SC23
Denver, CO | i am hpc.

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)

Tutorial logistics

- This is a full-day tutorial featuring *hands-on exercises*
  - Browser-based access available to JUWELS-Booster where the presented tools are installed
  - Exercise materials and slides are provided for you to do on your own
  - We'll be showing how the tools are used and the key functionality that they offer
    - Using provided example measurements & a simple benchmark code
  - But insufficient time to include everything, so for additional details refer to reference slides (marked by the book icon  in the upper right)
- We want the tutorial to be interactive!
  - Ask questions
  - Indicate when you need assistance

Access to JUWELS-Booster

- JUWELS-Booster at JSC will be used for the hands-on exercises
  - request an account on JSC HPC systems with the training project for tutorial

`https://judoor.fz-juelich.de/projects/join/training2341`

- If you don't already have an account for JSC HPC systems, you'll first need to register
  - provide your E-mail address
  - wait for account registration E-mail from `user-services.jsc@fz-juelich.de`
  - follow URL in E-mail to complete registration
  - provide your name, affiliation, etc
  - read & accept privacy policy

- Account setup may take 30 minutes

Virtual Institute – High Productivity Supercomputing

- **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
RESEARCH FOR GRAND CHALLENGES

- Activities
  - Development and integration of HPC application tools
    - Primarily correctness checking & performance analysis
  - Academic workshops: e.g. ProTools@SC23 (Sunday 12 November 2023)
  - Tools training via conference tutorials and multi-day "bring-your-own-code" Tuning Workshops
    - Face-to-face & side-by-side hands-on coaching now successfully migrated to virtual/on-line events

http://www.vi-hps.org

VI-HPS partners (founders)

Forschungszentrum Jülich

- Jülich Supercomputing Centre

RWTH Aachen University

- Centre for Computing & Communication

Technische Universität Dresden

- Centre for Information Services & HPC

University of Tennessee (Knoxville)

- Innovative Computing Laboratory

VI-HPS partners (cont.)

Barcelona Supercomputing Center
- Centro Nacional de Supercomputación

Lawrence Livermore National Lab.
- Center for Applied Scientific Computing

Leibniz Supercomputing Centre

Linaro Ltd.
- Allinea software

Technical University of Darmstadt
- Laboratory for Parallel Programming

VI-HPS partners (cont.)

Friedrich-Alexander-Universität

- Erlangen Regional Computing Center (RRZE)

Technical University of Munich

- Chair for Computer Architecture

University of Oregon

- Performance Research Laboratory

University of Stuttgart

- HPC Centre

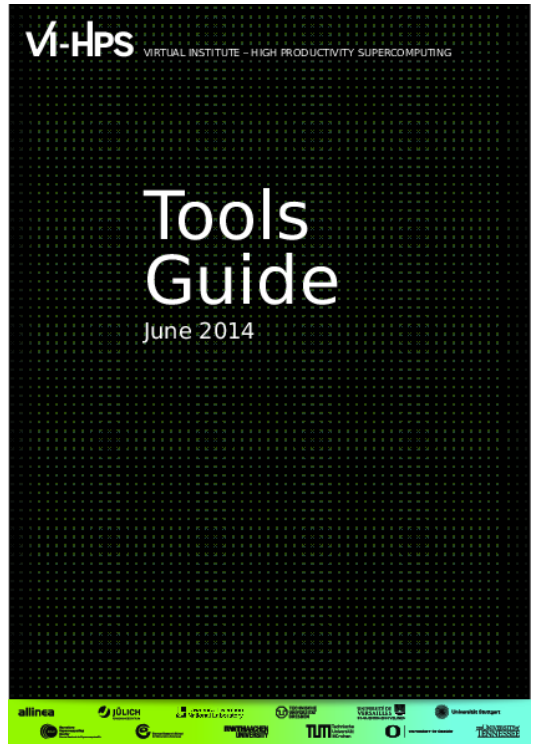
University of Versailles St-Quentin

- LRC ITACA

Productivity tools

- MUST / ARCHER
  - MPI & OpenMP usage correctness checking
- PAPI
  - Interfacing to hardware performance counters
- **CUBE**
  - Analysis report exploration & processing
- **Scalasca**
  - Large-scale parallel performance analysis
- **TAU**
  - Integrated parallel performance system
- **Vampir**
  - Interactive graphical trace visualization & analysis
- **Score-P**
  - Community-developed instrumentation & measurement infrastructure

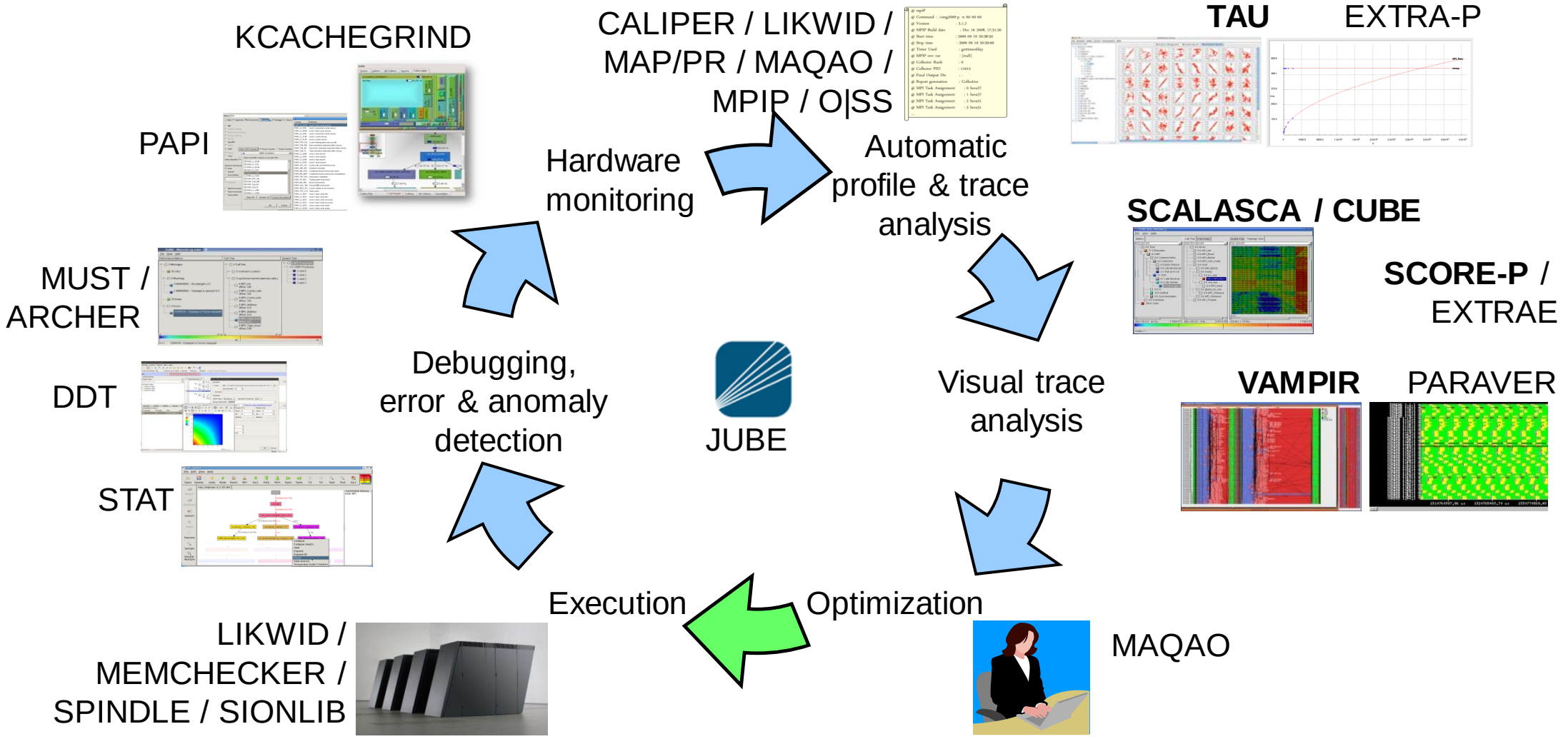
For a brief overview of tools consult the VI-HPS Tools Guide:



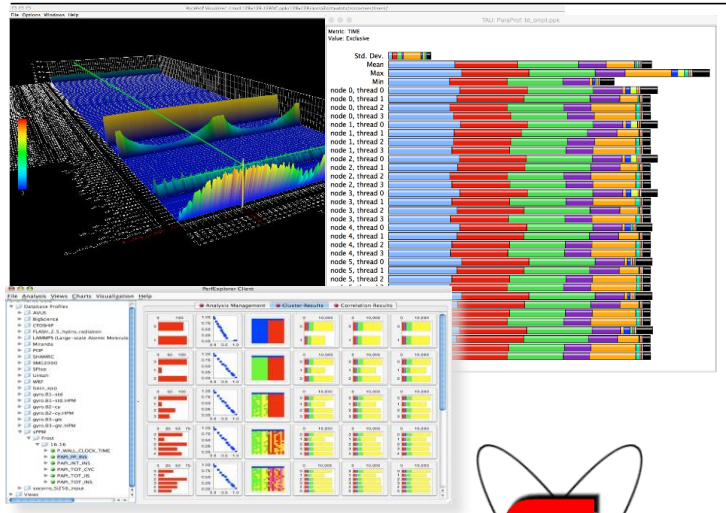
Productivity tools (cont.)

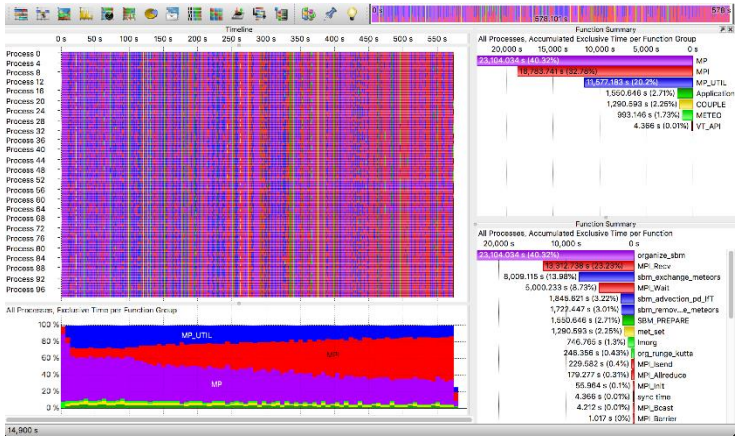
- Caliper: Library for application event annotation, logging and profiling
- Extra-P: Automated performance modelling
- FORGE DDT/MAP/PR: Parallel debugging, profiling & performance reports
- JUBE: Automatic workflow execution for benchmarking, testing & production
- Kcachegrind: Callgraph-based cache analysis [x86 only]
- LIKWID: Performance monitoring & benchmarking
- MAQAO: Assembly instrumentation & optimization [x86-64 only]
- mpiP/mpiPview: MPI profiling tool and analysis viewer
- Open MPI MemChecker: Integrated memory checking
- Open|SpeedShop: Integrated parallel performance analysis environment
- Paraver/Dimemas/Extrae: Event tracing and graphical trace visualization & analysis
- SIONlib/Spindle: Optimized native parallel file I/O & shared library loading
- STAT: Stack trace analysis tools

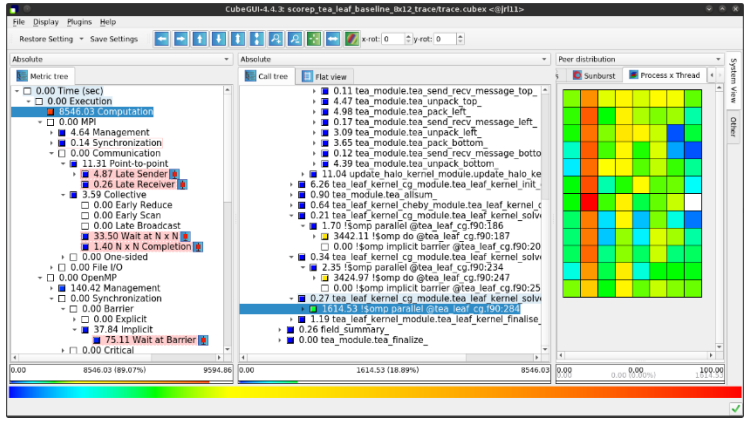
Technologies and their integration



Tools featured in this tutorial







Tuning and Analysis Utilities

VAMPIR

scalasca

Score-P
Scalable performance measurement infrastructure for parallel codes

Agenda (Part 1)

Time	Topic	Presenter
08:30	Welcome & Introduction to VI-HPS	Wylie
08:45	Introduction to parallel performance engineering	Geimer
09:15	Setup for hands-on exercises	Wylie
09:30	Instrumentation & measurement of applications with ***Score-P***	Wesarg
10:00	*Break*	
10:30	Exploration of execution call-path profiles with ***CUBE***	Wylie
11:00	Configuration & customization of *Score-P* measurements	Wesarg
11:30	Examination of profiles with ***TAU ParaProf***	Shende
12:00	*Lunch break*	

Agenda (Part 2)

Time	Topic	Presenter
13:30	Résumé + *Score-P* trace collection	Wylie
13:45	Interactive trace visualization & exploration with ***Vampir***	Wesarg
14:30	Automated analysis of traces for inefficiencies with ***Scalasca***	Geimer
15:00	*Break*	
15:30	Performance data management with ***TAU PerfExplorer***	Shende
15:45	Specialized *Score-P* measurements & analyses	Wesarg
16:15	Finding typical parallel performance bottlenecks	Wesarg
16:45	Review	Geimer
17:00	*Adjourn*	

SC23 Tutorial: Hands-on Practical Hybrid Parallel Application Performance Engineering

Sameer Shende
(sameer@cs.uoregon.edu)
University of Oregon

Markus Geimer
(m.geimer@fz-juelich.de)
Brian Wylie
(b.wylie@fz-juelich.de)
Jülich Supercomputing Centre

