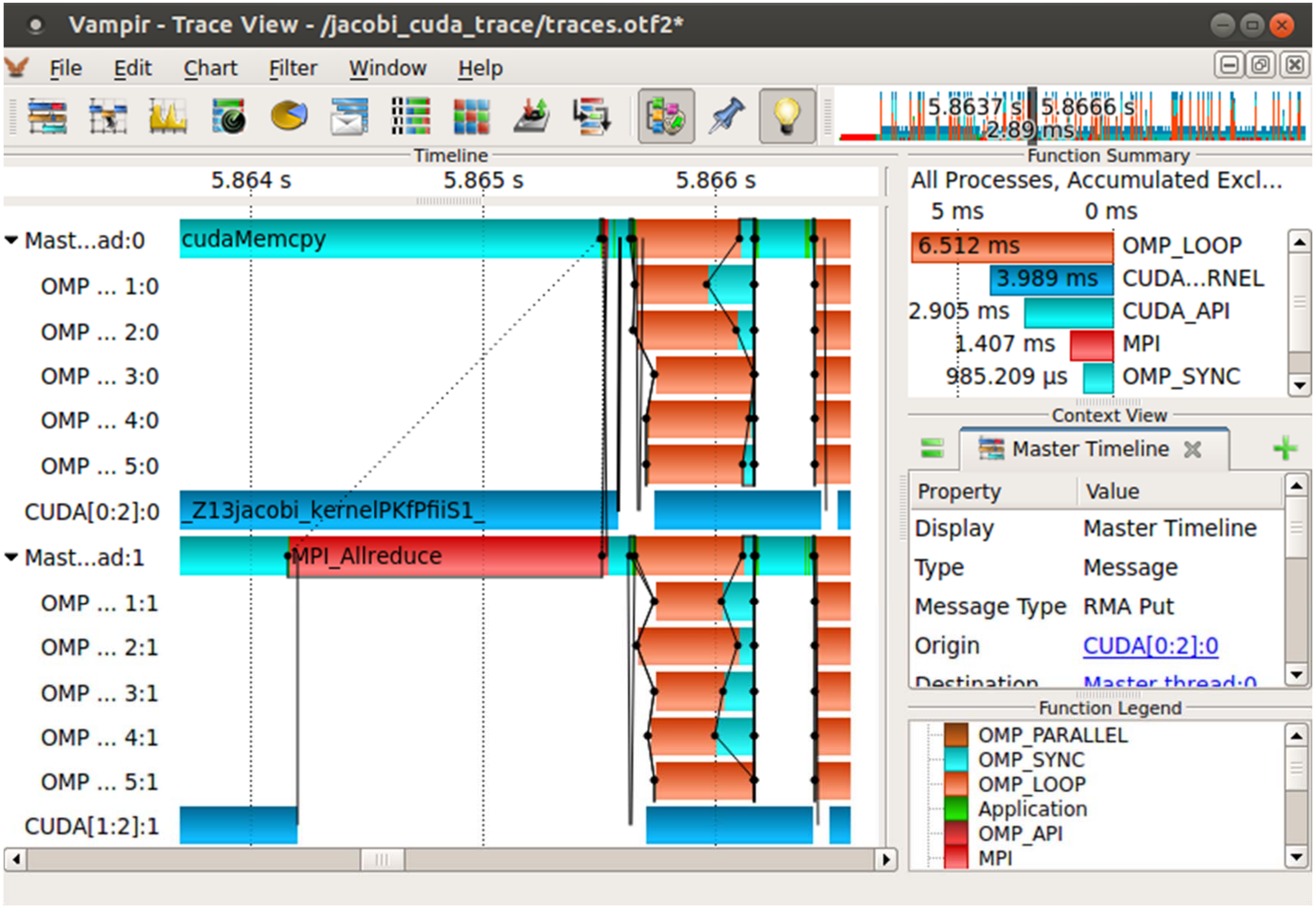
☞ Very small example => no filtering

Measurement (Tracing)

```
% export OMP_NUM_THREADS=6
% export SCOREP_CUDA_ENABLE=yes
% export SCOREP_CUDA_BUFFER=500000
% export SCOREP_EXPERIMENT_DIRECTORY=jacobi_cuda_trace
% export SCOREP_ENABLE_PROFILING=false
% export SCOREP_ENABLE_TRACING=true

% mpirun -n 2 ./jacobi_mpi+cuda 4096 4096 0.15

Jacobi relaxation Calculation: 4096 x 4096 mesh with
 2 processes and 6 threads + one Tesla T10 Processor for each process.
 307 of 2049 local rows are calculated on the CPU to balance the load
 between the CPU and the GPU.
    0, 0.113429
  … … … … … …
  900, 0.000101
 total: 12.875220 s
```

Vampir Analysis

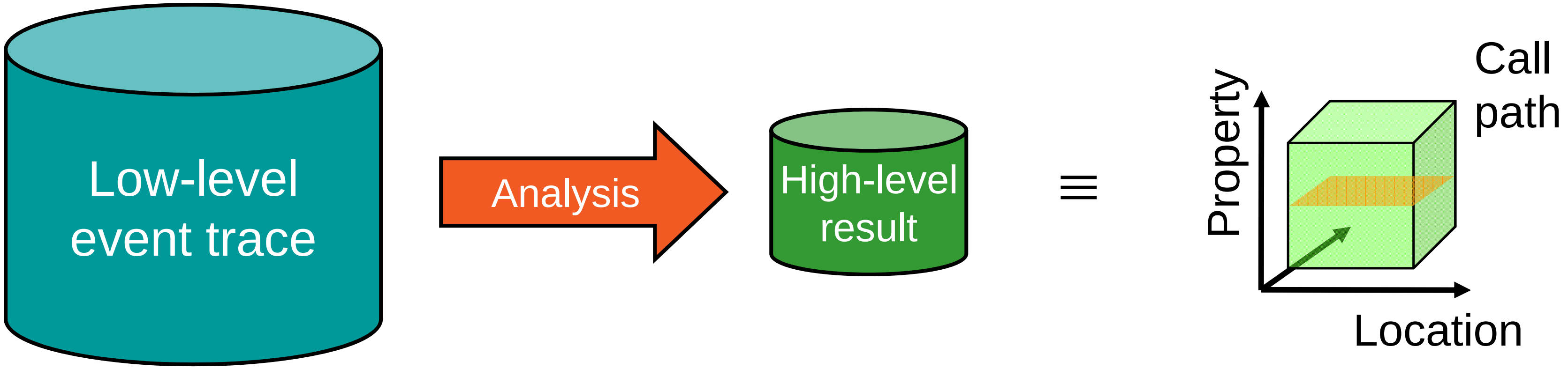
```
% vampir jacobi_cuda_trace/traces.otf2
```



Automatic trace analysis with Scalasca

Markus Geimer, Brian Wylie
Jülich Supercomputing Centre

- Idea
  - Automatic search for patterns of inefficient behavior
  - Classification of behavior & quantification of significance



  - Guaranteed to cover the entire event trace
  - Quicker than manual/visual trace analysis
  - Parallel replay analysis exploits available memory & processors to deliver scalability

The Scalasca project: Overview

- Project started in 2006
  - Initial funding by Helmholtz Initiative & Networking Fund
  - Many follow-up projects
- Follow-up to pioneering KOJAK project (started 1998)
  - Automatic pattern-based trace analysis
- Now joint development of
  - Jülich Supercomputing Centre
  - German Research School for Simulation Sciences

JÜLICH FORSCHUNGSZENTRUM

German Research School for Simulation Sciences

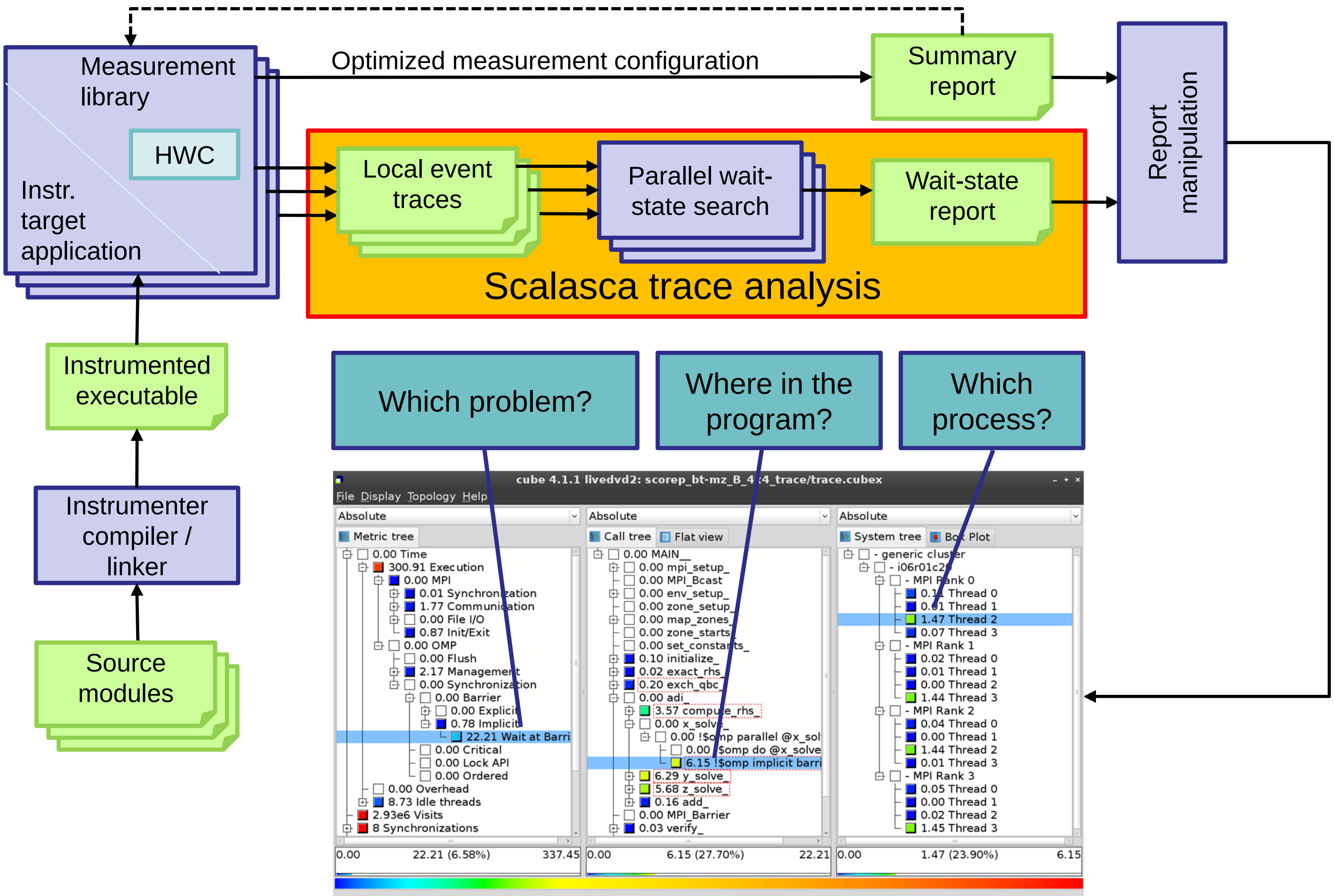
The Scalasca project: Objective

- Development of a **scalable** performance analysis toolset for most popular parallel programming paradigms
- Specifically targeting **large-scale** parallel applications
  - such as those running on IBM BlueGene or Cray XT systems with one million or more processes/threads
- Latest release:
  - Scalasca v2.0 with Score-P support (August 2013)

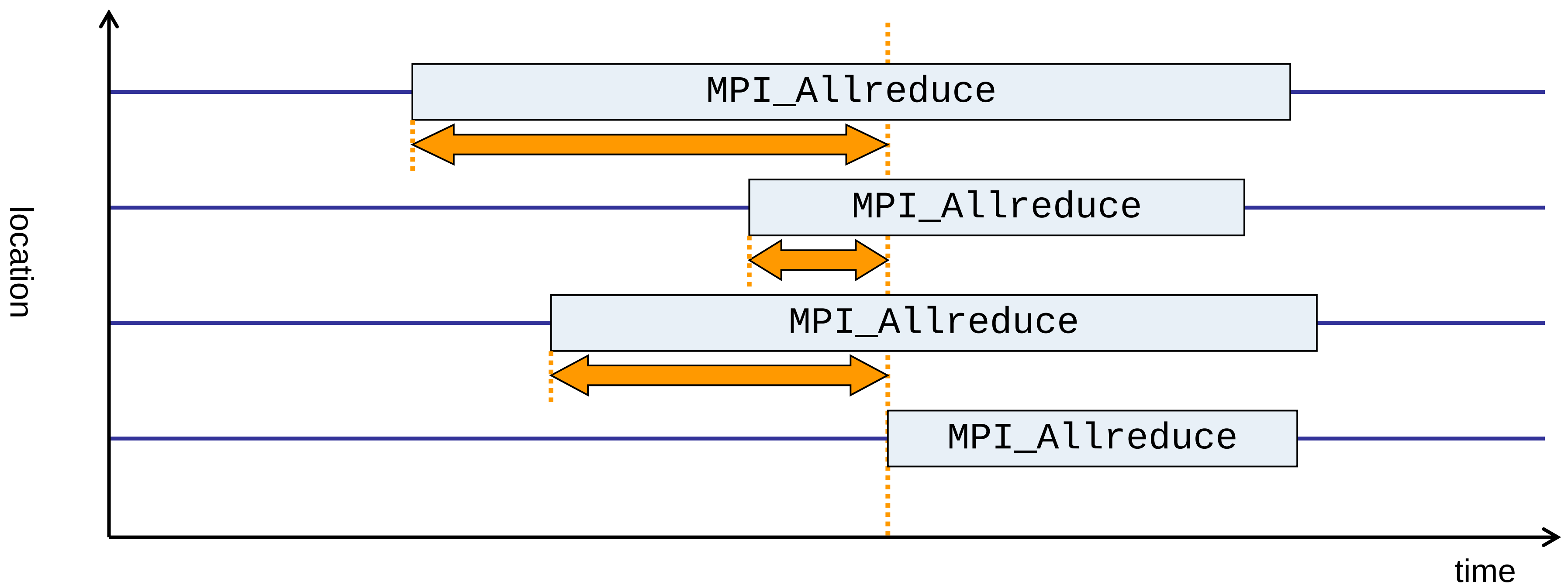
Scalasca 2.0 features

- Open source, New BSD license
- Fairly portable
  - IBM Blue Gene, IBM SP & blade clusters, Cray XT, SGI Altix, Solaris & Linux clusters, ...
- Uses Score-P instrumenter & measurement libraries
  - Scalasca 2.0 core package focuses on trace-based analyses
  - Supports common data formats
    - Reads event traces in OTF2 format
    - Writes analysis reports in CUBE4 format
- Current limitations:
  - No support for nested OpenMP parallelism and tasking
  - Unable to handle OTF2 traces containing CUDA events

Scalasca workflow

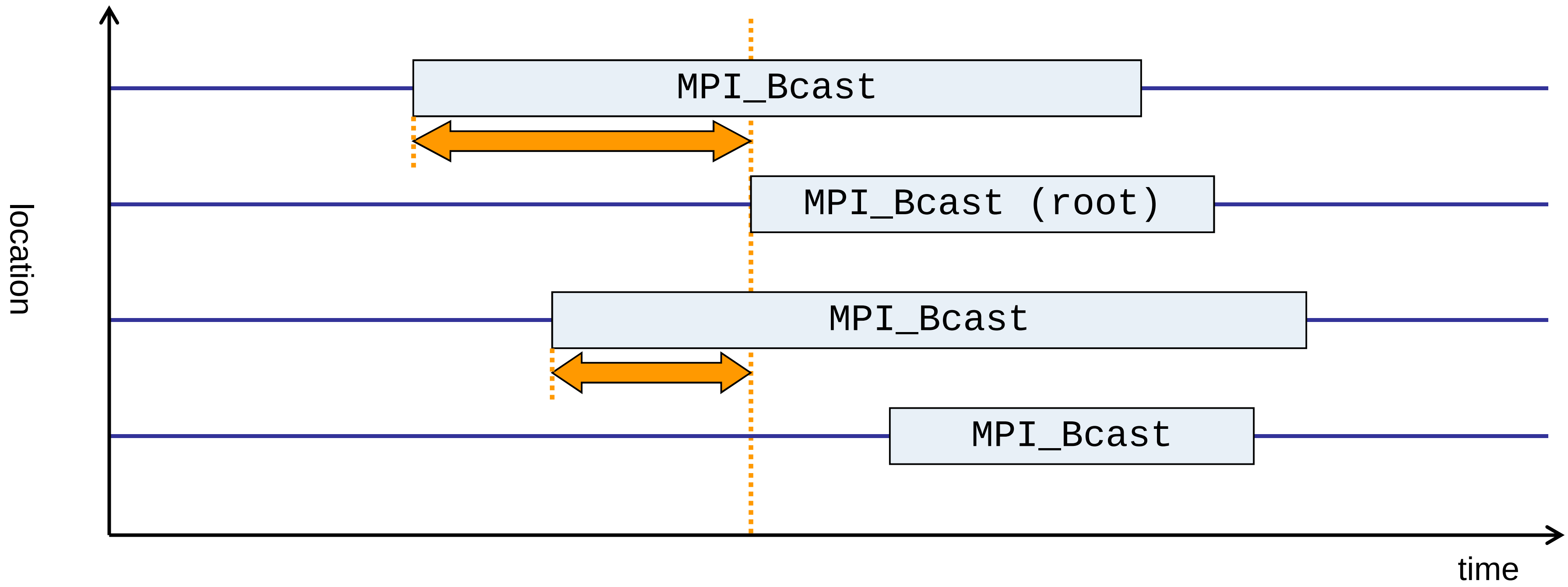


Example: Wait at NxN



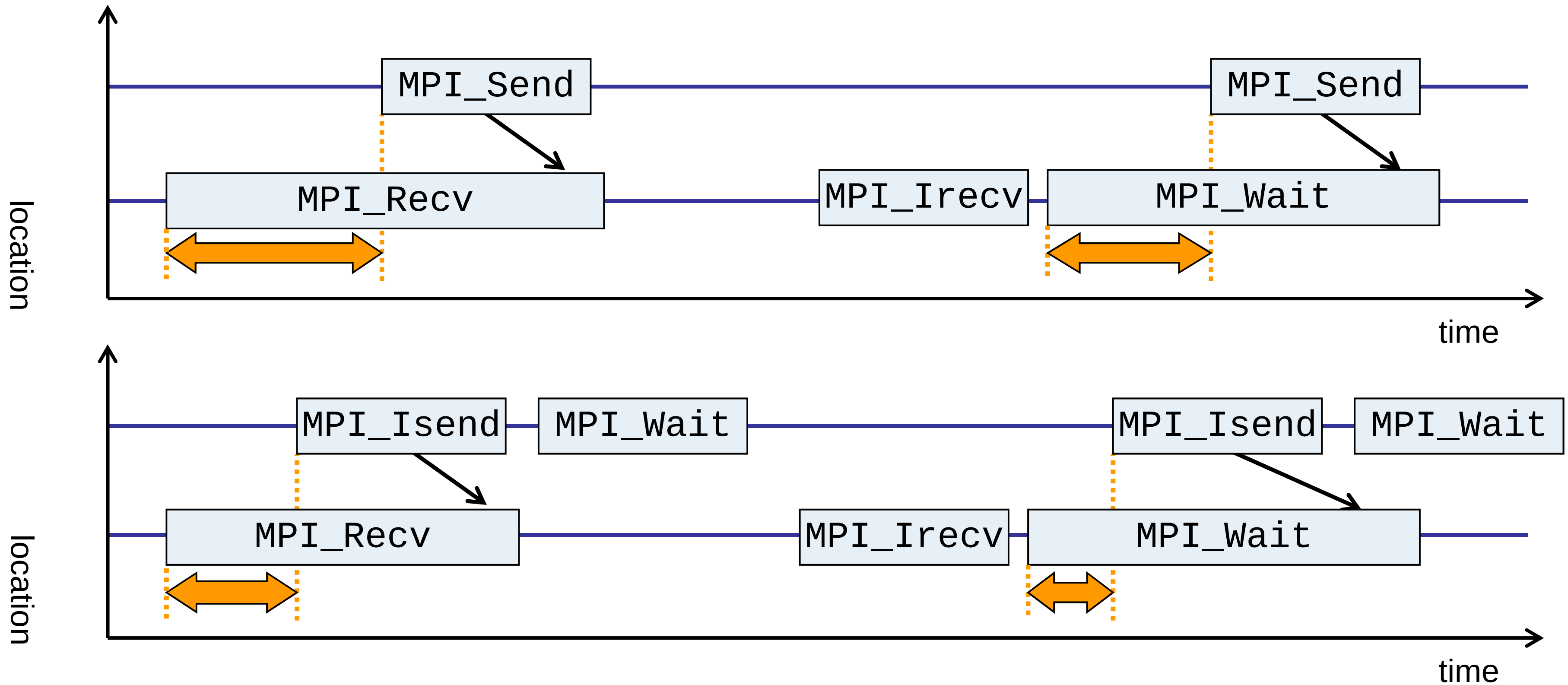
- Time spent waiting in front of synchronizing collective operation until the last process reaches the operation
- Applies to: MPI_Allgather, MPI_Allgatherv, MPI_Alltoall, MPI_Reduce_scatter, MPI_Reduce_scatter_block, MPI_Allreduce

Example: Late Broadcast



- Waiting times if the destination processes of a collective 1-to-N operation enter the operation earlier than the source process (root)
- Applies to: MPI_Bcast, MPI_Scatter, MPI_Scatterv

Example: Late Sender



- Waiting time caused by a blocking receive operation posted earlier than the corresponding send
- Applies to blocking as well as non-blocking communication

Hands-on:
NPB-MZ-MPI / BT

Scalasca command

- One command for (almost) everything…

```
% scalasca
Scalasca 2.0
Toolset for scalable performance analysis of large-scale applications
usage: scalasca [-v][-n][c] {action}
    1. prepare application objects and executable for measurement:
       scalasca –instrument <compile-or-link-command> # skin (using scorep)
    2. run application under control of measurement system:
       scalasca –analyze <application-launch-command> # scan
    3. interactively explore measurement analysis report:
       scalasca –examine <experiment-archive|report>  # square

   -v, --verbose      enable verbose commentary
   -n, --dry-run      show actions without taking them
   -c, --show-config  show configuration and exit
```

  – The '`scalasca -instrument`' command is deprecated and only provided for backwards compatibility with Scalasca 1.x.
  – Recommended: use Score-P instrumenter directly

Scalasca compatibility command: *skin*

- Scalasca application instrumenter

```
% skin
Scalasca 2.0: application instrumenter using scorep
usage: skin [-v] [–comp] [-pdt] [-pomp] [-user] <compile-or-link-cmd>
   -comp={all|none|...}: routines to be instrumented by compiler
          (... custom instrumentation specification for compiler)
   -pdt:  process source files with PDT instrumenter
   -pomp: process source files for POMP directives
   -user: enable EPIK user instrumentation API macros in source code
   -v:    enable verbose commentary when instrumenting

   --*:   options to pass to Score-P instrumenter
```

  – Provides compatibility with Scalasca 1.x
  – Recommended: use Score-P instrumenter directly

Scalasca convenience command: *scan*

- Scalasca measurement collection & analysis nexus

```
% scan
Scalasca 2.0: measurement collection & analysis nexus
usage: scan {options} [launchcmd [launchargs]] target [targetargs]
      where {options} may include:
  -h    Help: show this brief usage message and exit.
  -v    Verbose: increase verbosity.
  -n    Preview: show command(s) to be launched but don't execute.
  -q    Quiescent: execution with neither summarization nor tracing.
  -s    Summary: enable runtime summarization. [Default]
  -t    Tracing: enable trace collection and analysis.
  -a    Analyze: skip measurement to (re-)analyze an existing trace.
  -e exptdir   : Experiment archive to generate and/or analyze.
                 (overrides default experiment archive title)
  -f filtfile  : File specifying measurement filter.
  -l lockfile  : File that blocks start of measurement.
```

Scalasca convenience command: *square*

- Scalasca analysis report explorer

```
% square
Scalasca 2.0: analysis report explorer
usage: square [-v] [-s] [-f filtfile] [-F] <experiment archive
                                            | cube file>
   -F          : Force remapping of already existing reports
   -f filtfile : Use specified filter file when doing scoring
   -s          : Skip display and output textual score report
   -v          : Enable verbose mode
```

Automatic measurement configuration

- **scan** configures Score-P measurement by setting some environment variables automatically
  - e.g., experiment title, profiling/tracing mode, filter file, …
  - Precedence order:
    - Command-line arguments
    - Environment variables already set
    - Automatically determined values
- Also, **scan** includes consistency checks and prevents corrupting existing experiment directories
- For tracing experiments, after trace collection completes then automatic parallel trace analysis is initiated
  - uses identical launch configuration to that used for measurement (i.e., the same allocated compute resources)

BT-MZ summary measurement

- Run the application using the Scalasca measurement collection & analysis nexus prefixed to launch command

```
% export SCOREP_EXPERIMENT_DIRECTORY=scorep_bt-mz_W_4x4_sum
% OMP_NUM_THREADS=4 scan mpiexec –np 4 ./bt-mz_W.4
S=C=A=N: Scalasca 2.0 runtime summarization
S=C=A=N: ./scorep_bt-mz_W_4x4_sum experiment archive
S=C=A=N: Thu Sep 13 18:05:17 2012: Collect start
mpiexec –np 4 ./bt-mz_W.4

 NAS Parallel Benchmarks (NPB3.3-MZ-MPI) - BT-MZ MPI+OpenMP Benchmark

 Number of zones:   8 x   8
 Iterations: 200    dt:   0.000300
 Number of active processes:     4

 [... More application output ...]

S=C=A=N: Thu Sep 13 18:05:39 2012: Collect done (status=0) 22s
S=C=A=N: ./scorep_bt-mz_W_4x4_sum complete.
```

- Creates experiment directory ./scorep_bt-mz_W_4x4_sum

BT-MZ summary analysis report examination

- Score summary analysis report

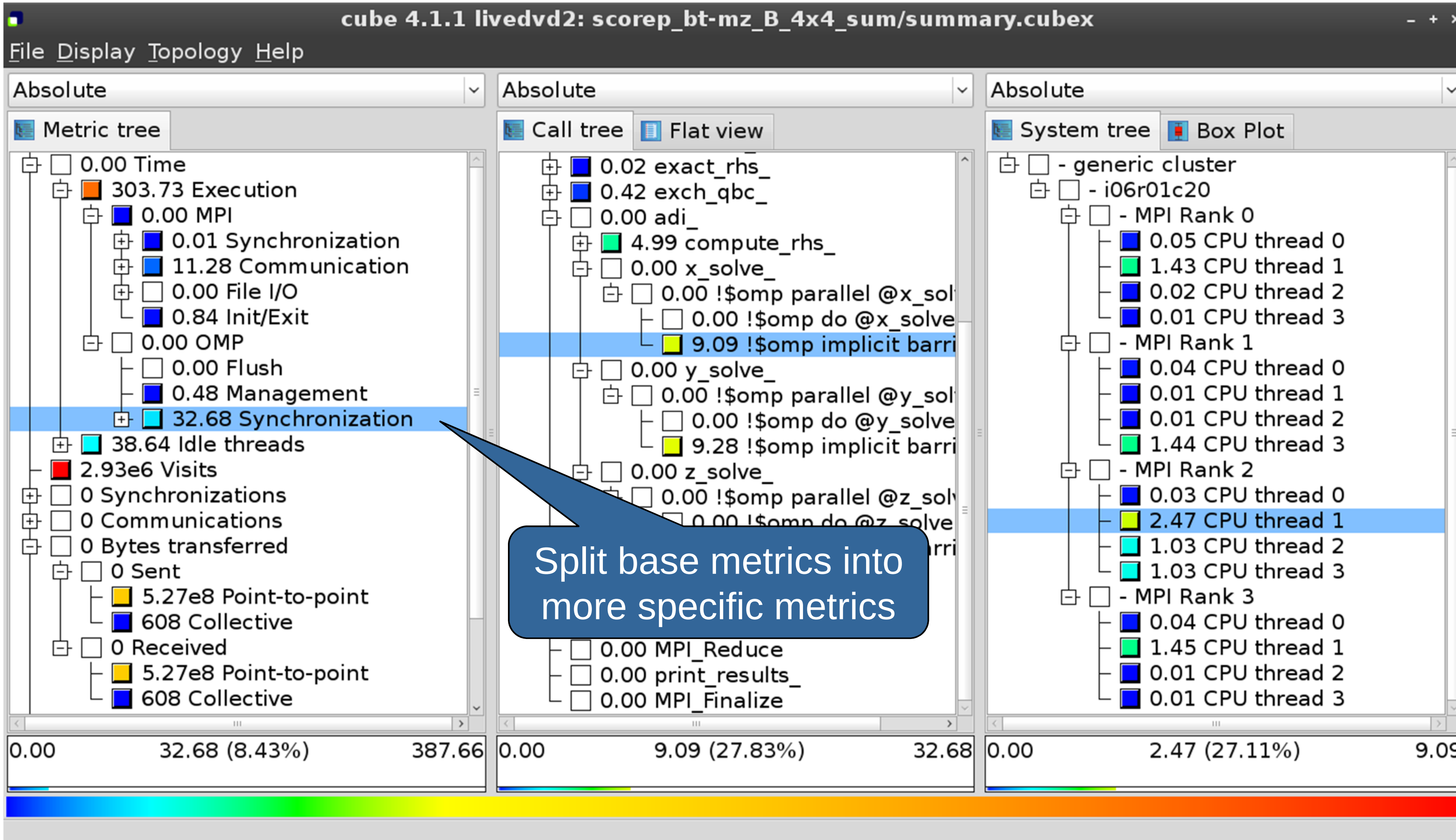
```
% square -s  scorep_bt-mz_W_4x4_sum
INFO: Post-processing runtime summarization result...
INFO: Score report written to ./scorep_bt-mz_W_4x4_sum/scorep.score
```

- Post-processing and interactive exploration with CUBE

```
% square  scorep_bt-mz_W_4x4_sum
INFO: Displaying ./scorep_bt-mz_W_4x4_sum/summary.cubex...

                  [GUI showing summary analysis report]
```

- The post-processing derives additional metrics and generates a structured metric hierarchy

Post-processed summary analysis report



0.0 Reference preparation for validation

1.0 Program instrumentation

1.1 Summary measurement collection

1.2 Summary analysis report examination

2.0 Summary experiment scoring

2.1 Summary measurement collection with filtering

2.2 Filtered summary analysis report examination

3.0 Event trace collection

3.1 Event trace examination & analysis

BT-MZ trace measurement collection...

- Re-run the application using Scalasca nexus with **"-t"** flag

```
% export SCOREP_EXPERIMENT_DIRECTORY=scorep_bt-mz_W_4x4_trace
% OMP_NUM_THREADS=4 scan -t mpiexec –np 4 ./bt-mz_W.4
S=C=A=N: Scalasca 2.0 trace collection and analysis
S=C=A=N: ./scorep_bt-mz_W_4x4_trace experiment archive
S=C=A=N: Thu Sep 13 18:05:39 2012: Collect start
mpiexec –np 4 ./bt-mz_B.4
 NAS Parallel Benchmarks (NPB3.3-MZ-MPI) - BT-MZ MPI+OpenMP Benchmark

 Number of zones:   8 x   8
 Iterations: 200    dt:   0.000300
 Number of active processes:     4

 [... More application output ...]

S=C=A=N: Thu Sep 13 18:05:58 2012: Collect done (status=0) 19s
 [... continued ...]
```

BT-MZ trace measurement ... analysis

- Continues with automatic (parallel) analysis of trace files

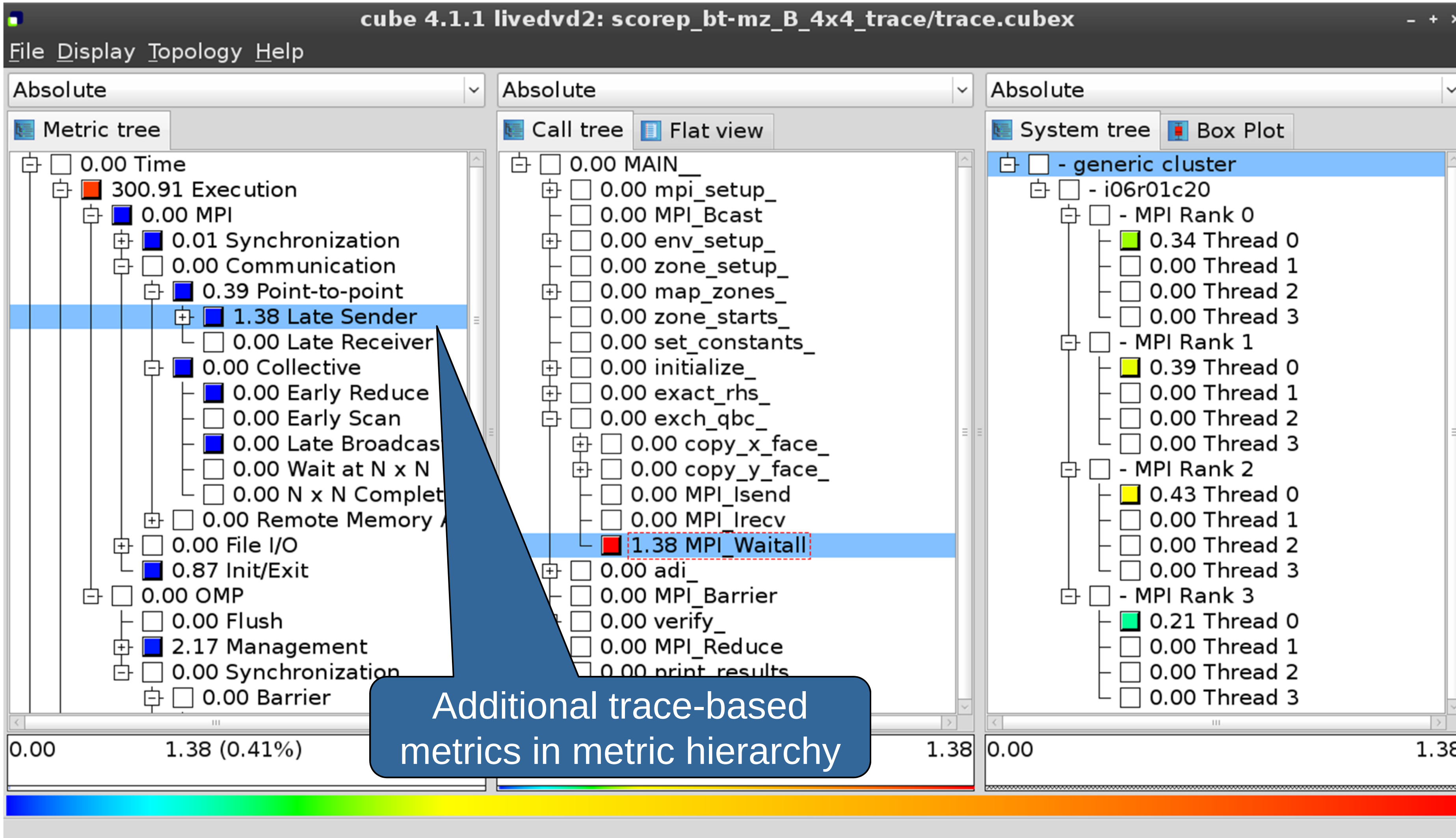
```
S=C=A=N: Thu Sep 13 18:05:58 2012: Analyze start
mpiexec –np 4 scout.hyb ./scorep_bt-mz_W_4x4_trace/traces.otf2
SCOUT   Copyright (c) 1998-2012 Forschungszentrum Juelich GmbH
        Copyright (c) 2009-2012 German Research School for Simulation
                                Sciences GmbH

Analyzing experiment archive ./scorep_bt-mz_W_4x4_trace/traces.otf2

Opening experiment archive ... done (0.002s).
Reading definition data    ... done (0.004s).
Reading event trace data   ... done (0.669s).
Preprocessing              ... done (0.975s).
Analyzing trace data       ... done (0.675s).
Writing analysis report    ... done (0.112s).

Max. memory usage         : 145.078MB

Total processing time     : 2.785s
S=C=A=N: Thu Sep 13 18:06:02 2012: Analyze done (status=0) 4s
```

BT-MZ trace analysis report exploration

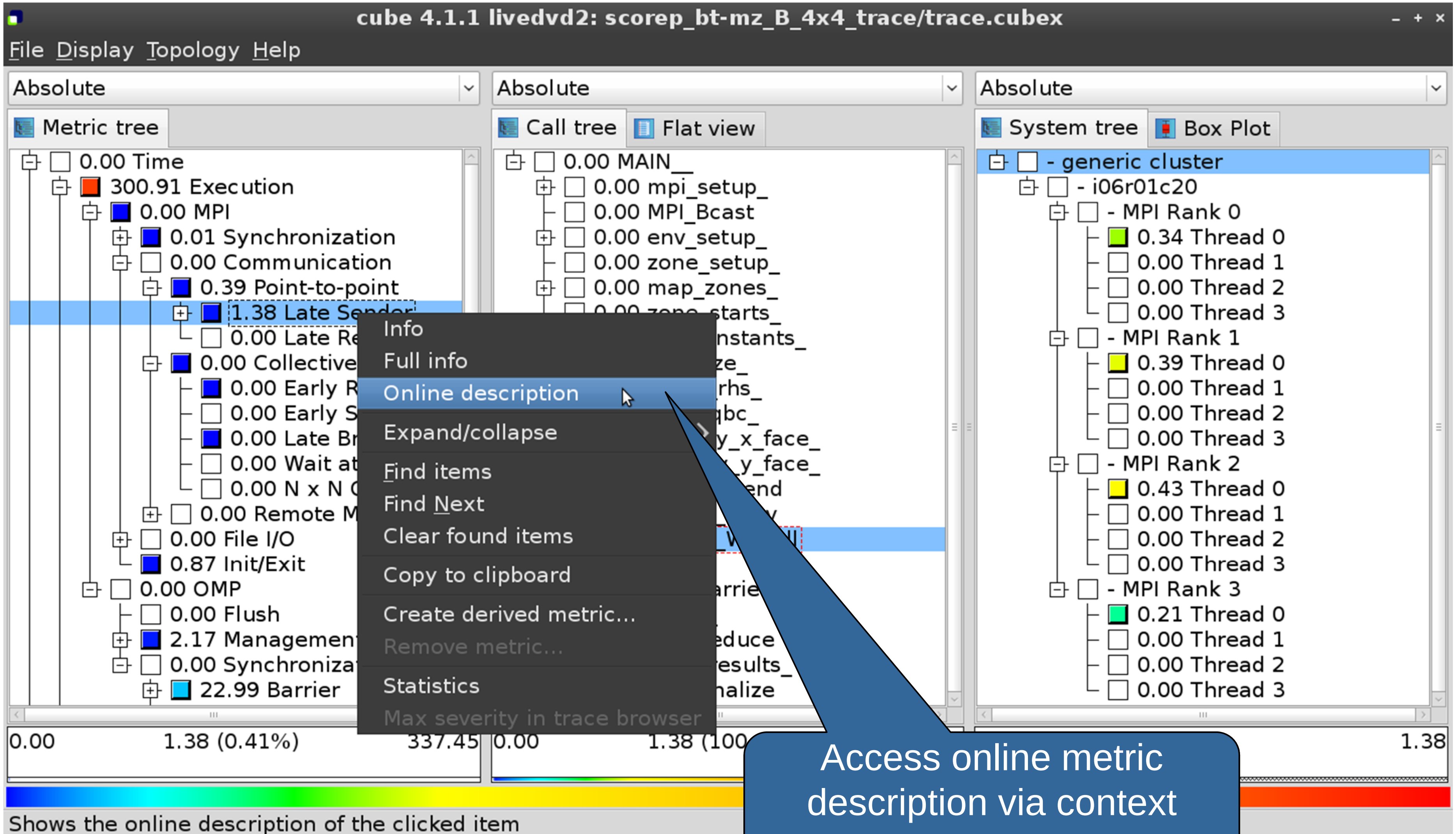
- Produces trace analysis report in experiment directory containing trace-based wait-state metrics

```
% square  scorep_bt-mz_W_4x4_trace
INFO: Post-processing runtime summarization result...
INFO: Post-processing trace analysis report...
INFO: Displaying ./scorep_bt-mz_W_4x4_trace/trace.cubex...

                  [GUI showing trace analysis report]
```

Post-processed trace analysis report



Online metric description



Access online metric description via context menu

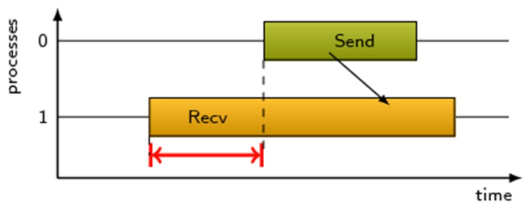
Online metric description

Performance properties

Late Sender Time

Description:

Refers to the time lost waiting caused by a blocking receive operation (e.g., `MPI_Recv` or `MPI_Wait`) that is posted earlier than the corresponding send operation.



If the receiving process is waiting for multiple messages to arrive (e.g., in an call to `MPI_Waitall`), the maximum waiting time is accounted, i.e., the waiting time due to the latest sender.

Unit:

Seconds

Diagnosis:

Try to replace `MPI_Recv` with a non-blocking receive `MPI_Irecv` that can be posted earlier, proceed concurrently with computation, and complete with a wait operation after the message is expected to have been sent. Try to post sends earlier, such that they are available when receivers need them. Note that outstanding messages (i.e., sent before the receiver is ready) will occupy internal message buffers, and that large numbers of posted receive buffers will also introduce message management overhead, therefore moderation is advisable.

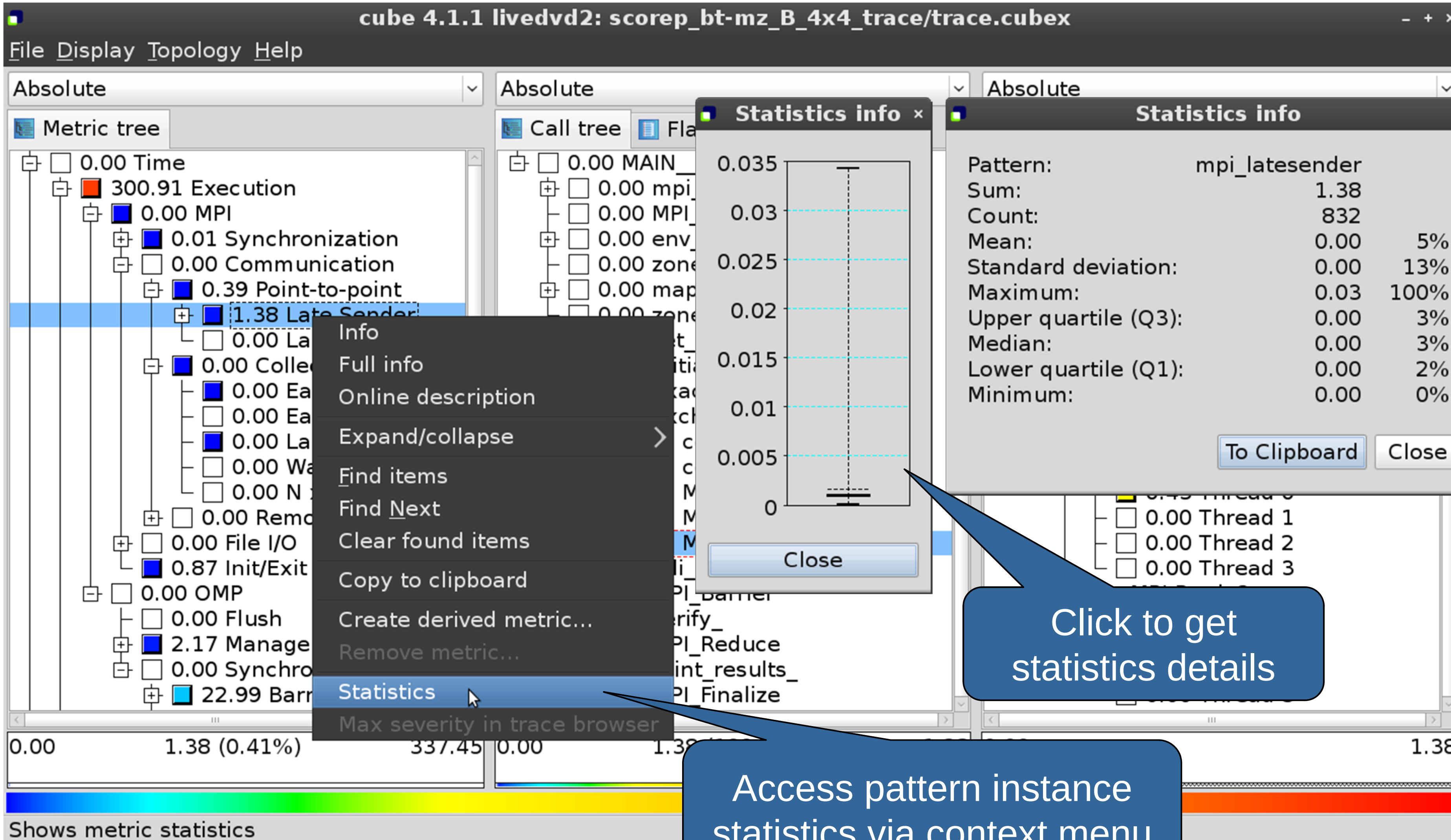
Parent:

[MPI Point-to-point Communication Time]

Children:

Close

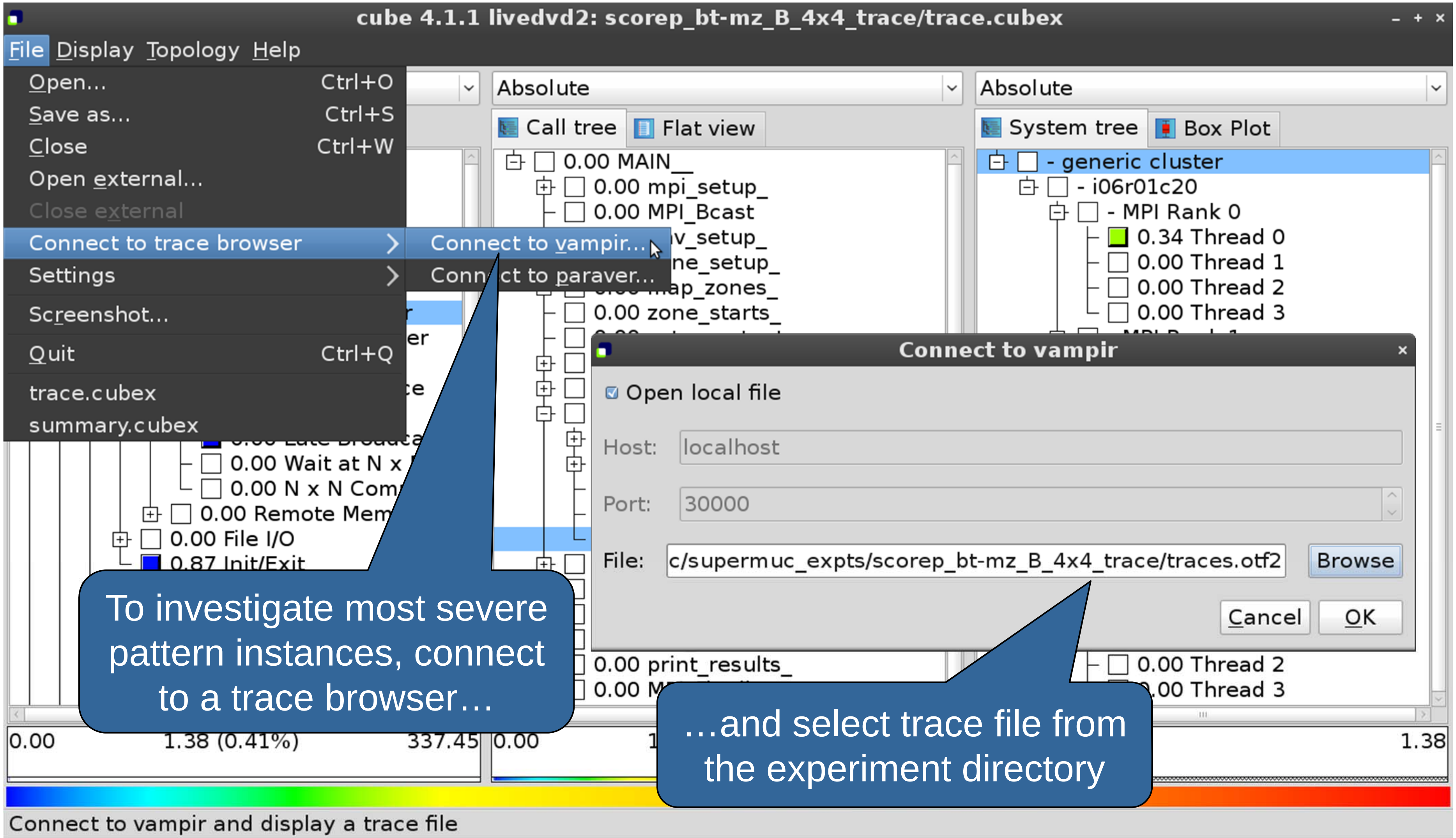
Pattern instance statistics



Click to get statistics details

Access pattern instance statistics via context menu

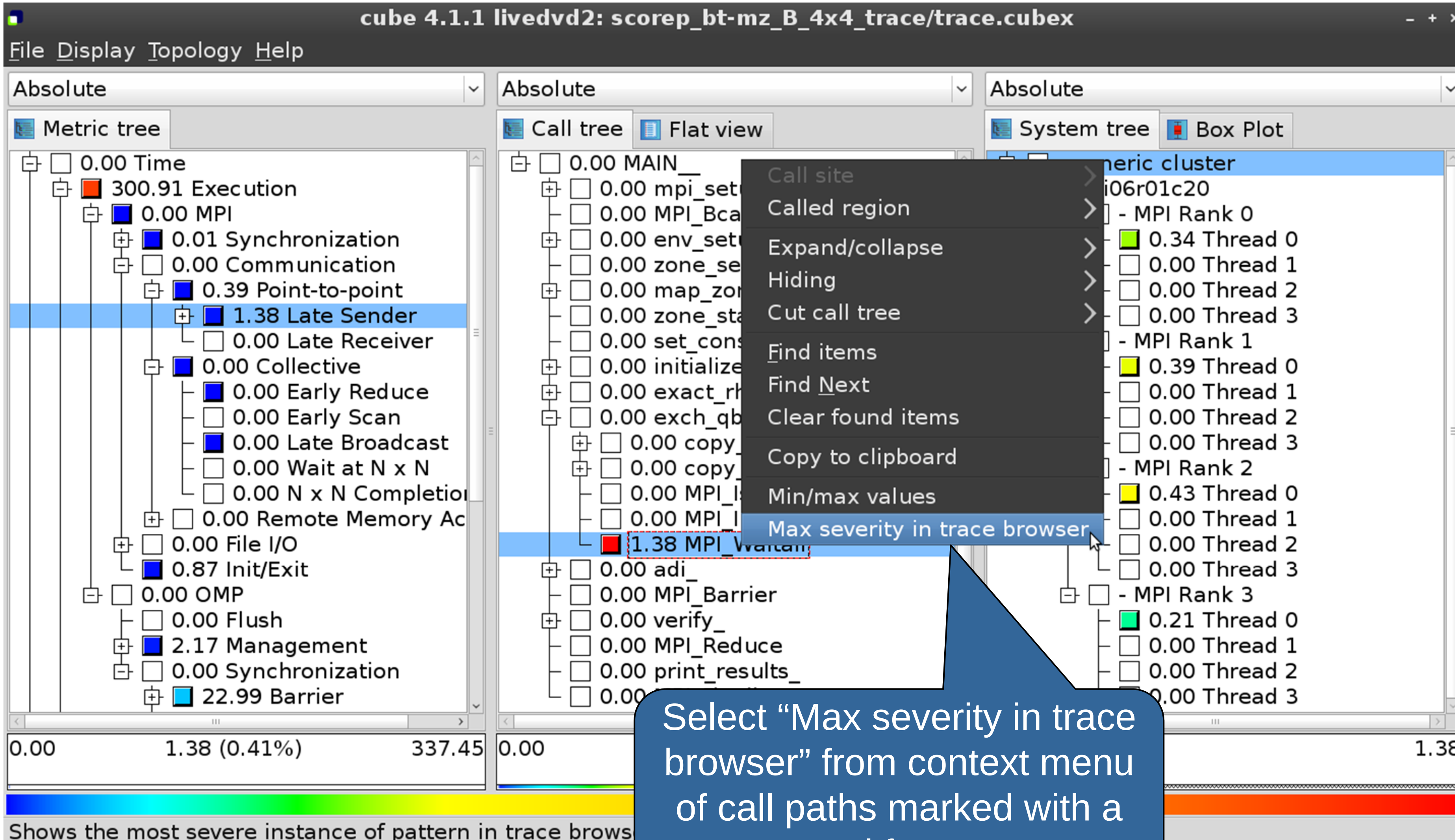
Connect to Vampir trace browser



To investigate most severe pattern instances, connect to a trace browser…

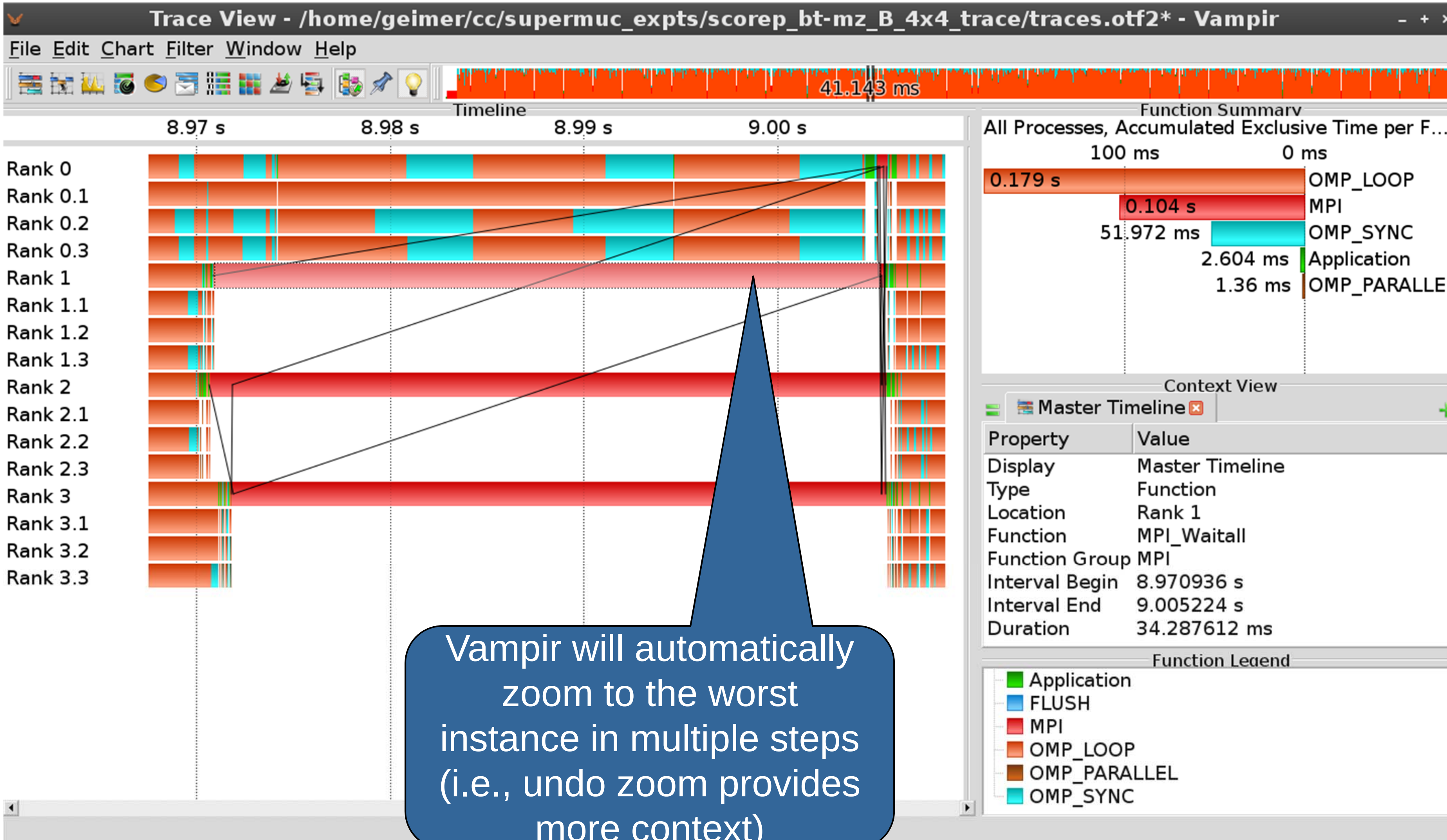
…and select trace file from the experiment directory

Show most severe pattern instances



Select "Max severity in trace browser" from context menu of call paths marked with a red frame

Investigate most severe instance in Vampir



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

- toolset for scalable performance measurement & analysis of MPI, OpenMP & hybrid parallel applications
- supporting most popular HPC computer systems
- available under New BSD open-source license
- sources, documentation & publications:
  - http://www.scalasca.org
  - mailto: scalasca@fz-juelich.de

Performance Analysis with Vampir

Bert Wesarg, Andreas Knüpfer
ZIH, Technische Universität Dresden

VI-HPS

Part I: Welcome to the Vampir Tool Suite

- Mission
- Event Trace Visualization
- Vampir & VampirServer
- The Vampir Displays

Part II: Vampir Hands On

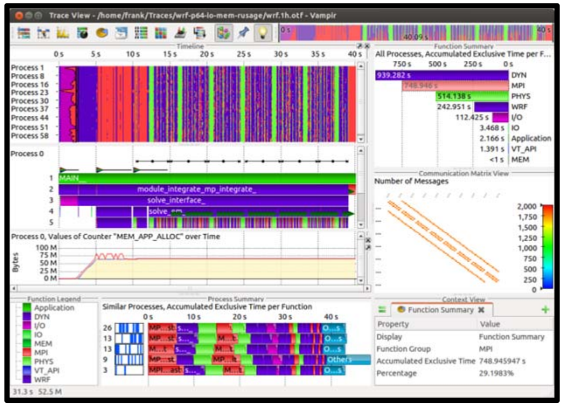
- Visualizing and analyzing NPB-MZ-MPI / BT

Part III: Summary and Conclusion

VAMPIR

Mission

- Visualization of dynamics of complex parallel processes
- Requires two components
  - Monitor/Collector (Score-P)
  - Charts/Browser (Vampir)



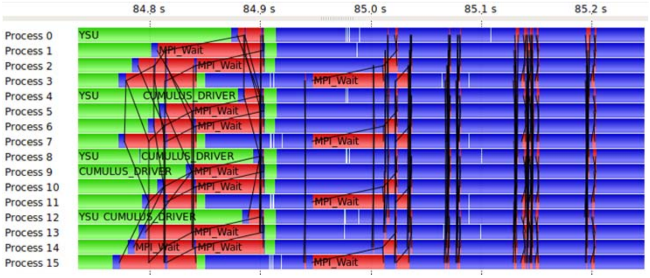
Typical questions that Vampir helps to answer:

- What happens in my application execution during a given time in a given process or thread?
- How do the communication patterns of my application execute on a real system?
- Are there any imbalances in computation, I/O or memory usage and how do they affect the parallel execution of my application?

- Alternative and supplement to automatic analysis
- Show dynamic run-time behavior graphically at any level of detail
- Provide statistics and performance metrics

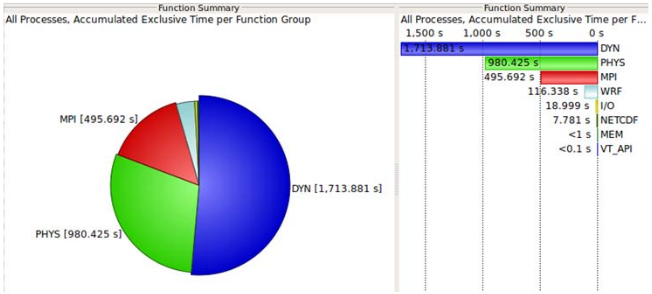
Timeline charts

– Show application activities and communication along a time axis



Summary charts

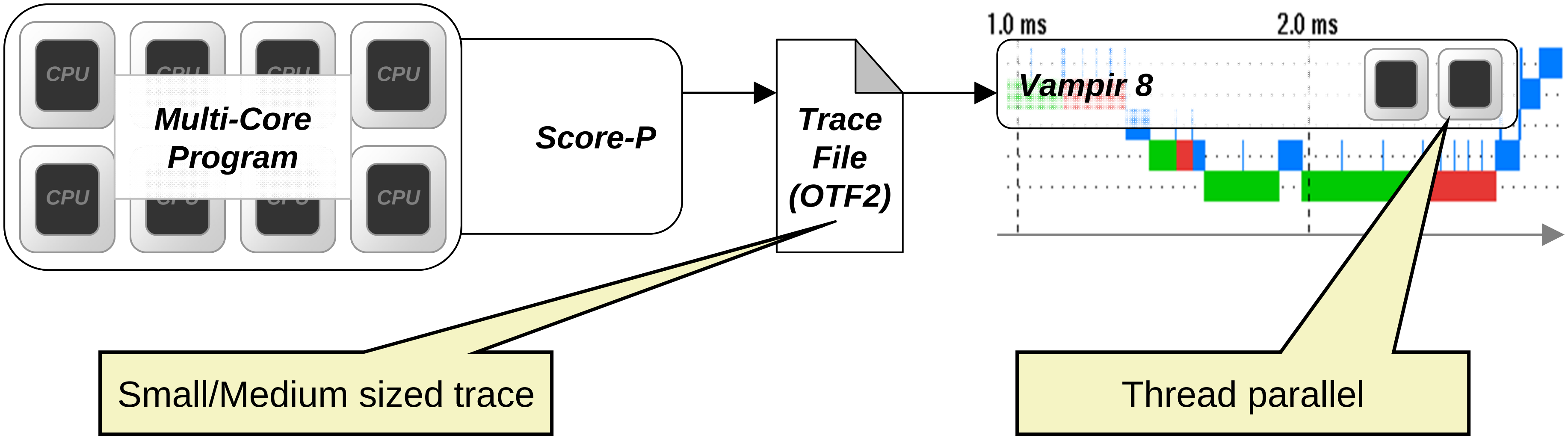
– Provide quantitative results for the currently selected time interval



Vampir – Visualization Modes (1)

- Directly on front end or local machine

```
% vampir
```

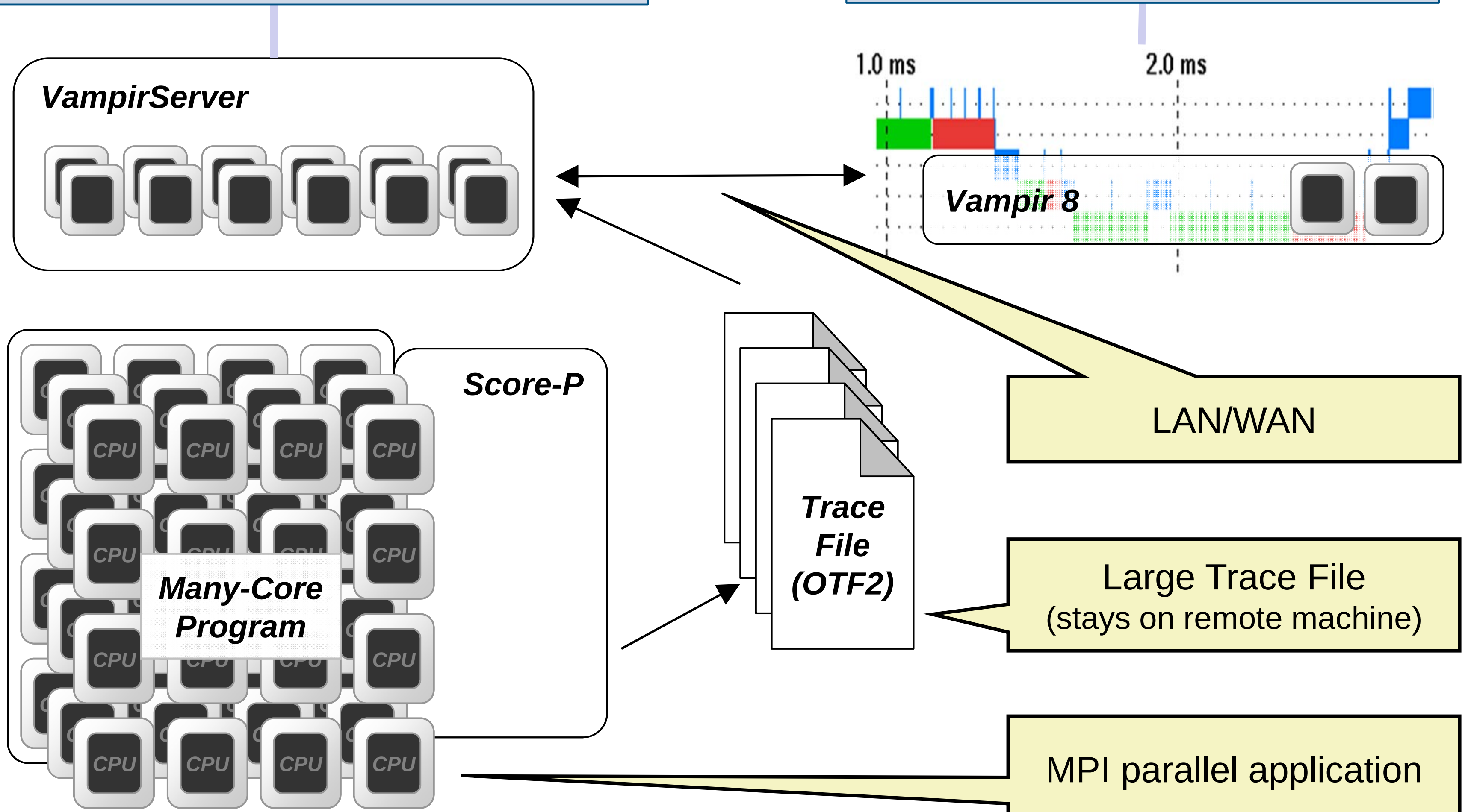


Vampir – Visualization Modes (2)

- On local machine with remote VampirServer

```
% vampirserver start –n 12
```

```
% vampir
```



Usage order of the Vampir Performance Analysis Toolset

1. Instrument your application with Score-P

2. Run your application with an appropriate test set

3. Analyze your trace file with Vampir
   - Small trace files can be analyzed on your local workstation
     1. Start your local Vampir
     2. Load trace file from your local disk
   - Large trace files should be stored on the HPC file system
     1. Start VampirServer on your HPC system
     2. Start your local Vampir
     3. Connect local Vampir with the VampirServer on the HPC system
     4. Load trace file from the HPC file system

The main displays of Vampir

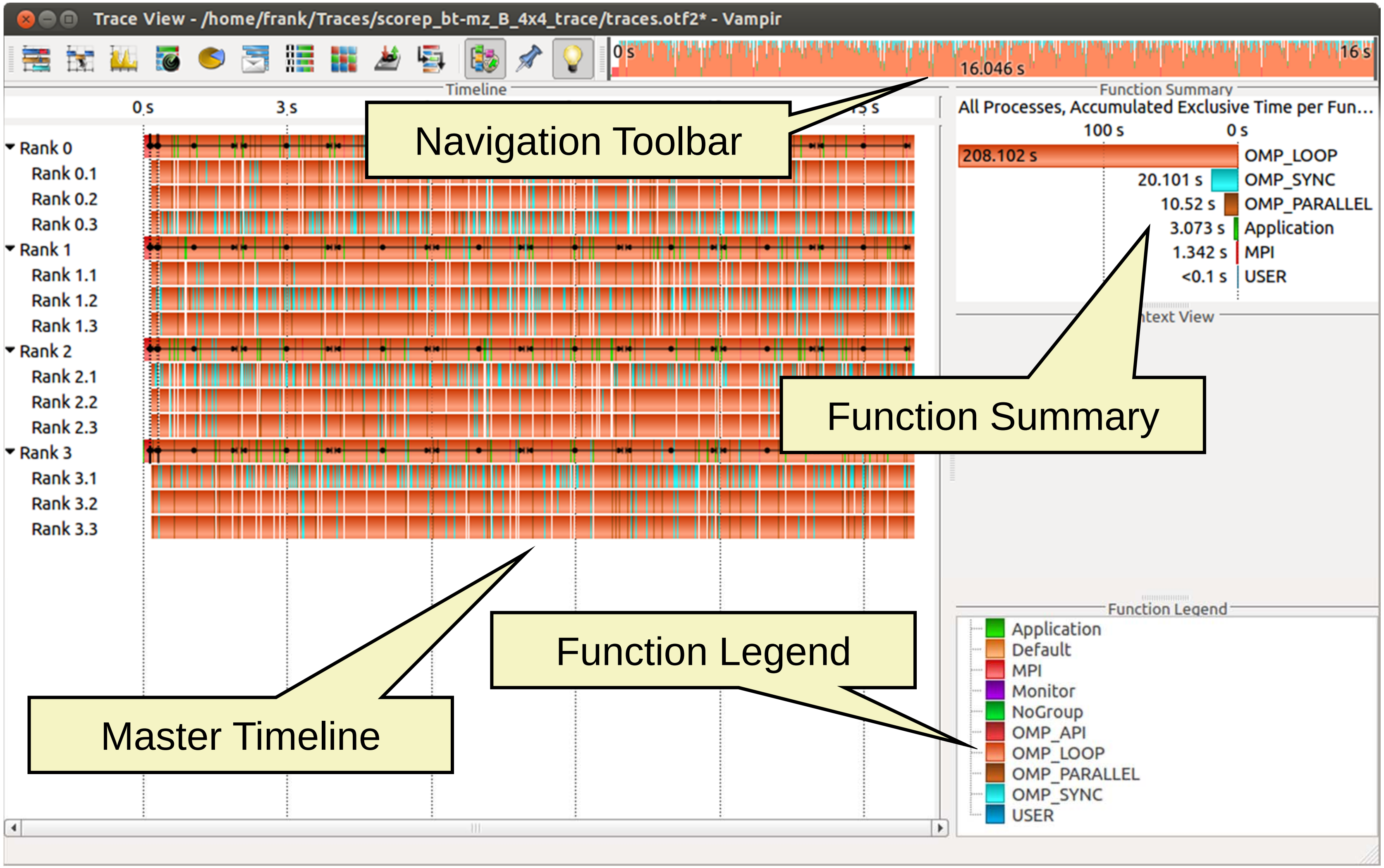
- **Timeline Charts:**
  - Master Timeline
  - Process Timeline
  - Counter Data Timeline
  - Performance Radar
- **Summary Charts:**
  - Function Summary
  - Message Summary
  - Process Summary
  - Communication Matrix View

Vampir hands-on

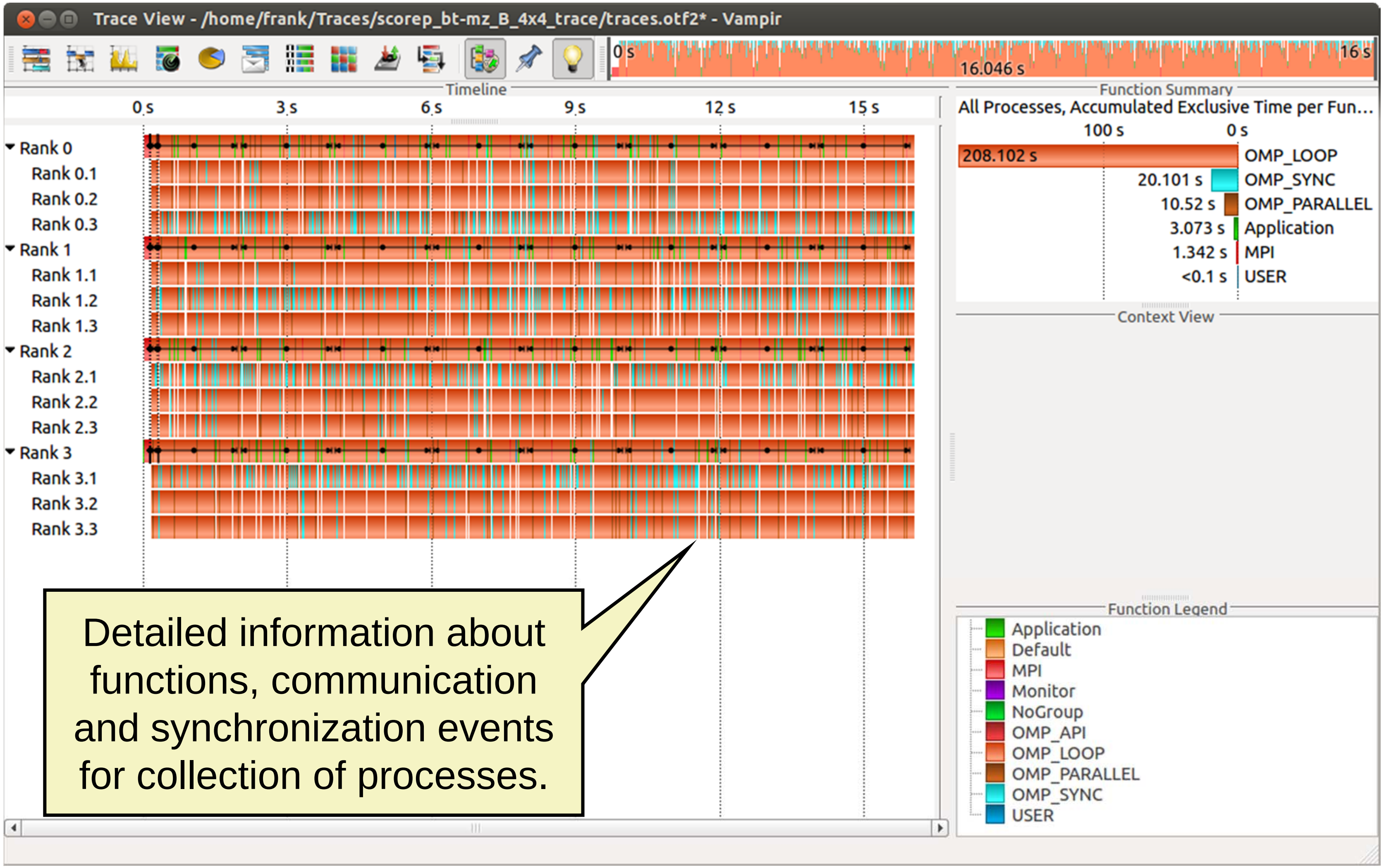
Visualizing and analyzing NPB-MZ-MPI / BT

Vampir: Visualization of the NPB-MZ-MPI / BT trace

```
% vampir scorep_bt-mz_B_4x4_trace
```



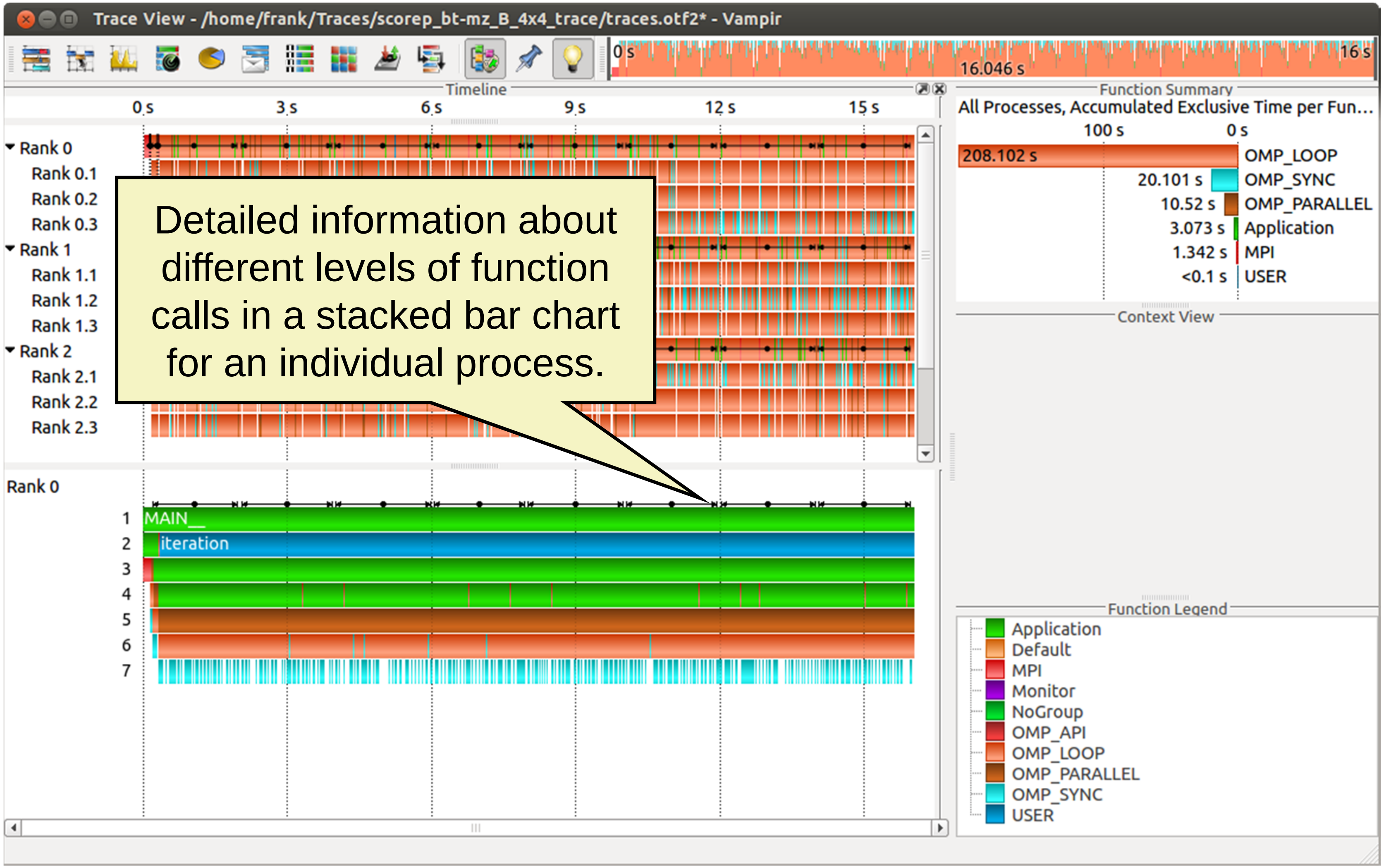
Master Timeline



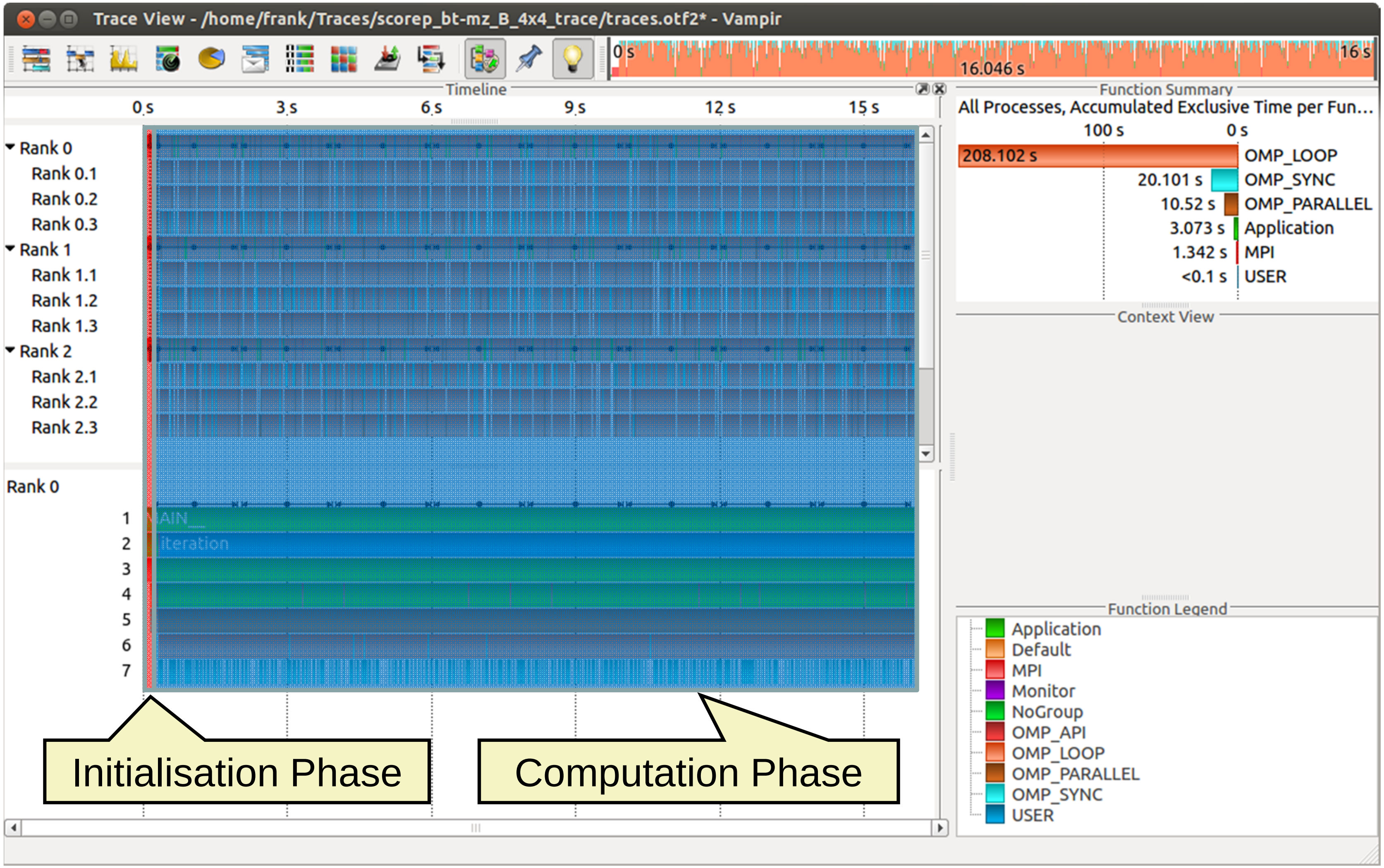
Detailed information about functions, communication and synchronization events for collection of processes.