Bert Wesarg
(bert.wesarg@tu-dresden.de)
Technische Universität Dresden

DOE booth #**243**

JSC booth #**401**

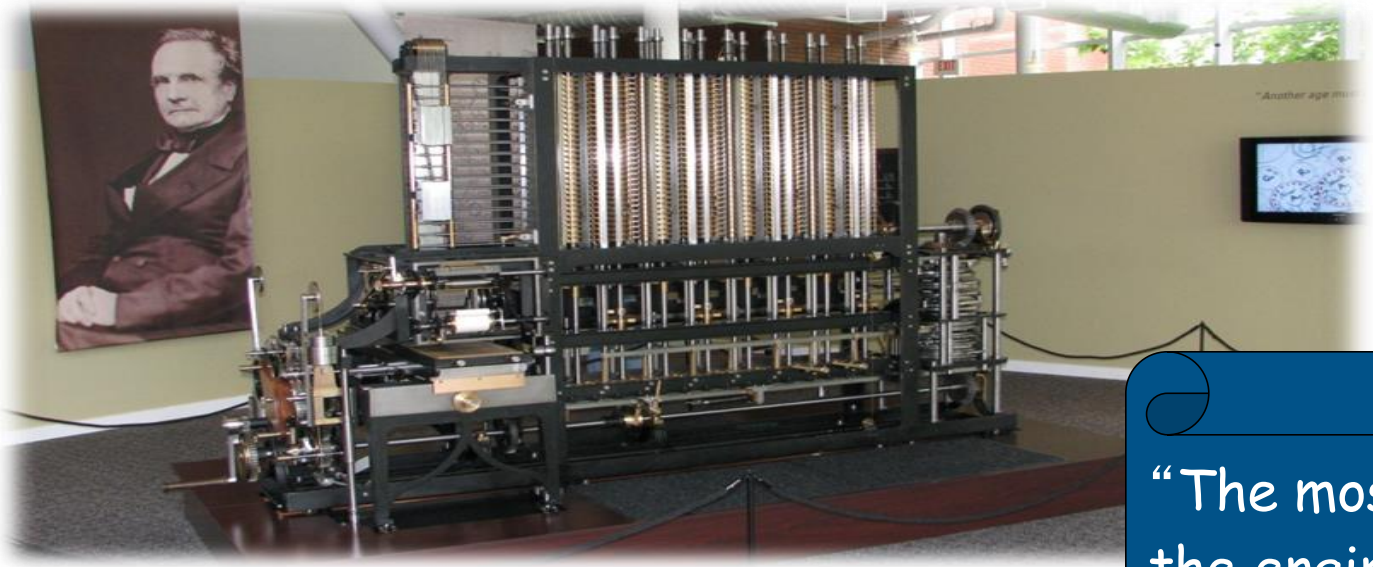
ZIH booth #**1533**

Introduction to Parallel Performance Engineering

Markus Geimer
Jülich Supercomputing Centre

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

Performance: an old problem



Difference Engine

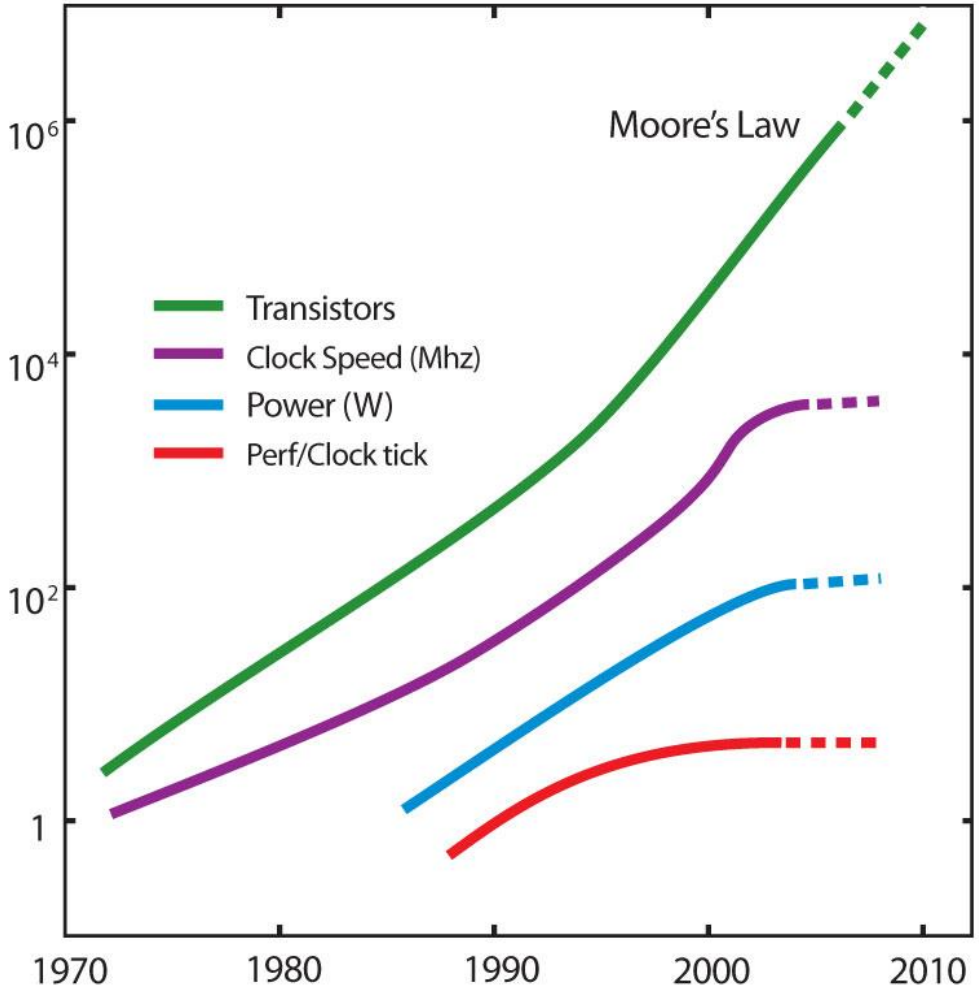
"The most constant difficulty in contriving the engine has arisen from the desire to reduce the time in which the calculations were executed to the shortest which is possible."

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!



Consequences

- Machine complexity
  - Increasing number of different architectures
  - Additional optimisation challenges related to parallelism
  - Single-core performance issues tied to increased vector length and memory hierarchy
- ➔ The optimisation process remains key to maintain a reasonable performance level

- Code complexity
  - Codes are harder to optimize and maintain manually
  - Optimisation is time consuming and error-prone
- ➔ Understanding application behaviour is critical

Performance factors of parallel applications

- "Sequential" performance 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

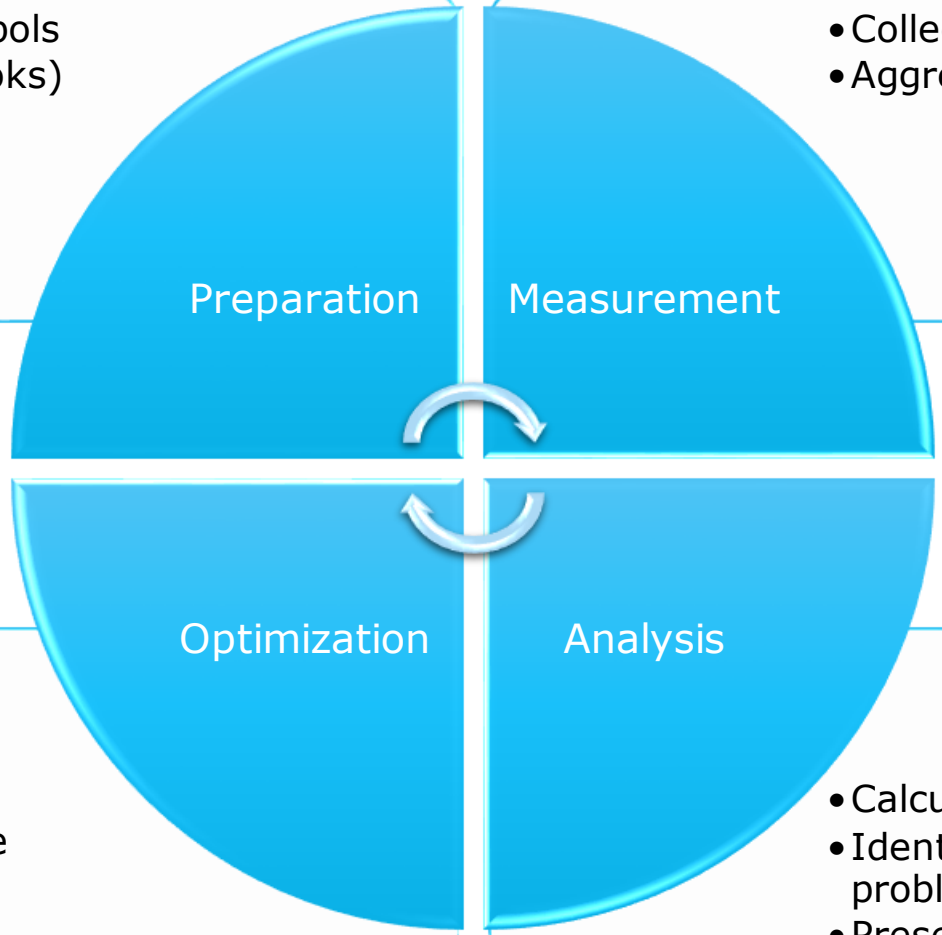
- "Parallel" performance factors
  - Partitioning / decomposition
  - Communication (i.e., message passing)
  - Multithreading
  - Synchronization / locking
    - More or less understood, good tool support

Tuning basics

- Successful engineering is a combination of
  - Careful setting of various tuning parameters
  - 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!
- Modeling is extremely useful but very difficult and rarely available
  - Allows to evaluate performance impact of optimization without implementing it
  - Simplifies search in large parameter space

Performance engineering workflow

- Prepare application with symbols
- Insert extra code (probes/hooks)



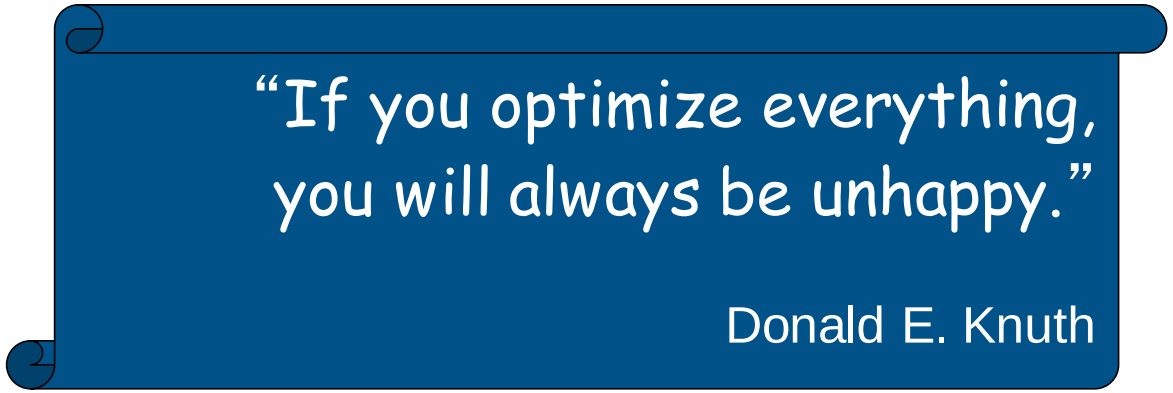
- Collection of performance data
- Aggregation of performance data

- Modifications intended to eliminate/reduce performance problem

- Calculation of metrics
- Identification of performance problems
- Presentation of results

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!