Vampir: Visualization of the NPB-MZ-MPI / BT trace

Process Timeline

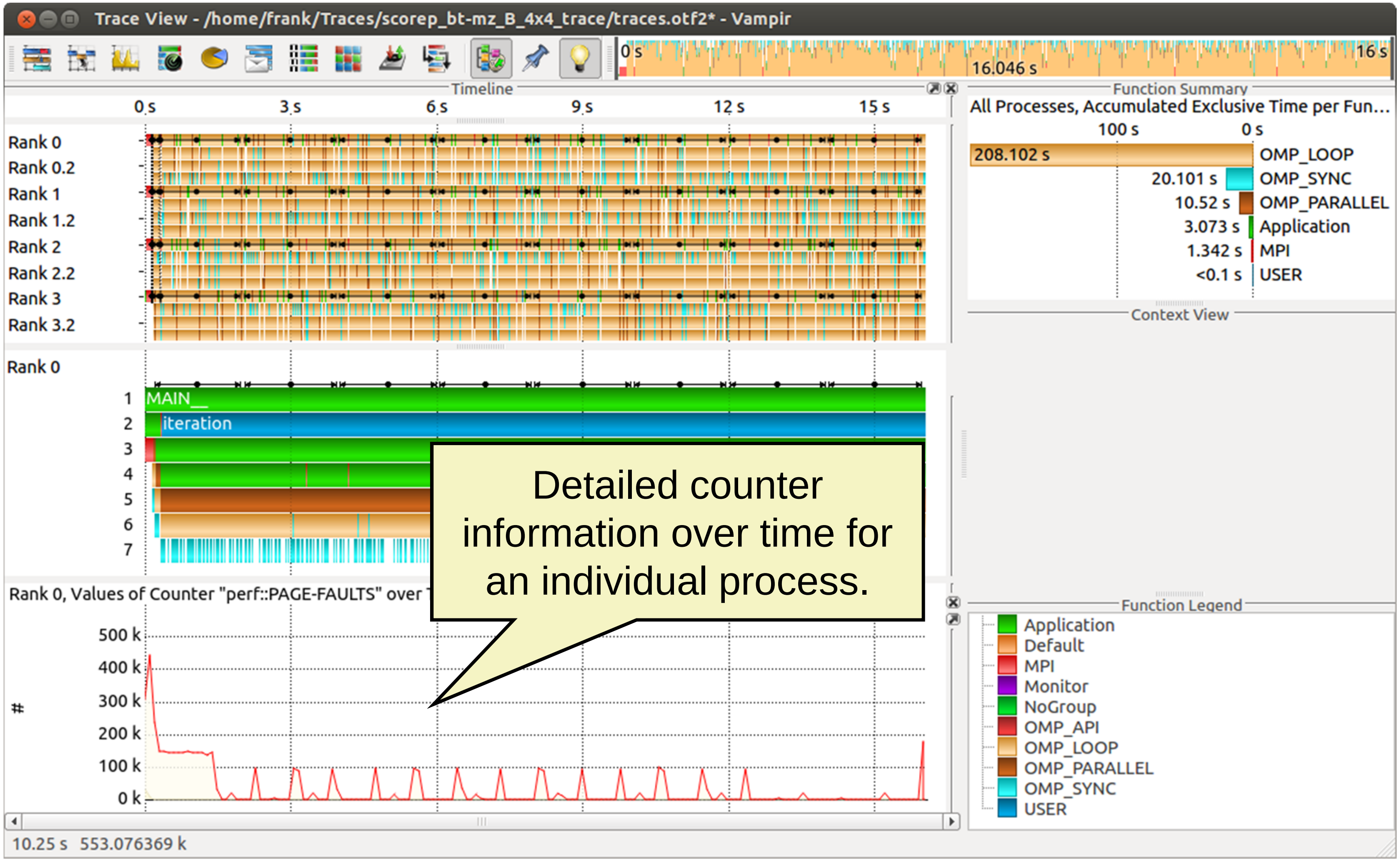


Typical program phases

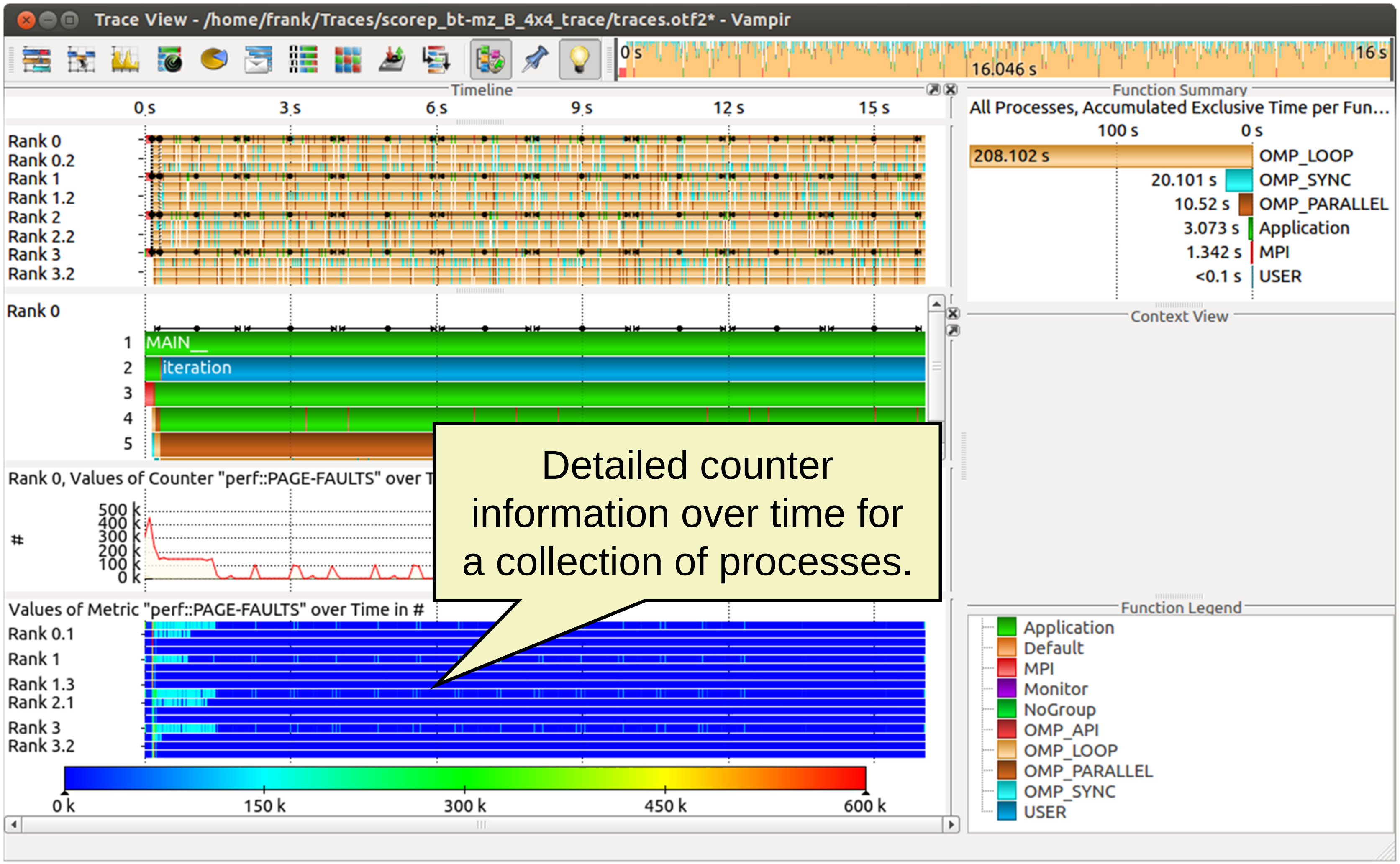


Vampir: Visualization of the NPB-MZ-MPI / BT trace

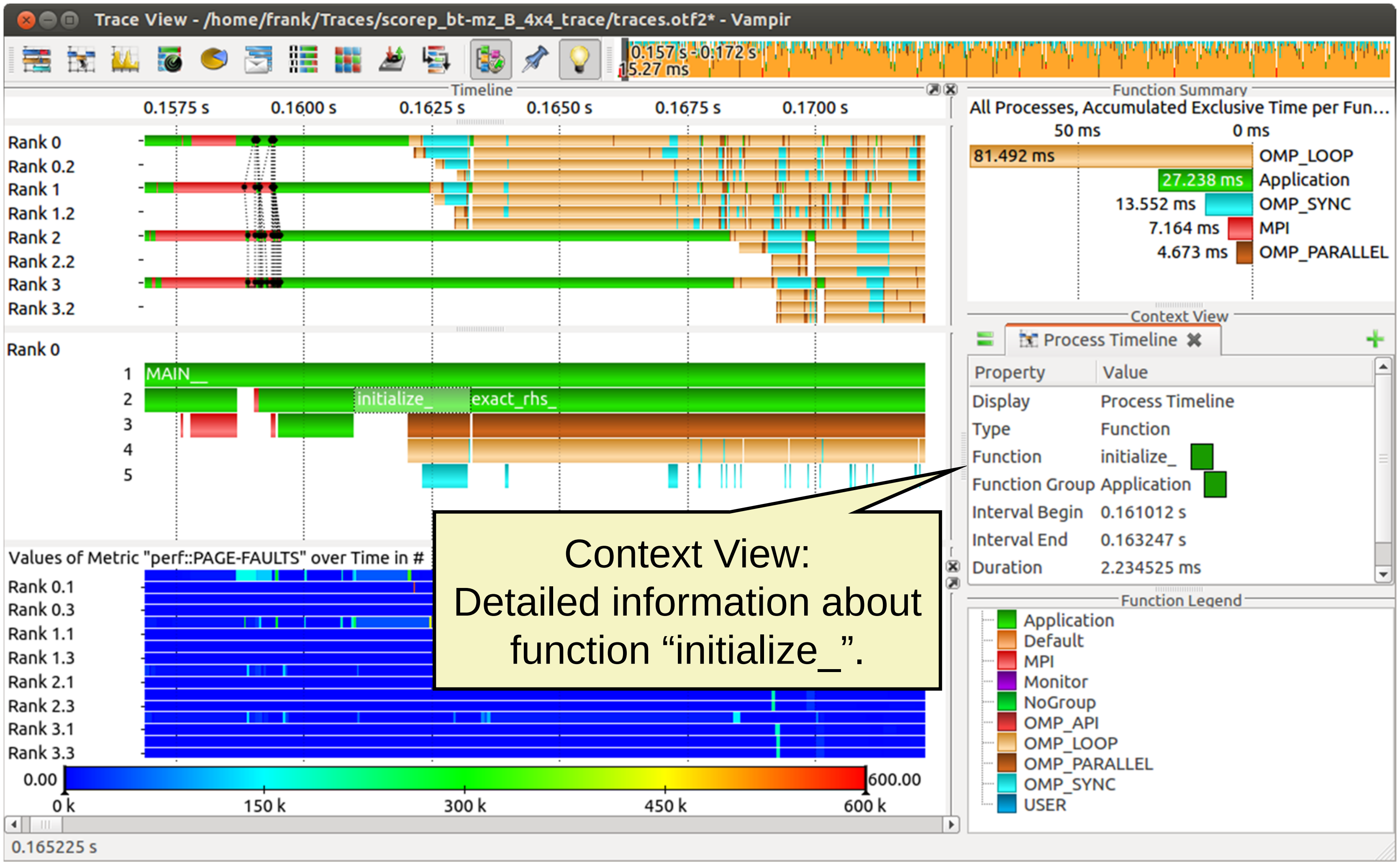
Counter Data Timeline



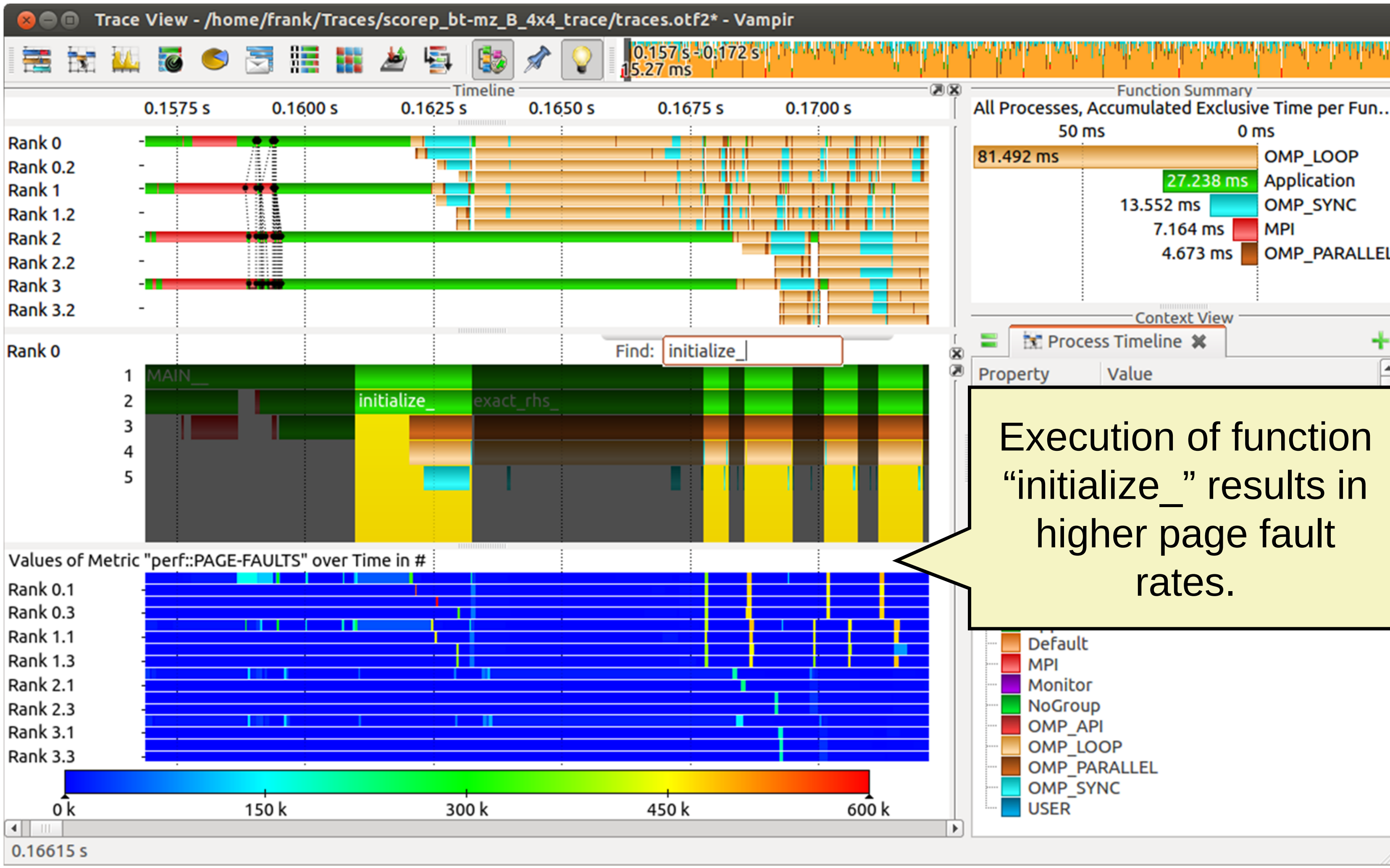
Performance Radar



Zoom in: Inititialisation Phase

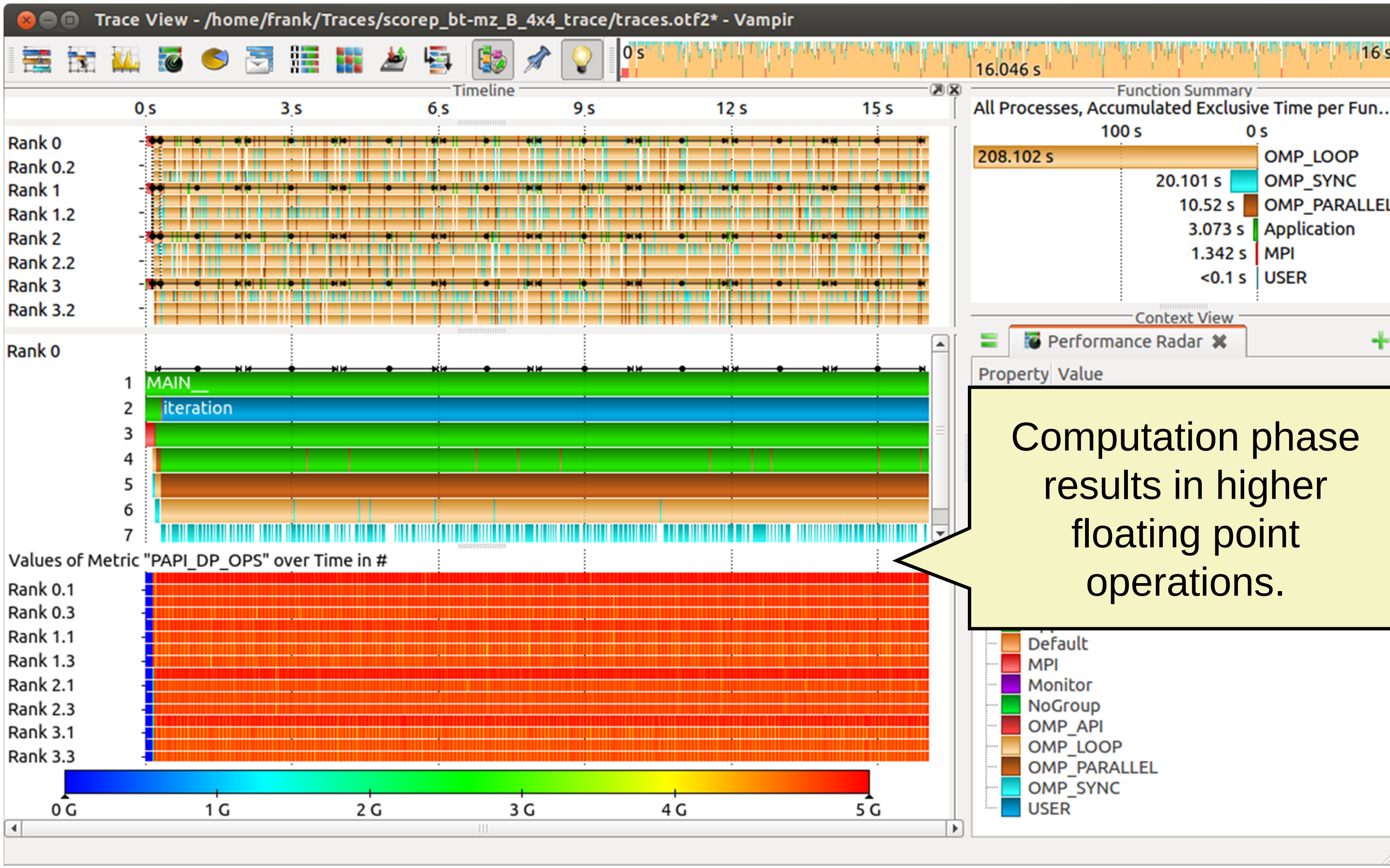


Feature: Find Function



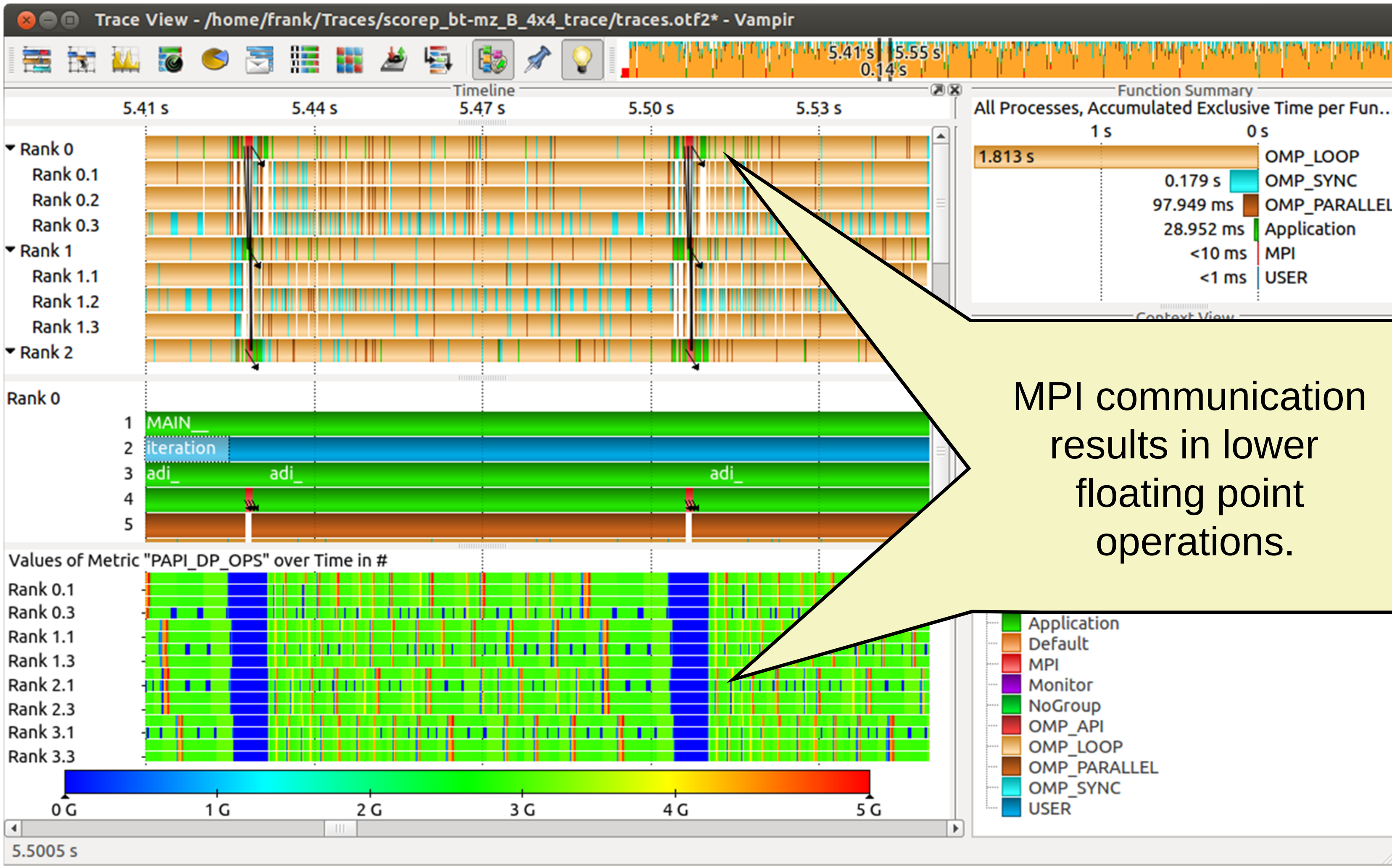
Execution of function “initialize_” results in higher page fault rates.

Computation Phase

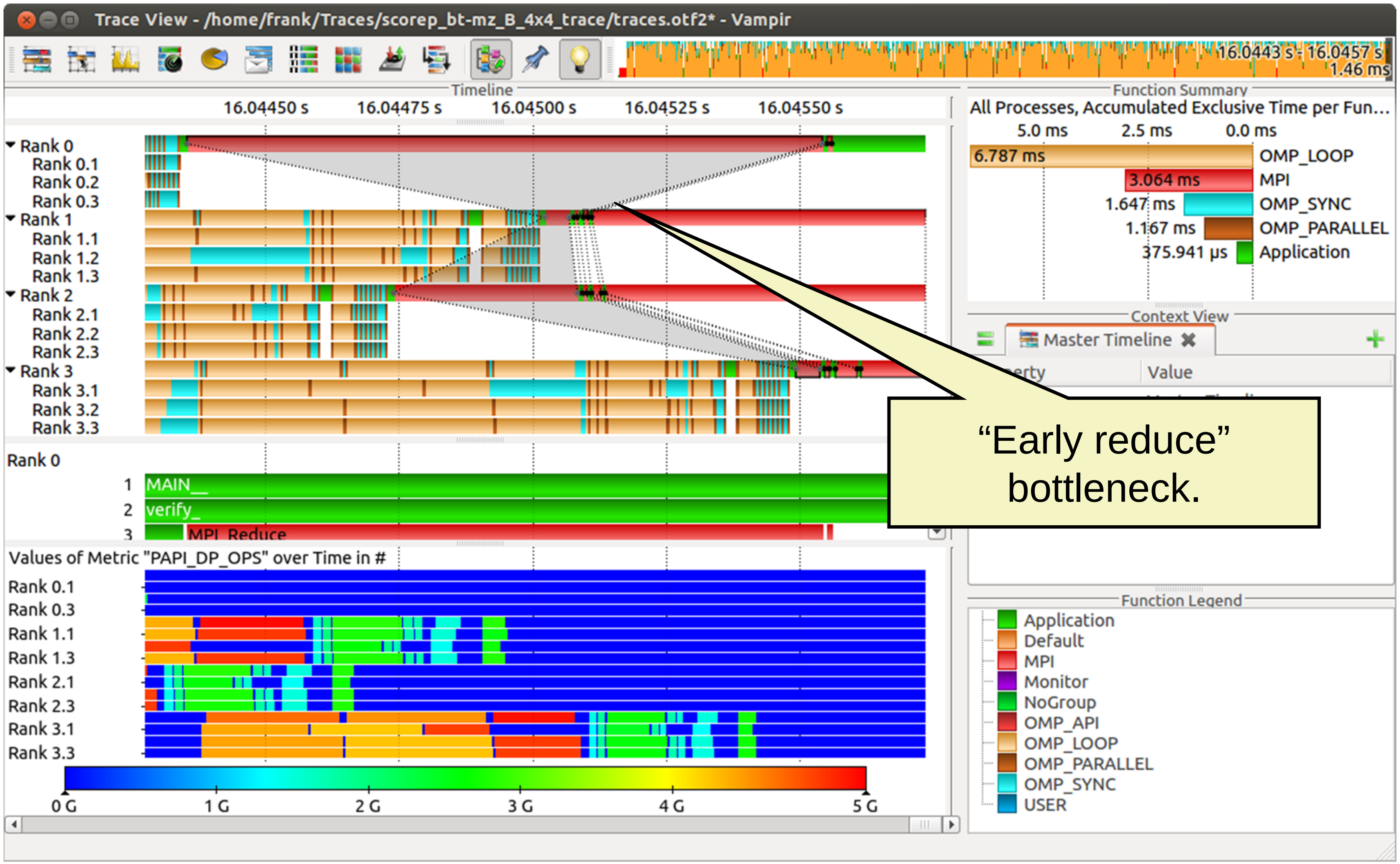


Computation phase results in higher floating point operations.

Zoom in: Computation Phase

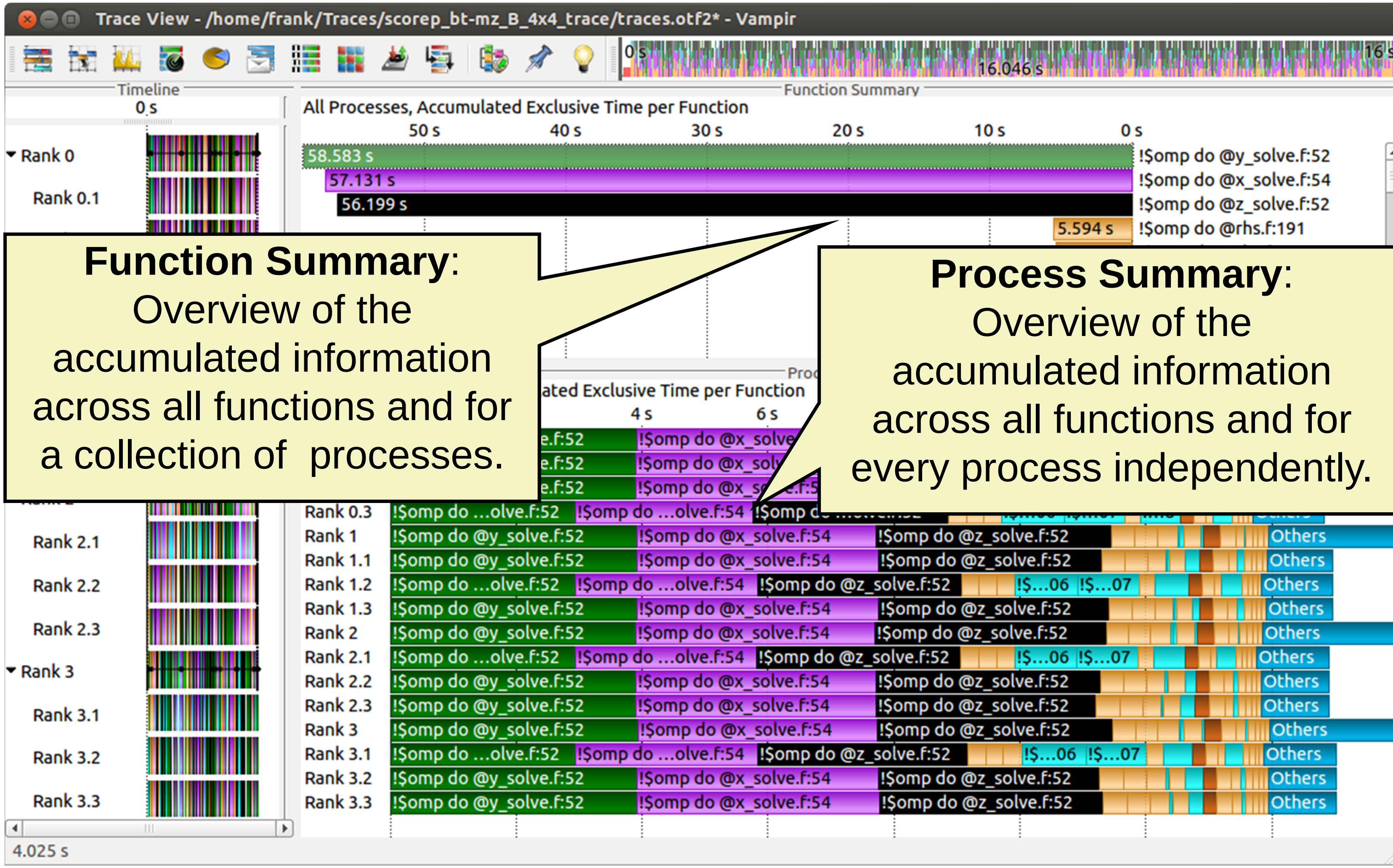


Zoom in: Finalisation Phase



Vampir: Visualization of the NPB-MZ-MPI / BT trace

Process Summary

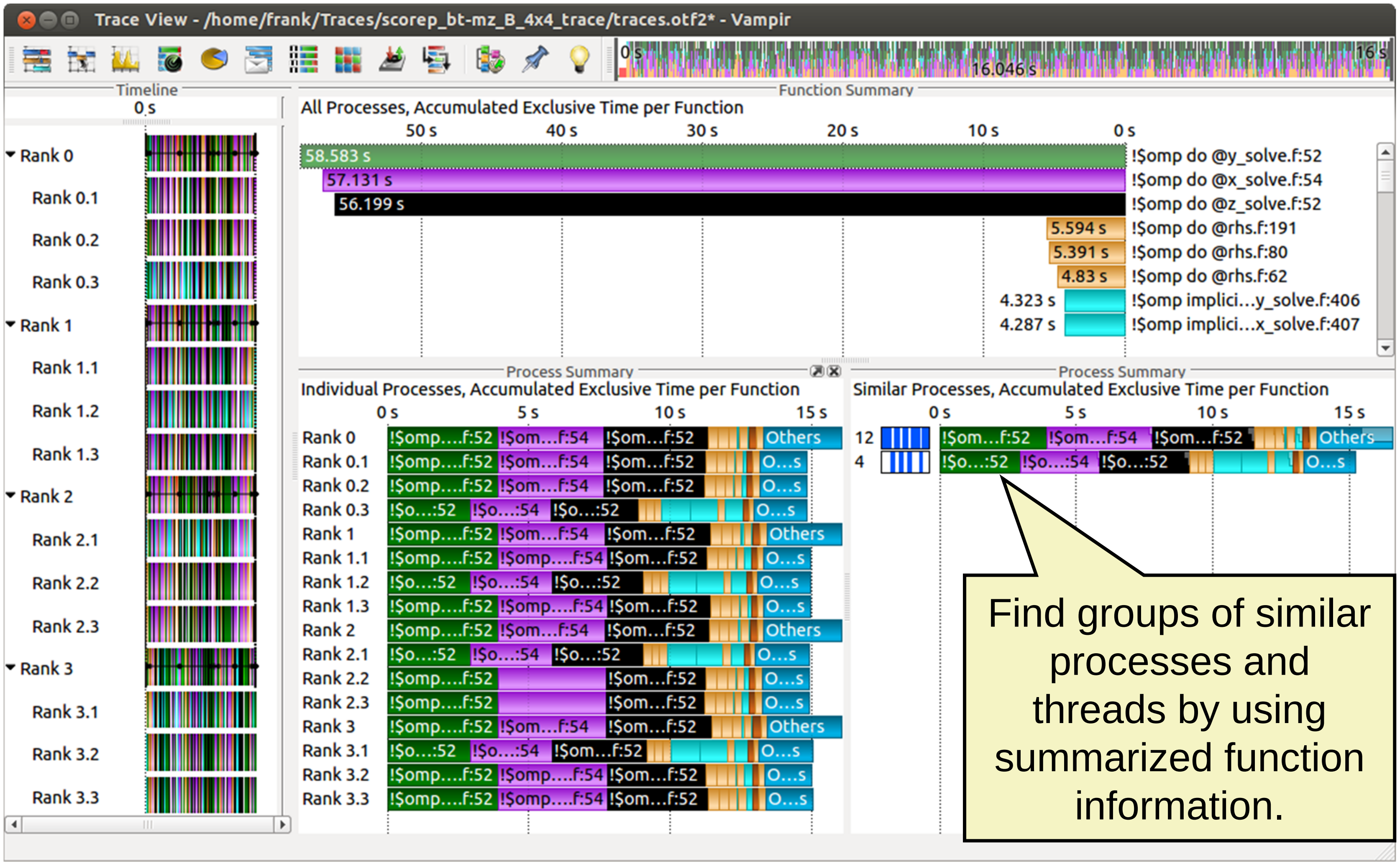


Function Summary: Overview of the accumulated information across all functions and for a collection of processes.

Process Summary: Overview of the accumulated information across all functions and for every process independently.

Vampir: Visualization of the NPB-MZ-MPI / BT trace

Process Summary



Summary and Conclusion

Summary

- Vampir & VampirServer
  - Interactive trace visualization and analysis
  - Intuitive browsing and zooming
  - Scalable to large trace data sizes (20 TByte)
  - Scalable to high parallelism (200000 processes)
- Vampir for Linux, Windows and Mac OS X

- **Note:** Vampir does neither solve your problems automatically nor point you directly at them. It does, however, give you FULL insight into the execution of your application.

Conclusion

- performance analysis very important in HPC

- use performance analysis tools for profiling and tracing
- do not spend effort in DIY solutions,
  e.g. like printf-debugging

- use tracing tools with some precautions
  - overhead
  - data volume

- let us know about problems and about feature wishes
- vampirsupport@zih.tu-dresden.de

Vampir is available at http://www.vampir.eu,

get support via vampirsupport@zih.tu-dresden.de

Acknowledgement

Staff at ZIH - TU Dresden:

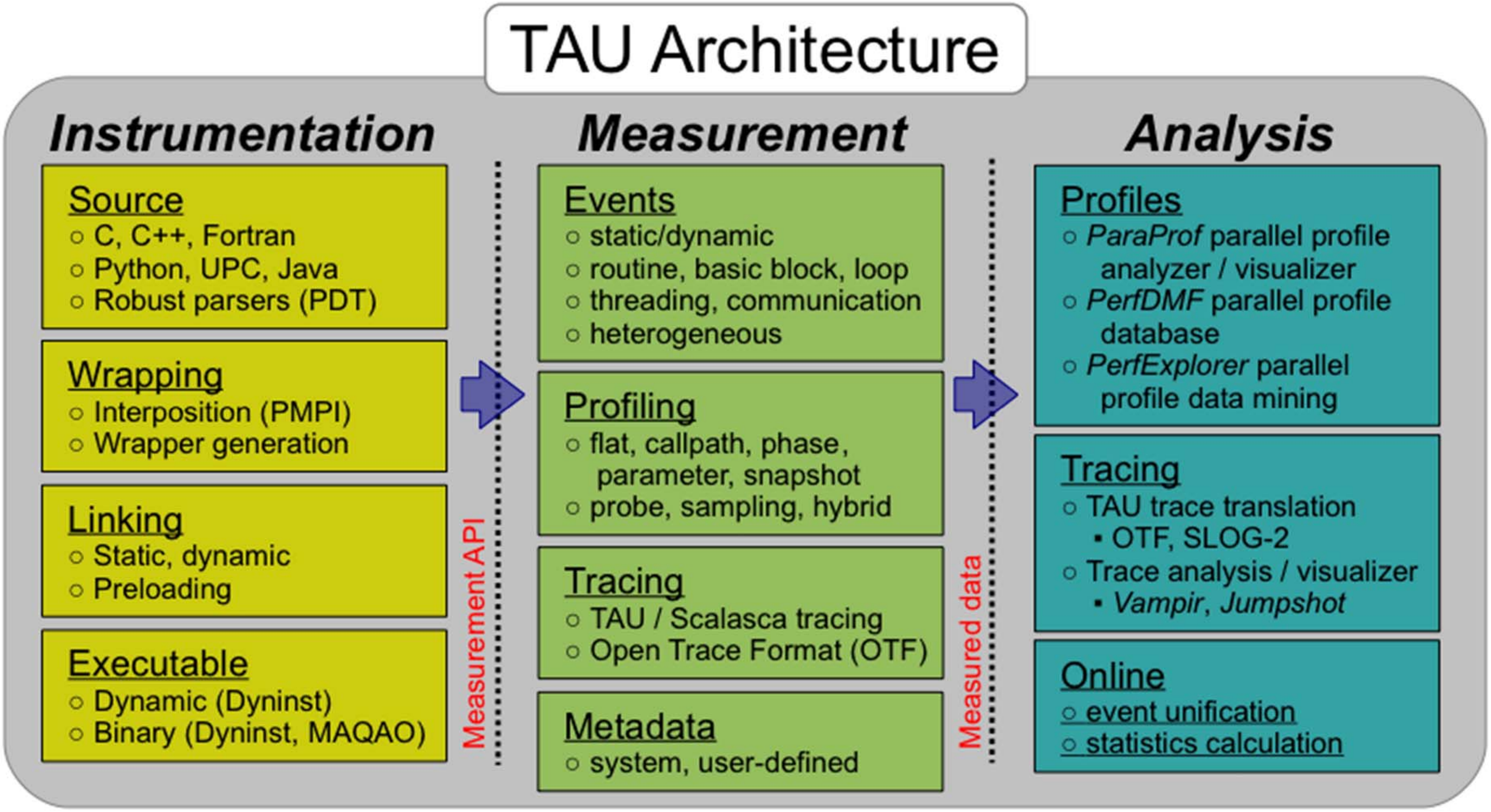
Ronny Brendel, Holger Brunst, Jens Doleschal, Ronald Geisler, Daniel Hackenberg, Michael Heyde, Matthias Jurenz, Michael Kluge, Andreas Knüpfer, Matthias Lieber, Holger Mickler, Hartmut Mix, Matthias Weber, Bert Wesarg, Frank Winkler, Matthias Müller, Wolfgang E. Nagel

Profile Analysis with ParaProf

Tuning and Analysis Utilities

Sameer Shende
Performance Reseaerch Lab, University of Oregon
http://TAU.uoregon.edu

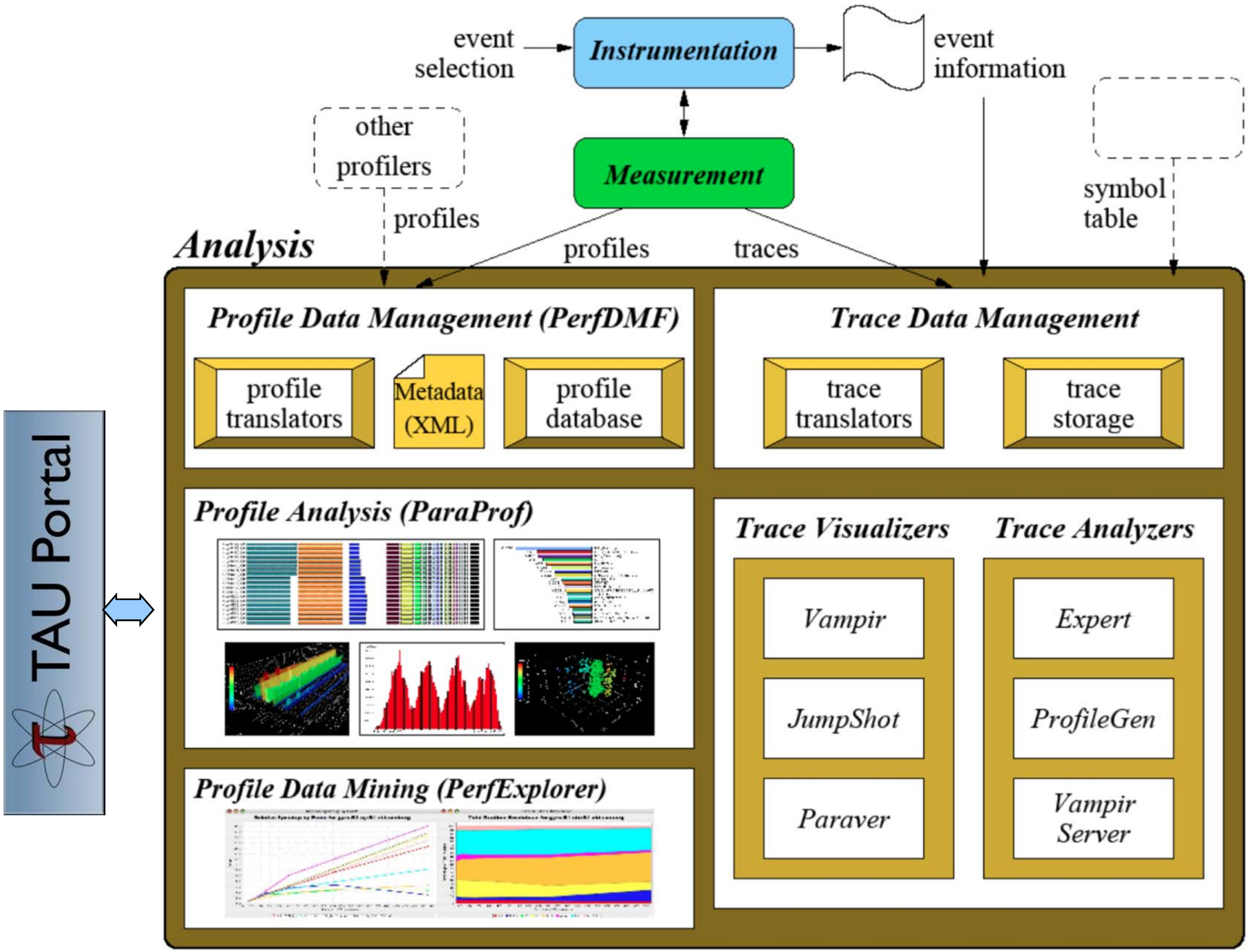
- Parallel performance framework and toolkit
  - Supports all HPC platforms, compilers, runtime system
  - Provides portable instrumentation, measurement, analysis