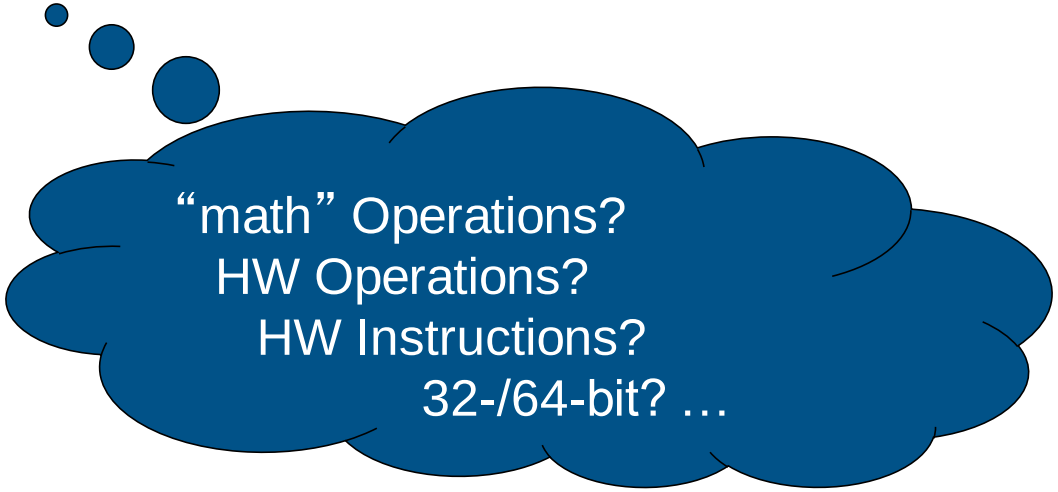


Metrics of performance

- 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

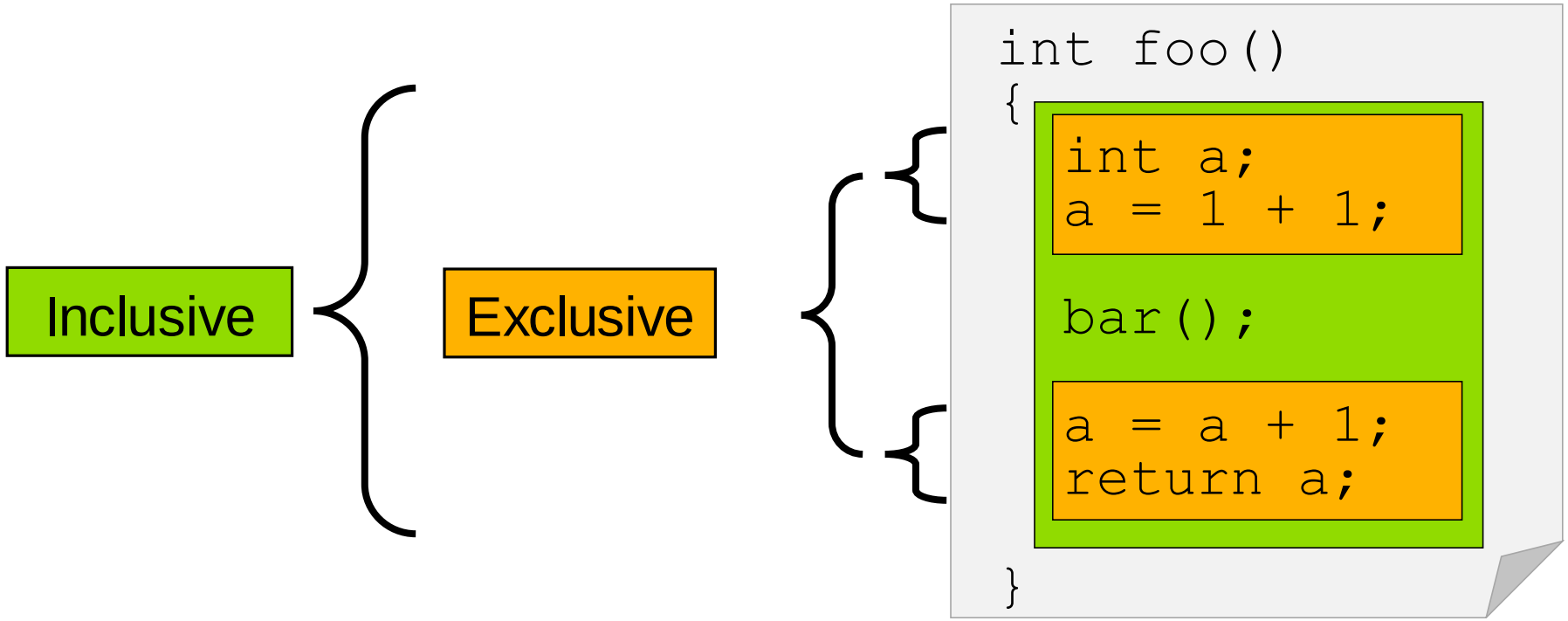


Execution time

- 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

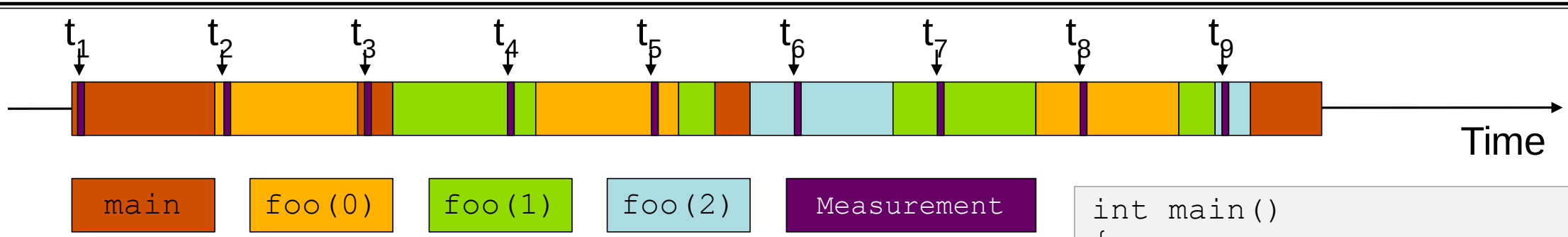
- Inclusive
  - Information of all sub-elements aggregated into single value
- Exclusive
  - Information cannot be subdivided further



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

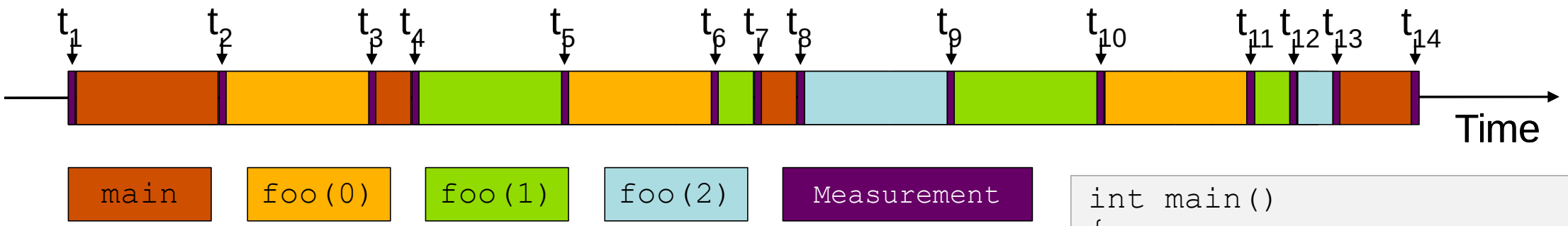
Sampling



- 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

```
int main()
{
  int i;

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

  return 0;
}

void foo(int i)
{

  if (i > 0)
    foo(i - 1);

}
```

Instrumentation



- 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

```
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");
}
```

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 detailed 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

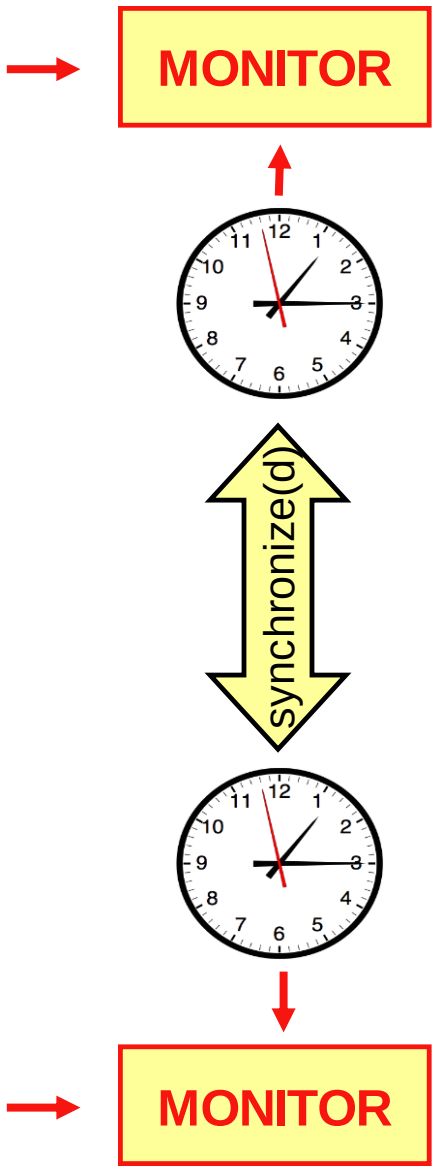
Process A

```
void foo() {
  trc_enter("foo");
  ...
  trc_send(B);
  send(B, tag, buf);
  ...
  trc_exit("foo");
}
```

instrument

Process B

```
void bar() {
  trc_enter("bar");
  ...
  recv(A, tag, buf);
  trc_recv(A);
  ...
  trc_exit("bar");
}
```



Local trace A

...	
58	ENTER foo
62	SEND to B
64	EXIT foo
...	

Local trace B

...	
60	ENTER bar
68	RECV from A
69	EXIT bar
...	

(Virtual merge)

Global trace view

...		
58	A	ENTER foo
60	B	ENTER bar
62	A	SEND to B
64	A	EXIT foo
68	B	RECV from A
69	B	EXIT bar
...		

Tracing Pros & Cons

- Tracing advantages
  - Event traces preserve the **temporal** and **spatial** relationships among individual events (☞ context)
  - Allows reconstruction of **dynamic** application behavior 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 may causes perturbation

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
  - Requires formalized knowledge about performance bottlenecks
  - More sophisticated inter-process analysis using
    - "Piggyback" messages
    - Hierarchical network of analysis agents
- Online analysis often involves application steering to interrupt and re-configure the measurement

Post-mortem analysis

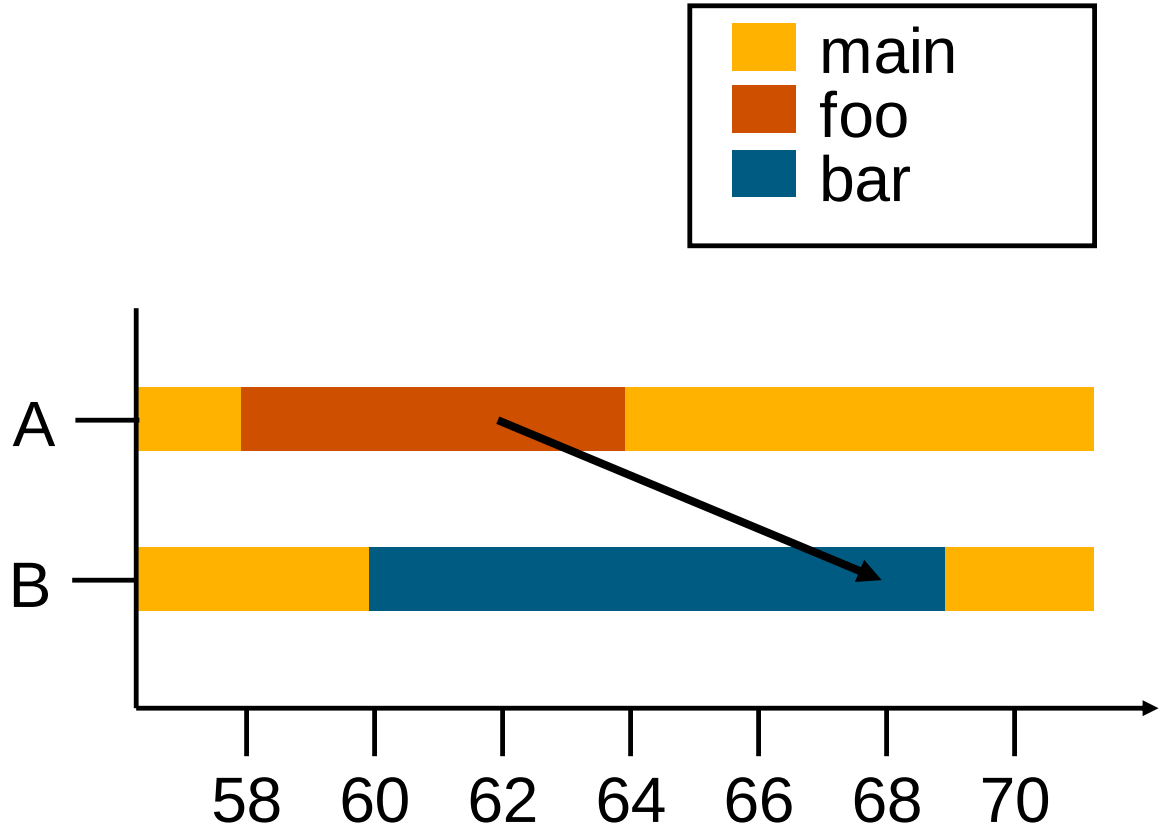
- Performance data is stored at end of measurement run
- Data analysis is performed afterwards
  - Automatic search for bottlenecks
  - Visual trace analysis
  - Calculation of statistics

Example: Time-line visualization

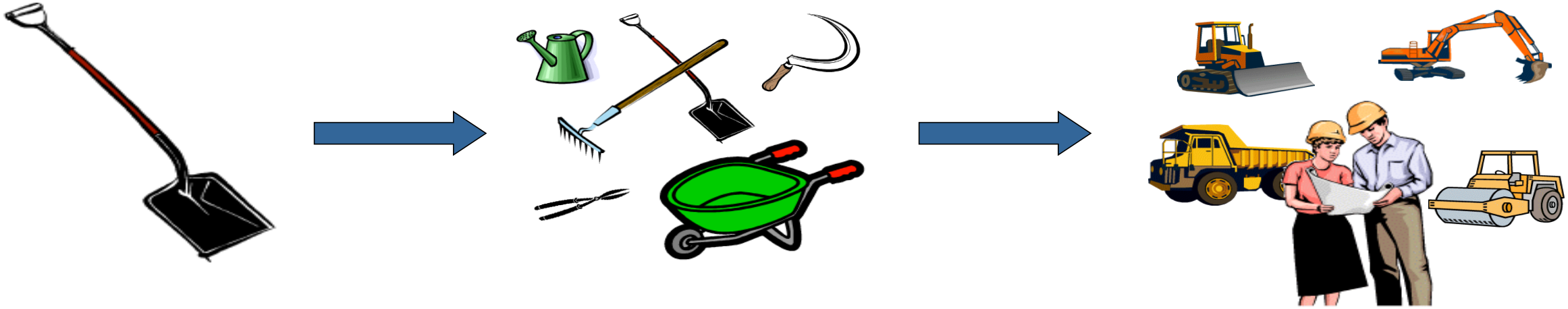
Global trace view

...		
58	A	ENTER foo
60	B	ENTER bar
62	A	SEND to B
64	A	EXIT foo
68	B	RECV from A
69	B	EXIT bar
...		

Post-Mortem Analysis



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

JUWELS Booster & JupyterJSC/Xpra

Brian Wylie

JUWELS-Booster (juwels-booster@fz-juelich.de)

JUWELS-Booster system overview

- 936 *accelerated* compute nodes
  - dual AMD EPYC 7402 processors, each with 24 cores, 2.8 GHz, 512 GB DDR4 RAM
  - *quad Nvidia A100 'Ampere' GPUs with 40GB HBM*
- 4 *login* nodes (and additional service nodes)
  - dual AMD EPYC 7402 processors, each with 24 cores, 2.8 GHz, 512 GB DDR4 RAM

- Mellanox InfiniBand HDR full fat-tree interconnect network
- All compute nodes are diskless: OS image in RAM
- IBM Spectrum Scale (GPFS) parallel file system connection to JUST storage & HPST

Software environment

- Rocky Linux 8 OS
- Batch system workload/resource management based on SLURM from ParTec
- Programming environment
  - GNU, Intel & NVHPC compilers (for C, C++ & Fortran)
    - all supporting OpenMP and other multithreading
  - ParaStation MPI (based on MPICH3) & OpenMPI
  - Optimized mathematical libraries (Intel MKL, etc.)