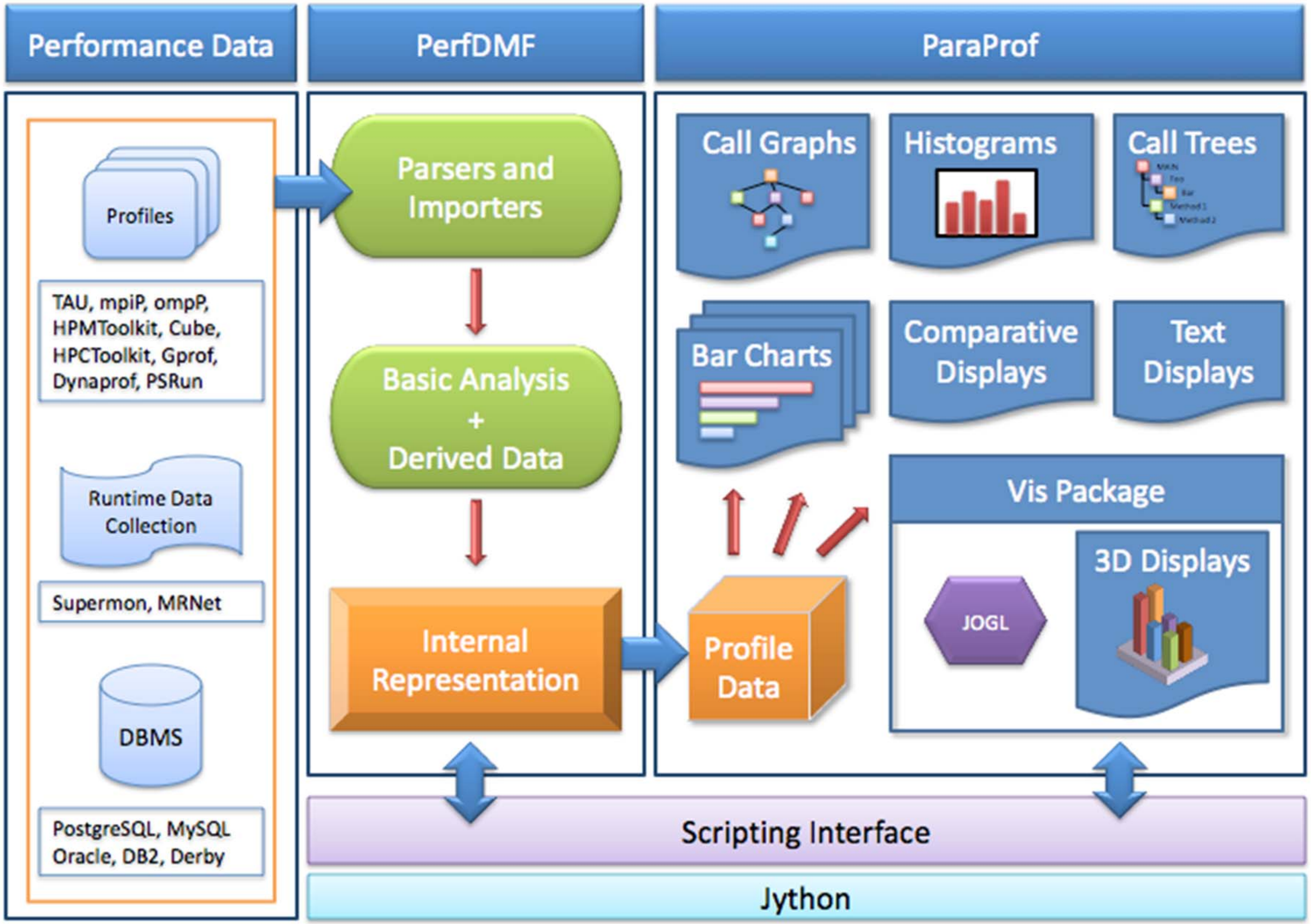
TAU Performance System®

- Instrumentation
  - Fortran, C++, C, UPC, Java, Python, Chapel
  - Automatic instrumentation
- Measurement and analysis support
  - MPI, OpenSHMEM, ARMCI, PGAS, DMAPP
  - pthreads, OpenMP, hybrid, other thread models
  - GPU, CUDA, OpenCL, OpenACC
  - Parallel profiling and tracing
  - Use of Score-P for native OTF2 and CUBEX generation
  - Efficient callpath proflles and trace generation using Score-P
- Analysis
  - Parallel profile analysis (ParaProf), data mining (PerfExplorer)
  - Performance database technology (PerfDMF, TAUdb)
  - 3D profile browser

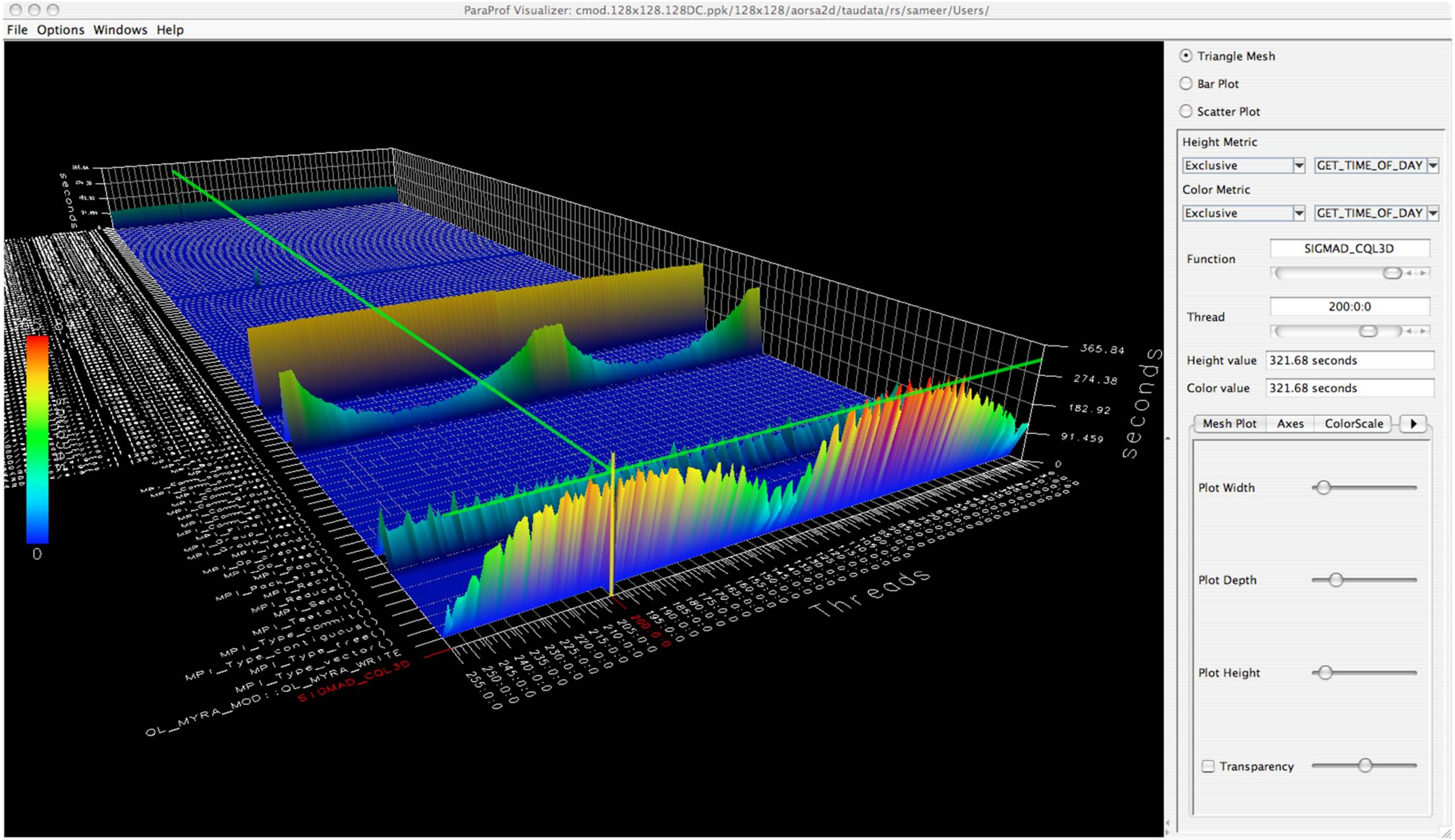
TAU Analysis



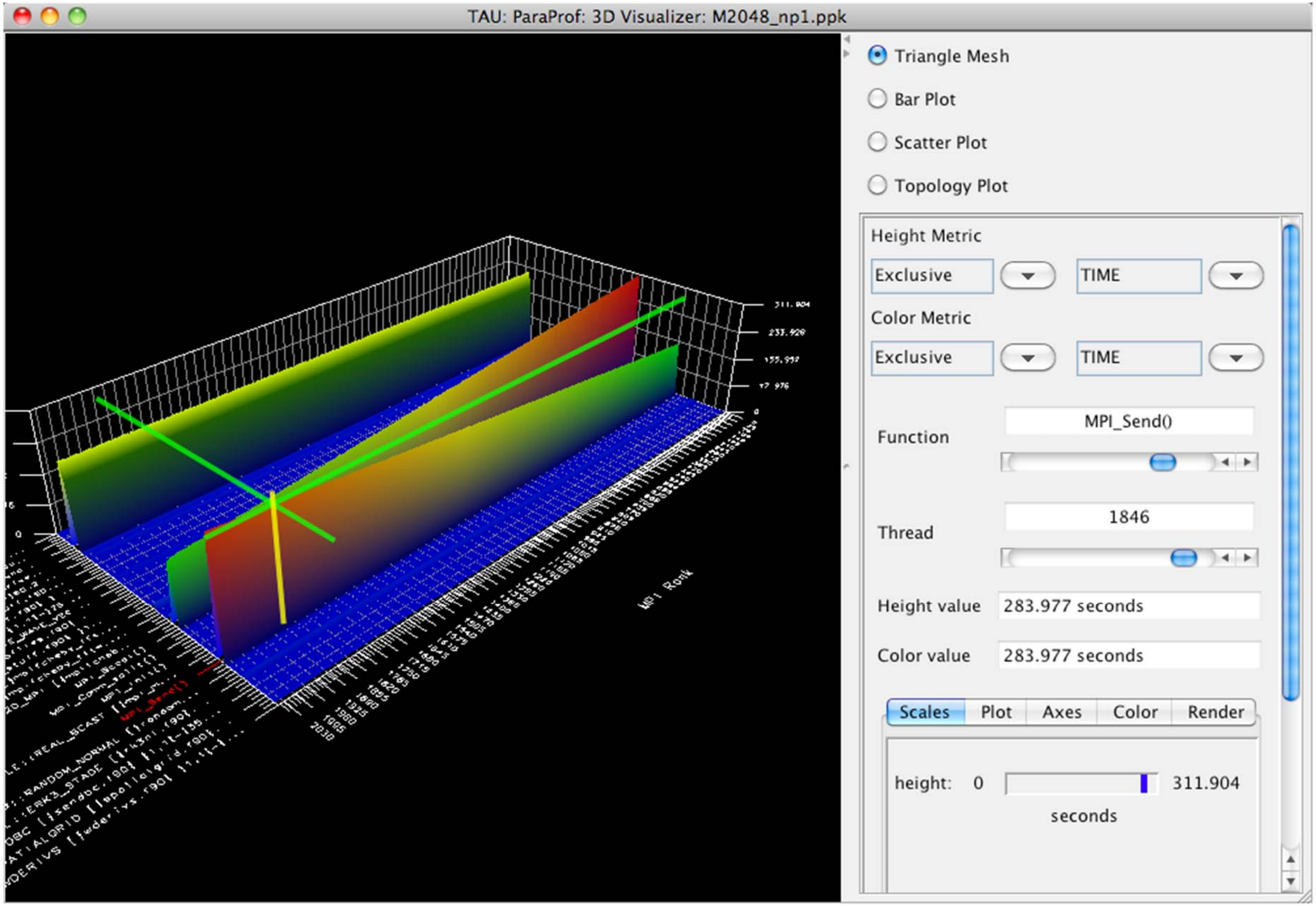
ParaProf Profile Analysis Framework



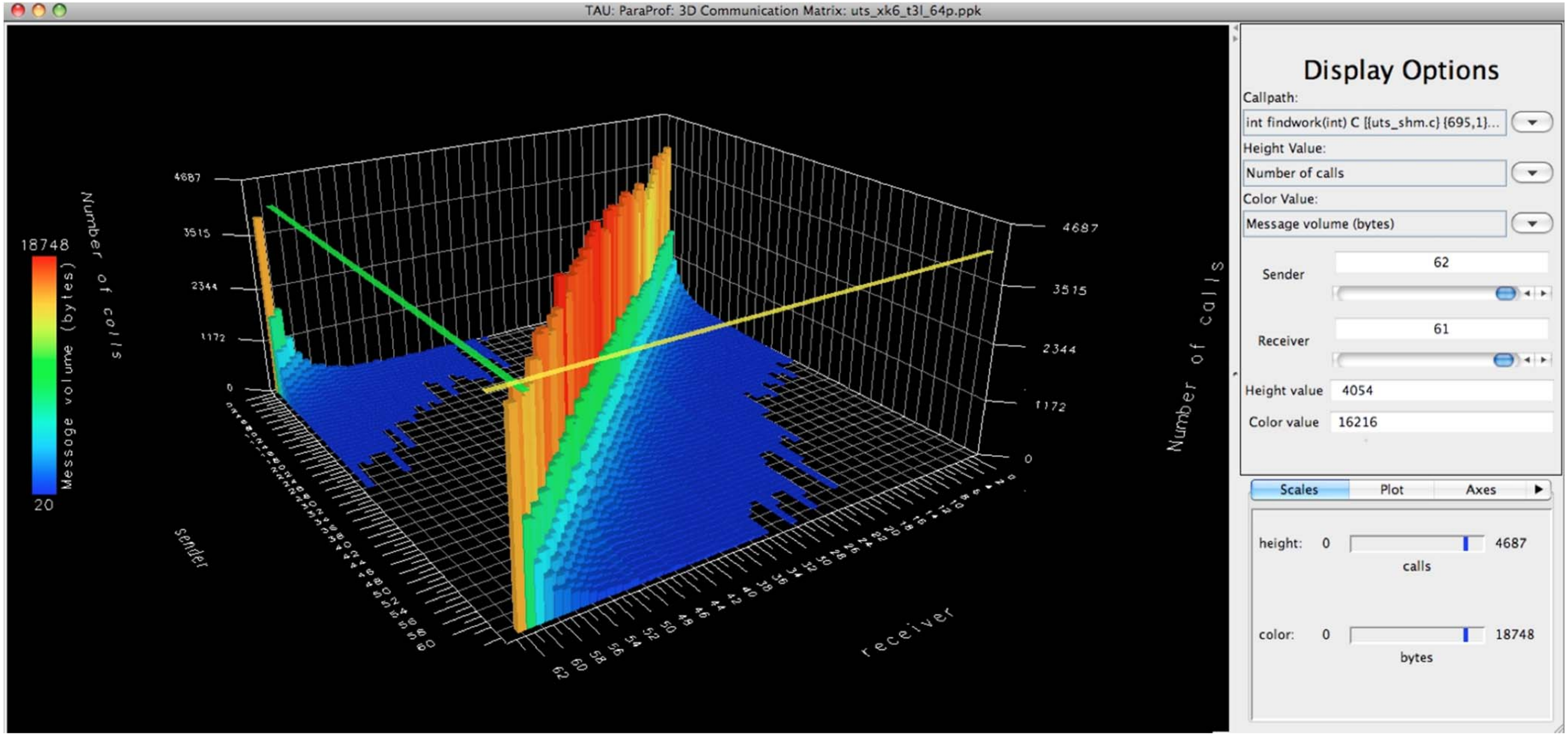
Parallel Profile Visualization: ParaProf



Parallel Profile Visualization: ParaProf



ParaProf: 3D Communication Matrix



Hands-on: Profile report exploration

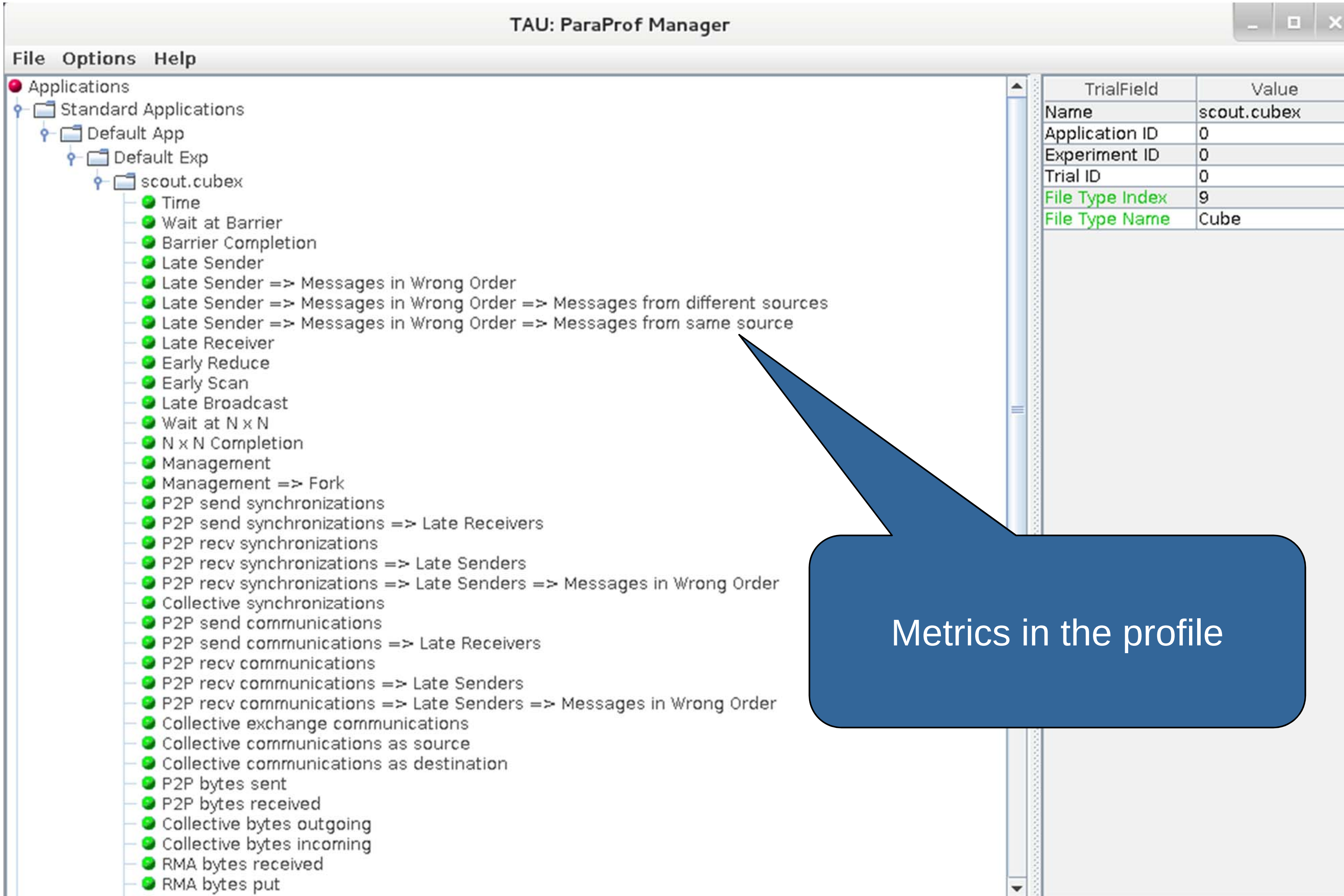
- The Live-DVD contains Score-P experiments of BT-MZ
  - class “B“, 4 processes with 4 OpenMP threads each
  - collected on a dedicated node of the SuperMUC HPC system at Leibniz Rechenzentrum (LRZ), Munich, Germany

```
% cd
% cd workshop-vihps/supermuc_expts
% ls
periscope-1.5                          scorep_bt-mz_B_4x4_sum
README                                 scorep_bt-mz_B_4x4_sum+mets
run.out                                scorep_bt-mz_B_4x4_trace
scorep-20120913_1740_557443655223384
```

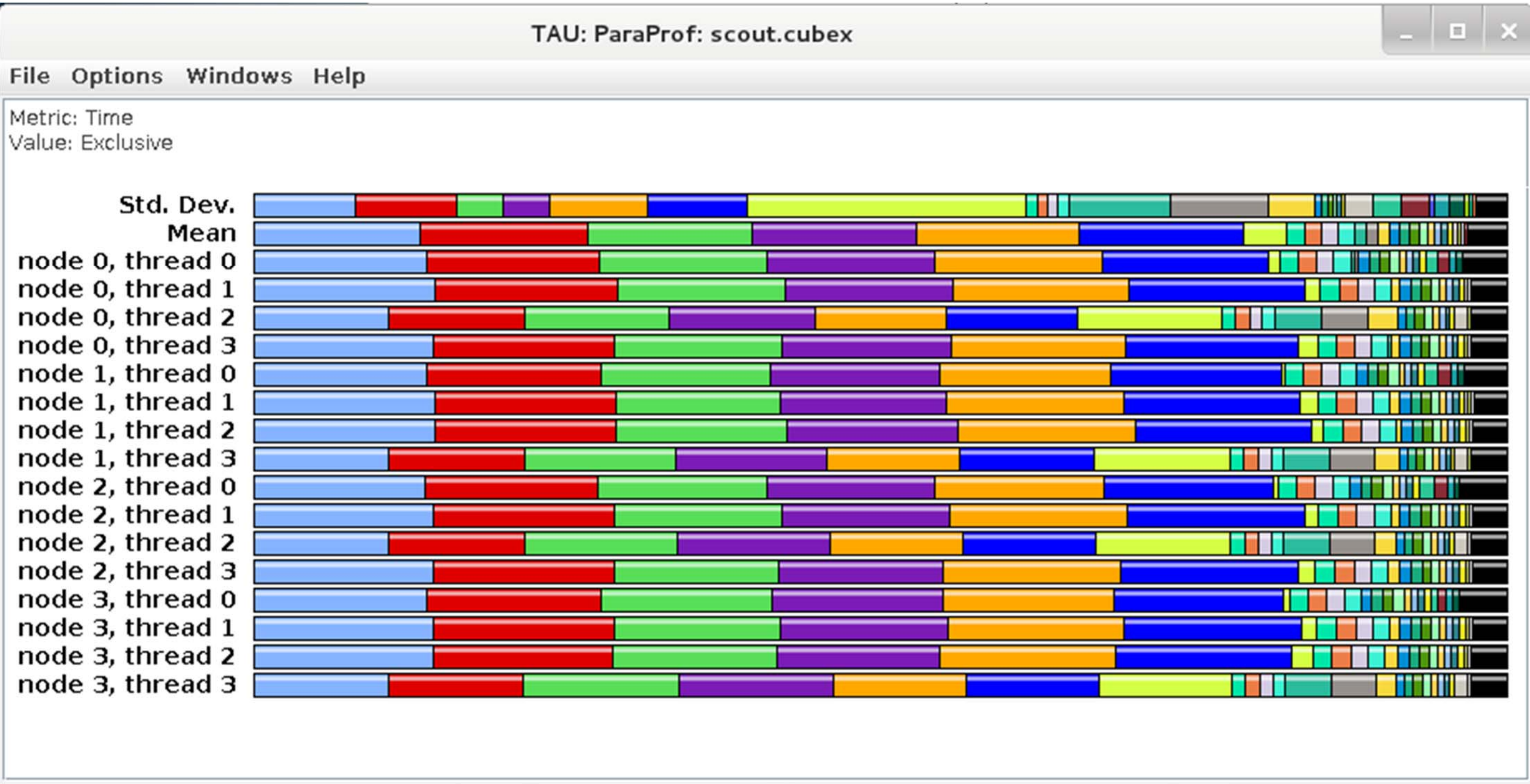
- Start TAU‘s paraprof GUI with default profile report

```
% paraprof scorep-20120913_1740_557443655223384/profile.cubex
OR
% paraprof scorep_bt-mz_B_4x4_trace/scout.cubex
```

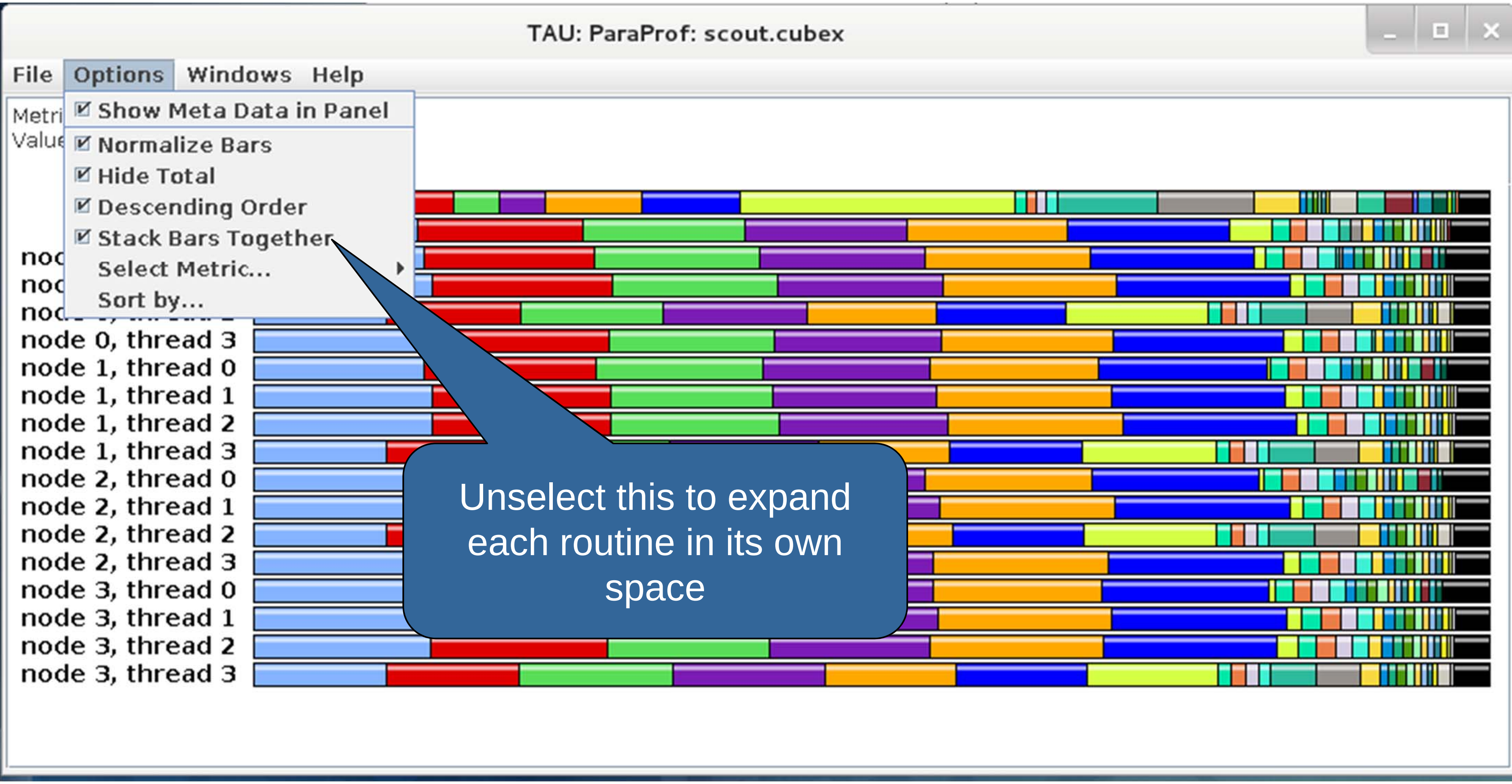
ParaProf: Manager Window: scout.cubex



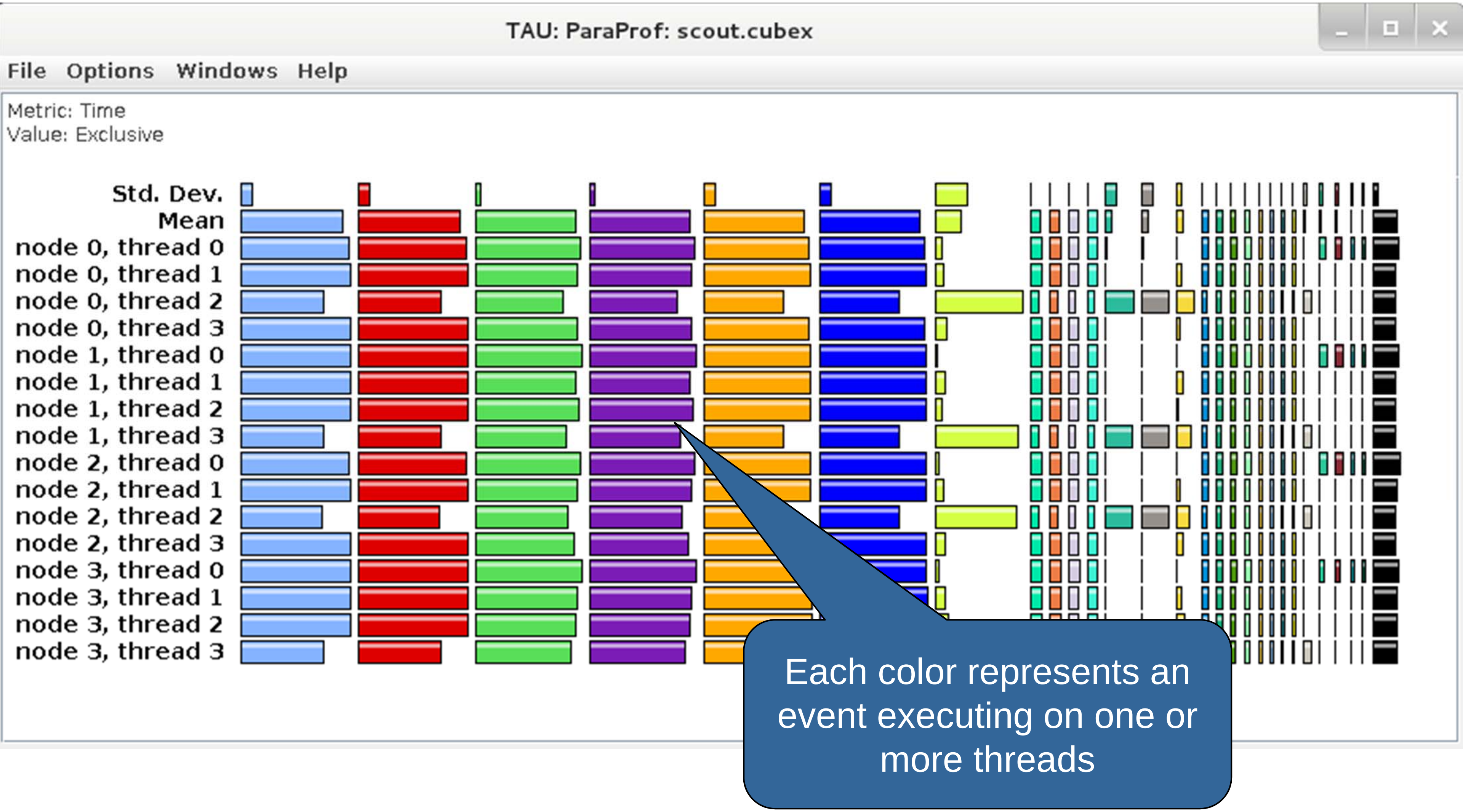
ParaProf: Main window



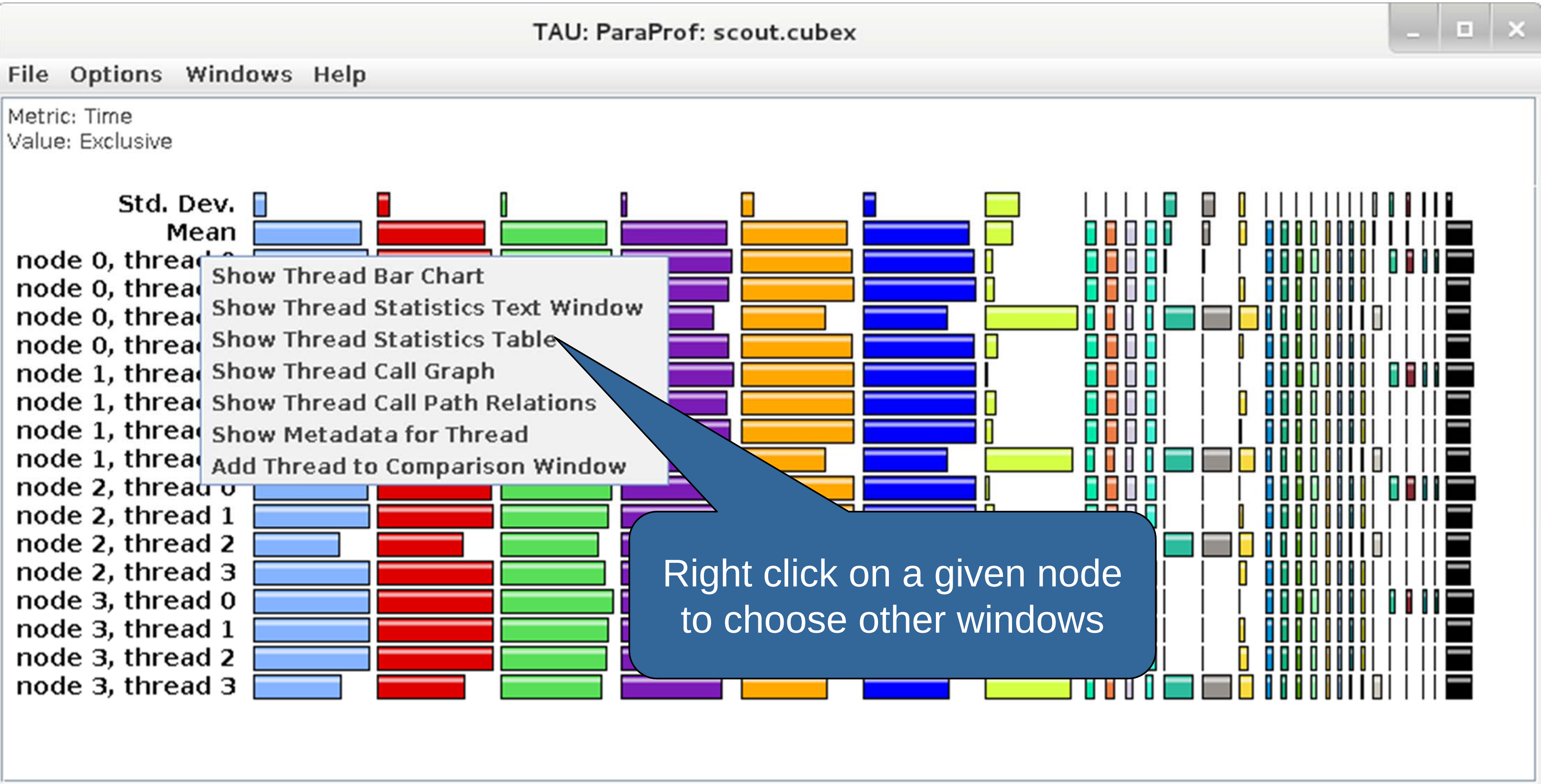
ParaProf: Options



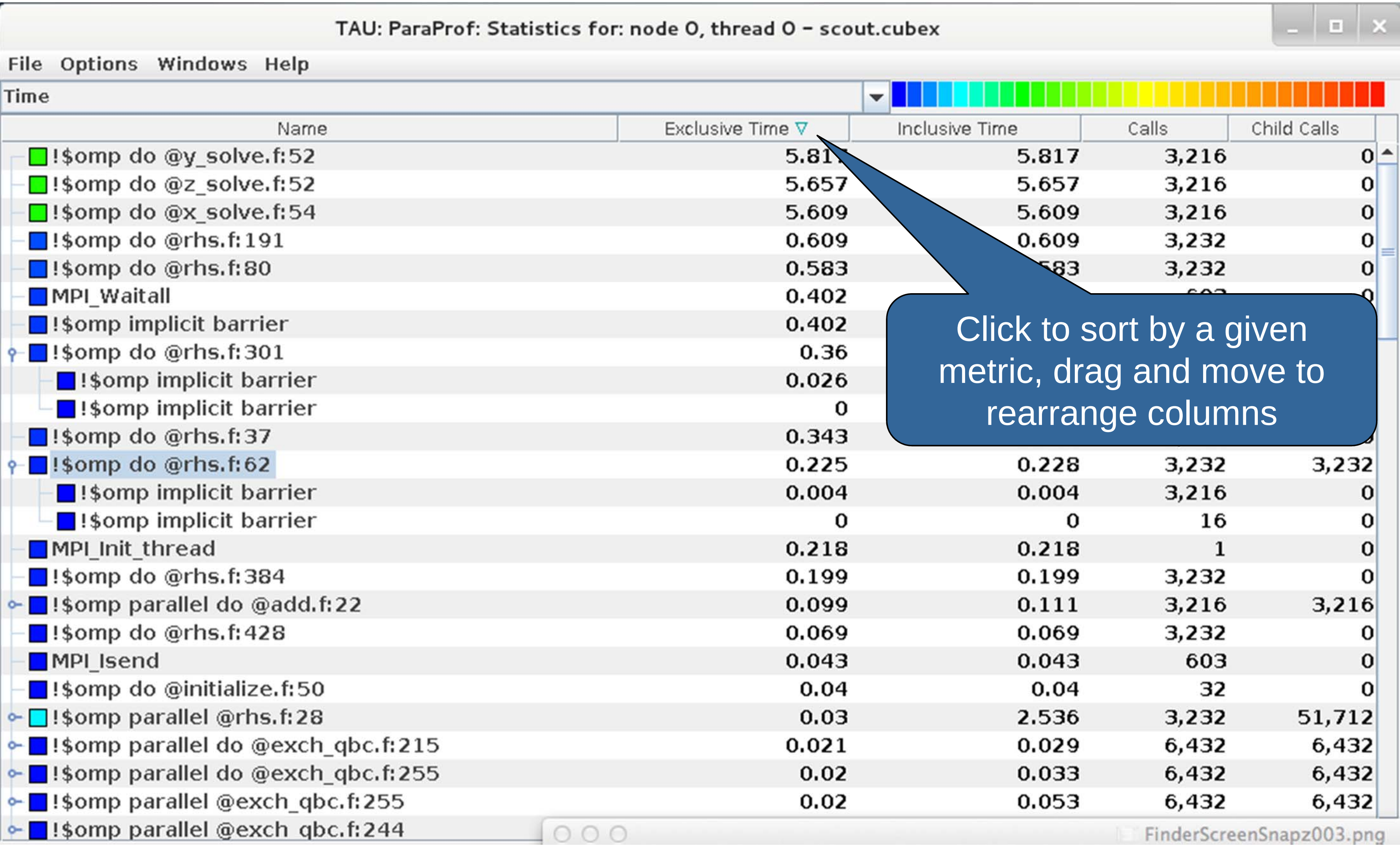
ParaProf:


TAU: ParaProf: scout.cubex
File Options Windows Help
Metric: Time
Value: Exclusive
Std. Dev.
Mean
node 0, thread 0
node 0, thread 1
node 0, thread 2
node 0, thread 3
node 1, thread 0
node 1, thread 1
node 1, thread 2
node 1, thread 3
node 2, thread 0
node 2, thread 1
node 2, thread 2
node 2, thread 3
node 3, thread 0
node 3, thread 1
node 3, thread 2
node 3, thread 3
Each color represents an event executing on one or more threads

ParaProf: Windows



ParaProf: Thread Statistics Table



Name	Exclusive Time ▽	Inclusive Time	Calls	Child Calls
!$omp do @y_solve.f:52	5.81[illegible]	5.817	3,216	0
!$omp do @z_solve.f:52	5.657	5.657	3,216	0
!$omp do @x_solve.f:54	5.609	5.609	3,216	0
!$omp do @rhs.f:191	0.609	0.609	3,232	0
!$omp do @rhs.f:80	0.583	[illegible]	3,232	0
MPI_Waitall	0.402	[illegible]	[illegible]	0
!$omp implicit barrier	0.402	[illegible]	[illegible]	[illegible]
!$omp do @rhs.f:301	0.36	[illegible]	[illegible]	[illegible]
!$omp implicit barrier	0.026	[illegible]	[illegible]	[illegible]
!$omp implicit barrier	0	[illegible]	[illegible]	[illegible]
!$omp do @rhs.f:37	0.343	[illegible]	[illegible]	[illegible]
!$omp do @rhs.f:62	0.225	0.228	3,232	3,232
!$omp implicit barrier	0.004	0.004	3,216	0
!$omp implicit barrier	0	0	16	0
MPI_Init_thread	0.218	0.218	1	0
!$omp do @rhs.f:384	0.199	0.199	3,232	0
!$omp parallel do @add.f:22	0.099	0.111	3,216	3,216
!$omp do @rhs.f:428	0.069	0.069	3,232	0
MPI_Isend	0.043	0.043	603	0
!$omp do @initialize.f:50	0.04	0.04	32	0
!$omp parallel @rhs.f:28	0.03	2.536	3,232	51,712
!$omp parallel do @exch_qbc.f:215	0.021	0.029	6,432	6,432
!$omp parallel do @exch_qbc.f:255	0.02	0.033	6,432	6,432
!$omp parallel @exch_qbc.f:255	0.02	0.053	6,432	6,432
!$omp parallel @exch_qbc.f:244				

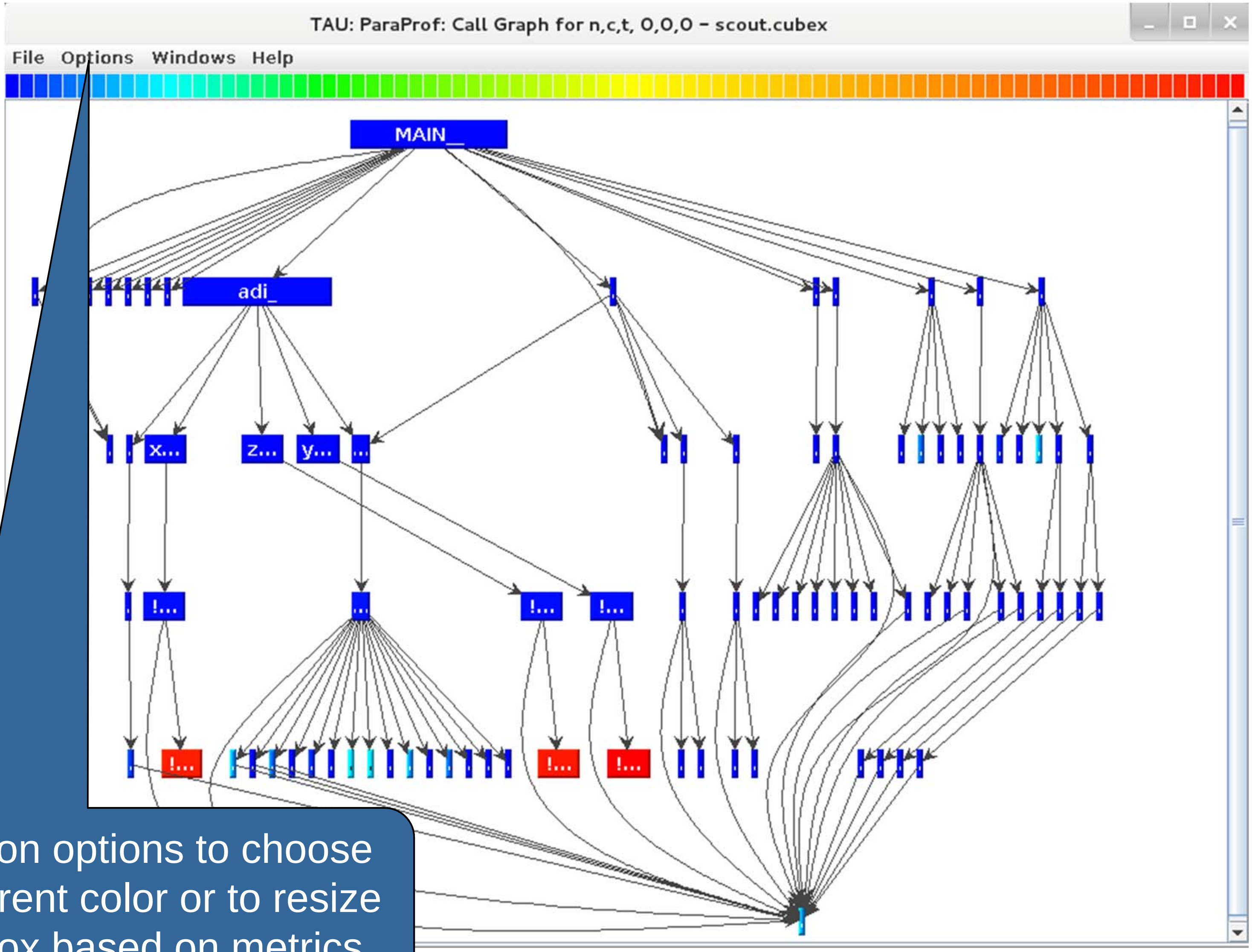
Example: Score-P with TAU (LU NPB)

TAU: ParaProf: Statistics for: node 0, thread 0 – profile.cubex

File Options Windows Help

Name	Exclusive Time ▽	Inclusive Time	Calls	Child Calls
APPLU [{lu.f} {46,7}-{162,9}]	0	8.035	1	19
SSOR [{ssor.f} {4,7}-{241,9}]	0.064	6.225	2	37,643
RHS [{rhs.f} {5,7}-{504,9}]	0.743	2.524	303	606
BLTS [{blts.f} {4,7}-{259,9}]	0.613	0.658	9,331	18,662
BUTS [{buts.f} {4,7}-{259,9}]	0.612	1.871	9,331	18,662
EXCHANGE_1 [{exchange_1.f} {5,7}-{177,9}]	0.024	1.259	18,662	18,662
MPI_Recv	1.235	1.235	18,662	0
MPI_Send	0	0	0	0
JACU [{jacu.f} {5,7}-{384,9}]	0.532	0.532	9,331	0
JACLD [{jacld.f} {5,7}-{384,9}]	0.522	0.522	9,331	0
MPI_Allreduce	0.018	0.018	2	0
L2NORM [{l2norm.f} {4,7}-{68,9}]	0	0.035	4	4
MPI_Barrier	0	0	2	0
TIMER_START [{timers.f} {23,7}-{37,9}]	0	0	2	0
TIMER_STOP [{timers.f} {43,7}-{59,9}]	0	0	2	0
TIMER_CLEAR [{timers.f} {4,7}-{17,9}]	0	0	2	0
TIMER_READ [{timers.f} {65,7}-{77,9}]	0	0	2	0
SETIV [{setiv.f} {4,7}-{67,9}]	0.043	0.111	2	95,232
PROC_GRID [{proc_grid.f} {5,7}-{34,9}]	0.011	0.011	1	0
ERHS [{erhs.f} {4,7}-{536,9}]	0.004	0.108	1	2
ERROR [{error.f} {4,7}-{81,9}]	0.004	0.009	1	7,937
SETBV [{setbv.f} {5,7}-{79,9}]	0.002	0.004	2	3,400
READ_INPUT [{read_input.f} {5,7}-{125,9}]	0	0.001	1	2
VERIFY [{verify.f} {5,9}-{403,11}]	0	0	1	0
PRINT_RESULTS [{print_results.f} {2,7}-{115,12}]	0	0	1	0
PINTGR [{pintgr.f} {5,7}-{288,9}]	0	0	1	6
INIT_COMM [{init_comm.f} {5,7}-{57,9}]	0	1.565	1	4
MPI_Finalize	0	0	1	0
SETHYPER [{sethyper.f} {5,7}-{94,9}]	0	0	1	0
NEIGHBORS [{neighbors.f} {5,7}-{48,9}]	0	0	1	0
SETCOEFF [{setcoeff.f} {5,7}-{157,9}]	0	0	1	0

ParaProf: Thread Callgraph Window



Click on options to choose a different color or to resize the box based on metrics

ParaProf: Callpath Thread Relations Window

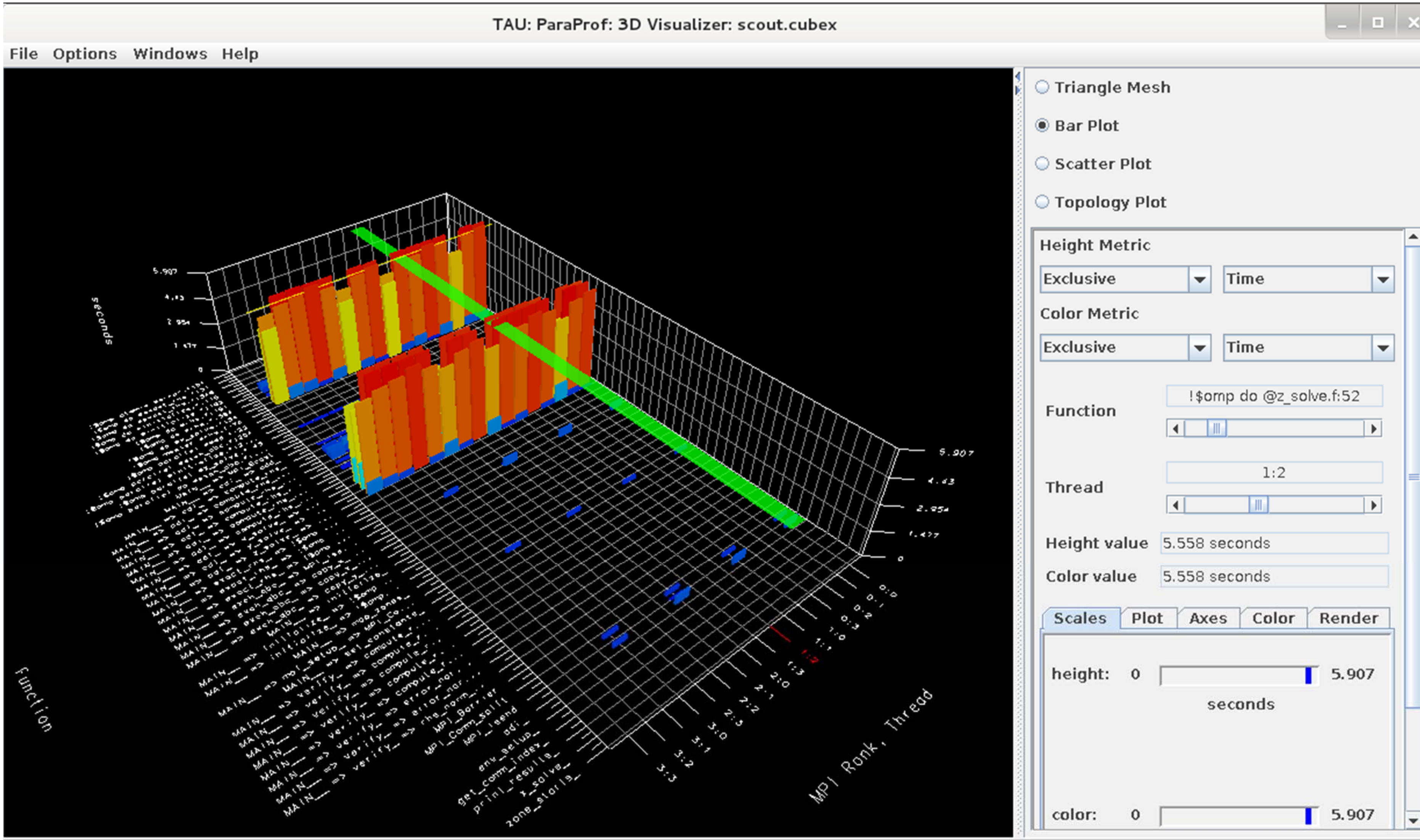
TAU: ParaProf: Call Path Data n,c,t, 0,0,0 - scout.cubex

File Options Windows Help

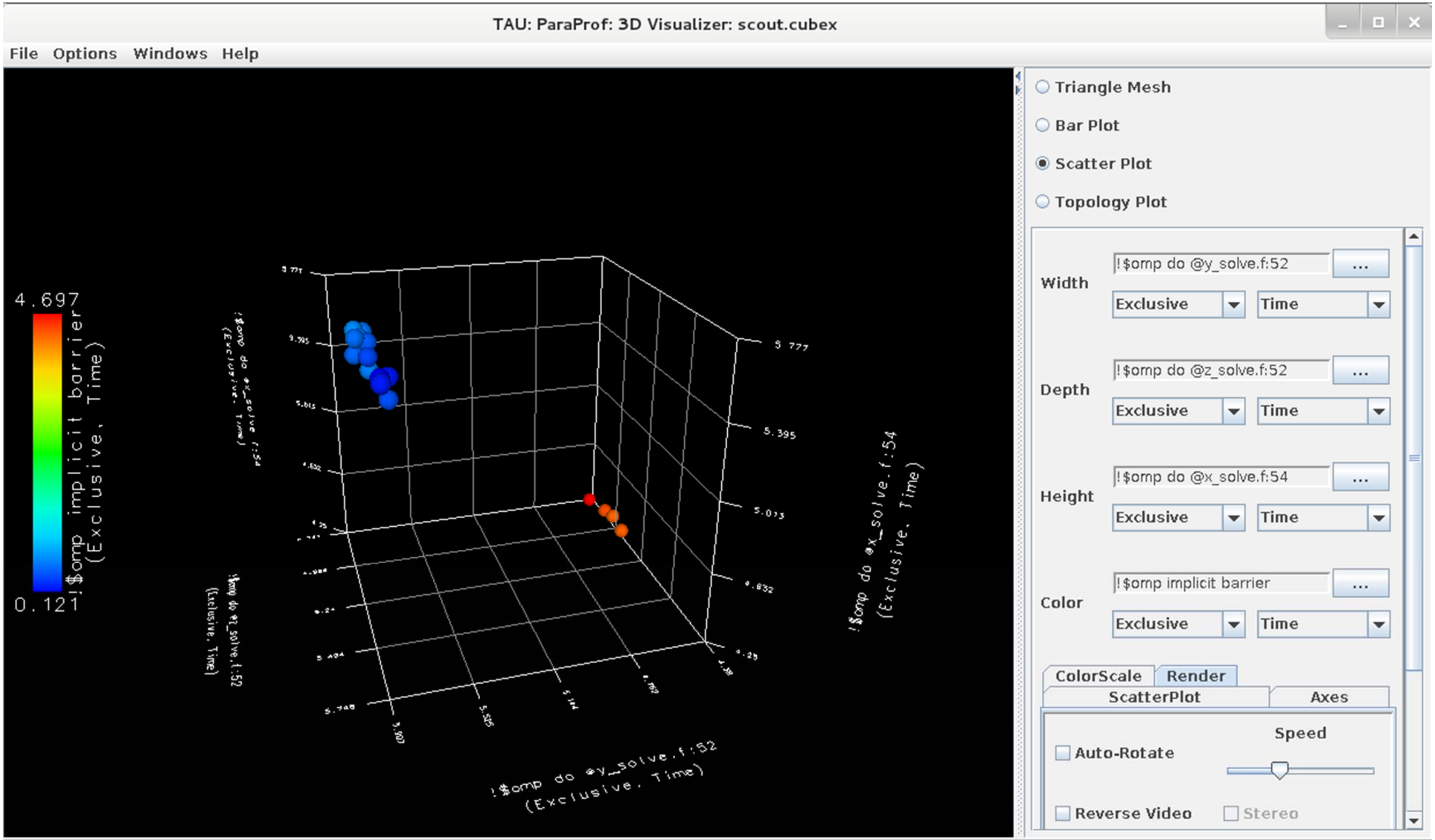
Metric Name: Time
Sorted By: Exclusive
Units: seconds

	0.04	0.04	32/32	!$omp parallel @initialize.f:28
-->	0.04	0.04	32	!$omp do @initialize.f:50
	0.03	2.536	3232/3232	compute_rhs_
-->	0.03	2.536	3232	!$omp parallel @rhs.f:28
	9.8E-4	9.8E-4	3232/3232	!$omp master @rhs.f:424
	0.225	0.228	3232/3232	!$omp do @rhs.f:62
	0.002	0.002	3232/3232	!$omp master @rhs.f:74
	0.002	0.002	3232/3232	!$omp master @rhs.f:293
	0.199	0.199	3232/3232	!$omp do @rhs.f:384
	0.002	0.002	3232/3232	!$omp master @rhs.f:183
	0.343	0.343	3232/3232	!$omp do @rhs.f:37
	0.016	0.016	3232/3232	!$omp do @rhs.f:372
	0.014	0.027	3232/3232	!$omp do @rhs.f:413
	0.609	0.609	3232/3232	!$omp do @rhs.f:191
	0.36	0.386	3232/3232	!$omp do @rhs.f:301
	0.583	0.583	3232/3232	!$omp do @rhs.f:80
	0.019	0.019	3232/3232	!$omp do @rhs.f:400
	0.006	0.006	3232/51680	!$omp implicit barrier
	0.069	0.069	3232/3232	!$omp do @rhs.f:428
	0.015	0.015	3232/3232	!$omp do @rhs.f:359
	0.021	0.029	6432/6432	!$omp parallel @exch_qbc.f:215
-->	0.021	0.029	6432	!$omp parallel do @exch_qbc.f:215
	0.007	0.007	6432/51680	!$omp implicit barrier
	0.02	0.033	6432/6432	!$omp parallel @exch_qbc.f:255
-->	0.02	0.033	6432	!$omp parallel do @exch_qbc.f:255
	0.013	0.013	6432/51680	!$omp implicit barrier

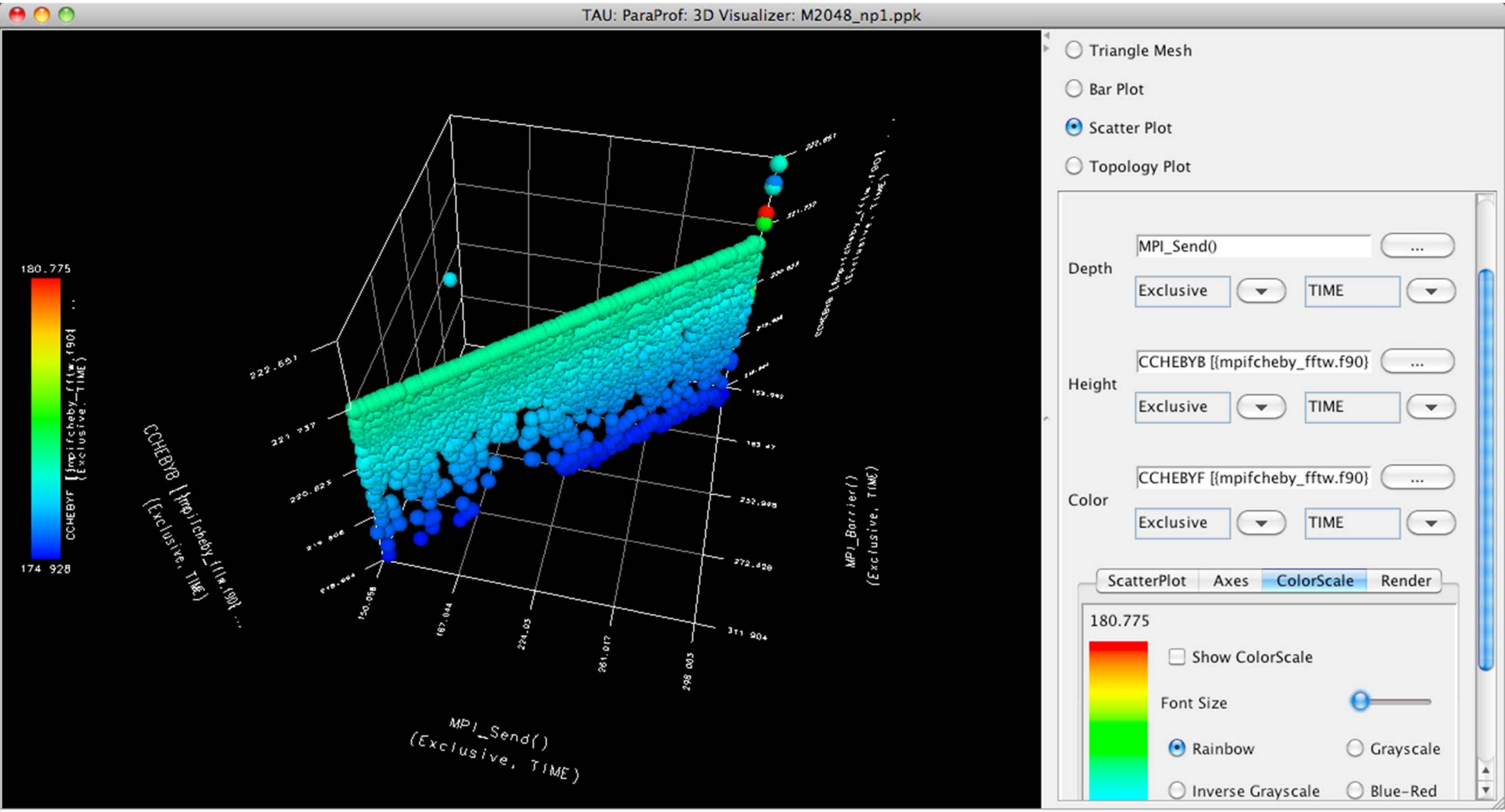
ParaProf:Windows -> 3D Visualization -> Bar Plot



ParaProf: 3D Scatter Plot

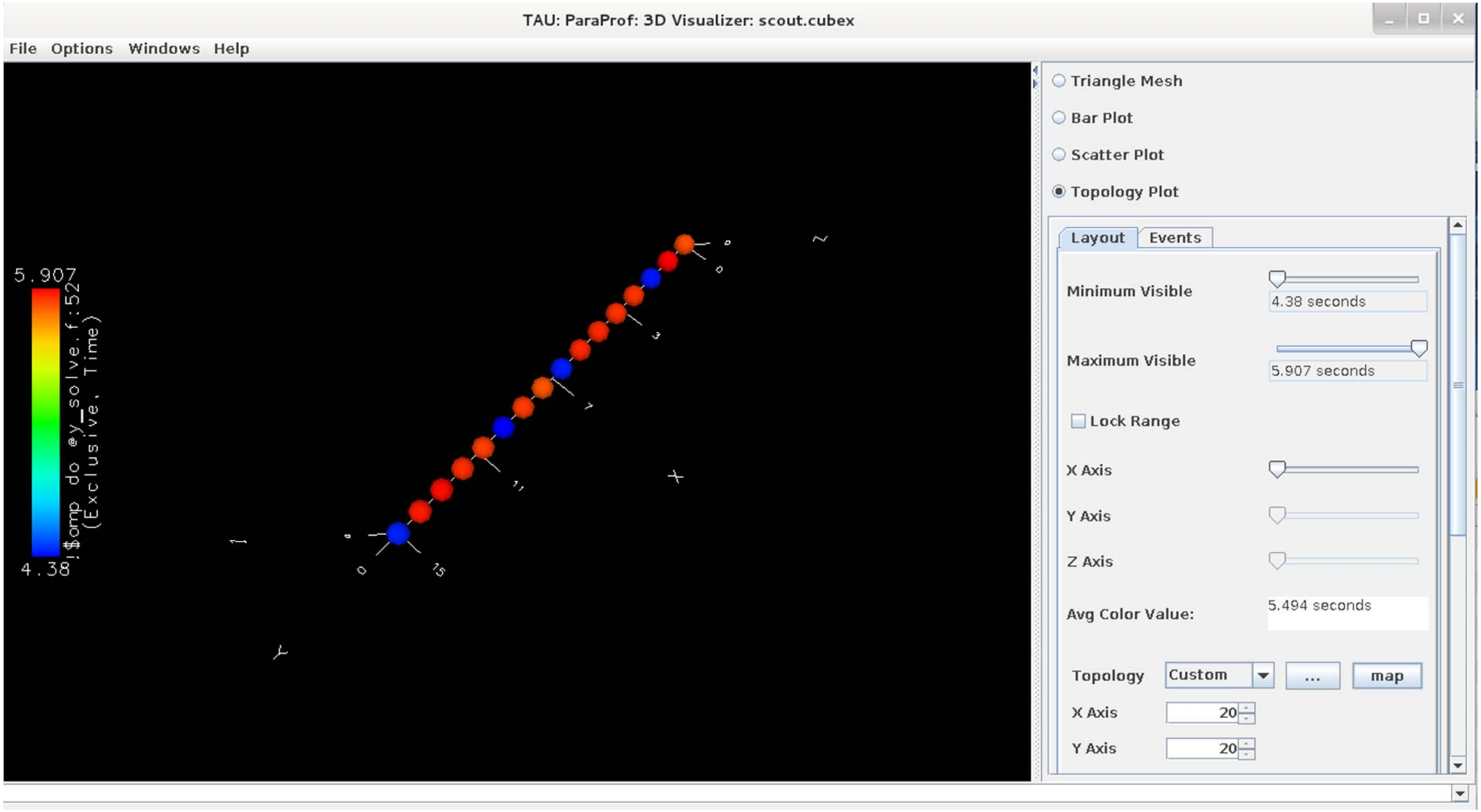


ParaProf: Scatter Plot

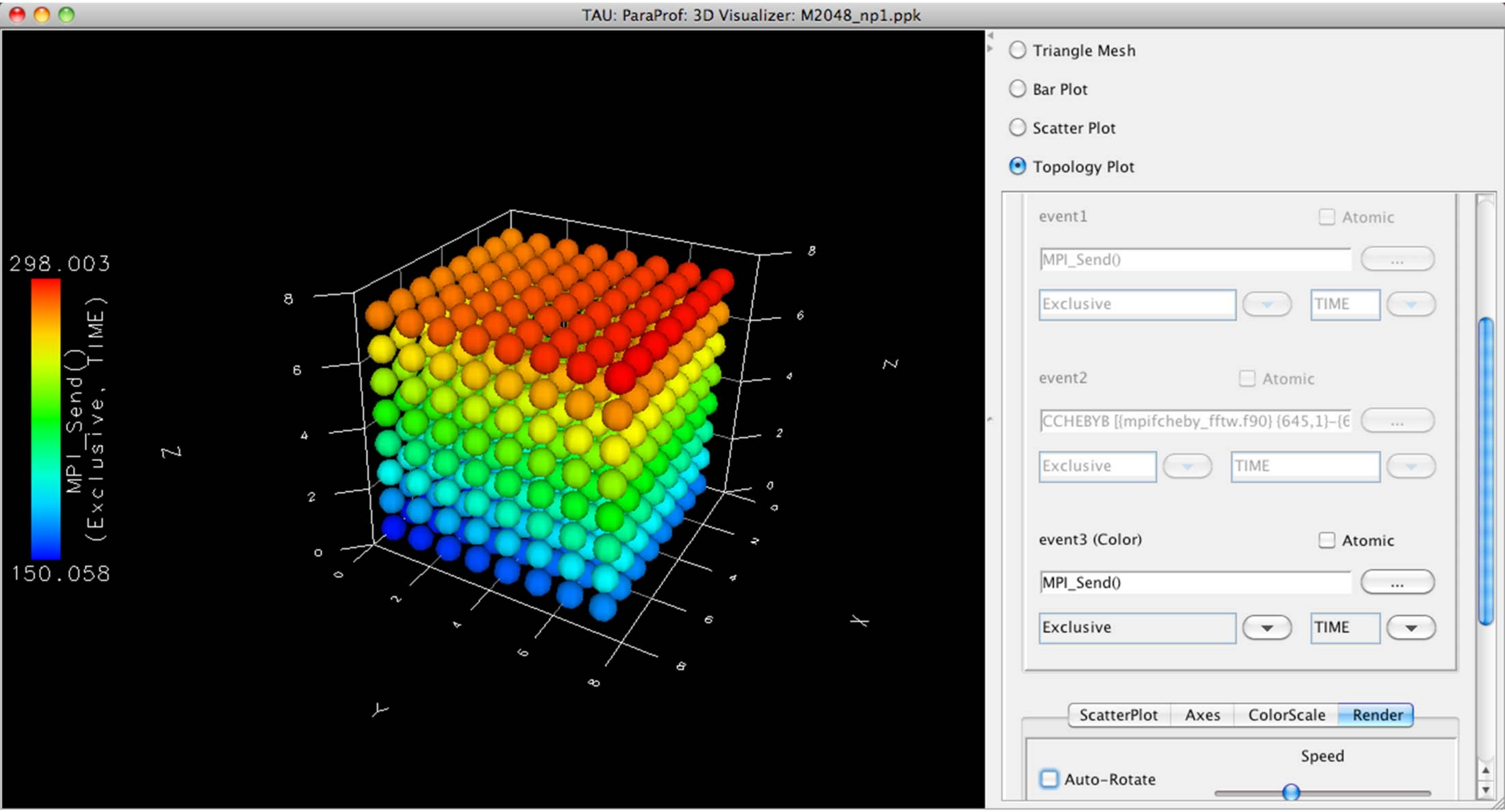


SC'13: Hands-on Practical Hybrid Parallel Application Performance Engineering

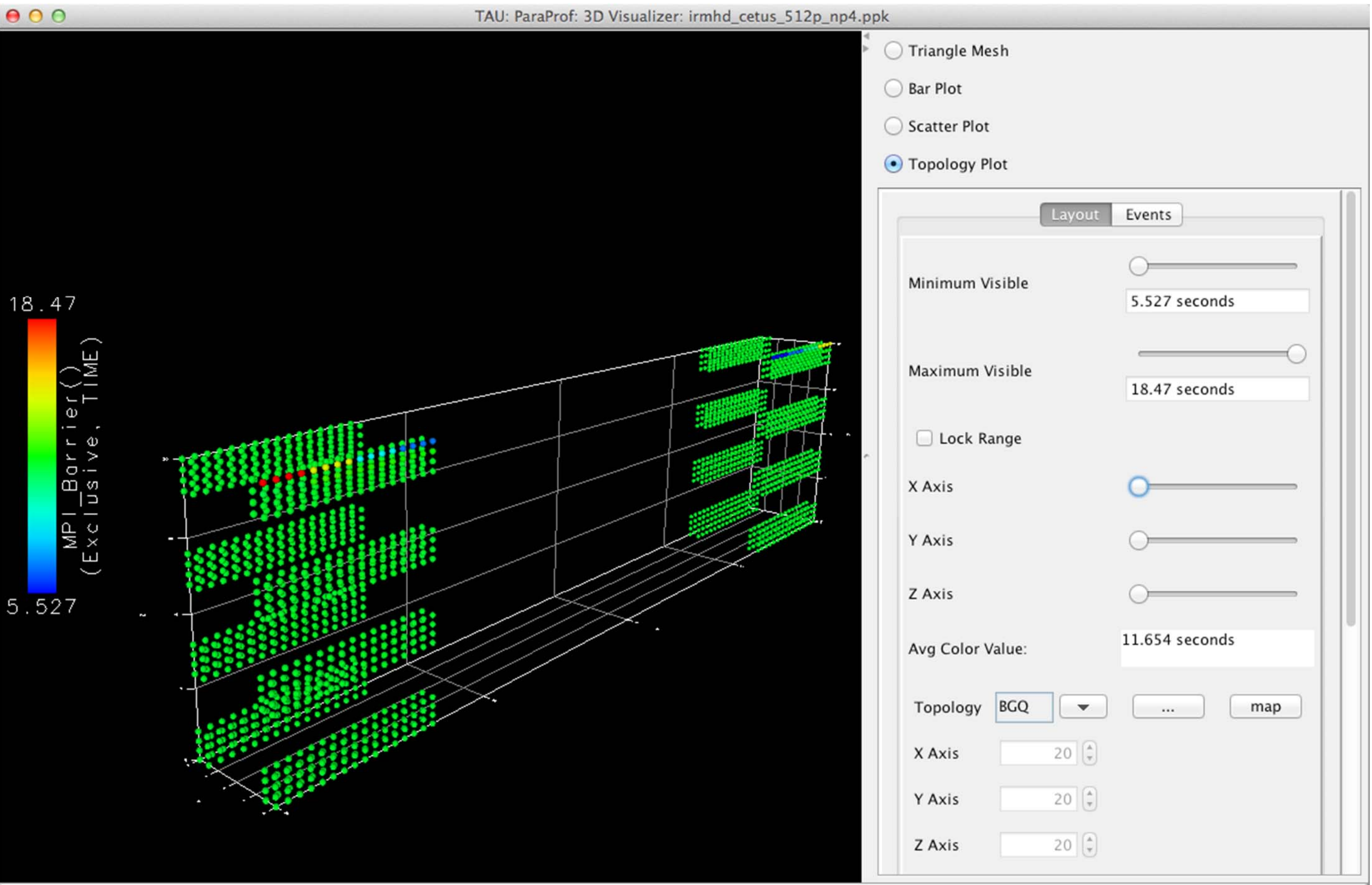
ParaProf: 3D Topology View for a Routine



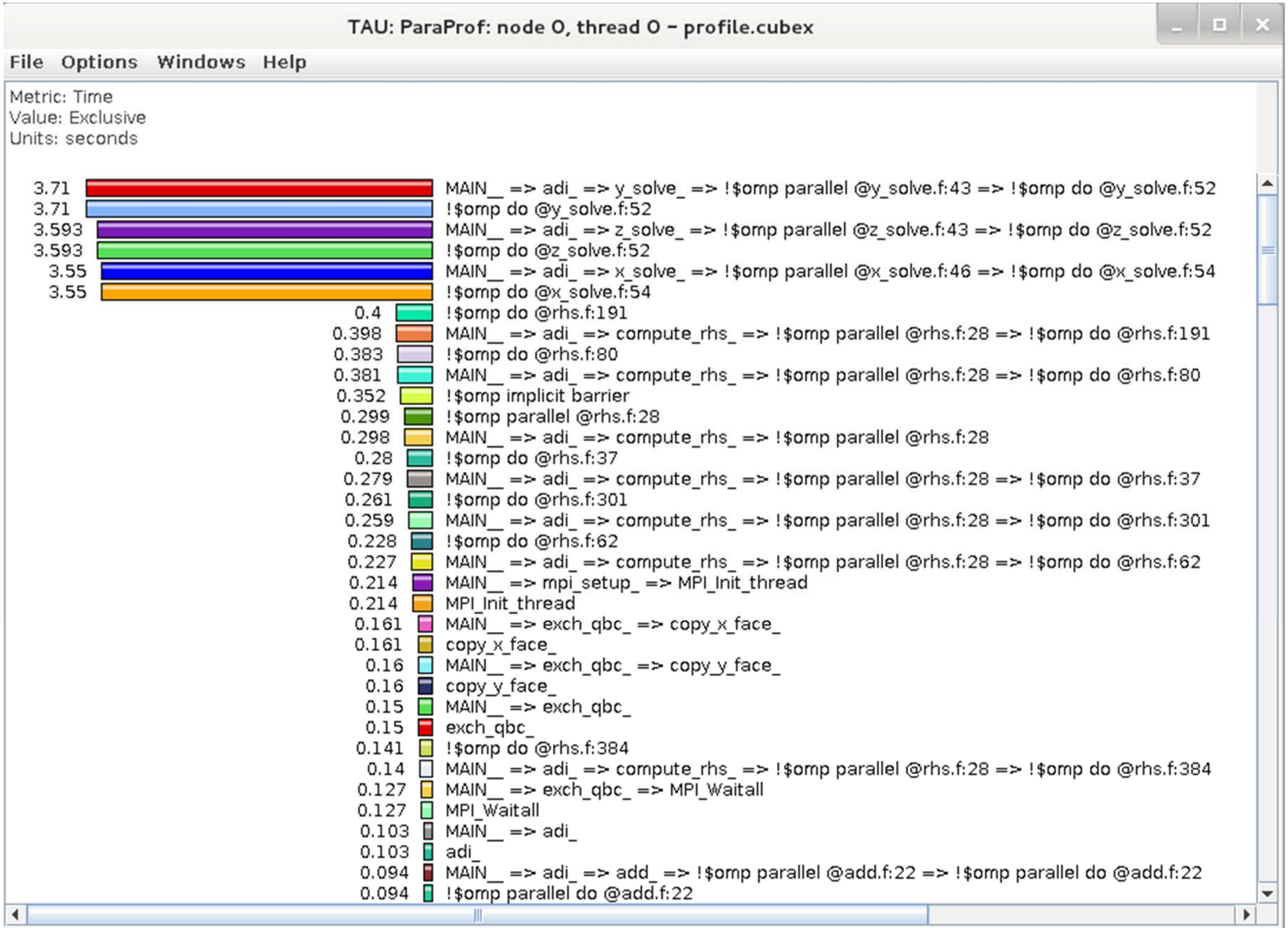
ParaProf: Topology View 3D Torus (IBM BG/P)



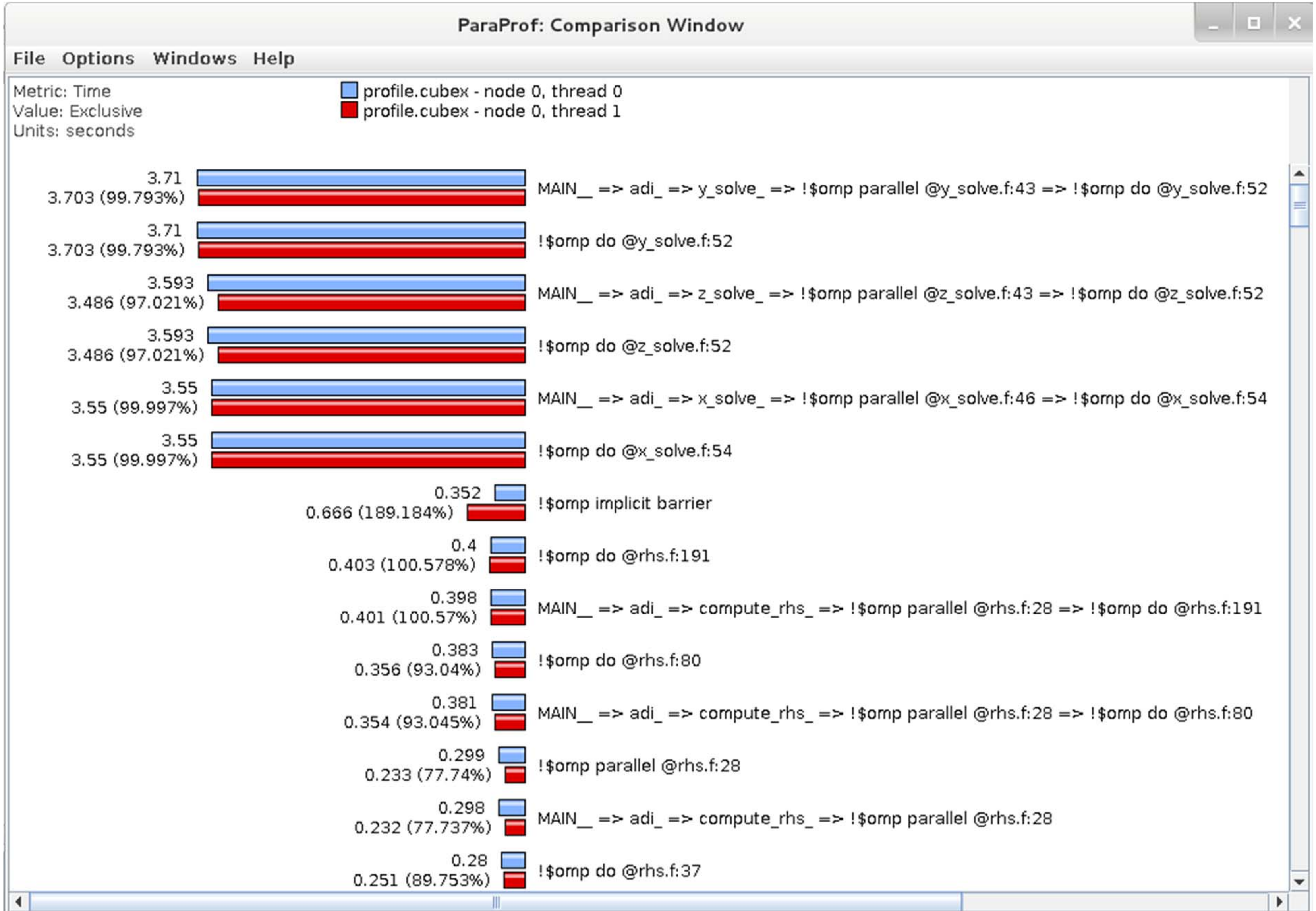
ParaProf:Topology View (6D Torus Coordinates BG/Q)