Accessing software

- Hierarchical modules: 'toolchain' constructed by loading compiler then MPI

- List available modules (ready to be loaded)
  - `module avail`
- Search for an application/library/tool
  - `module spider <name>`
- Load the desired compiler
  - `module load GCC`
- Load the desired MPI
  - `module load ParaStationMPI`
- Load additional applications/libraries/tools
  - `module load Score-P Scalasca Vampir`

- List currently loaded modules
  - `module list`
- Purge all loaded modules
  - `module purge`
- Unload an undesired compiler
  - `module unload CUDA`
- Save current collection of modules
  - `module save [<name>]`
- Restore a saved collection of modules
  - `module restore [<name>]`

Filesystems

- $HOME (/p/home/$USER)
  - private, small storage quota for each user account, poor parallel I/O performance
- $PROJECT (/p/project)
  - shared by project members, regular backup, optimized parallel I/O performance
- $SCRATCH (/p/scratch)
  - shared by project members, no backup, optimized parallel I/O performance, automatic purge based on last file access
- $ARCHIVE
- …

SLURM

- Show available partitions
  - `sinfo`
- Show queued jobs of user
  - `squeue –u $USER`
- Cancel queued/running job
  - `scancel jobid`
- Submit batch script (to partition)
  - `sbatch <script.sh>`
    - within script ***always*** use `srun` (rather than mpiexec/mpirun) to launch application on allocated compute nodes
      - `srun mpi_app.exe`

Jupyter-JSC webservice

Registration

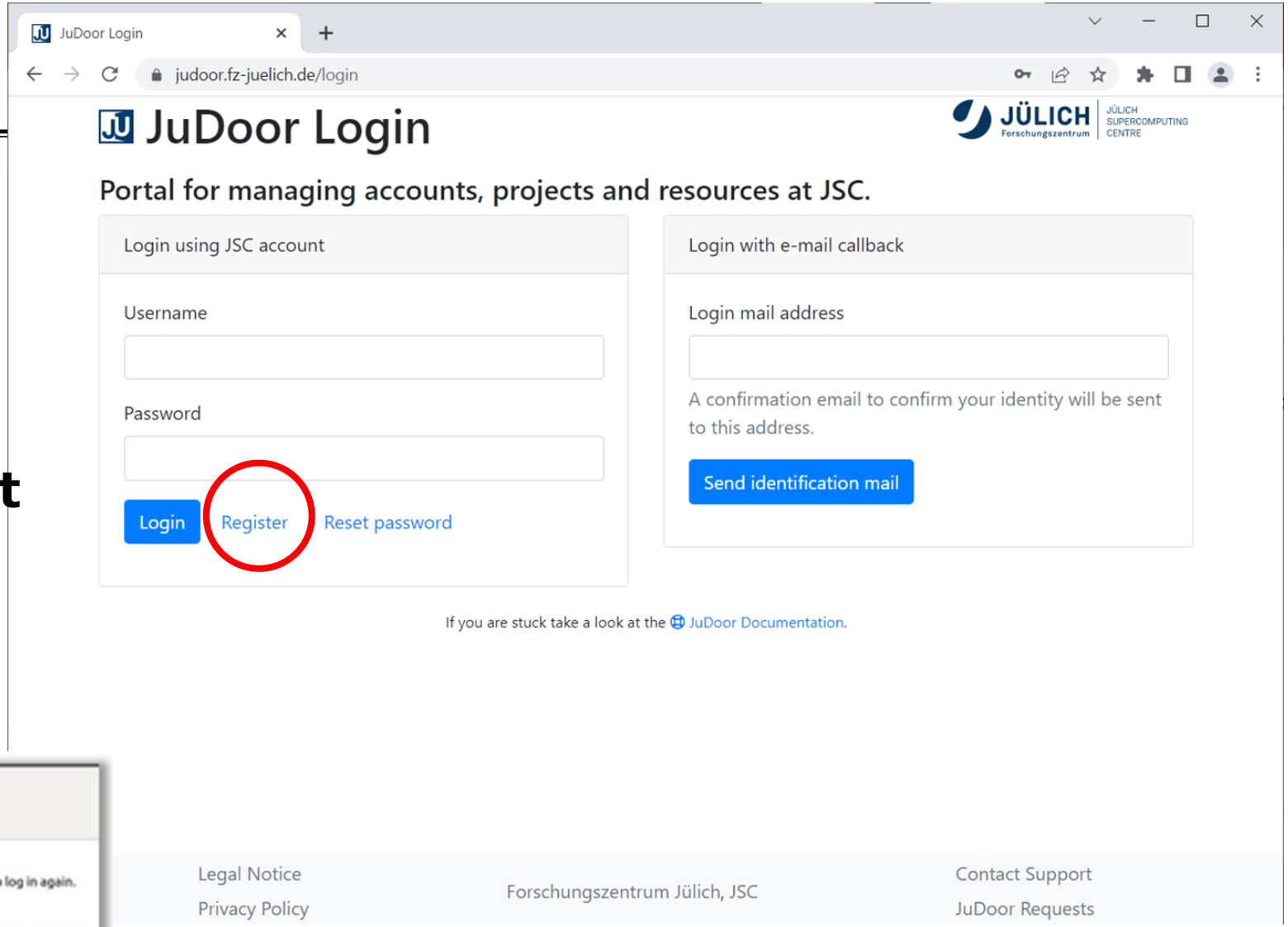
- Go to training project join link in web browser

  `https://judoor.fz-juelich.de/projects/join/training2341`

- Register for JSC HPC (JuDoor) account (if don't already have one)
  - Provide Email address and other details
  - Accept usage agreement
  - Submit the registration
  - Wait for registration Email and follow link to confirm

Register for account

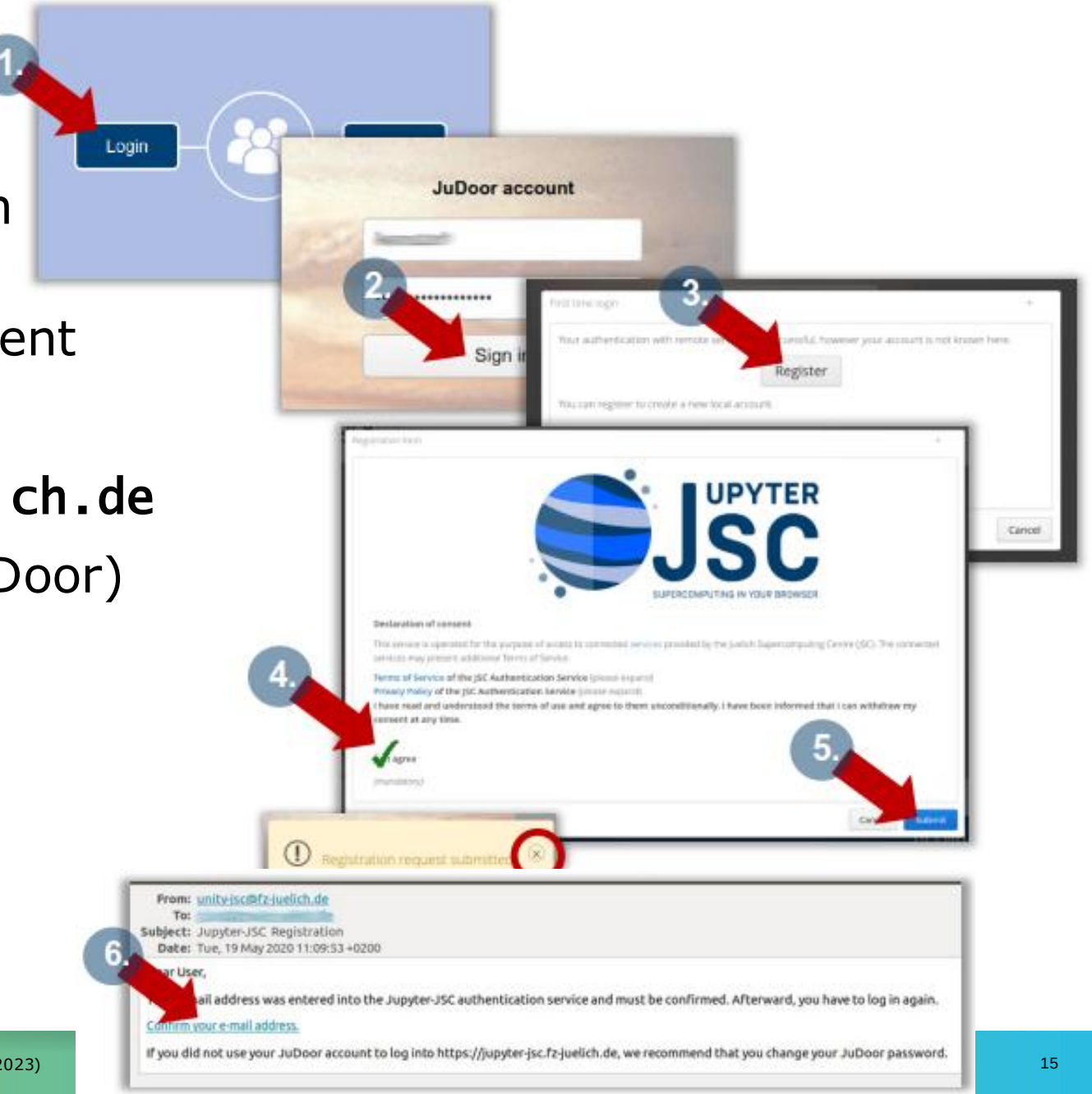
- Provide Email address
- Wait for Email with URL for account registration
- Enter name/affiliation/etc
- **Accept usage agreement**
- Submit the registration
- Wait for registration Email and follow link to confirm



First time login

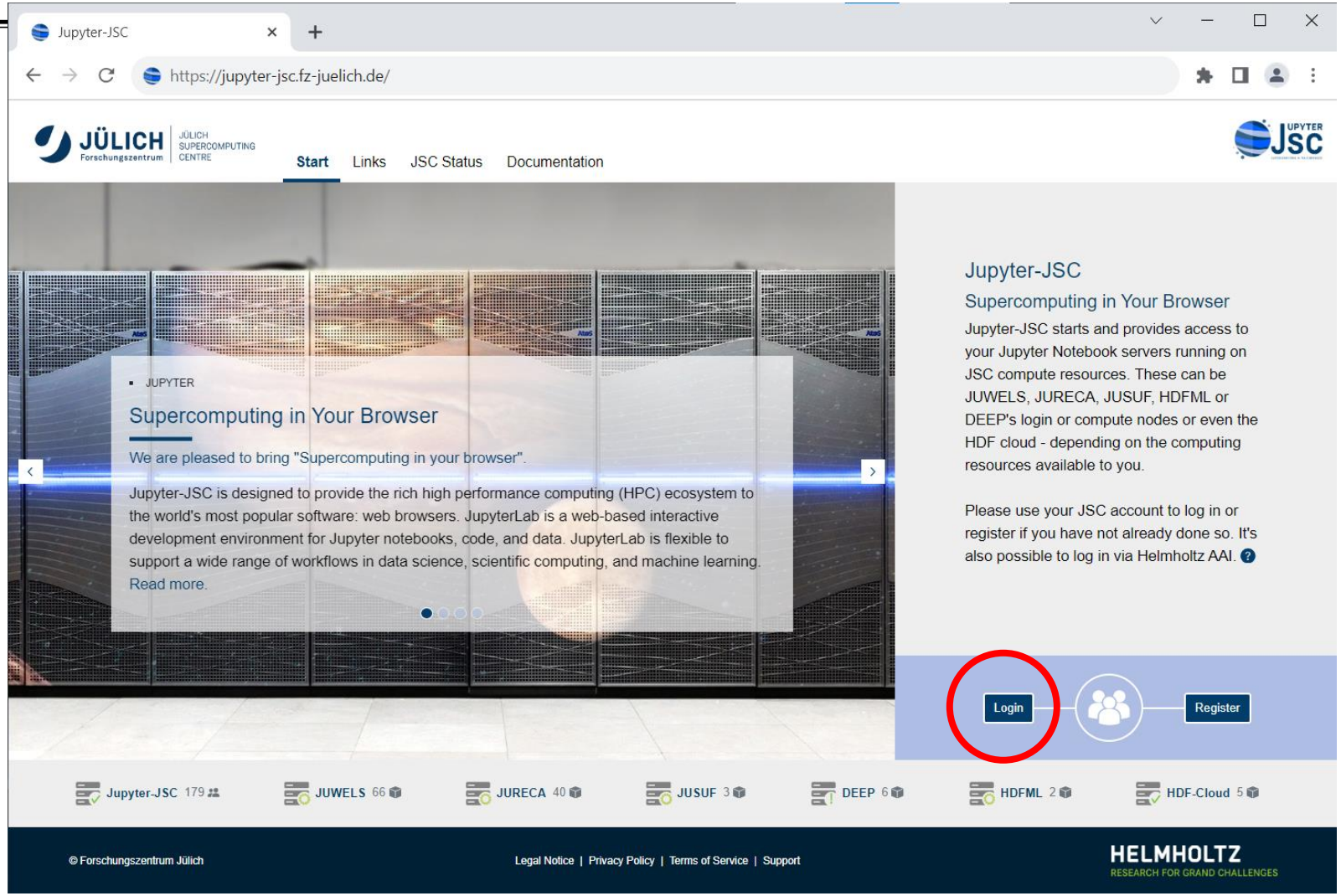
0. Pre-requisite: JuDoor registration with project membership and accepted systems usage agreement
1. Go to link in web browser

   `https://jupyter-jsc.fz-juelich.de`
2. Sign in with JSC HPC account (JuDoor)
3. Register to Jupyter-JSC
4. Accept usage agreement
5. Submit the registration
6. Wait for email and confirm



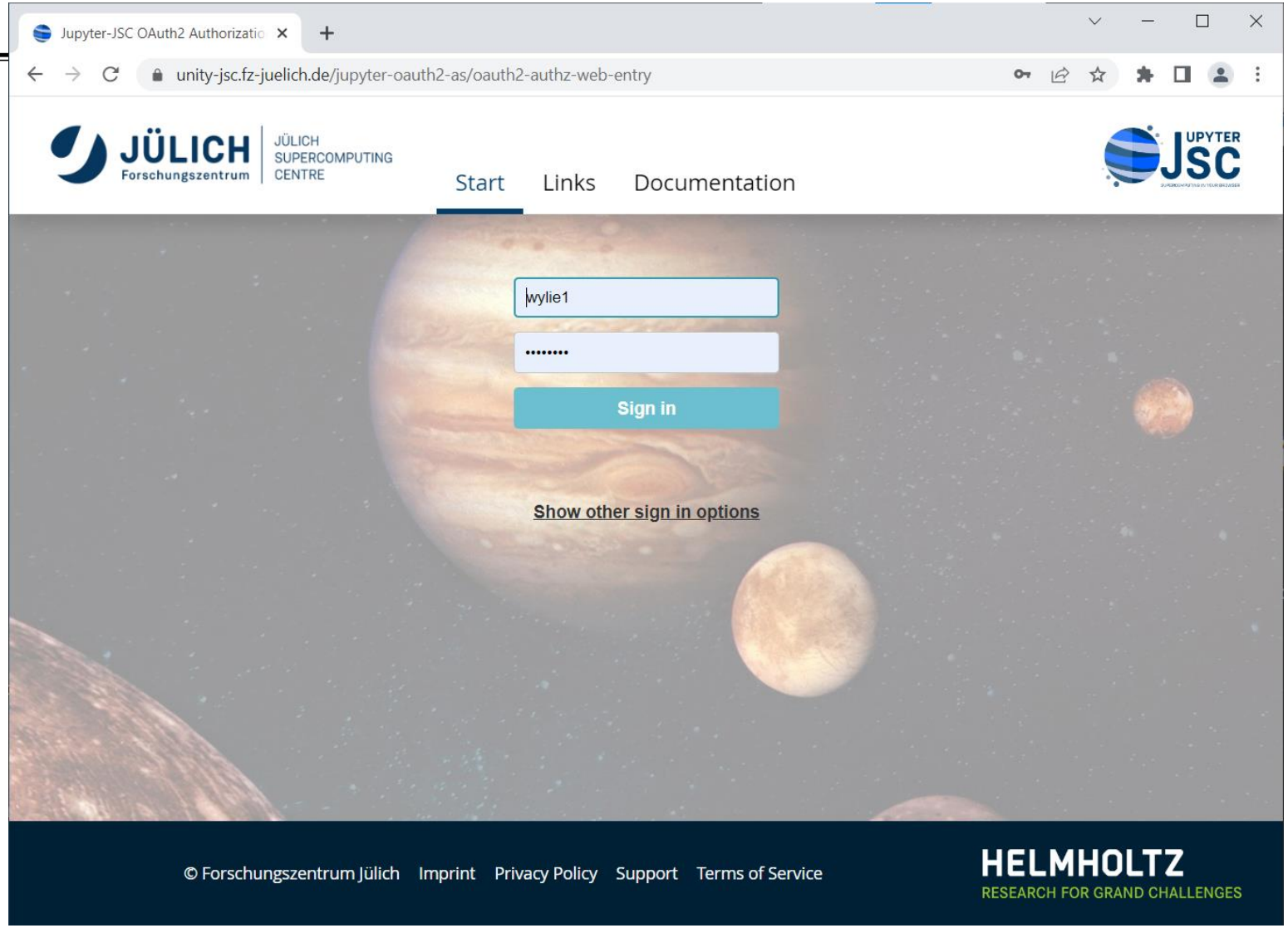
Login

- Click on Login button to enter



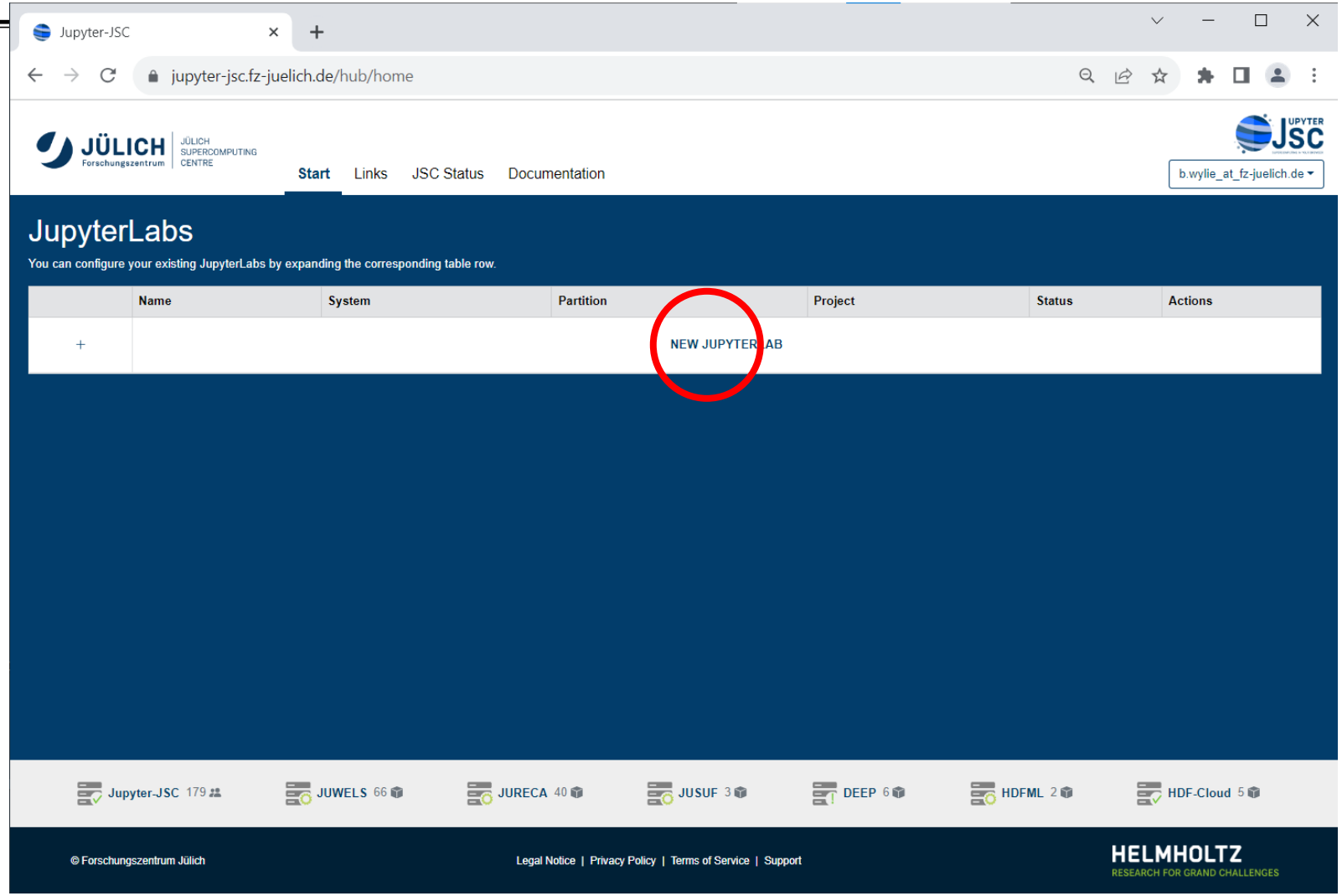
Sign in

- Provide JSC HPC account credentials to sign in



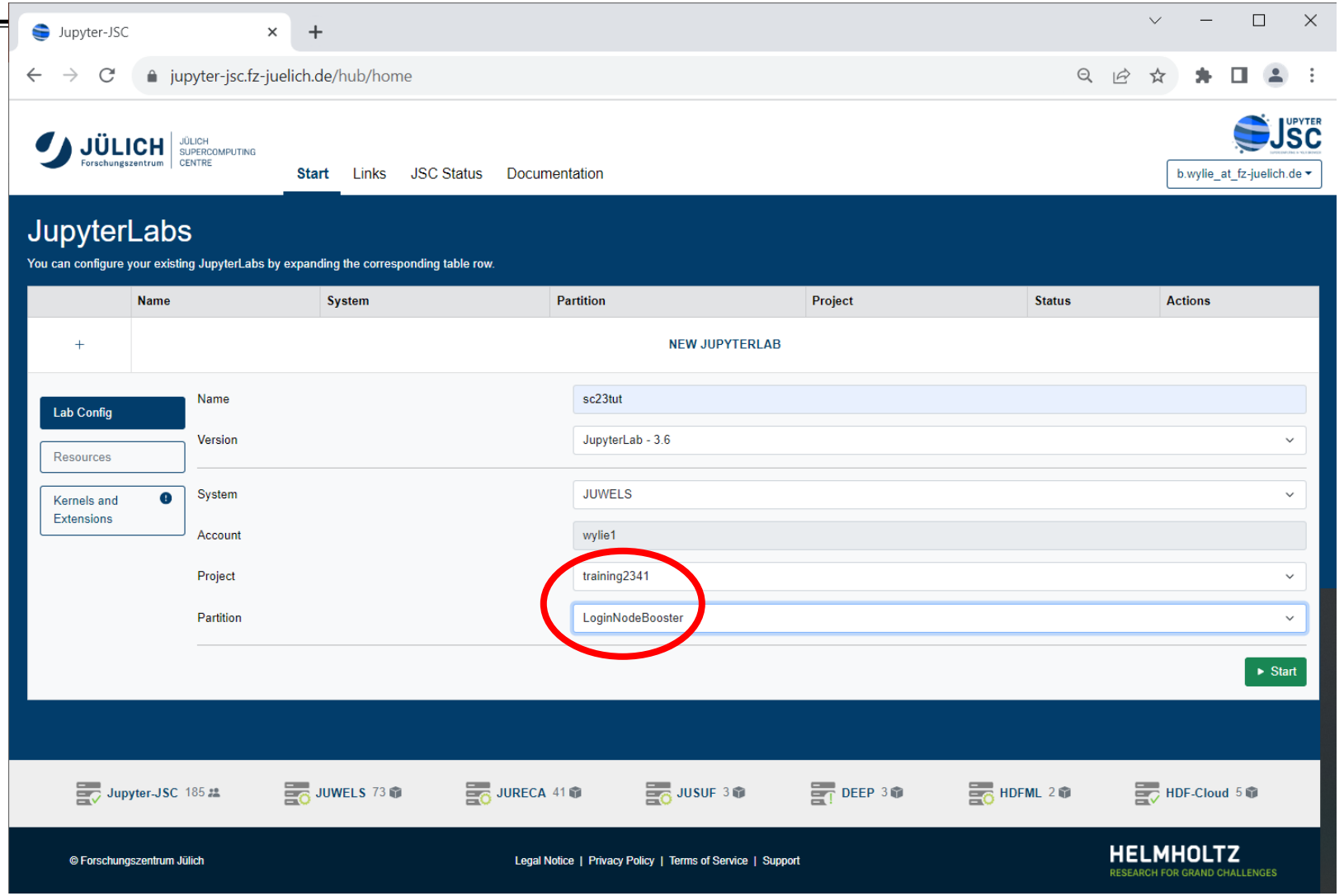
Configure

- Click on New JupyterLab configure a new service

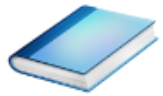


Define service

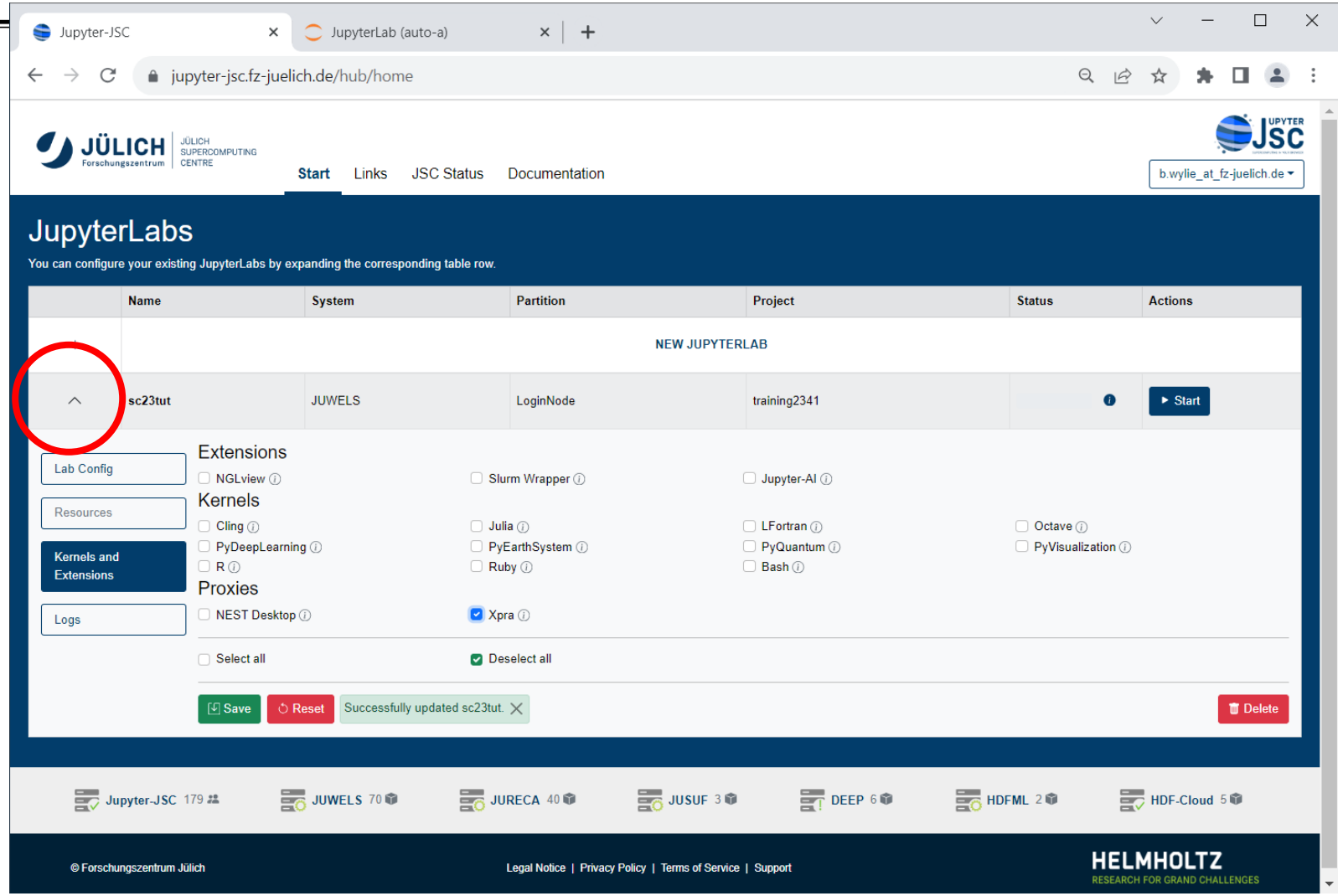
- Provide name of your choice for new lab configuration
- Configure from available (limited) options
  - Select partition **LoginNodeBooster**
- … and Start