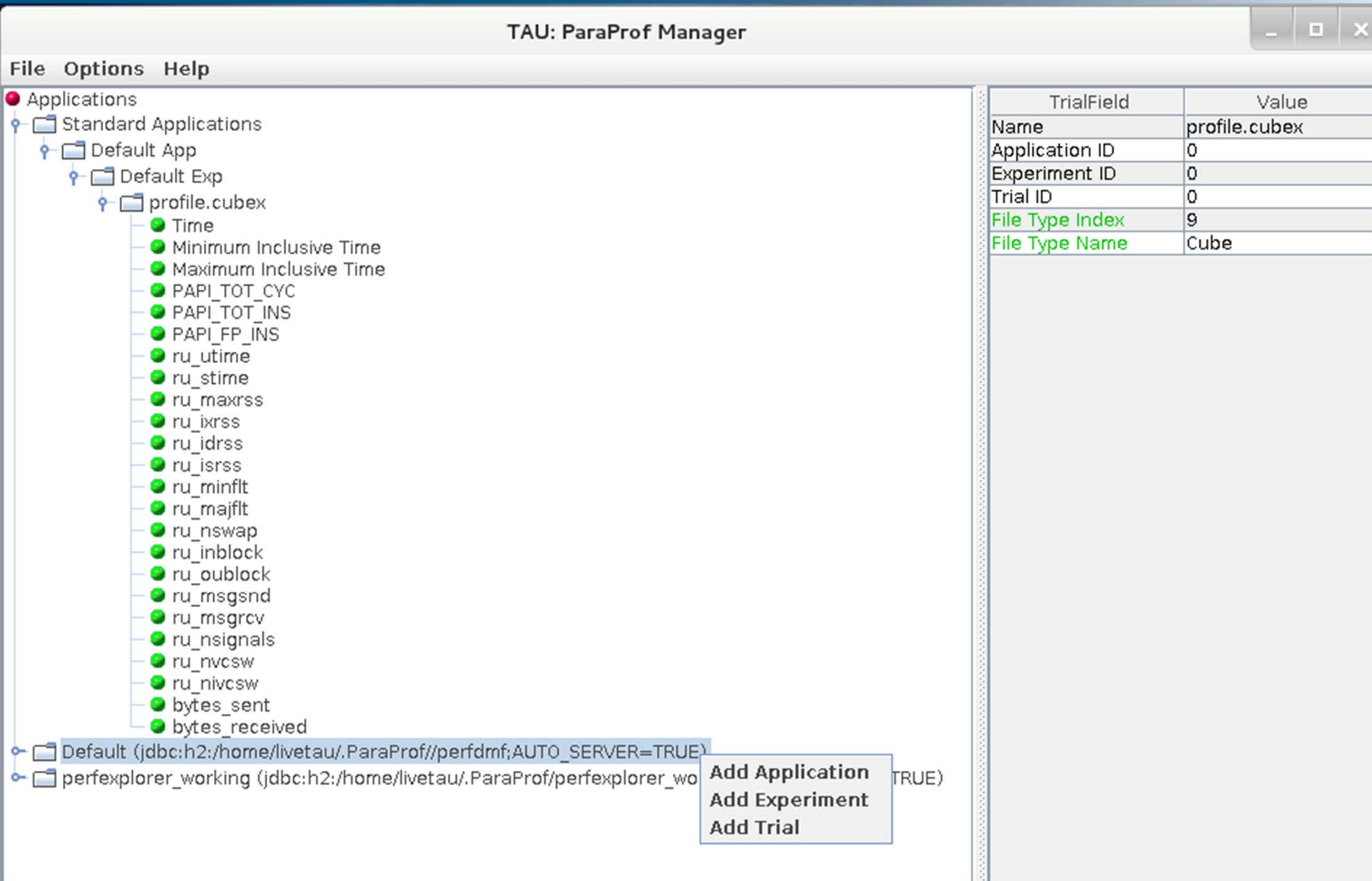
ParaProf: Node View



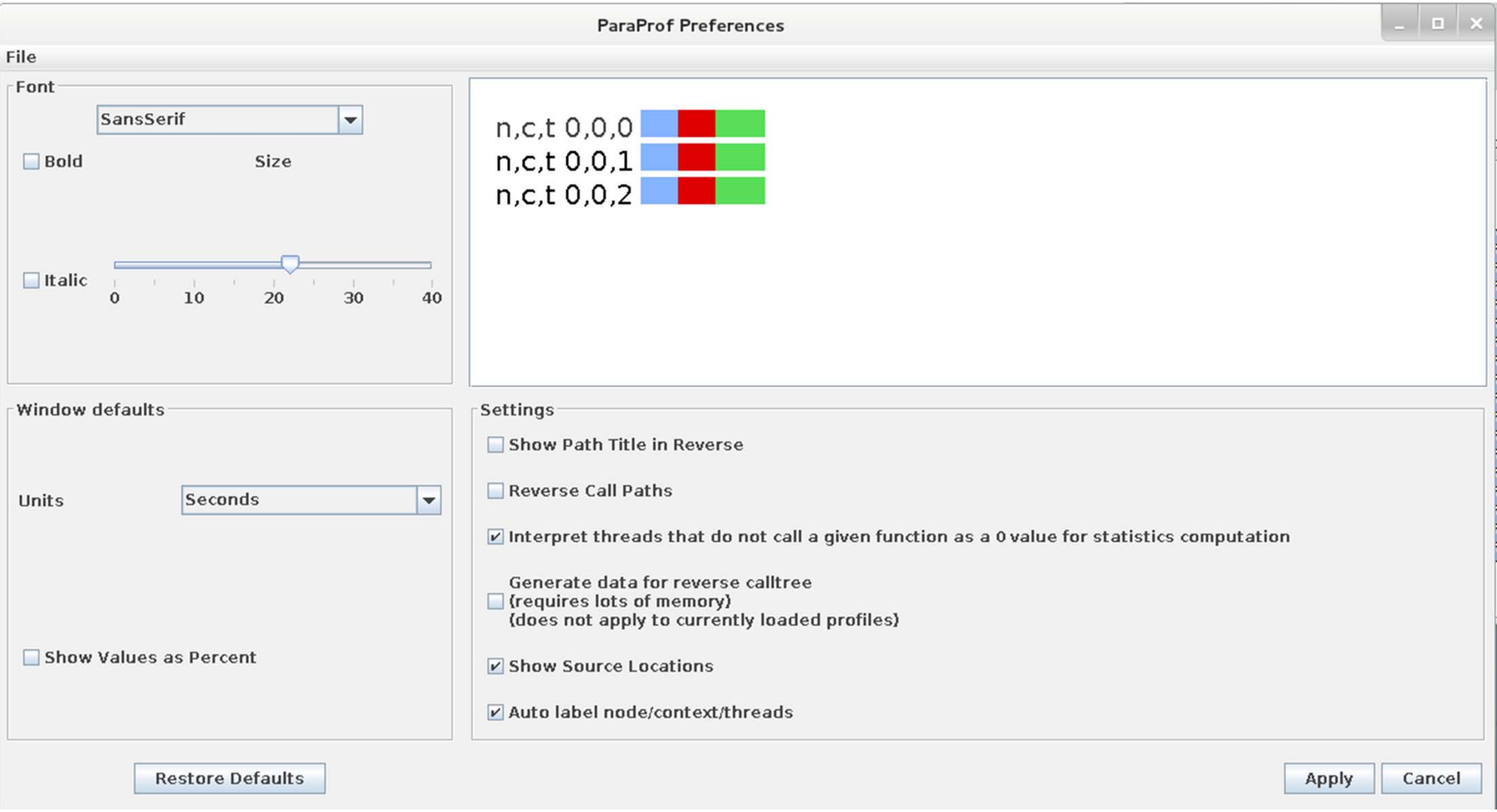
ParaProf: Add Thread to Comparison Window



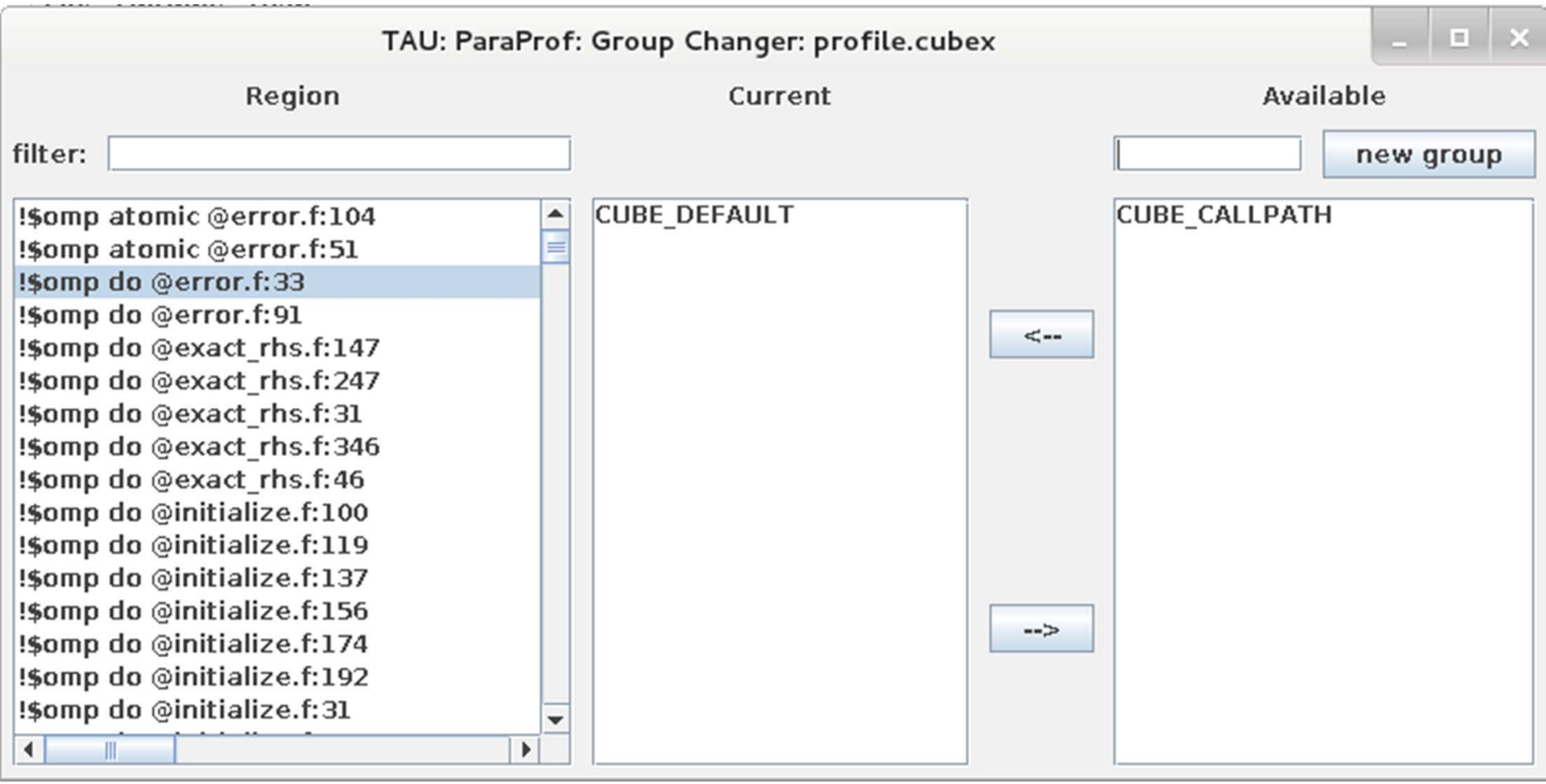
ParaProf: Score-P Profile Files, Database



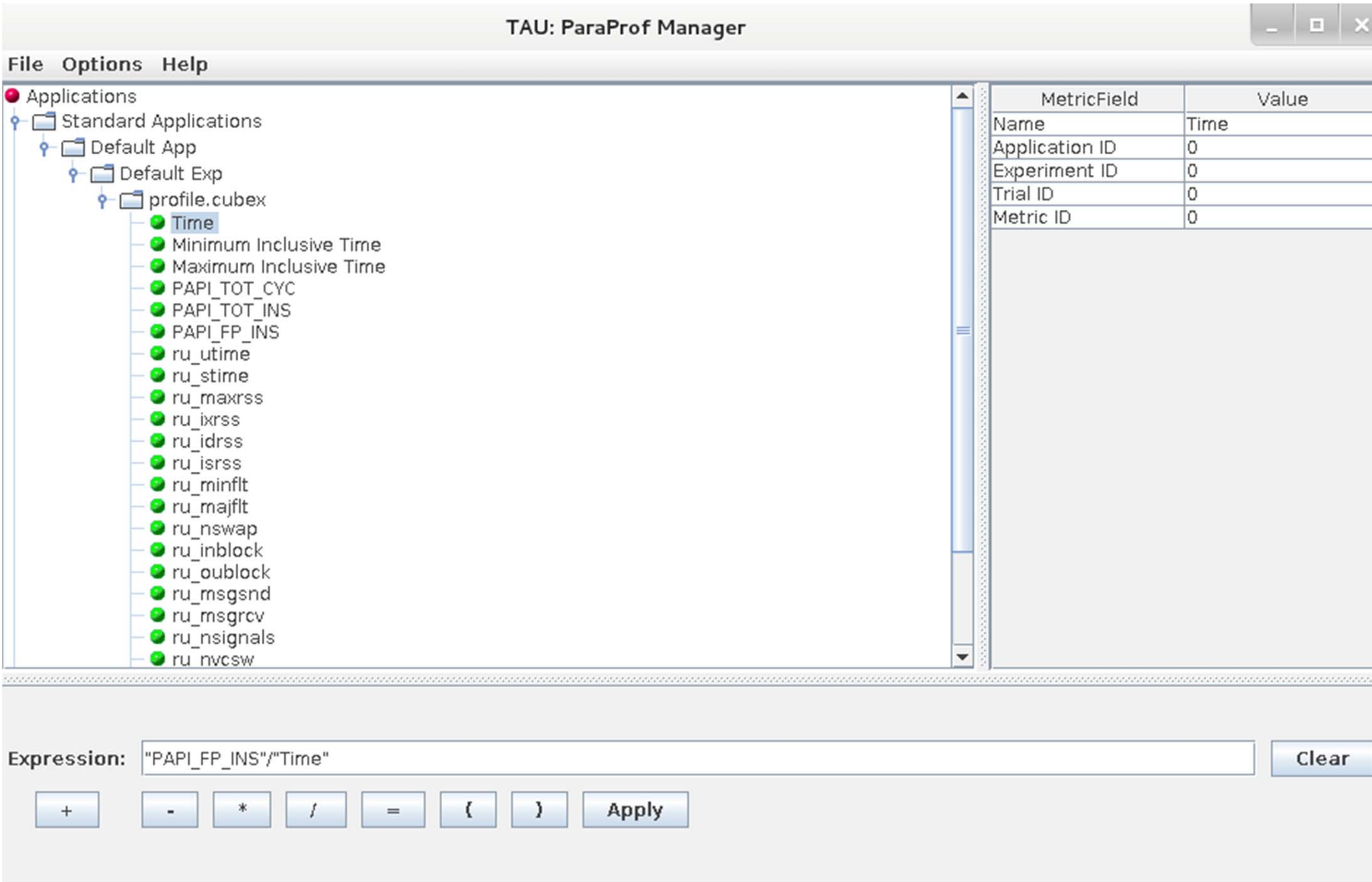
ParaProf: File -> Preferences



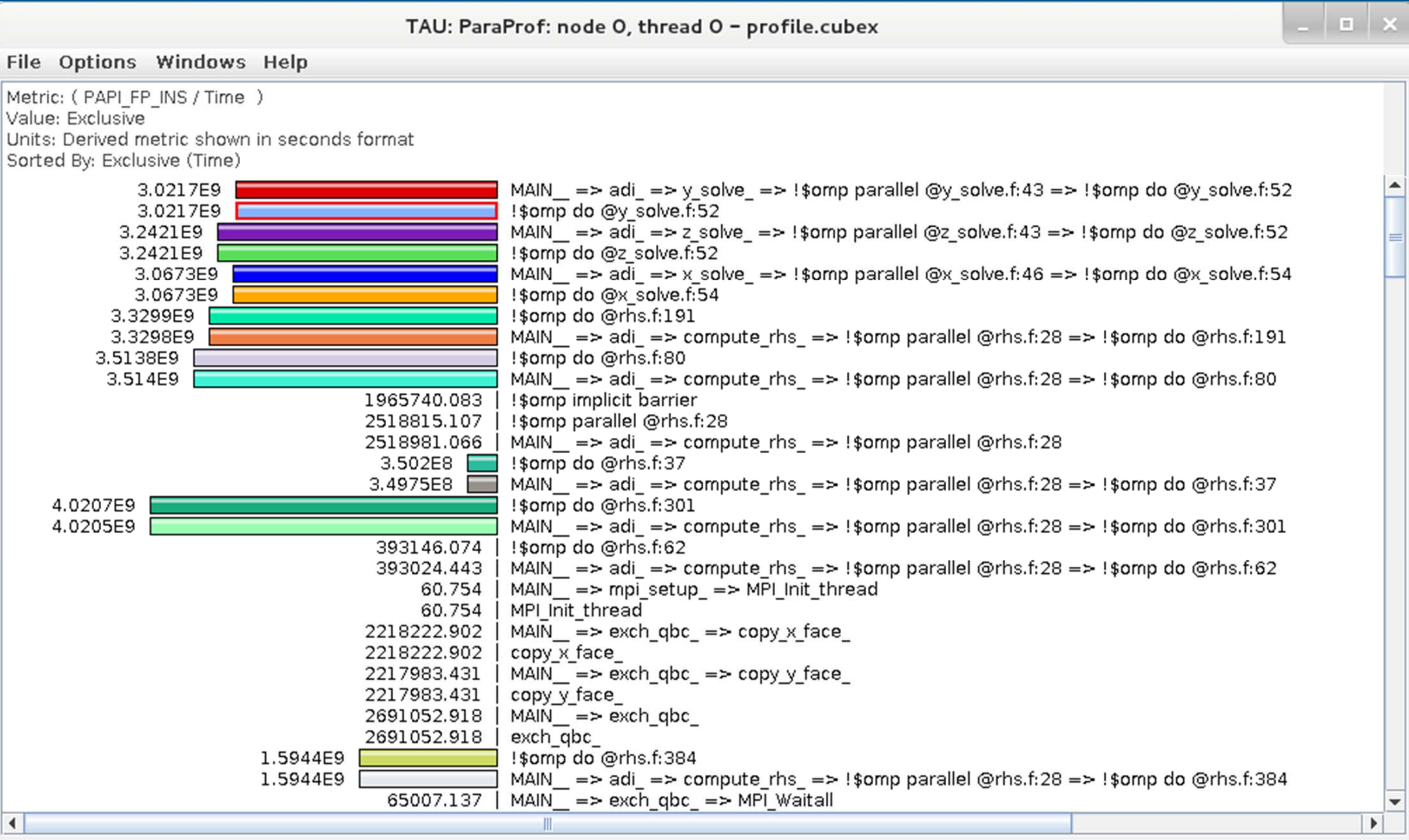
ParaProf: Group Changer Window



ParaProf: Options -> Derived Metric Panel



Sorting Derived Flops Metric by Exclusive Time



Support Acknowledgments

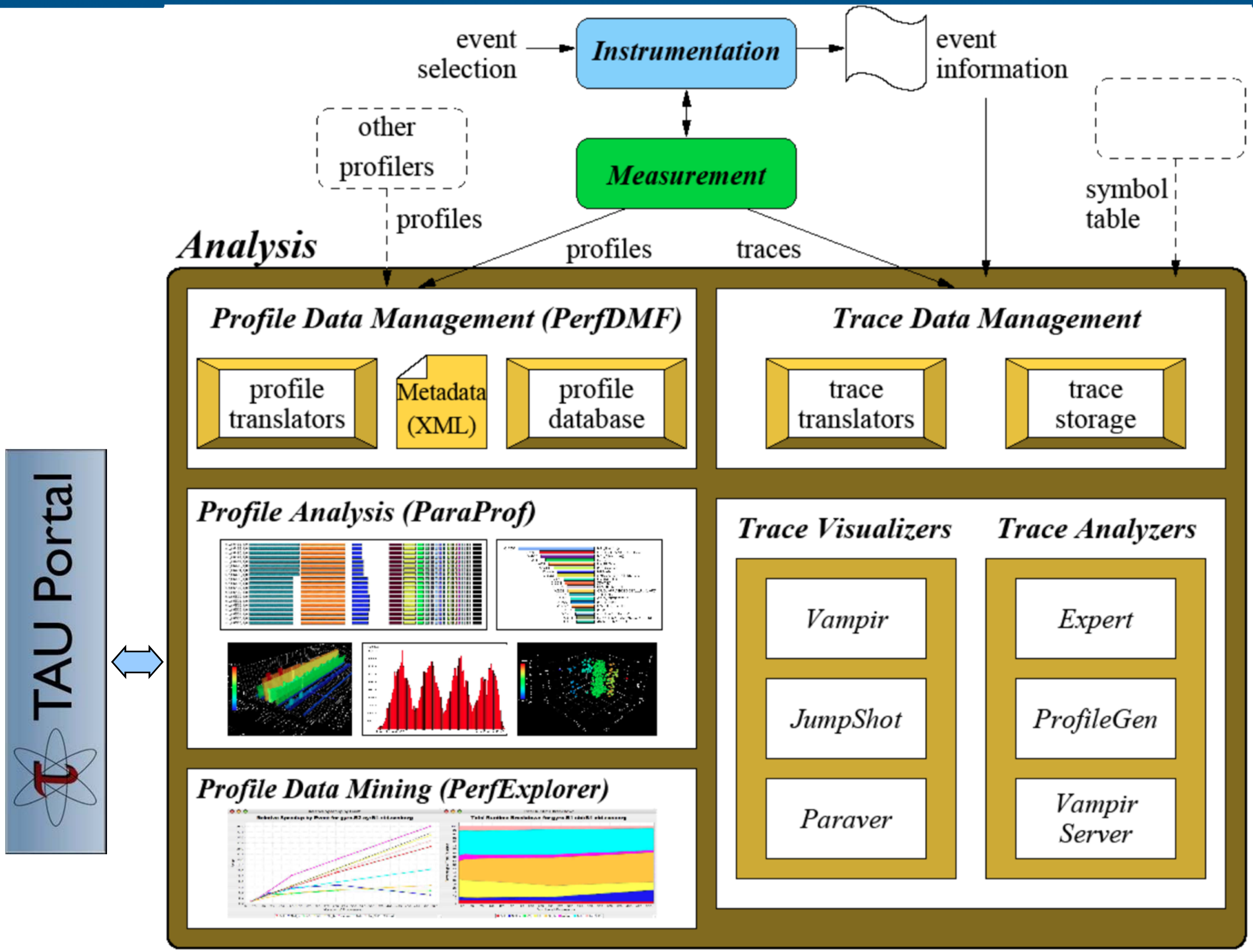
- U.S. Department of Energy (DOE)
  - Office of Science
  - ASC/NNSA, Tri-labs (LLNL,LANL, SNL)
- U.S. Department of Defense (DoD)
  - HPC Modernization Office (HPCMO)
- NSF Software Development for Cyberinfrastructure (SDCI)
- Juelich Supercomputing Center, NIC
- Argonne National Laboratory
- Technical University Dresden
- ParaTools, Inc.
- NVIDIA

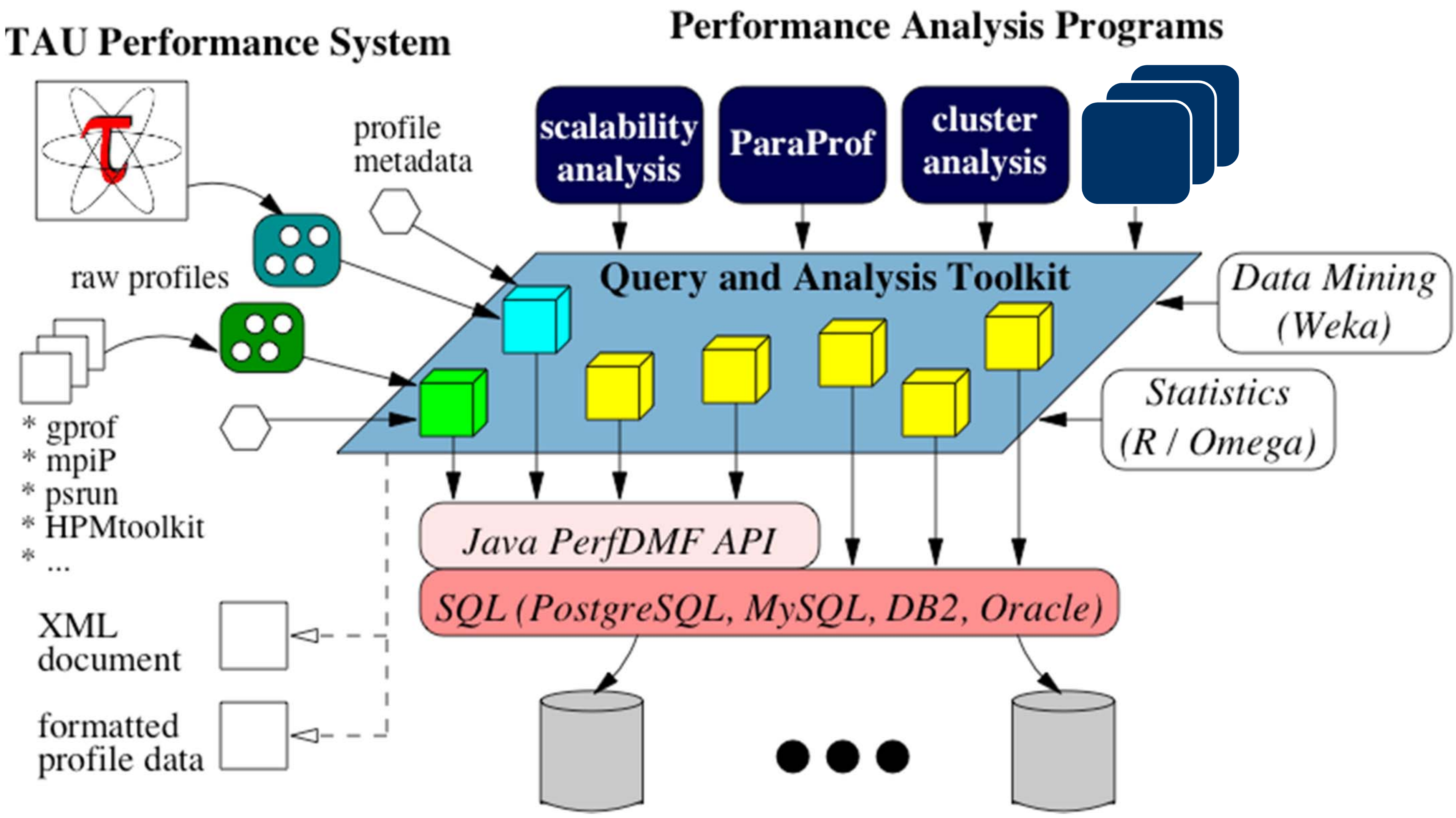
Profile Data Mining with PerfExplorer

Tuning and Analysis Utilities

Sameer Shende
Performance Reseaerch Lab, University of Oregon
http://TAU.uoregon.edu

TAU Analysis





TAU Performance System
Performance Analysis Programs
profile metadata
scalability analysis
ParaProf
cluster analysis
raw profiles
Query and Analysis Toolkit
Data Mining (Weka)
Statistics (R / Omega)
* gprof
* mpiP
* psrun
* HPMtoolkit
* ...
Java PerfDMF API
SQL (PostgreSQL, MySQL, DB2, Oracle)
XML document
formatted profile data

Using TAUdb

- Configure TAUdb (Done by each user)

  % taudb_configure --create-default

    - Choose derby, PostgreSQL, MySQL, Oracle or DB2
    - Hostname
    - Username
    - Password
    - Say yes to downloading required drivers (we are not allowed to distribute these)
    - Stores parameters in your ~/.ParaProf/taudb.cfg file
- Configure PerfExplorer (Done by each user)

  % perfexplorer_configure
- Execute PerfExplorer

  % perfexplorer

Using PerfExplorer

```
% wget http://tau.uoregon.edu/data.tgz  (Contains CUBE profiles from Score-P)
% taudb_configure --create-default
(Chooses derby, blank user/passwd, yes to save passwd, defaults)
% perfexplorer_configure
(Yes to load schema, defaults)
% paraprof
(load each trial: DB -> Add Trial -> Type (Paraprof Packed Profile) -> OK) OR use
    taudb_loadtrial –a “app” –x “experiment” –n “name” file.ppk
Then,
% perfexplorer
(Select experiment, Menu: Charts -> Speedup)
```

TAUdb and the TAU Portal

- Development of the TAU portal
  - Common repository for collaborative data sharing
  - Profile uploading, downloading, user management
  - Paraprof, PerfExplorer can be launched from the portal using Java Web Start (no TAU installation required)
- Portal URL

  http://tau.nic.uoregon.edu

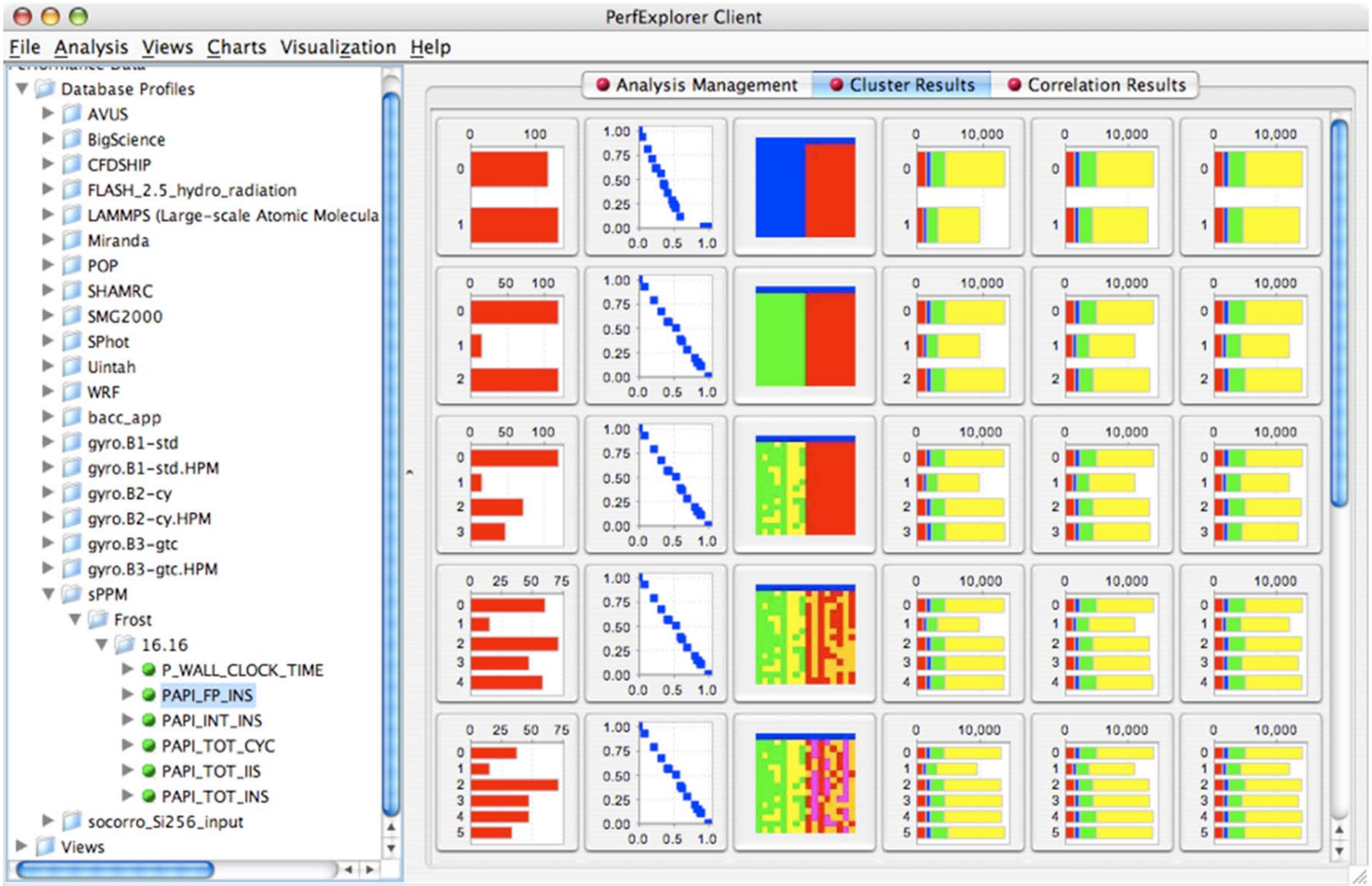
Performance Data Mining (PerfExplorer)

- Performance knowledge discovery framework
  - Data mining analysis applied to parallel performance data
    - comparative, clustering, correlation, dimension reduction, …
  - Use the existing TAU infrastructure
    - TAU performance profiles, taudb
  - Client-server based system architecture
- Technology integration
  - Java API and toolkit for portability
  - taudb
  - R-project/Omegahat, Octave/Matlab statistical analysis
  - WEKA data mining package
  - JFreeChart for visualization, vector output (EPS, SVG)

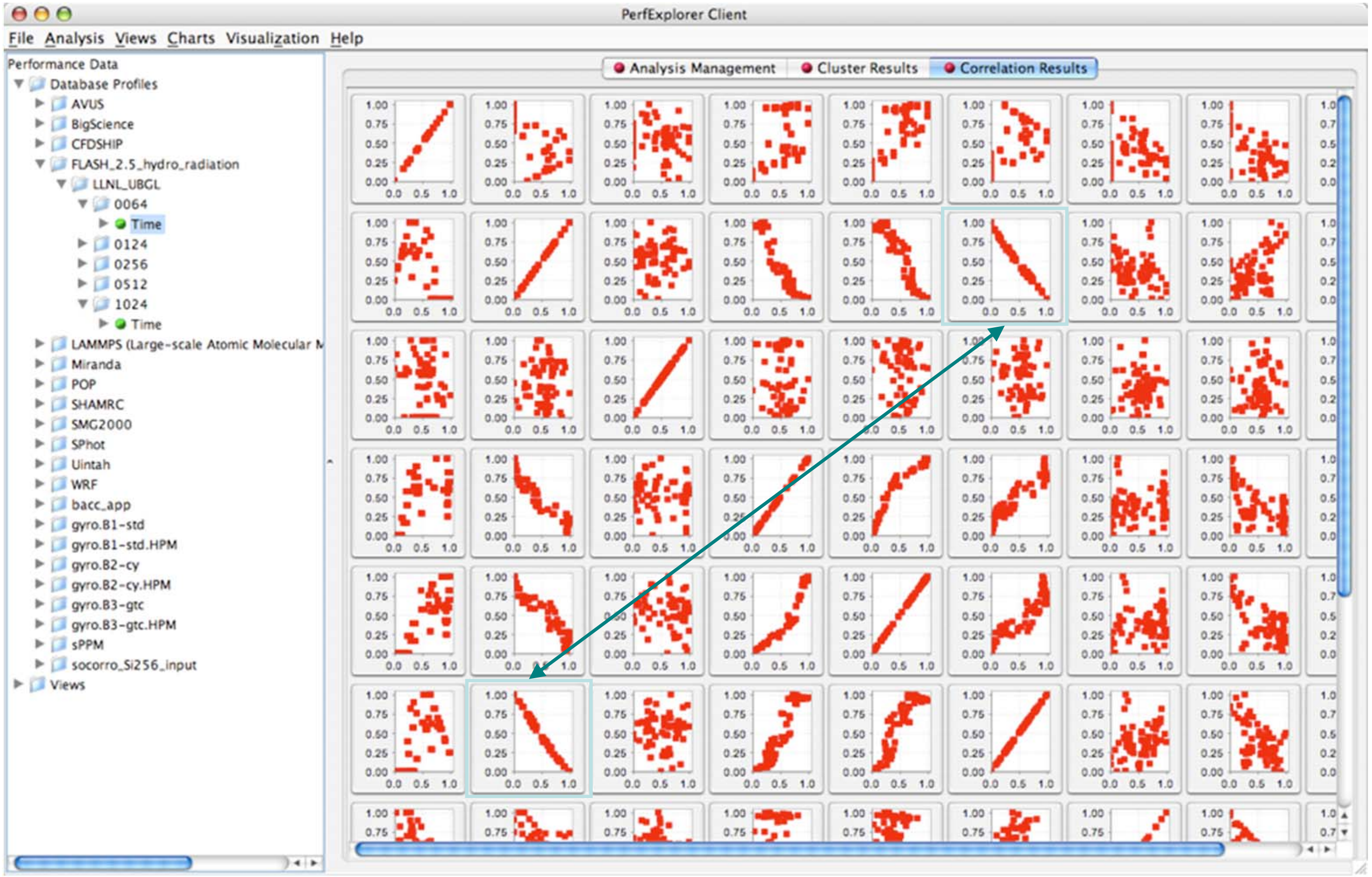
PerfExplorer - Cluster Analysis

- Performance data represented as vectors - each dimension is the cumulative time for an event
- *k*-means: *k* random centers are selected and instances are grouped with the "closest" (Euclidean) center
- New centers are calculated and the process repeated until stabilization or max iterations
- Dimension reduction necessary for meaningful results
- Virtual topology, summaries constructed

PerfExplorer - Cluster Analysis (sPPM)

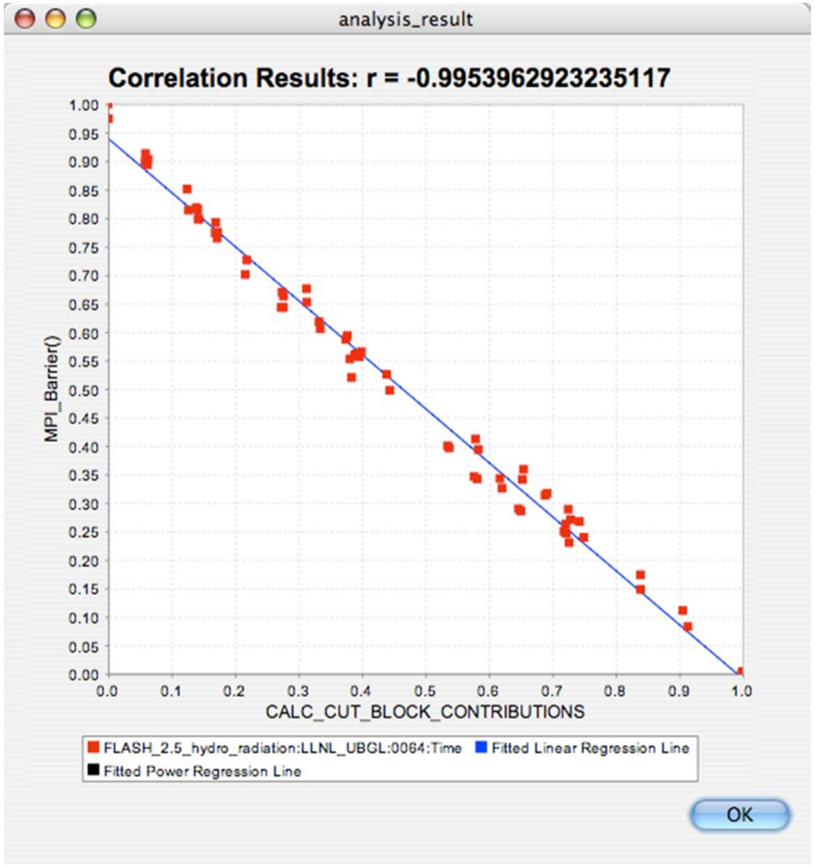


- Describes strength and direction of a linear relationship between two variables (events) in the data



PerfExplorer - Correlation Analysis (Flash)

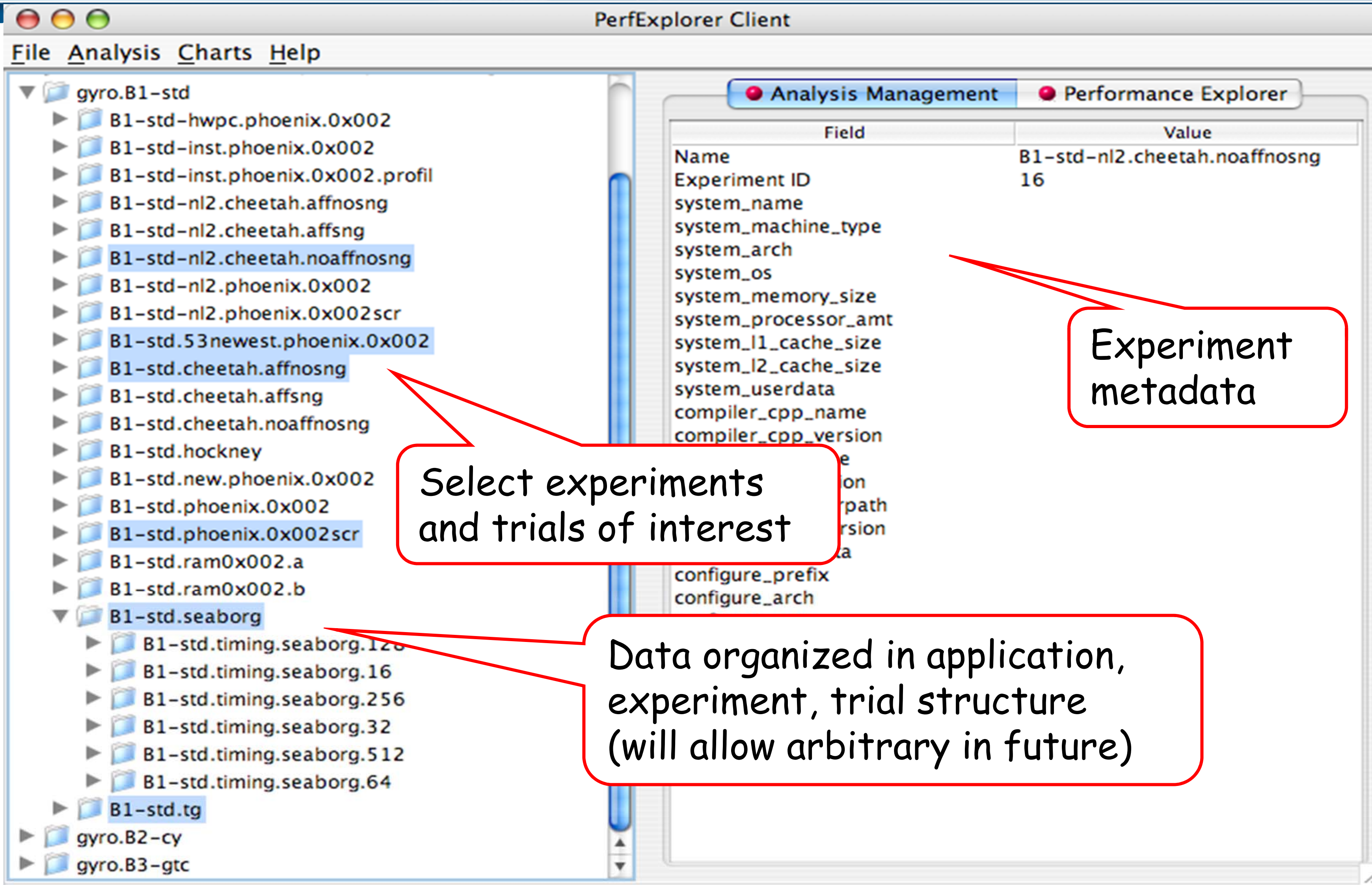
- -0.995 indicates strong, negative relationship
- As CALC_CUT_ BLOCK_CONTRIBUTIONS() increases in execution time, MPI_Barrier() decreases