Customize extensions

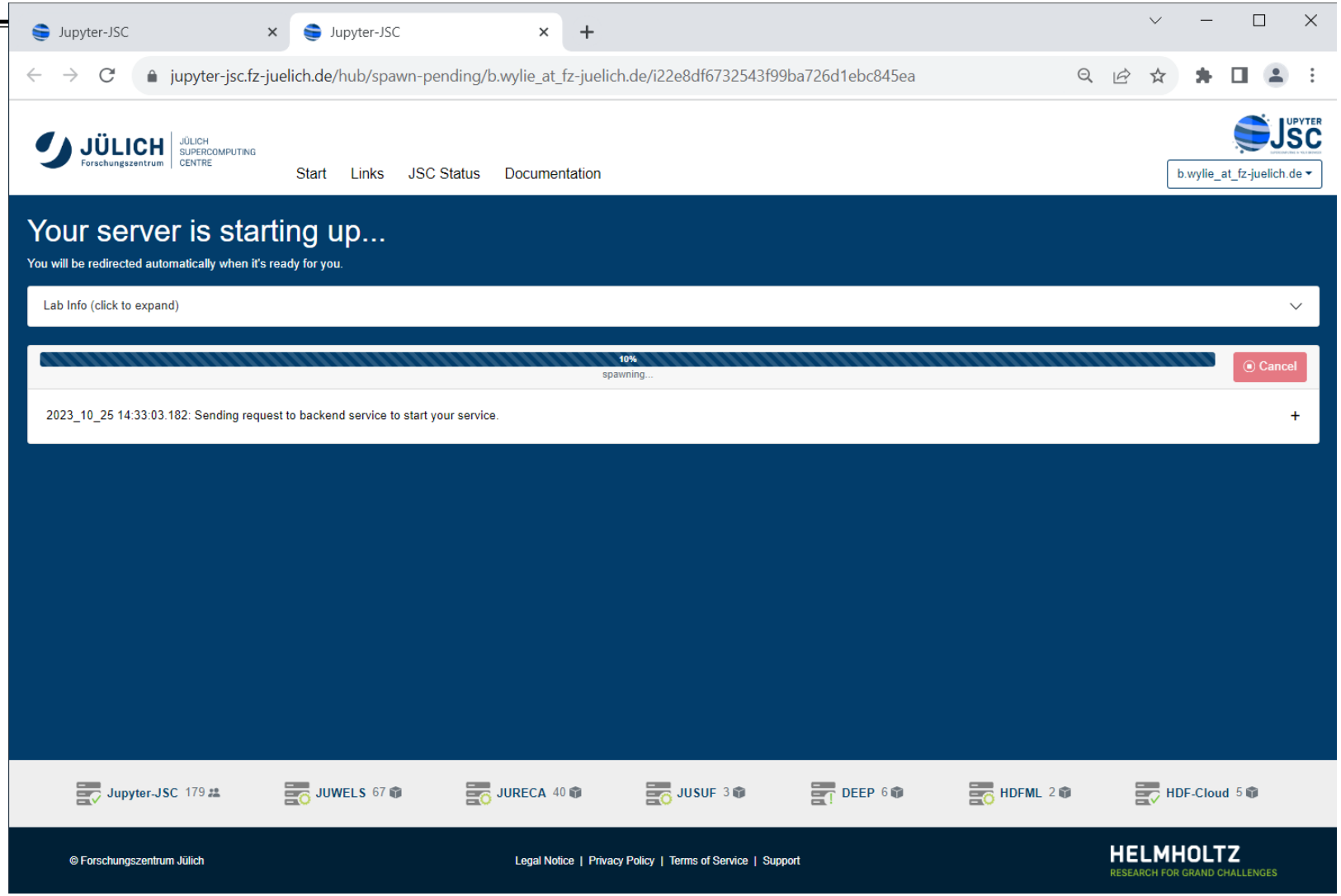


- For our tutorial, we only need Xpra desktop
  - could deselect everything else



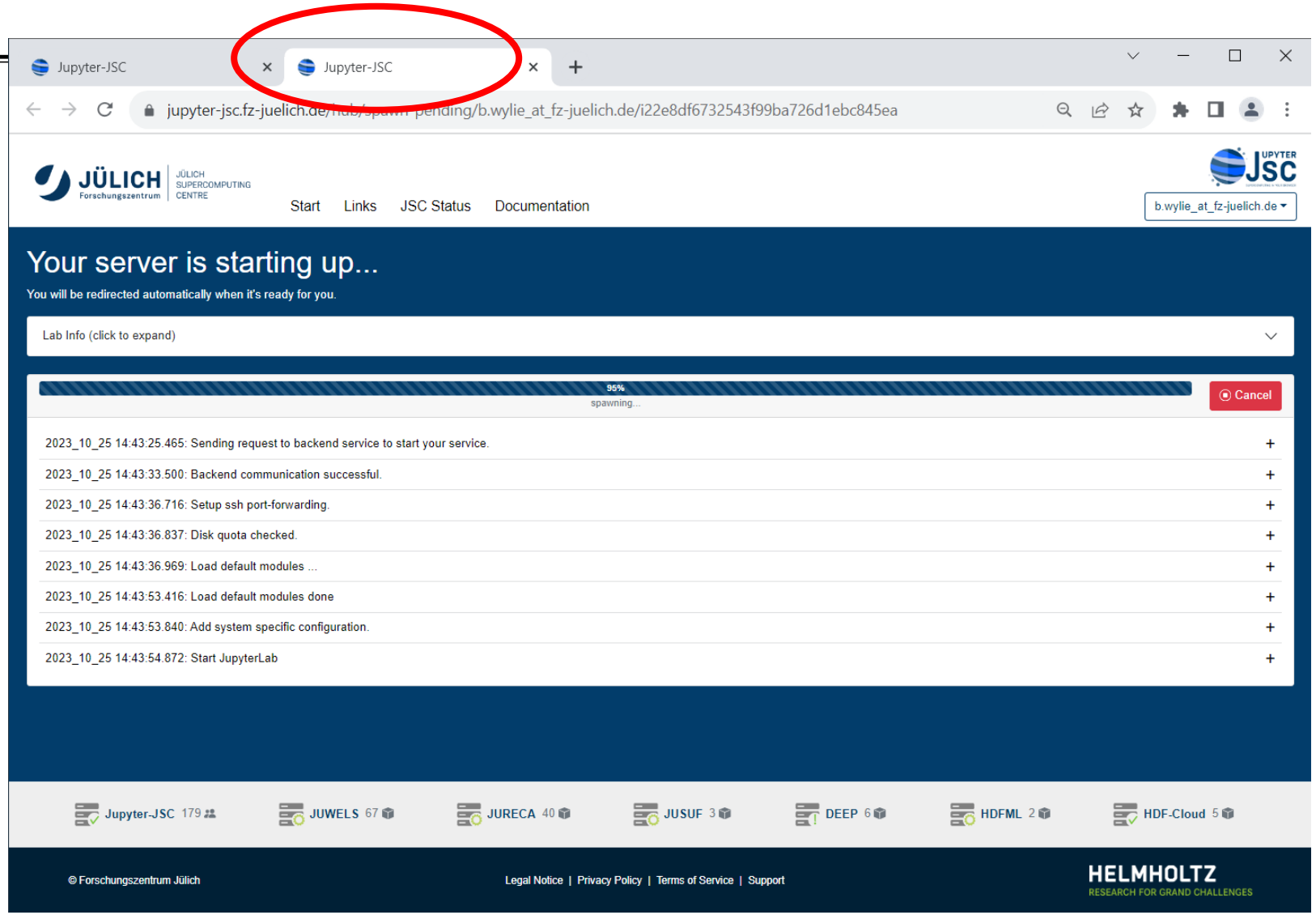
Starting

- Now wait for server starting up …

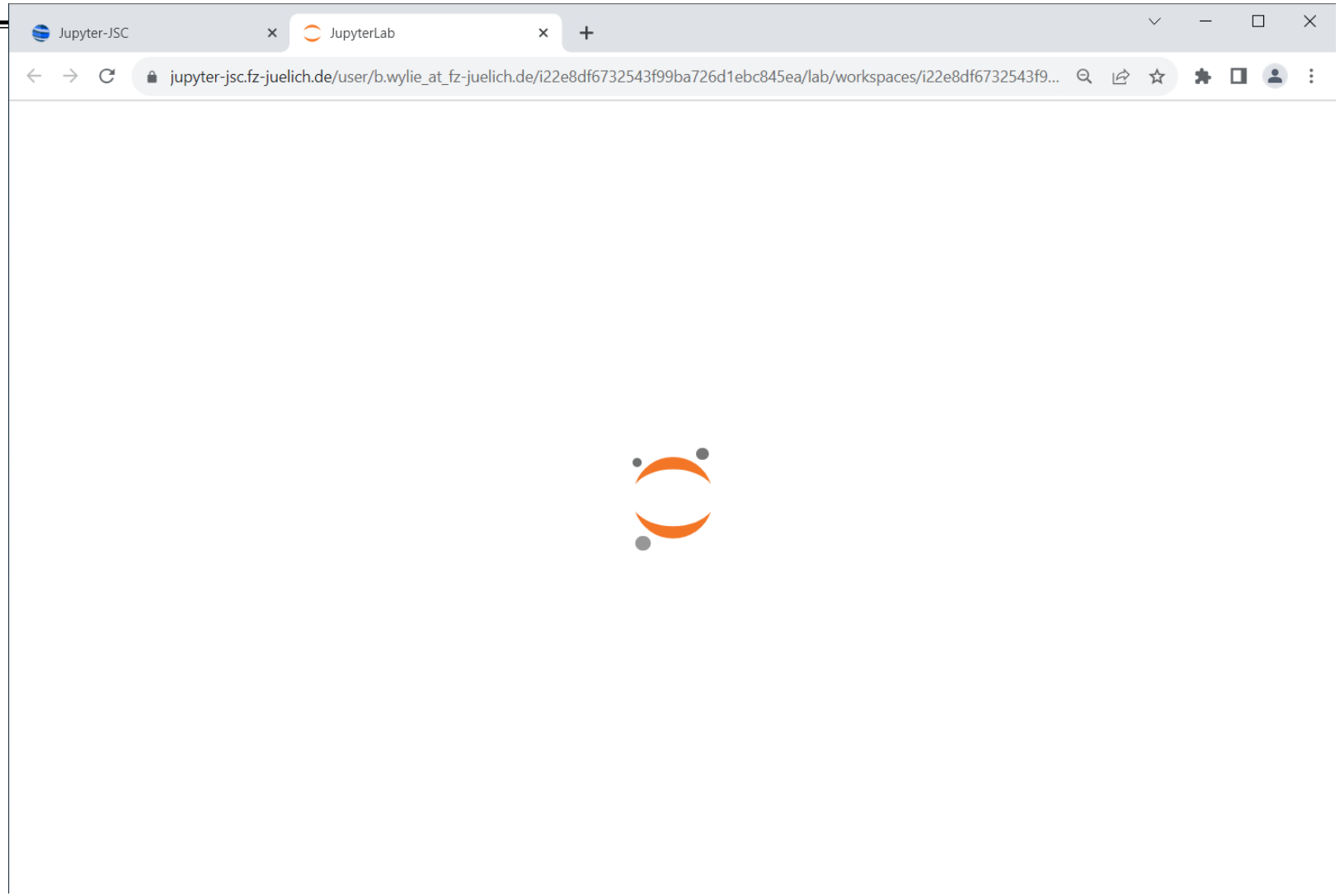


… still starting …

- Browser tab created for new service

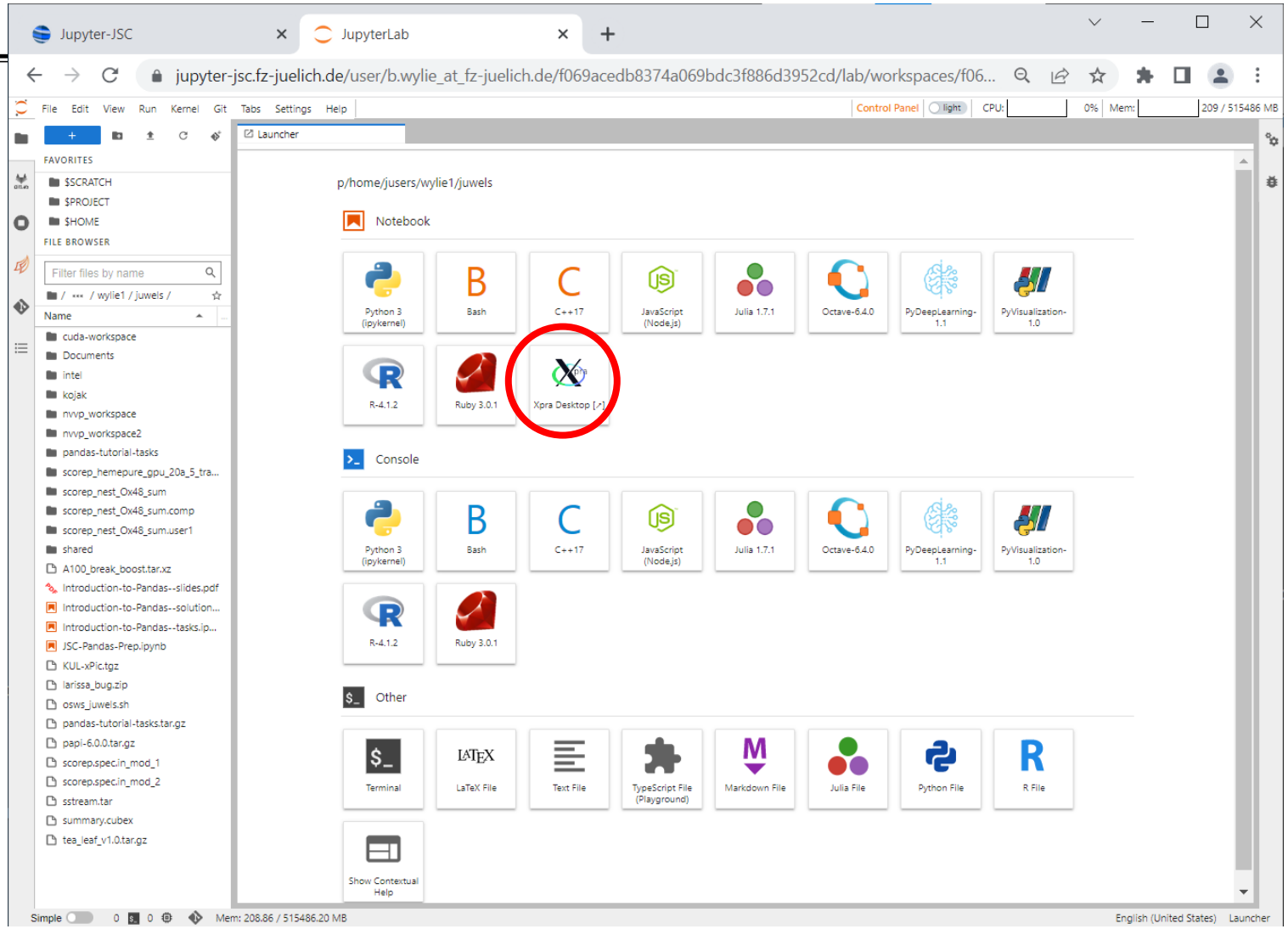


… almost there



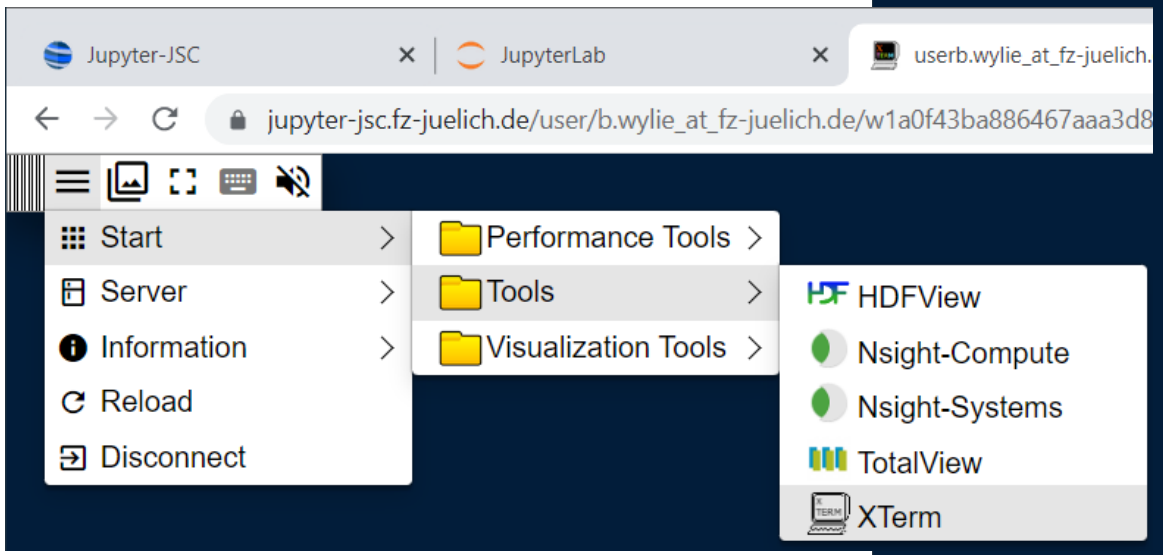
Lab service running!

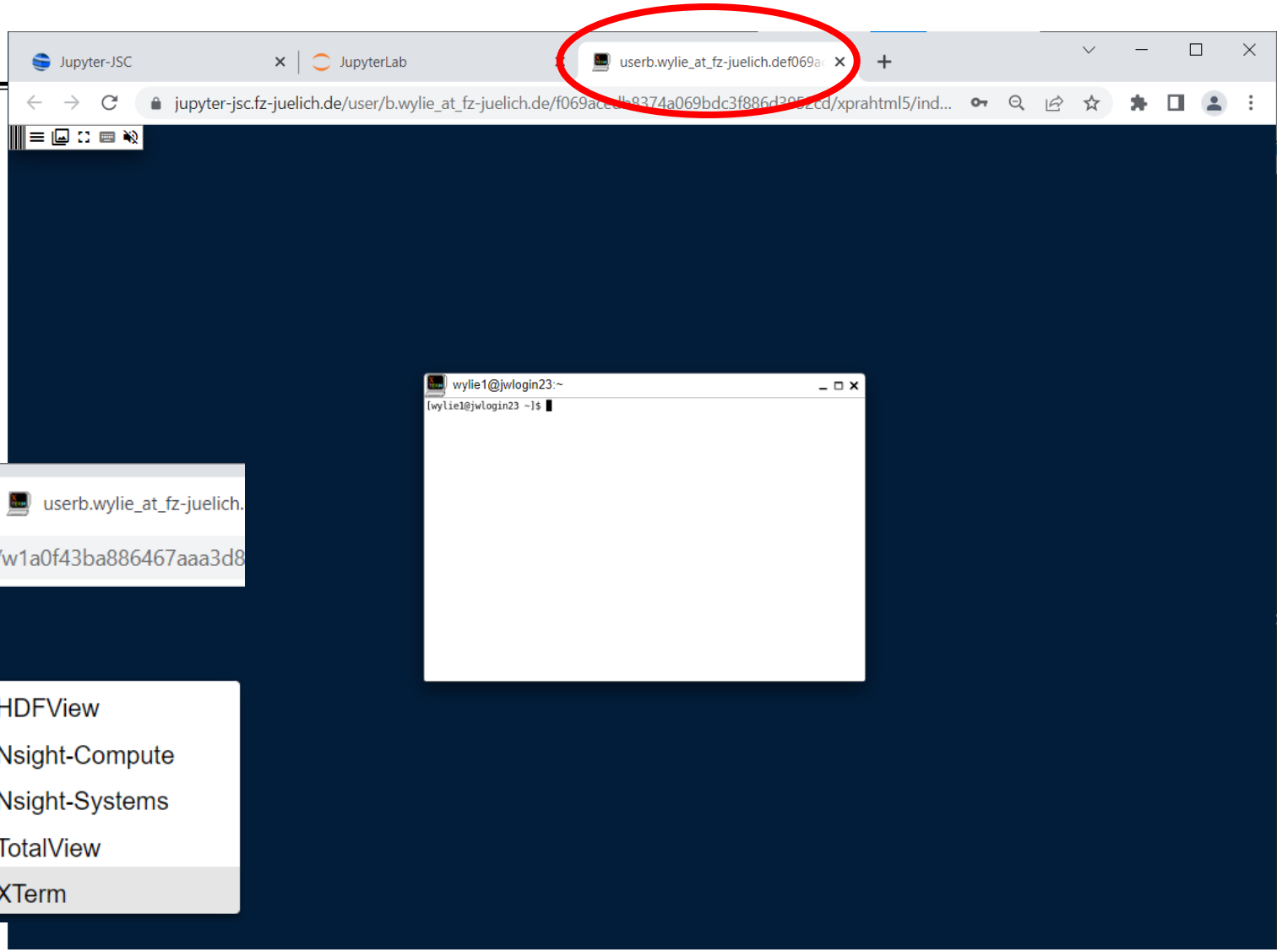
- Click on Xpra Desktop icon



Xpra desktop

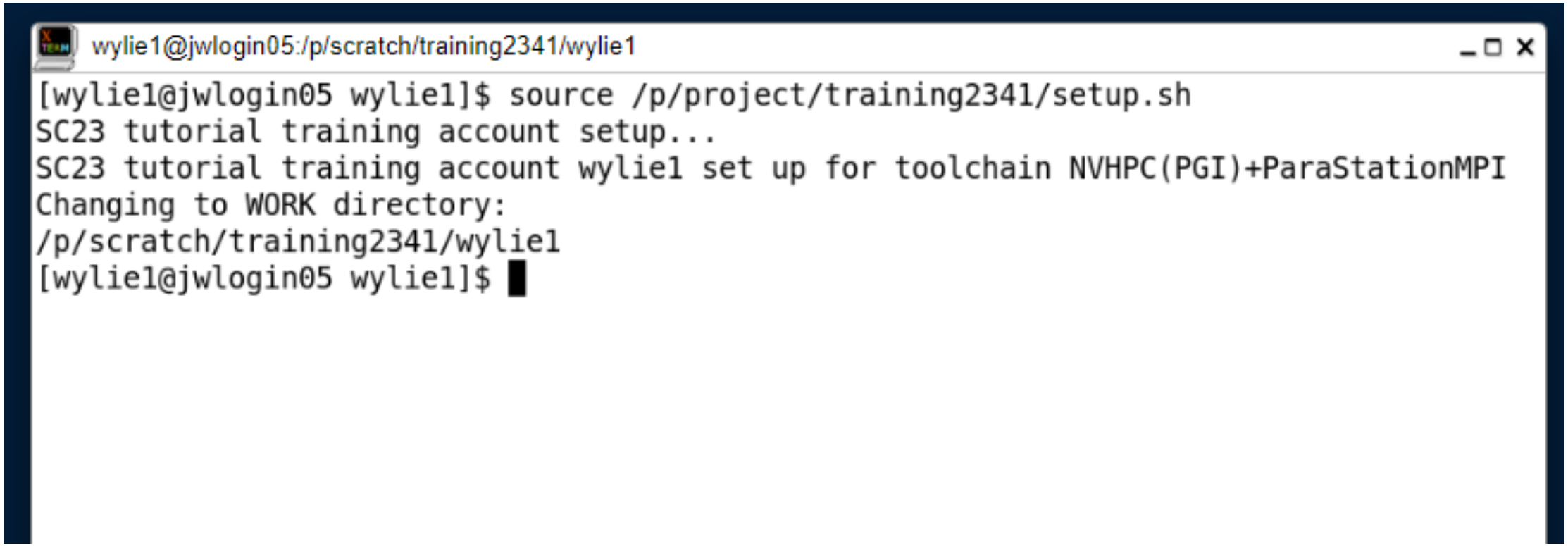
- Opens new browser tab containing virtual desktop
- Xterm opened by default
  - if not, use Start menu