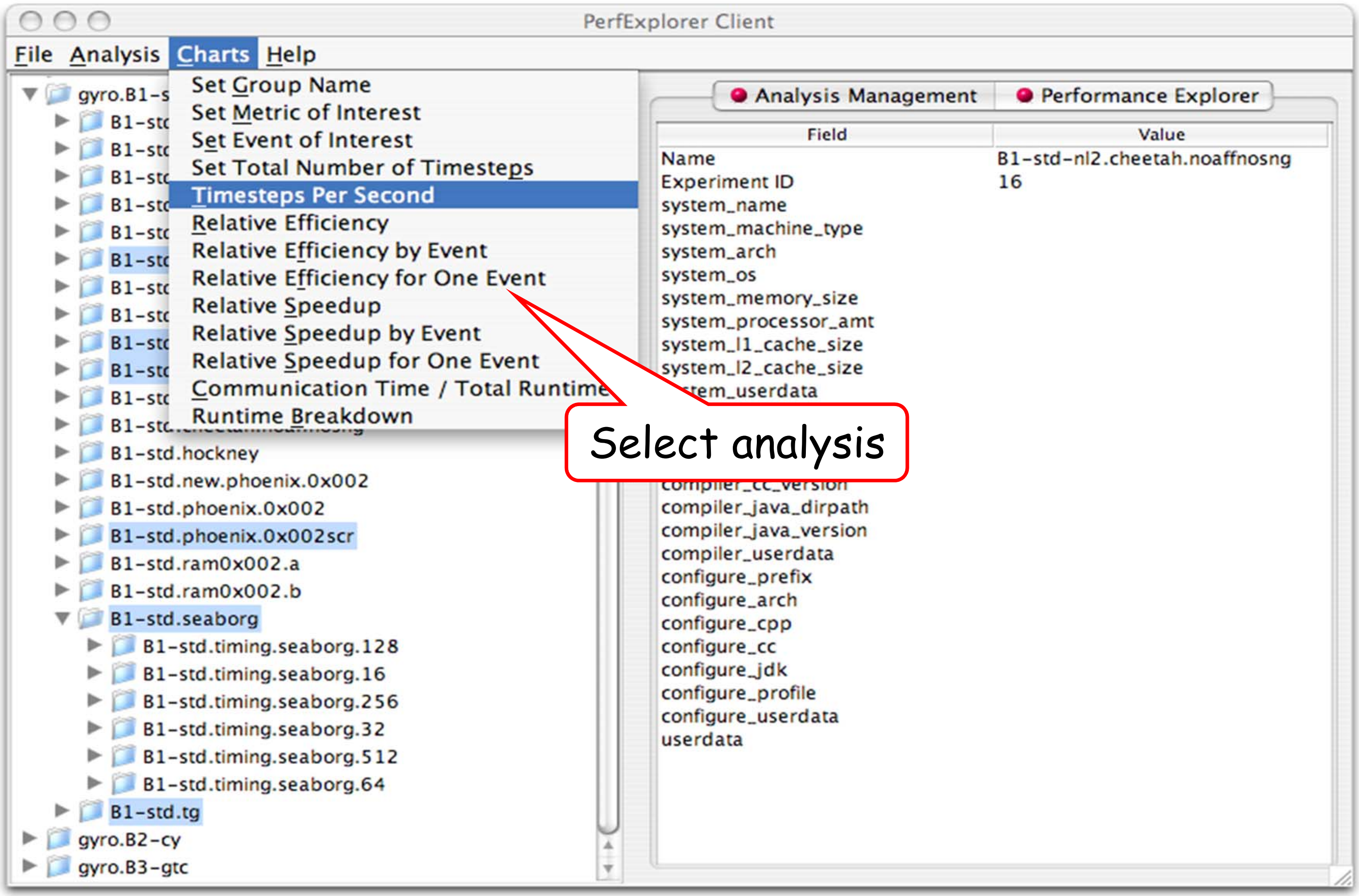
PerfExplorer - Comparative Analysis

- Relative speedup, efficiency
  - total runtime, by event, one event, by phase
- Breakdown of total runtime
- Group fraction of total runtime
- Correlating events to total runtime
- Timesteps per second

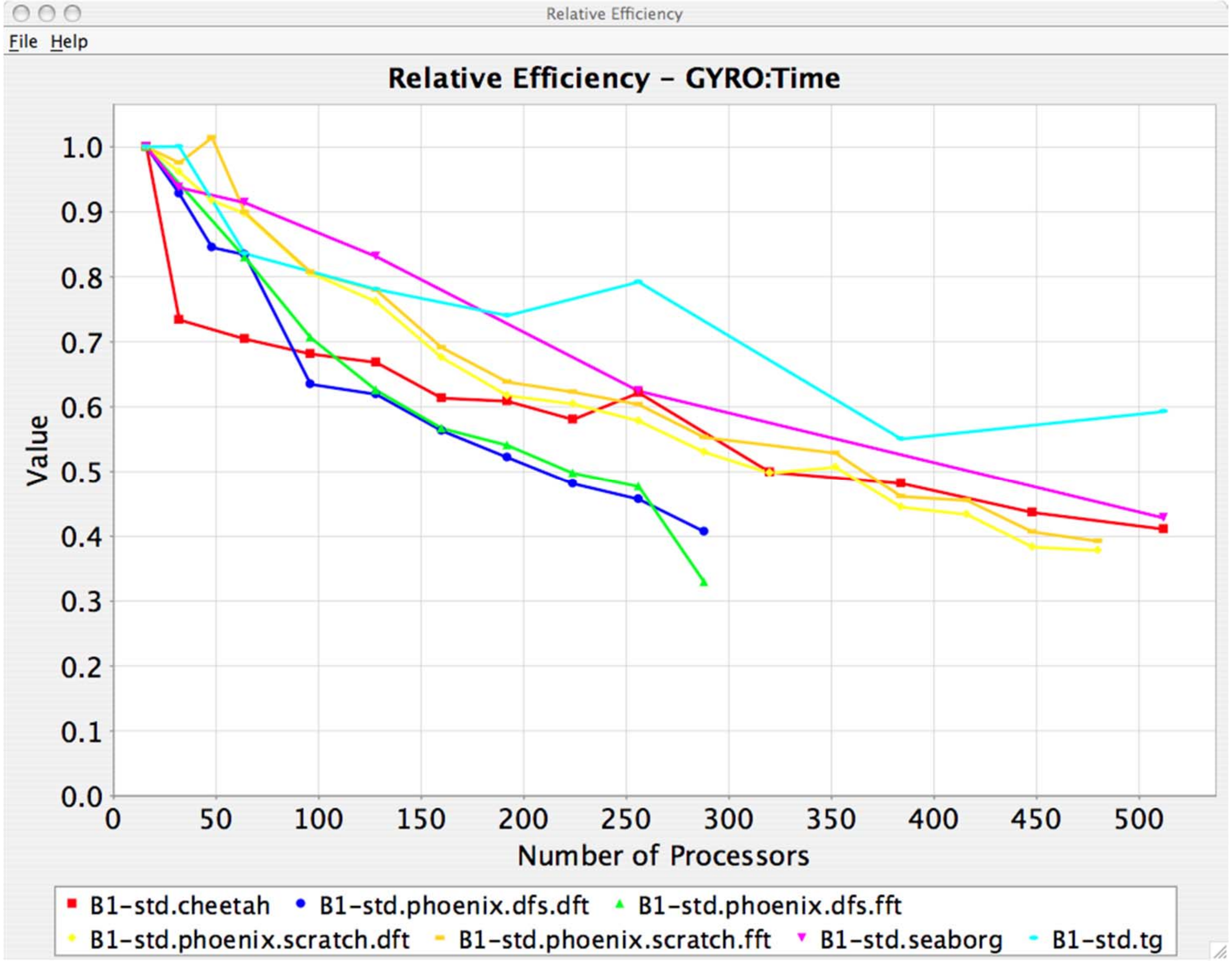
PerfExplorer - Interface



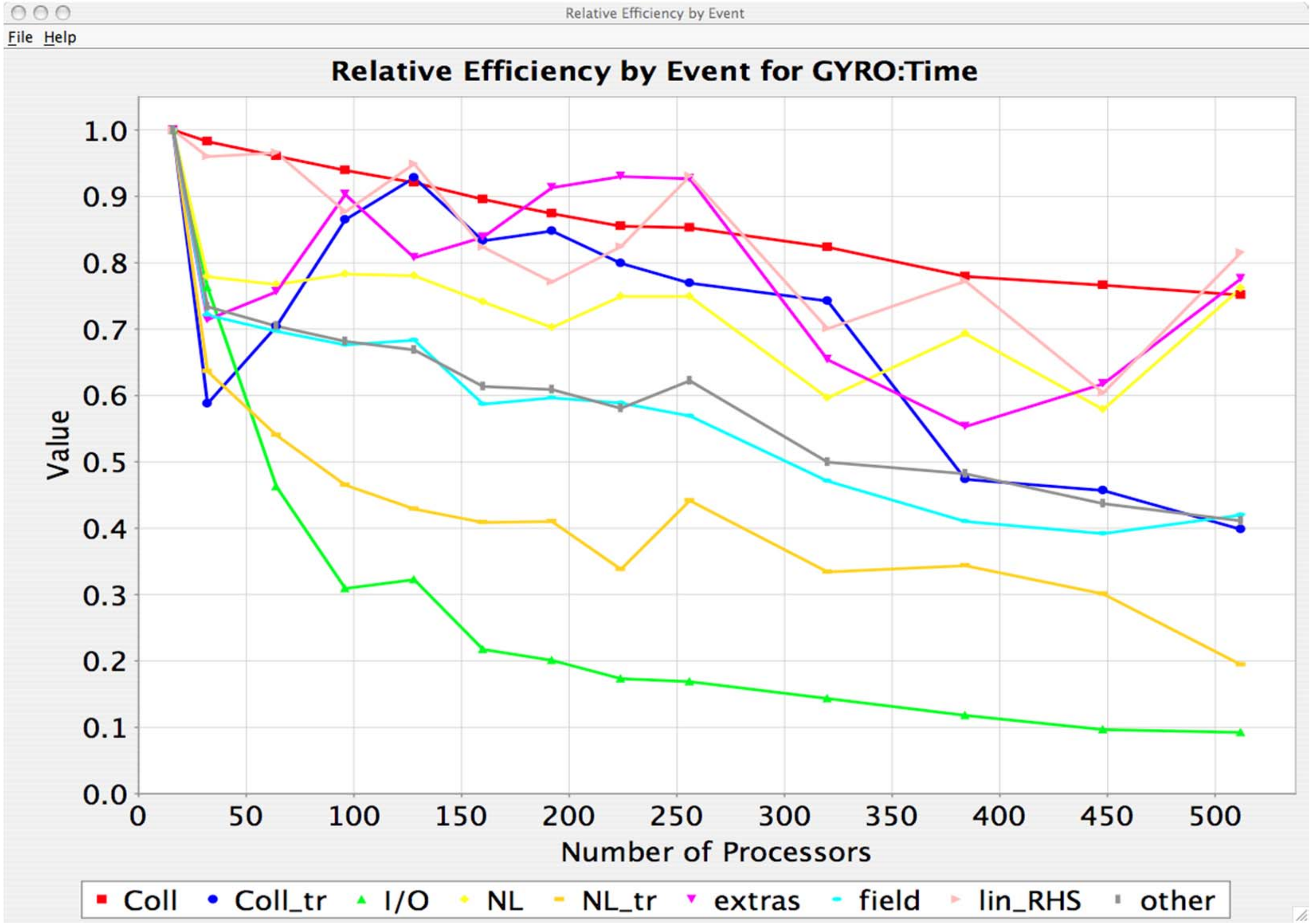
PerfExplorer - Interface



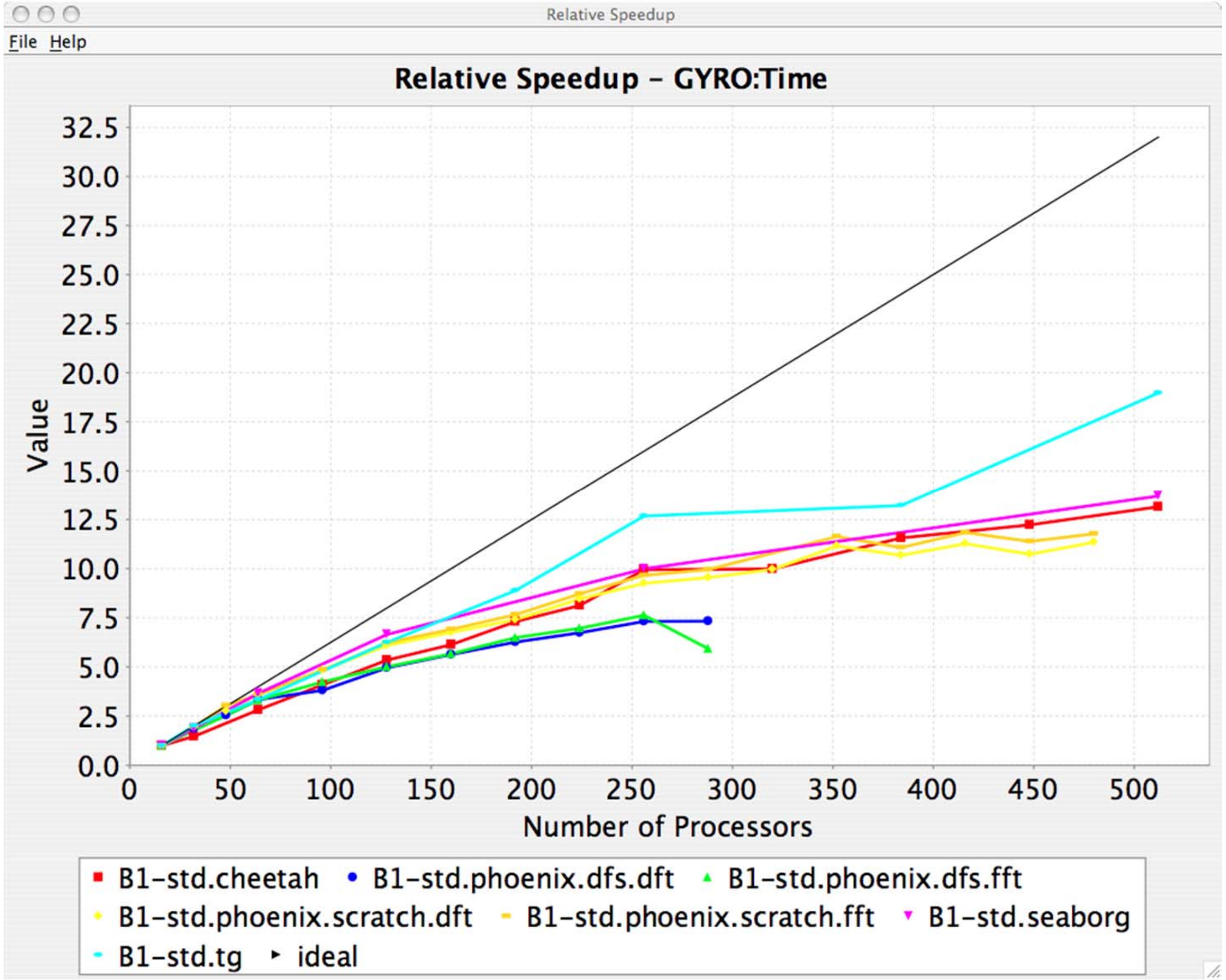
PerfExplorer - Relative Efficiency Plots



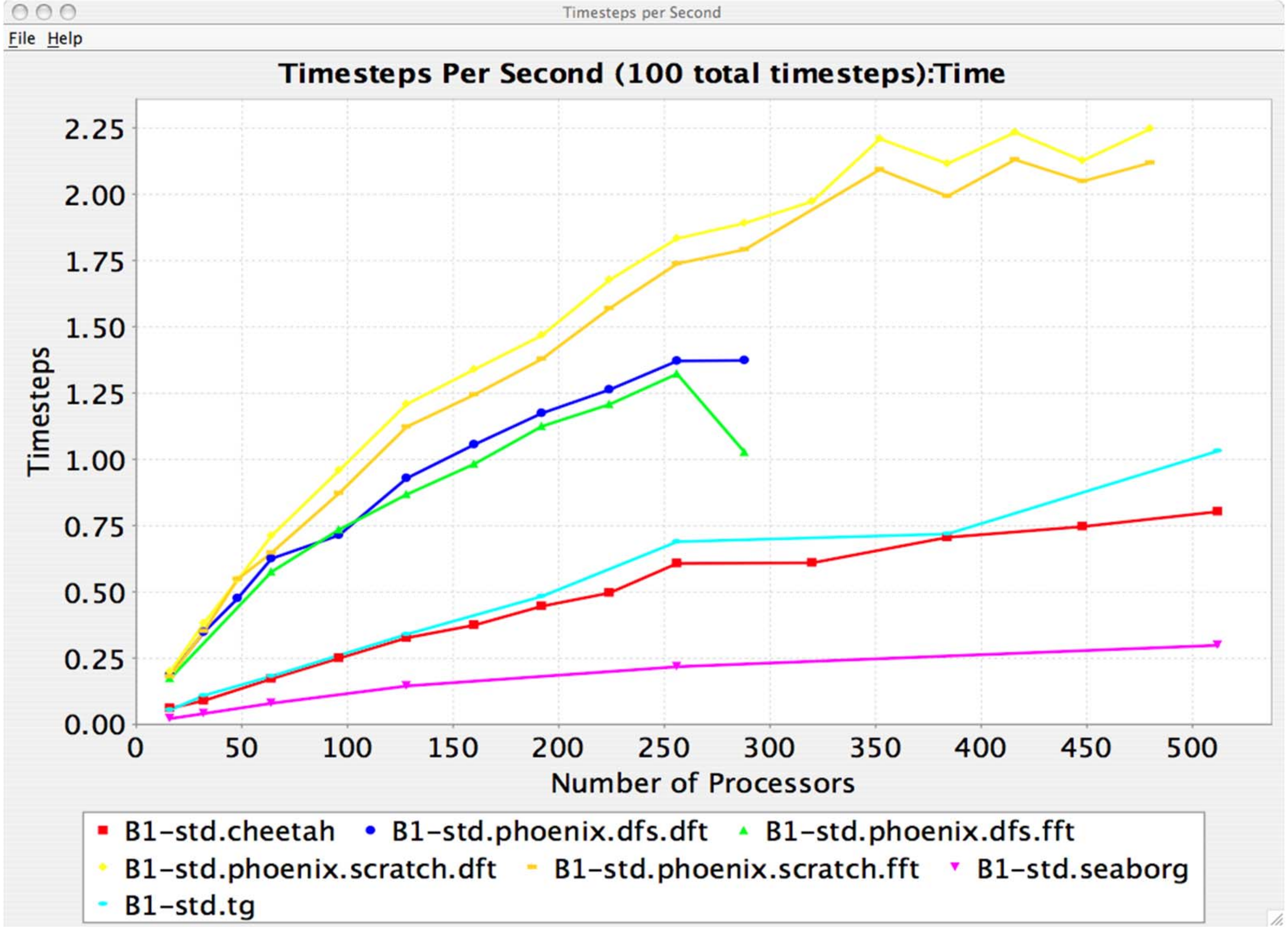
PerfExplorer - Relative Efficiency by Routine



PerfExplorer - Relative Speedup

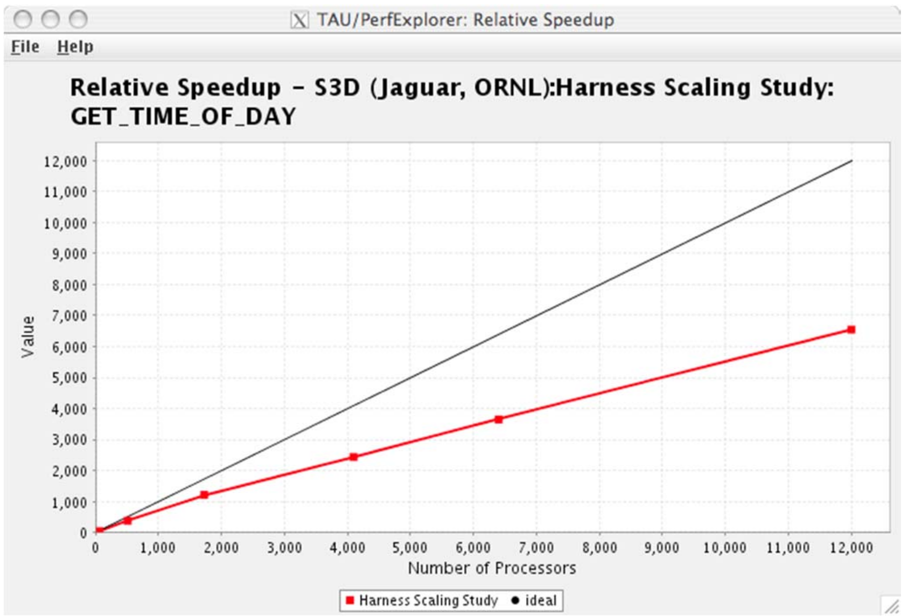


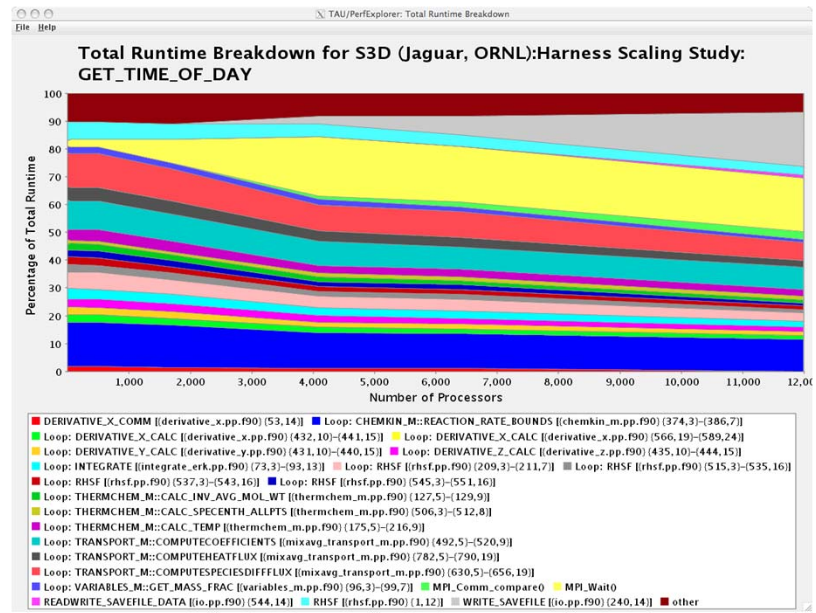
PerfExplorer - Timesteps Per Second



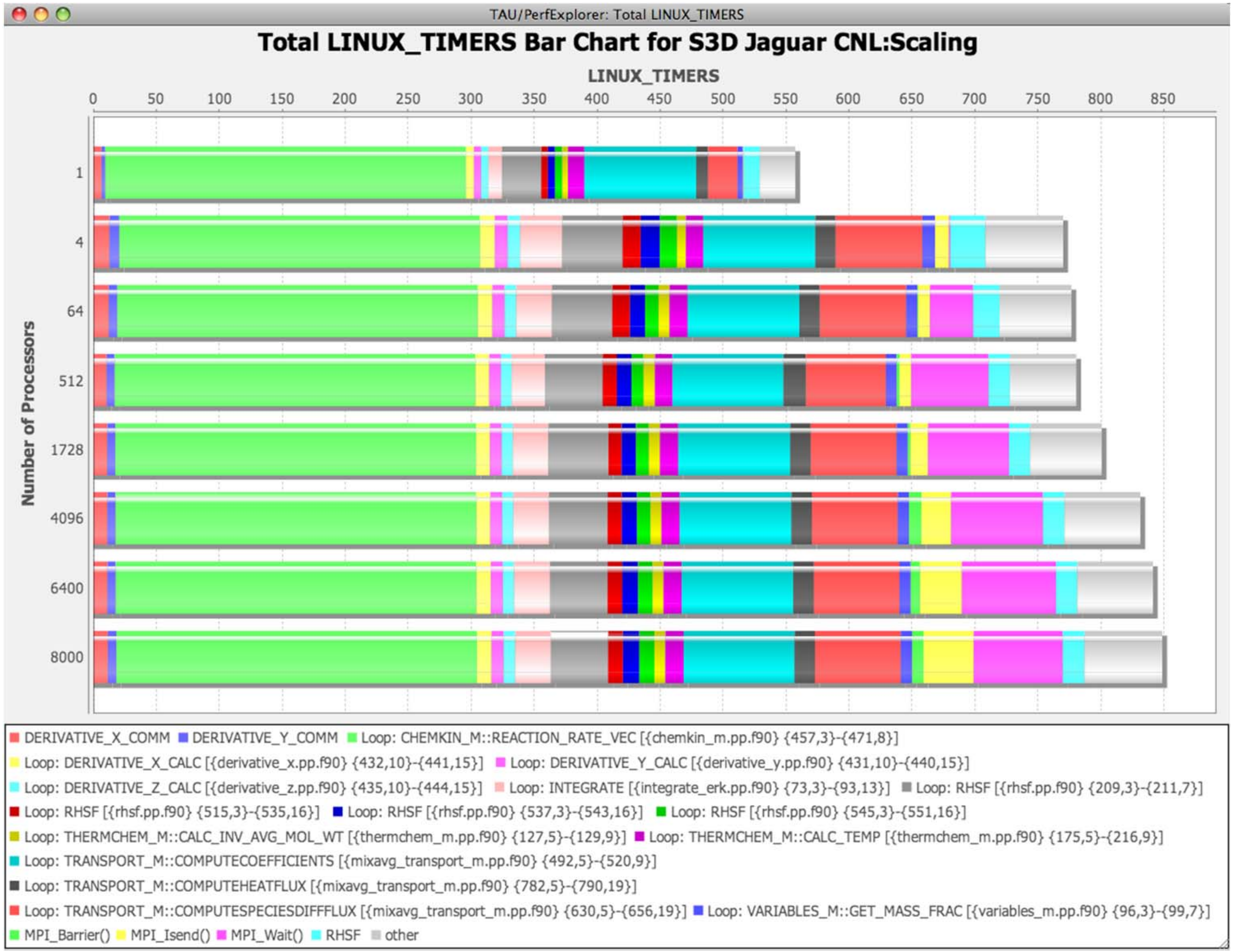
Usage Scenarios: Evaluate Scalability

- Goal: How does my application scale? What bottlenecks occur at what core counts?
- Load profiles in taudb database and examine with PerfExplorer

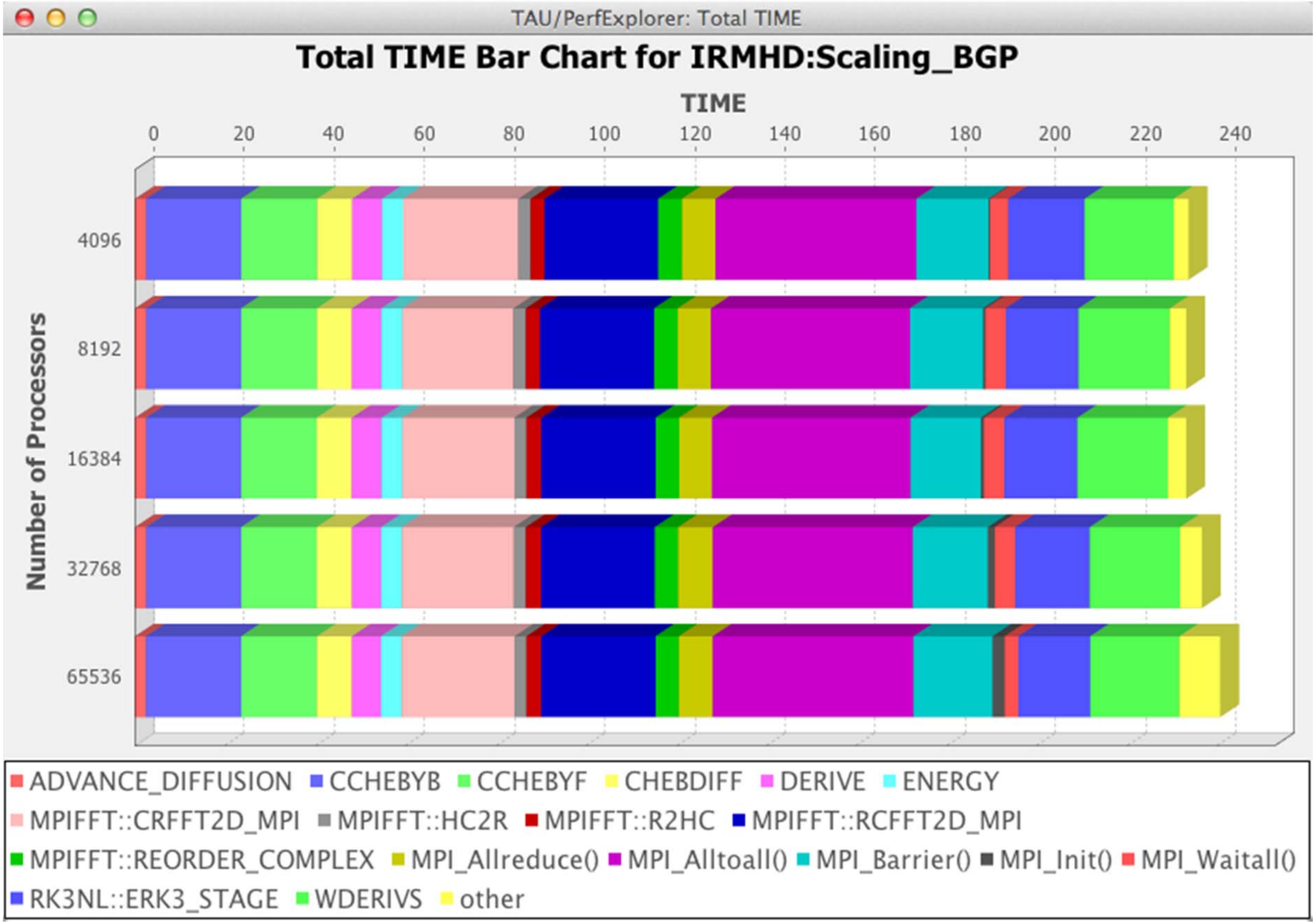




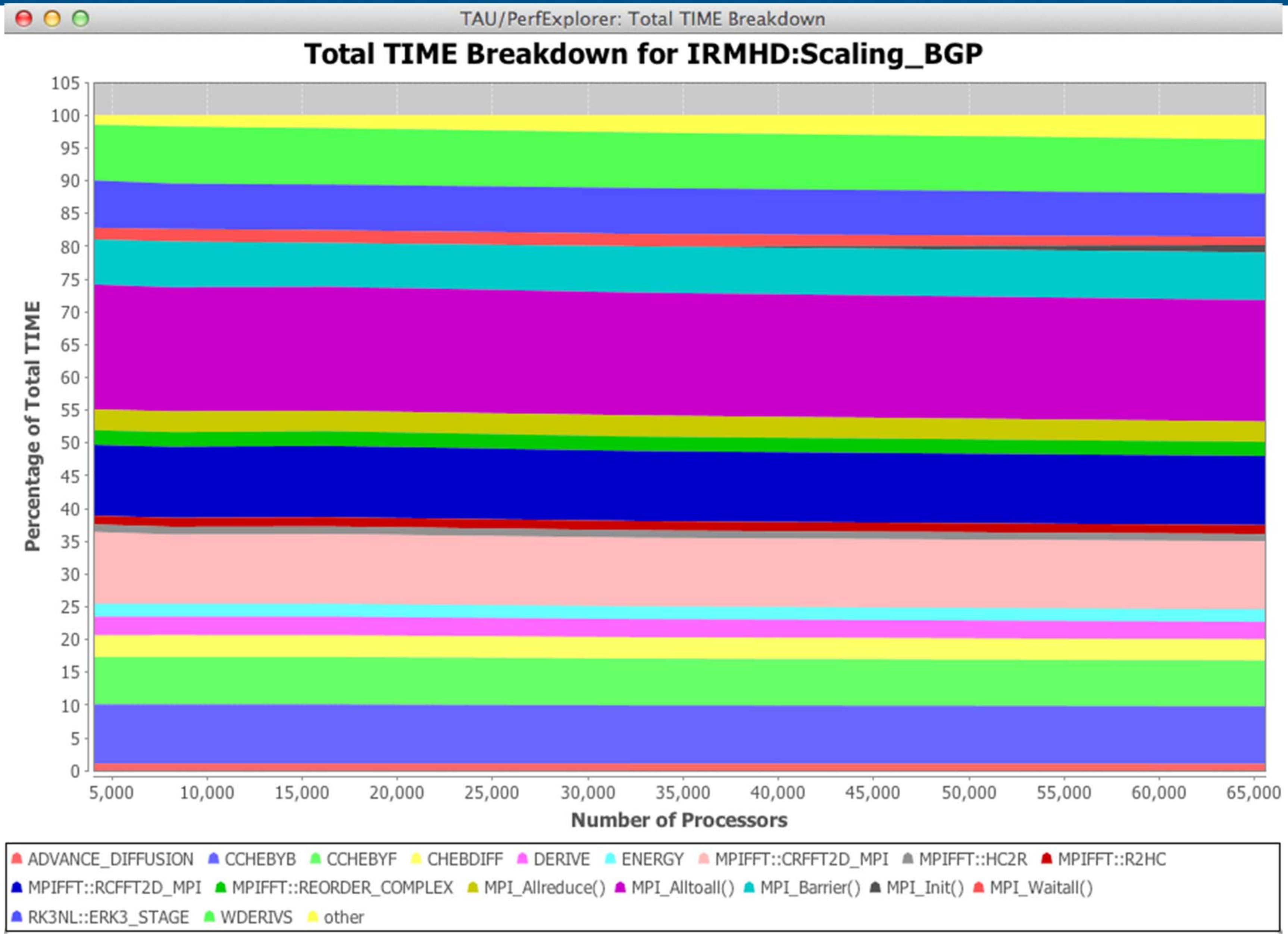
Usage Scenarios: Evaluate Scalability



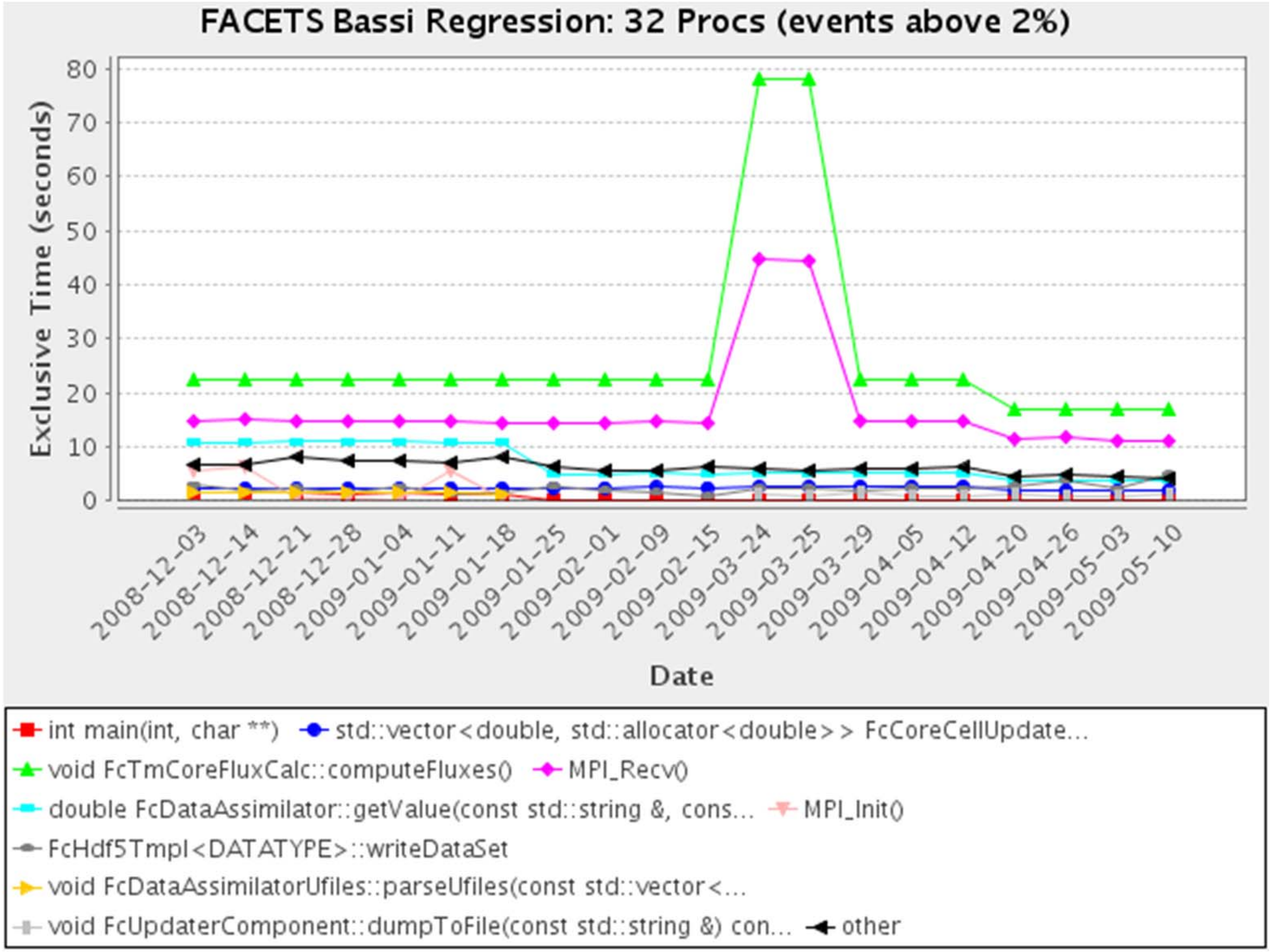
PerfExplorer



PerfExplorer



Performance Regression Testing



Typical performance bottlenecks and how they can be found

Bert Wesarg

ZIH, Technische Universität Dresden

Outline

- **Case I:**
  - **Finding load imbalances in OpenMP codes**

- **Case II:**
  - **Finding communication and computation imbalances in MPI codes**

Case I: Sparse Matrix Vector Multiplication

$$\begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix} = \begin{pmatrix} a_{11} & \cdots & a_{n1} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$$

- Matrix has significant more zero elements => sparse matrix
- Only non-zero elements of a_{ij} are saved efficiently in memory
- Algorithm:

```
foreach row r in A
  y[r.x] = 0
  foreach non-zero element e in row
    y[r.x] += e.value * x[e.y]
```

Case I: Sparse Matrix Vector Multiplication

- Naïve OpenMP Algorithm:

```
#pragma omp parallel for
foreach row r in A
  y[r.x] = 0
  foreach non-zero element e in row
    y[r.x] += e.value * x[e.y]
```

- Distributes the rows of A evenly across the threads in the parallel region
- The distribution of the non-zero elements may influence the load balance in the parallel application

Case I: Load imbalances in OpenMP codes

- Measuring the static OpenMP application

```
% cd ~/Bottlenecks/smxv
% make PREP=scorep
scorep   gcc -fopenmp -DLITTLE_ENDIAN \
    -DFUNCTION_INC='"y_Ax-omp.inc.c"' -DFUNCTION=y_Ax_omp \
    -o smxv-omp smxv.c -lm
scorep   gcc -fopenmp -DLITTLE_ENDIAN \
    -DFUNCTION_INC='"y_Ax-omp-dynamic.inc.c"‘ \
    -DFUNCTION=y_Ax_omp_dynamic -o smxv-omp-dynamic smxv.c -lm
% OMP_NUM_THREADS=8 scan –t ./smxv-omp yax_large.bin
```

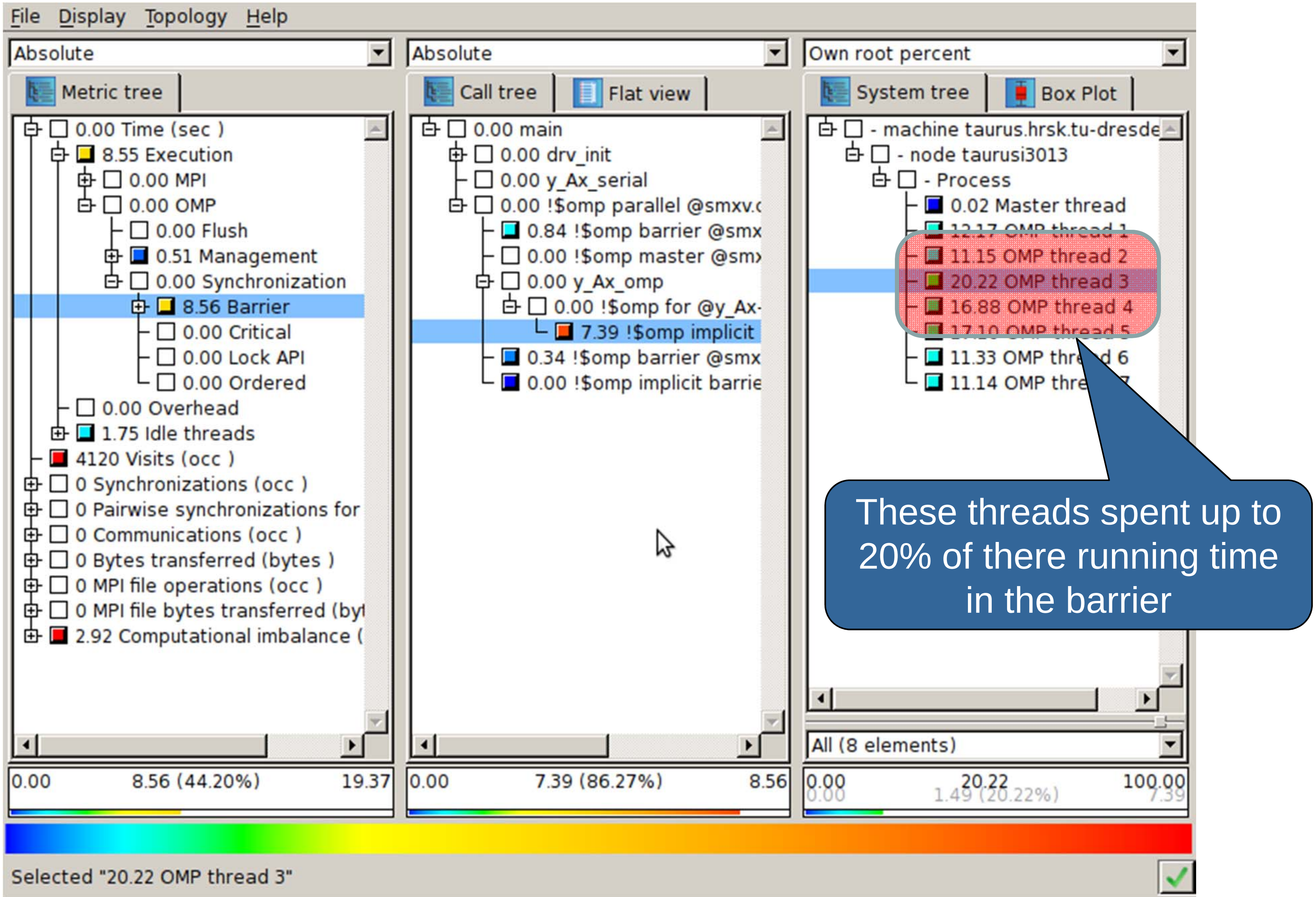
Case I: Load imbalances in OpenMP codes: Profile

- Two metrics which indicate load imbalances:
  - Time spent in OpenMP barriers
  - Computational imbalance

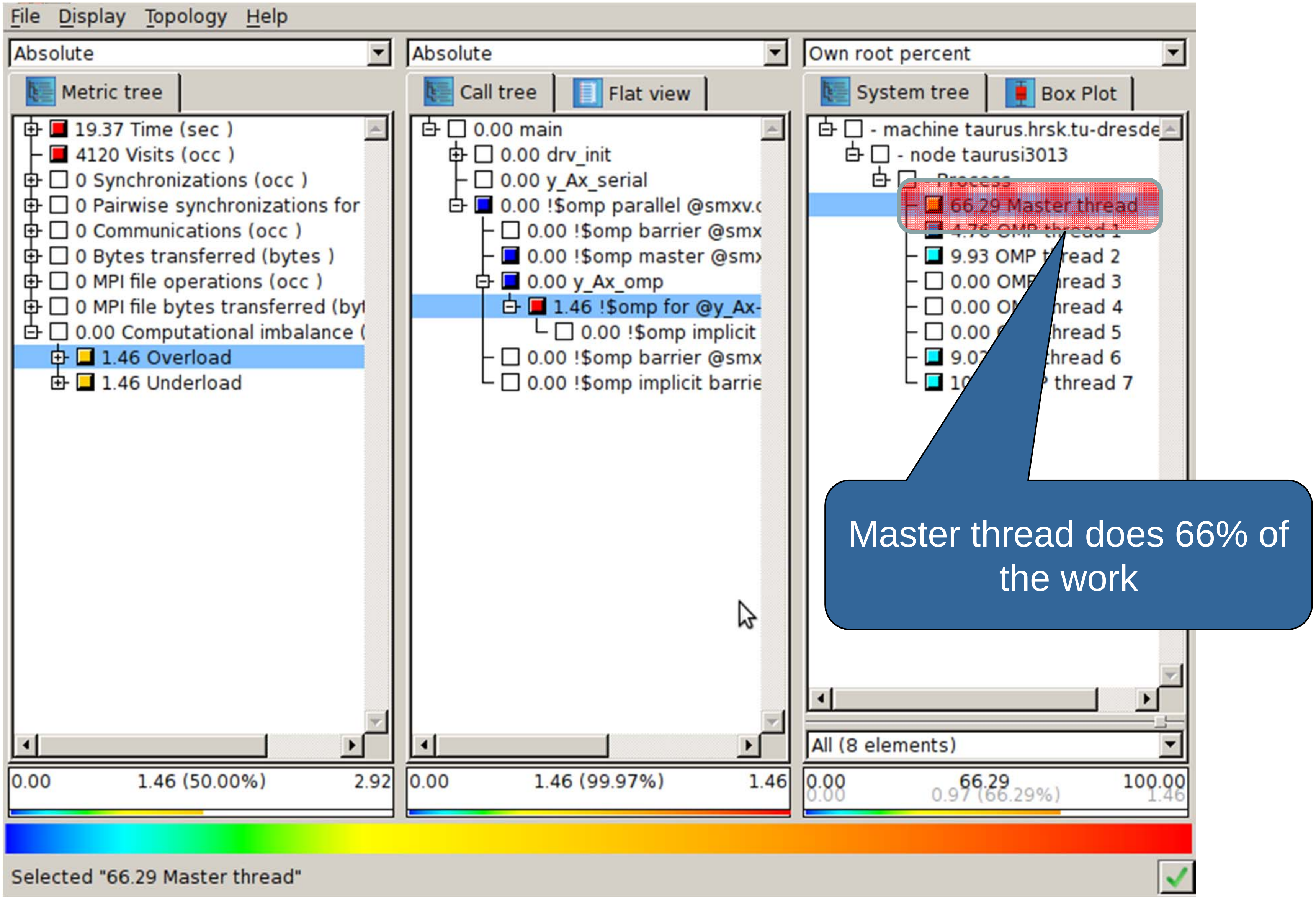
- Open prepared measurement on the LiveDVD with Cube

```
% cube ~/Bottlenecks/smxv/scorep_smxv-omp_large/trace.cubex

                [CUBE GUI showing trace analysis report]
```

Case I: Time spent in OpenMP barriers



Case I: Computational imbalance



- Improved OpenMP Algorithm

```
#pragma omp parallel for schedule(dynamic,1000)
foreach row r in A
  y[r.x] = 0
  foreach non-zero element e in row
    y[r.x] += e.value * x[e.y]
```

- Distributes the rows of A *dynamically* across the threads in the parallel region

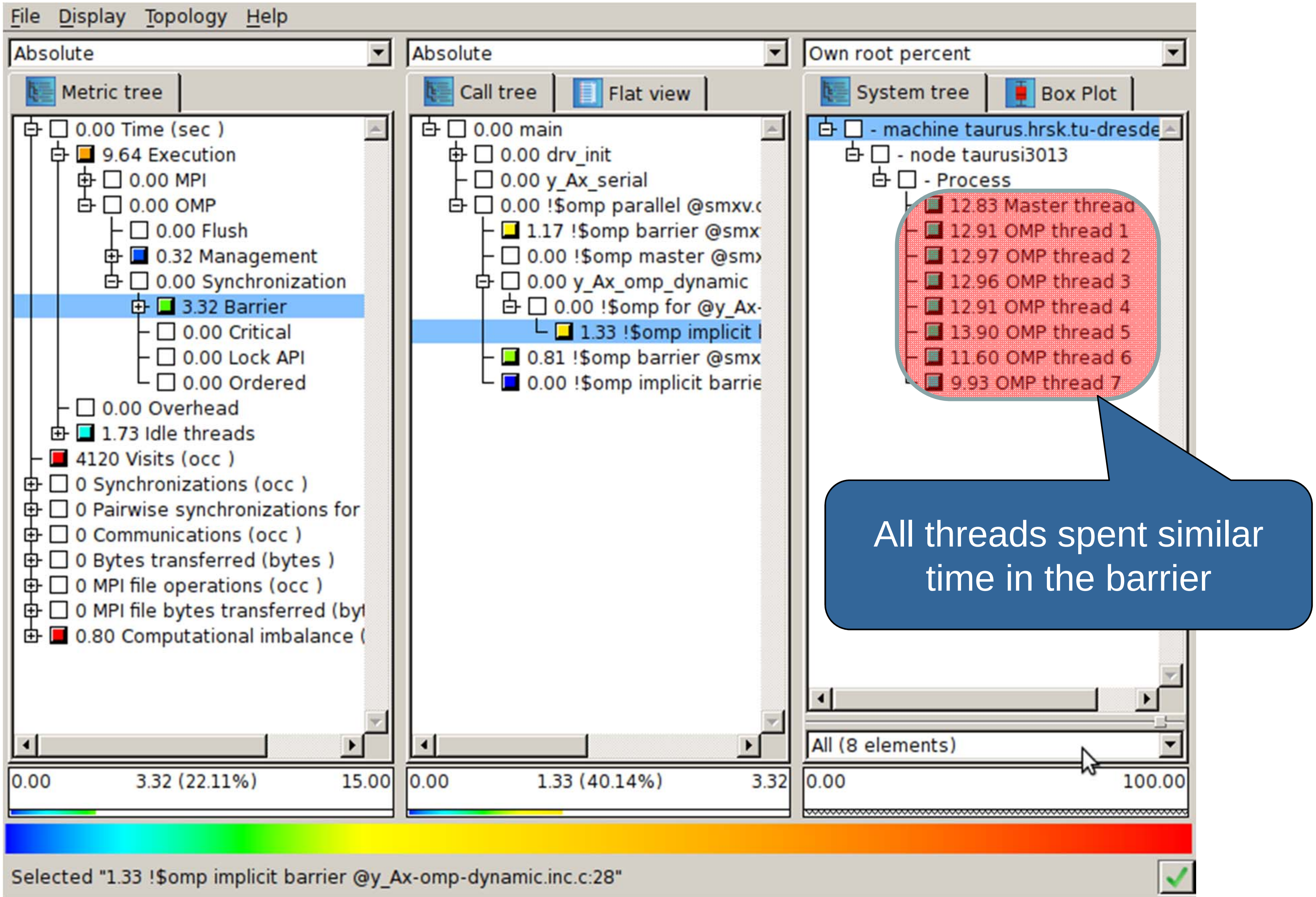
Case I: Profile Analysis

- Two metrics which indicate load imbalances:
  - Time spent in OpenMP barriers
  - Computational imbalance

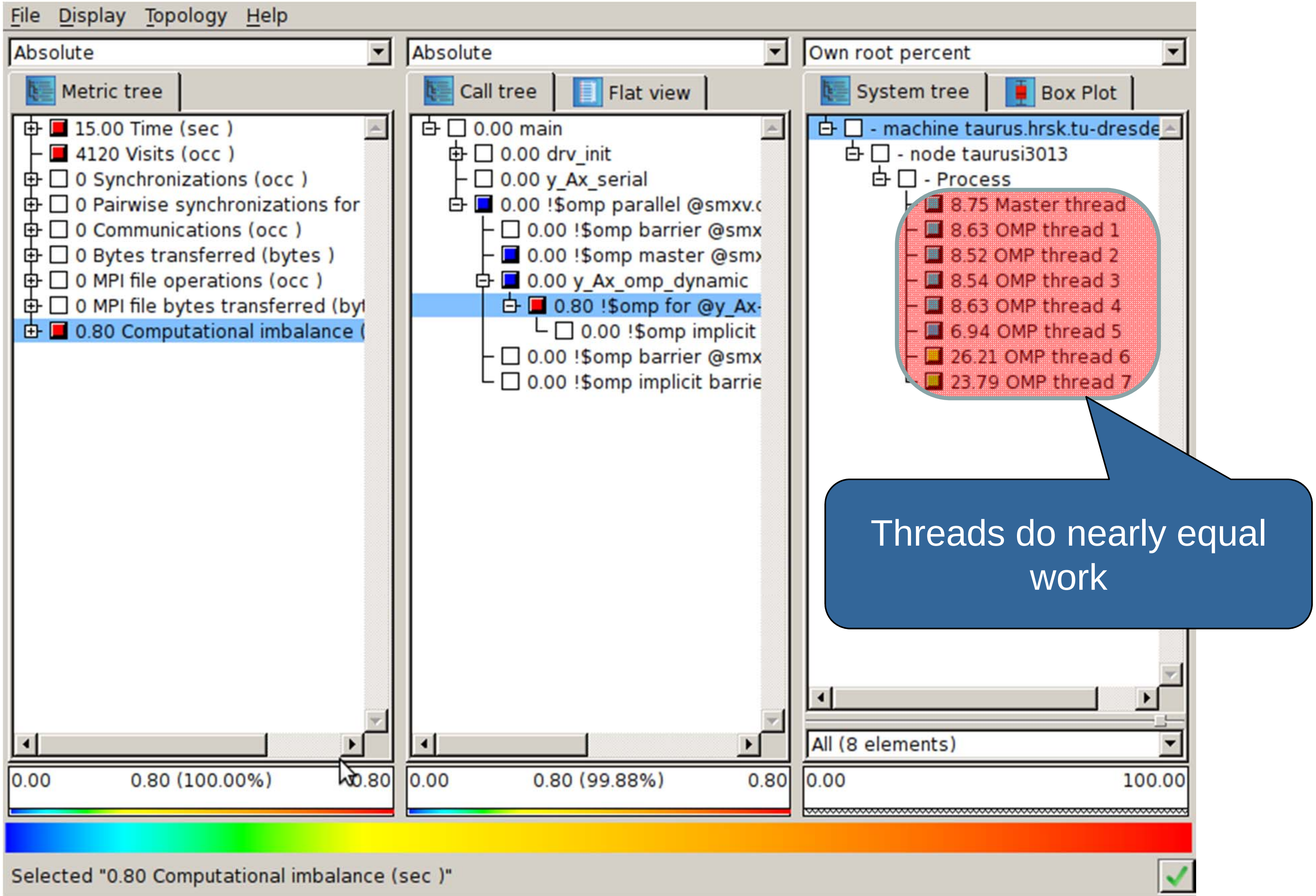
- Open prepared measurement on the LiveDVD with Cube:

```
% cube ~/Bottlenecks/smxv/scorep_smxv-omp-dynamic_large/trace.cubex

[CUBE GUI showing trace analysis report]
```

Case I: Time spent in OpenMP barriers



Case I: Computational imbalance

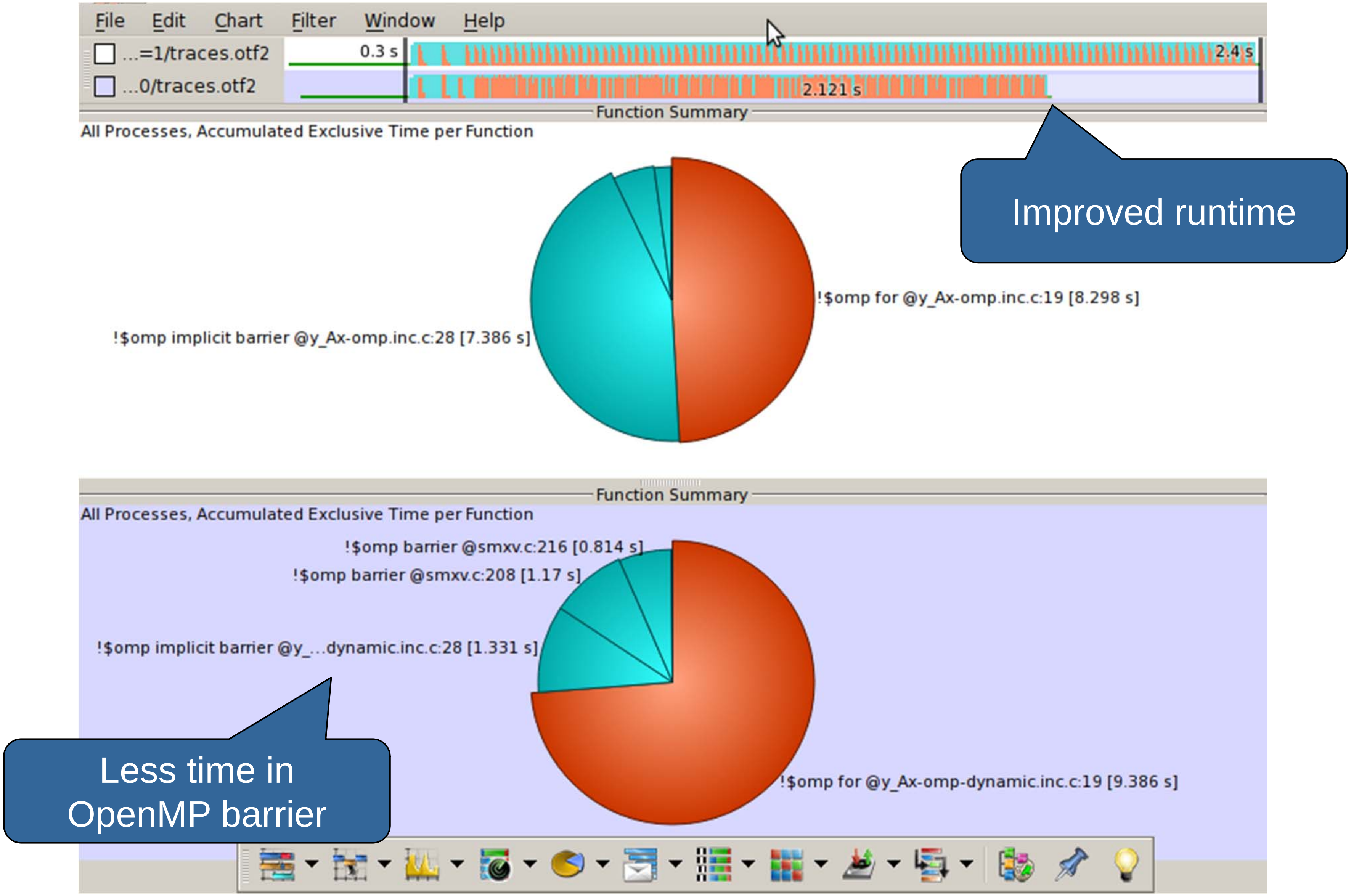


Case I: Trace Comparison

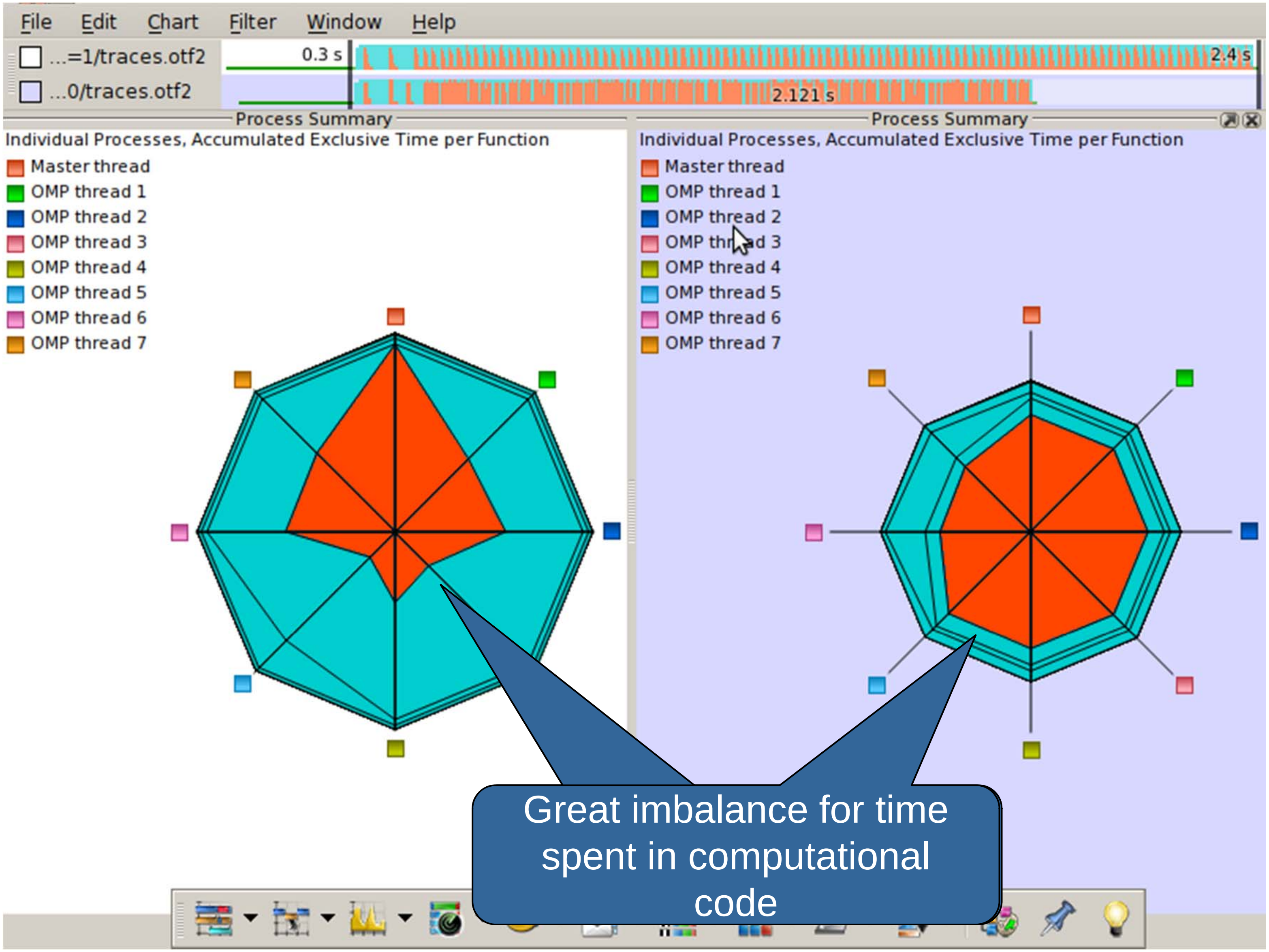
- Open prepared measurement on the LiveDVD with Vampir:

```
% vampir ~/Bottlenecks/smxv/scorep_smxv-omp_large/traces.otf2

                      [Vampir GUI showing trace]
```

Case I: Time spent in OpenMP barriers



Case I: Computational imbalance



Outline

- **Case I:**
  - **Finding load imbalances in OpenMP codes**

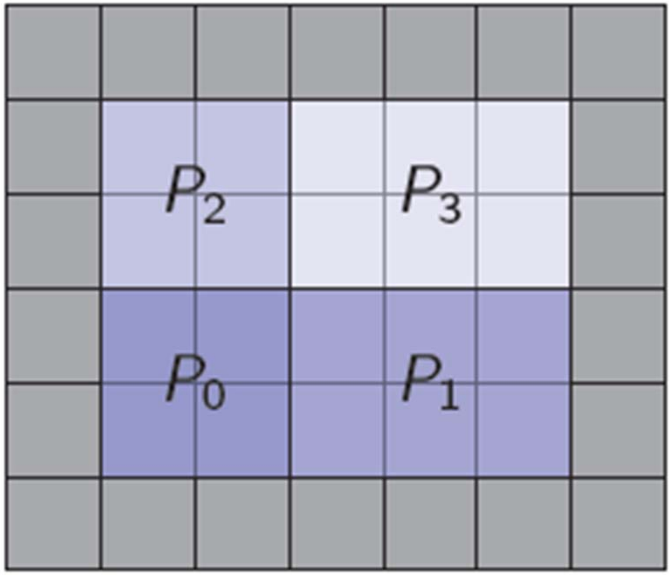
- **Case II:**
  - **Finding communication and computation imbalances in MPI codes**

- Calculating the heat conduction at each timestep
- Discretized formula for space dx, dy and time dt

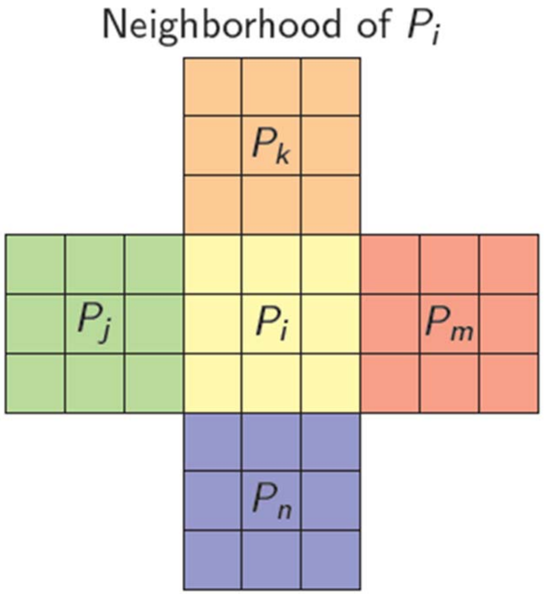
$$\theta_{i,j}^{t+1} = \theta_{i,j}^{t} + \left(\frac{\theta_{i+1,j}^{t} - 2\theta_{i,j}^{t} + 2\theta_{i-1,j}^{t}}{dx^2} + \frac{\theta_{i,j+1}^{t} - 2\theta_{i,j}^{t} + 2\theta_{i,j-1}^{t}}{dy^2} \right) \cdot k \cdot dt$$

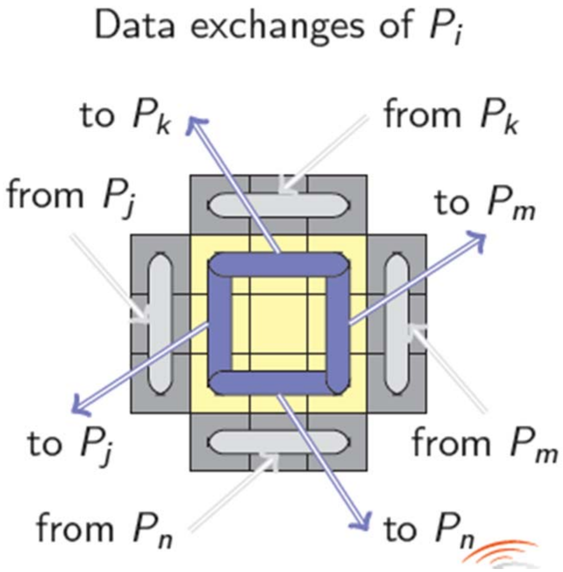
Case II: Heat Conduction Simulation

- Application uses MPI for boundary exchange and OpenMP for computation
- Simulation grid is distributed across MPI ranks



- Ranks need to exchange boundaries before next iteration step





- MPI Algorithm

```
foreach step in [1:nsteps]
  exchangeBoundaries
  computeHeatConduction
```

- Building and measuring the heat conduction application:

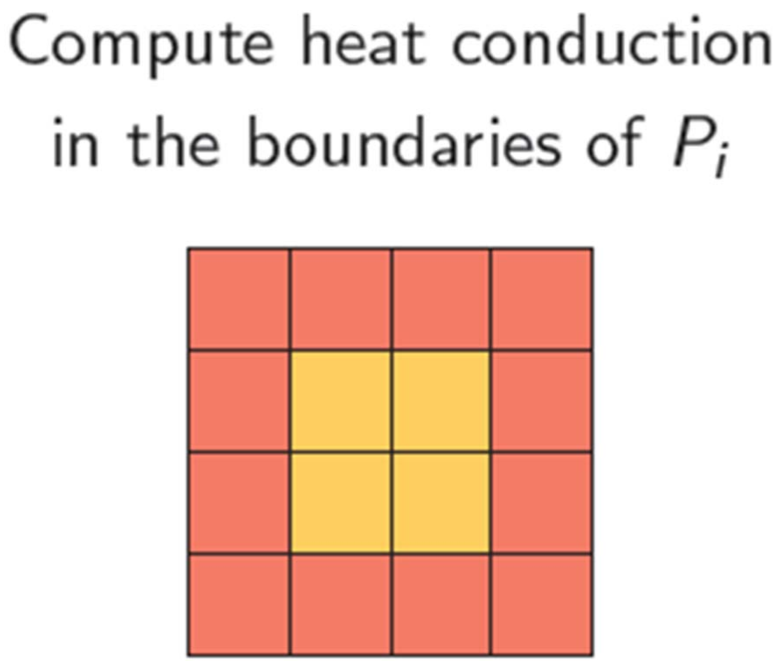
```
% cd ~/Bottlenecks/heat
% make PREP='scorep --user'
  [... make output ...]
% scan –t mpirun –np 16 ./heat-MPI 3072 32
```

- Open prepared measurement on the LiveDVD with Cube

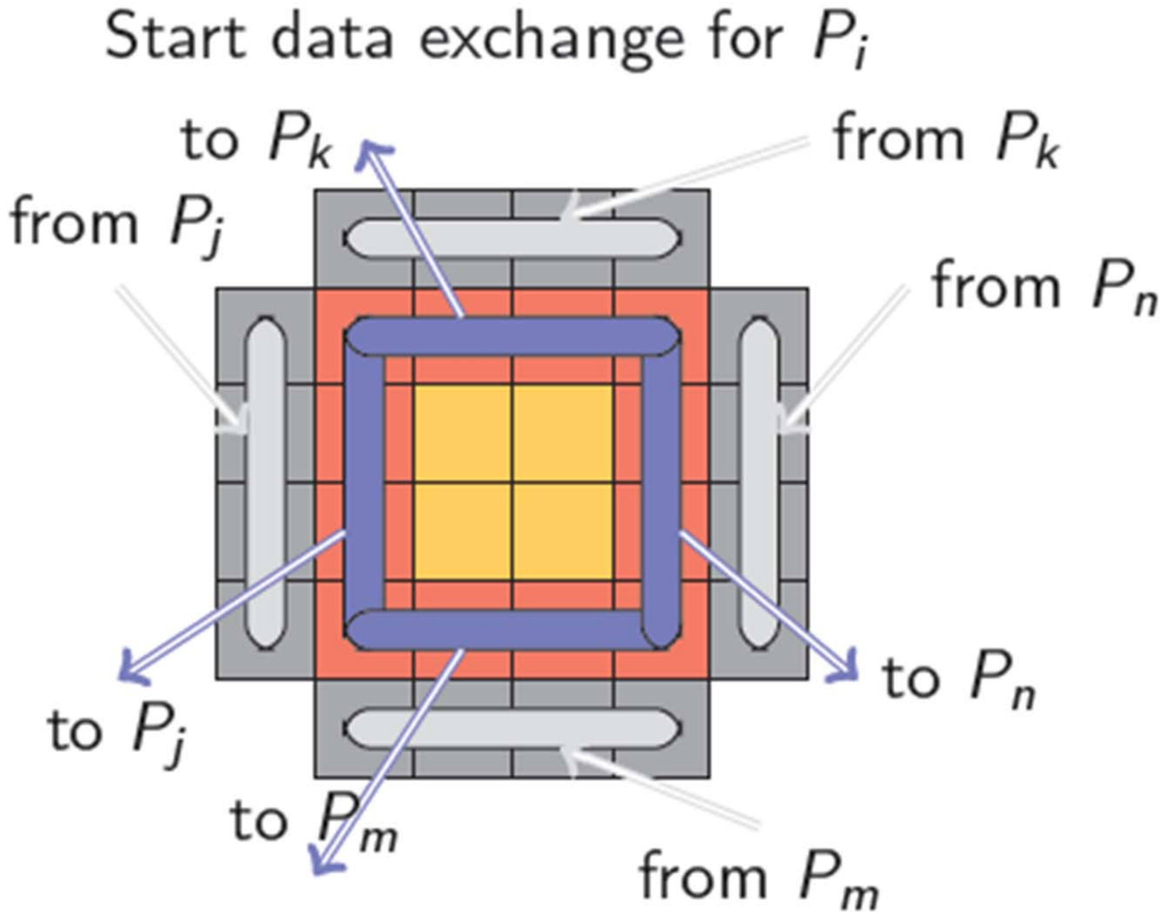
```
% cube ~/Bottlenecks/heat/scorep_heat-MPI_16/trace.cubex

               [CUBE GUI showing trace analysis report]
```

Case II: Hide MPI communication with computation

- Step 1: Compute heat in the area which is communicated to your neighbors

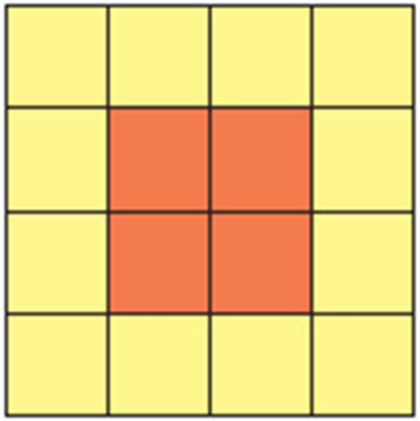


- Step 2: Start communicating boundaries with your neighbors



- Step 3: Compute heat in the interior area

Compute heat conduction
in the interior of P_i



Case II: Profile Analysis

- Improved MPI Algorithm

```
foreach step in [1:nsteps]
  computeHeatConductionInBoundaries
  startBoundaryExchange
  computeHeatConductionInInterior
  waitForCompletionOfBoundaryExchange
```

- Measuring the improved heat conduction application:

```
% scan –t mpirun –np 16 ./heat-MPI-overlap 3072 32
```

- Open prepared measurement on the LiveDVD with Cube

```
% cube ~/Bottlenecks/heat/scorep_heat-MPI-overlap_16/trace.cubex

                [CUBE GUI showing trace analysis report]
```

Case II: Trace Comparison

- Open prepared measurement on the LiveDVD with Vampir:

```
% vampir ~/Bottlenecks/heat/scorep_heat-MPI_16/traces.otf2

                      [Vampir GUI showing trace]
```

Acknowledgments

- Thanks to Dirk Schmidl, RWTH Aachen, for providing the sparse matrix vector multiplication code

Review

Brian Wylie
Jülich Supercomputing Centre

Summary

You've been introduced to a variety of tools, and had an opportunity to try them with a prepared example code

- with guidance to apply and use the tools most effectively

- Tools provide complementary capabilities
  - computational kernel & processor analyses
  - communication/synchronization analyses
  - load-balance, scheduling, scaling, …
- Tools are designed with various trade-offs
  - general-purpose versus specialized
  - platform-specific versus agnostic
  - simple/basic versus complex/powerful

Tool selection

- Which tools you use and when you use them likely to depend on situation
  - which are available on (or for) your computer system
  - which support your programming paradigms and languages
  - which you are familiar (comfortable) with using
- also depends on the type of issue you have or suspect
- Awareness of (potentially) available tools can help finding the most appropriate tools

Workflow (getting started)

- First ensure that the parallel application runs correctly
  - no-one will care how quickly you can get invalid answers or produce a directory full of corefiles
  - parallel debuggers help isolate known problems
  - correctness checking tools can help identify other issues
  - (that might not cause problems right now, but will eventually)
    - e.g., race conditions, invalid/non-compliant usage
- Generally valuable to start with an overview of execution performance
  - fraction of time spent in computation vs comm/synch vs I/O
  - which sections of the application/library code are most costly
- and how it changes with scale or different configurations
  - processes vs threads, mappings, bindings

Workflow (communication/synchronization)

- Communication/synchronization issues generally apply to every computer system (to different extents) and typically grow with the number of processes/threads
  - *Weak scaling*: fixed computation per thread, and perhaps fixed localities, but increasingly distributed
  - *Strong scaling*: constant total computation, increasingly divided amongst threads, while communication grows
  - Collective communication (particularly of type “all-to-all”) result in increasing data movement
  - Synchronizations of larger groups are increasingly costly
  - Load-balancing becomes increasingly challenging, and imbalances increasingly expensive
    - generally manifests as waiting time at following collective ops

Workflow (wasted waiting time)

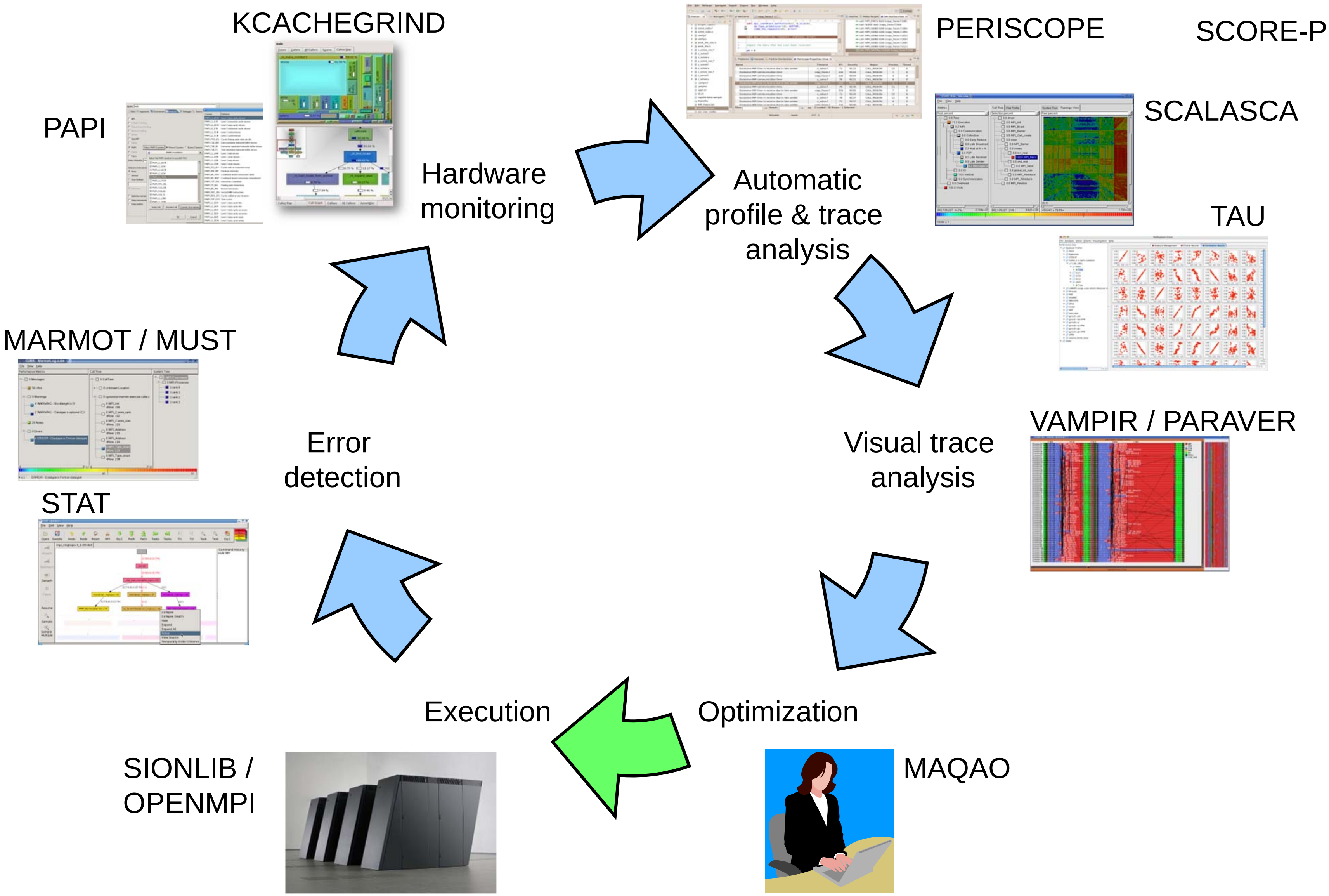
- Waiting times are difficult to determine in basic profiles
  - Part of the time each process/thread spends in communication & synchronization operations may be wasted waiting time
  - Need to correlate event times between processes/threads
    - *Periscope* uses augmented messages to transfer timestamps and additional on-line analysis processes
    - Post-mortem event trace analysis avoids interference and provides a complete history
    - *Scalasca* automates trace analysis and ensures waiting times are completely quantified
    - *Vampir* allows interactive exploration and detailed examination of reasons for inefficiencies

Workflow (core computation)

Effective computation within processors/cores is also vital

- Optimized libraries may already be available
- Optimizing compilers can also do a lot
  - provided the code is clearly written and not too complex
  - appropriate directives and other hints can also help
- Processor hardware counters can also provide insight
  - although hardware-specific interpretation required
- Tools available from processor and system vendors help navigate and interpret processor-specific performance issues

VI-HPS tools portfolio and their integration



- **Score-P**
  - community-developed instrumenter & measurement libraries for parallel profiling and event tracing
- **CUBE** & **ParaProf/PerfExplorer**
  - interactive parallel profile analyses
- **Scalasca**
  - automated event-trace analysis
- **Vampir**
  - interactive event-trace visualizations and analyses
- **TAU/PDT**
  - comprehensive performance system

Further information

- Website
  - Introductory information about the VI-HPS portfolio of tools for high-productivity parallel application development
    - links to individual tools sites for details and download
  - Training material
    - tutorial slides
    - latest ISO image of VI-HPS Linux DVD with productivity tools
    - user guides and reference manuals for tools
  - News of upcoming events
    - tutorials and workshops
    - mailing-list sign-up for announcements

http://www.vi-hps.org