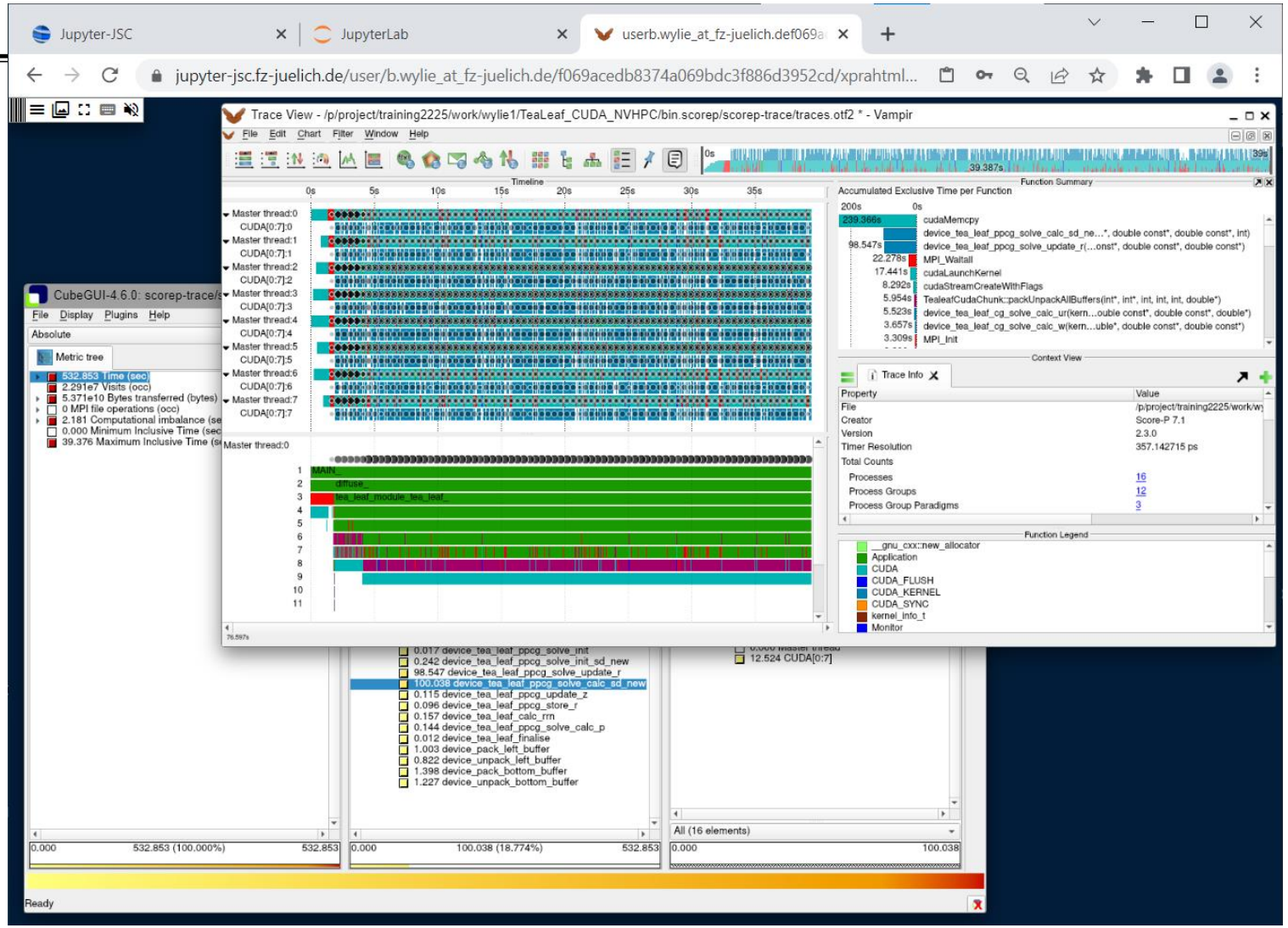
Source setup

- For tutorial, source the provided shell environment setup script



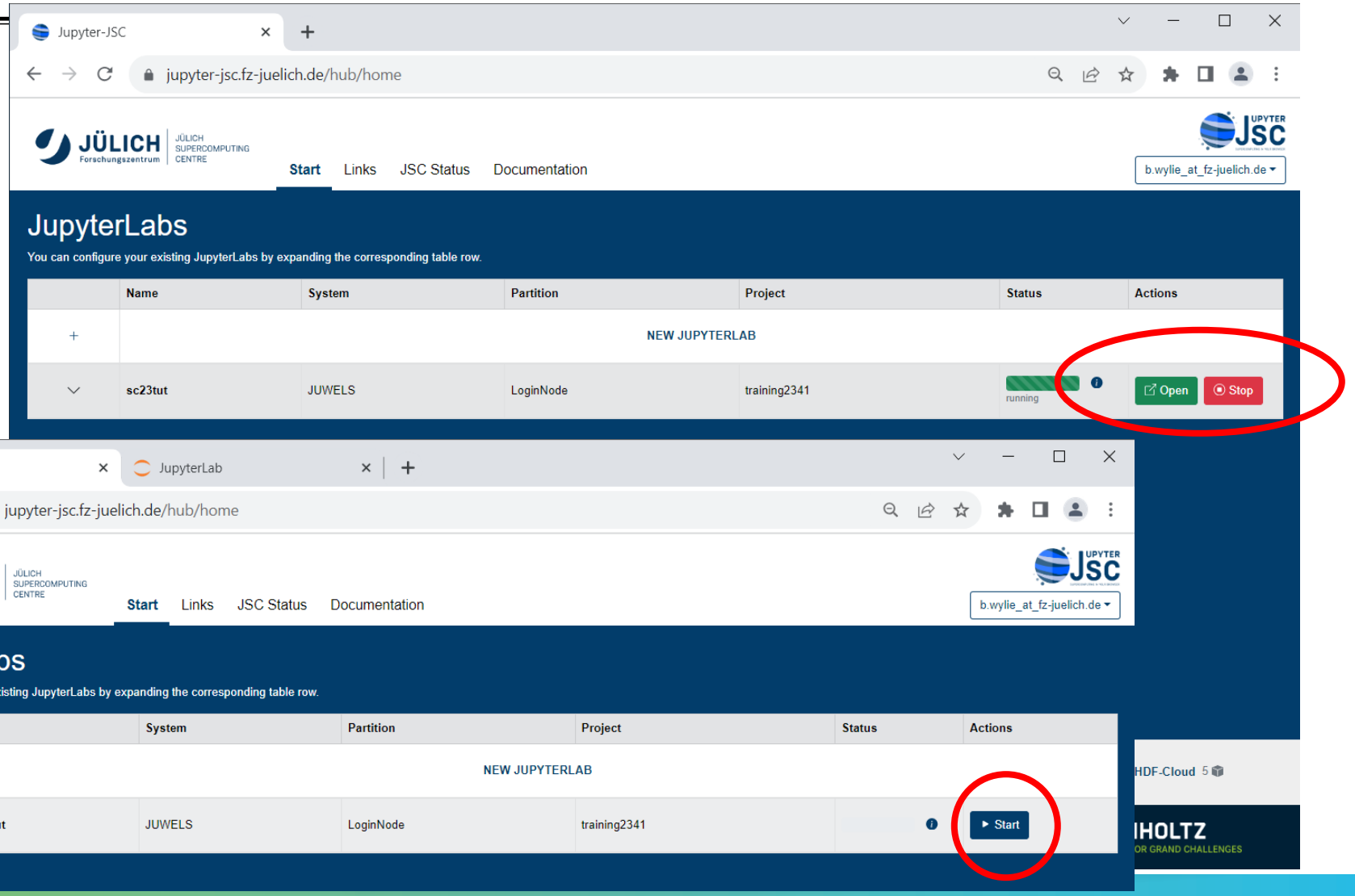
Start tools

- Graphical tools are windows within the virtual desktop



Manage service

- Stop or re-open existing service
- Re-start service when desired



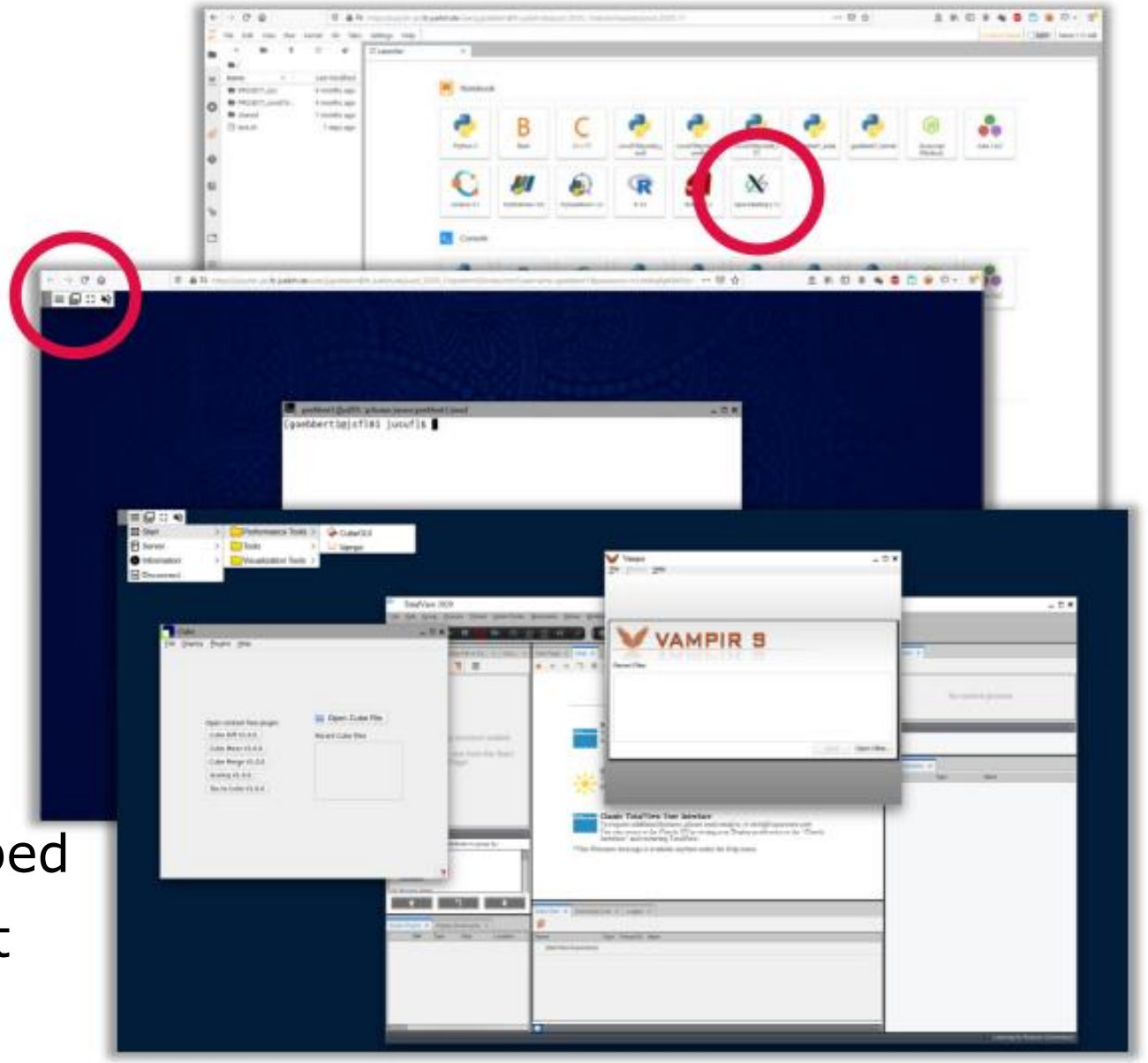
Xpra – remote desktop access

X Persistent Remote Applications

- runs X clients on a remote host and directs display to local machine
- runs in browser
- reconnection without disrupting the forwarded application session

Remote desktop runs on the same node as JupyterLab does

- killed when JupyterLab session stopped
- refresh browser tab if connection lost



https://jupyter-jsc.fz-juelich.de

Hands-on: *JUWELS Booster* (AMD EPYC Rome + 4 x A100) TeaLeaf_CUDA

VI-HPS Team

Tutorial exercise objectives

- Familiarise with usage of VI-HPS tools
  - complementary tools' capabilities & interoperability
- Prepare to apply tools productively to *your* applications(s)
- Exercise is based on a small portable benchmark code
  - unlikely to have significant optimisation opportunities

- Optional (recommended) exercise extensions
  - analyse performance of alternative configurations
  - investigate effectiveness of system-specific compiler/MPI optimisations and/or placement/binding/affinity capabilities
  - investigate scalability and analyse scalability limiters
  - compare performance on different HPC platforms
  - …

Setup for exercises

- Connect to your training account on JUWELS Booster (with X11-forwarding)

```
% ssh -X <yourid>@juwels-booster.fz-juelich.de
```

Not needed with JupyterJSC!

- Set account and default environment (NVHPC + ParaStationMPI) via helper script

```
% source /p/project/training2341/setup.sh
```

Needed in each new shell

- Copy tutorial sources to your WORK directory

```
% cd $WORK
% tar zxvf  $PROJECT/examples/tea_leaf.tar.gz
% cd TeaLeaf_CUDA
```

WORK=/p/scratch/training2341/$USER
PROJECT=/p/project/training2341

Case study: TeaLeaf_CUDA

- HPC mini-app developed by the UK Mini-App Consortium
  - Solves the linear 2D heat conduction equation on a spatially decomposed regular grid using a 5 point stencil with implicit solvers
  - Part of the Mantevo 3.0 suite
  - Available on GitHub: https://uk-mac.github.io/TeaLeaf/

- CUDA-enabled MPI version written in Fortran90, run using default testcase
  - Optional OpenMP (only used during initialization): vary the number of threads for each MPI process
  - Run with 4 MPI tasks-per-node so that each has a dedicated GPU
  - Run on 1 or more nodes to see its strong scaling performance: 2 is sufficient for exercise
  - Experiment with different MPI libraries, compilers & compiler optimisations
  - Experiment with different bindings/affinities of MPI processes and OpenMP threads
  - Experiment with different solvers (and other testcases)

- Provided version of Makefile and sources customized for this tutorial
  - builds **tea_leaf** executable in separate directory when using instrumentation

TeaLeaf_CUDA source directory

```
% ls
Benchmarks/                          generate_chunk.f90                  set_field_kernels_cuda.cu
Makefile                             generate_chunk_kernel_cuda.cu       start.f90
README.md                            global_mpi.f90                      tea.f90
build_field.f90                      host_reductions_kernel_cuda.cu      tea.in
calc_dt.f90                          init_cuda.cu                        tea_leaf.f90
config/                              initialise.f90                      tea_leaf_cg.f90
cuda_common.hpp                      initialise_chunk.f90                tea_leaf_cheby.f90
cuda_errors.cu                       initialise_chunk_kernel_cuda.cu     tea_leaf_common.f90
cuda_strings.cu                      jobscripts/                         tea_leaf_kernel_cuda.cu
cuda_strings.hpp                     kernel_files/                       tea_leaf_ppcg.f90
data.f90                             makefile.deps                       tea_solve.f90
definitions.f90                      pack_kernel_cuda.cu                 timer.f90
diffuse.f90                          parse.f90                           timer_c.c
field_summary.f90                    read_input.f90                      timestep.f90
field_summary_kernel_cuda.cu         report.f90                          update_halo.f90
ftocmacros.h                         set_field.f90                       update_halo_kernel_cuda.cu
```

25 Fortran90 modules, 1 C module, 10 CUDA modules

TeaLeaf_CUDA: Makefile

```
#Crown Copyright 2014 AWE
#
# This file is part of TeaLeaf.
#
# TeaLeaf is free software...
#
# Agnostic, platform independent Makefile for the TeaLeaf benchmark code.
# It is not meant to be clever in any way, just a simple build script.
#
# this works as well:-
#
# make COMPILER=GNU [OPENMP=1]
#

...

#PREP=scorep --cuda

MPI_COMPILER=$(PREP) mpifort
C_MPI_COMPILER=$(PREP) mpicc
# No preposition for CXX_MPI_COMPILER!
CXX_MPI_COMPILER=mpic++
NVCC=$(PREP) nvcc -ccbin $(CXX_MPI_COMPILER)

...
```

Specify the suite of compilers (and optionally OpenMP)

No instrumentation by default

Building tea_leaf

```
% make COMPILER=PGI
mpif90 -fastsse -gopt -g -c data.f90 -o data.o
[ ... ]
mpif90 -fastsse -gopt -g -c tea_leaf.f90 -o tea_leaf.o
mpif90 -fastsse -gopt -g -c diffuse.f90 -o diffuse.o
mpicc -fastsse -gopt -g -c timer_c.c -o timer_c.o
nvcc -ccbin mpicxx -std=c++14 -I/p/software/juwelsbooster/stages/2024/software/CUDA/12/include \
 -gencode arch=compute_80,code=sm_80 -restrict -Xcompiler "-fastsse -gopt -c -g" -DNO_ERR_CHK \
 -O3 -c cuda_errors.cu -o cuda_errors.o
[ ... ]
mpif90 -fastsse -gopt -g  \
 data.o definitions.o global_mpi.o tea.o report.o timer.o parse.o read_input.o initialise_chunk.o \
 build_field.o update_halo.o start.o generate_chunk.o initialise.o field_summary.o calc_dt.o \
 timestep.o set_field.o tea_leaf_common.o tea_leaf_cg.o tea_leaf_cheby.o tea_leaf_ppcg.o \
 tea_leaf_jacobi.o tea_solve.o tea_leaf.o diffuse.o \
 timer_c.o       \
 cuda_errors.o cuda_strings.o field_summary_kernel_cuda.o generate_chunk_kernel_cuda.o init_cuda.o \
 initialise_chunk_kernel_cuda.o pack_kernel_cuda.o set_field_kernel_cuda.o tea_leaf_kernel_cuda.o \
 update_halo_kernel_cuda.o \
 -L/p/software/juwelsbooster/stages/2024/software/CUDA/12/lib64 \
 -lstdc++ -lcudart \
 -o bin/tea_leaf
```

TeaLeaf_CUDA jobscript for reference execution

```
% cd bin
% cp ../jobscripts/juwelsbooster/run.sbatch .
% cat run.sbatch

#!/bin/bash
#SBATCH --job-name=TeaLeaf         # Job name
#SBATCH --nodes=2                  # Total number of nodes requested
#SBATCH --gres=gpu:4               # Number of GPUs per node
#SBATCH --ntasks-per-node=4        # Number of MPI tasks per node (one per GPU)
#SBATCH --time=00:05:00            # Max. wall-clock time (hh:mm:ss)
#SBATCH --account=training2341     # Project account to be charged
#SBATCH --output=%x.%j.out         # Output files
#SBATCH --error=%x.%j.out
#SBATCH --partition=develbooster   # Job partition

srun ./tea_leaf

% sbatch run.sbatch
```

- Copy jobscript and check/modify its contents
- then submit

TeaLeaf_CUDA Reference Execution

```
% cat TeaLeaf.<job_id>.out

Tea Version    1.400
       MPI Version
   Task Count      8

 Output file tea.out opened. All output will go there.

CUDA in rank 1 using jwb0149.juwels GPU device 0 NVIDIA A100-SXM4-40GB (0:68:0)
CUDA in rank 3 using jwb0149.juwels GPU device 0 NVIDIA A100-SXM4-40GB (0:196:0)
CUDA in rank 2 using jwb0149.juwels GPU device 0 NVIDIA A100-SXM4-40GB (0:132:0)
CUDA in rank 0 using jwb0149.juwels GPU device 0 NVIDIA A100-SXM4-40GB (0:3:0)
Solver to use: PPCG
Preconditioner to use: None
 Step       1 time   0.0000000 timestep   4.00E-03
CUDA in rank 5 using jwb0150.juwels GPU device 0 NVIDIA A100-SXM4-40GB (0:68:0)
CUDA in rank 7 using jwb0150.juwels GPU device 0 NVIDIA A100-SXM4-40GB (0:196:0)
CUDA in rank 6 using jwb0150.juwels GPU device 0 NVIDIA A100-SXM4-40GB (0:132:0)
CUDA in rank 4 using jwb0150.juwels GPU device 0 NVIDIA A100-SXM4-40GB (0:3:0)
[...]
 This test is considered PASSED
 First step overhead  -0.3670756816864014
 Wall clock     33.43737912178040
```

- Verify the reported execution configuration and that the test execution passed

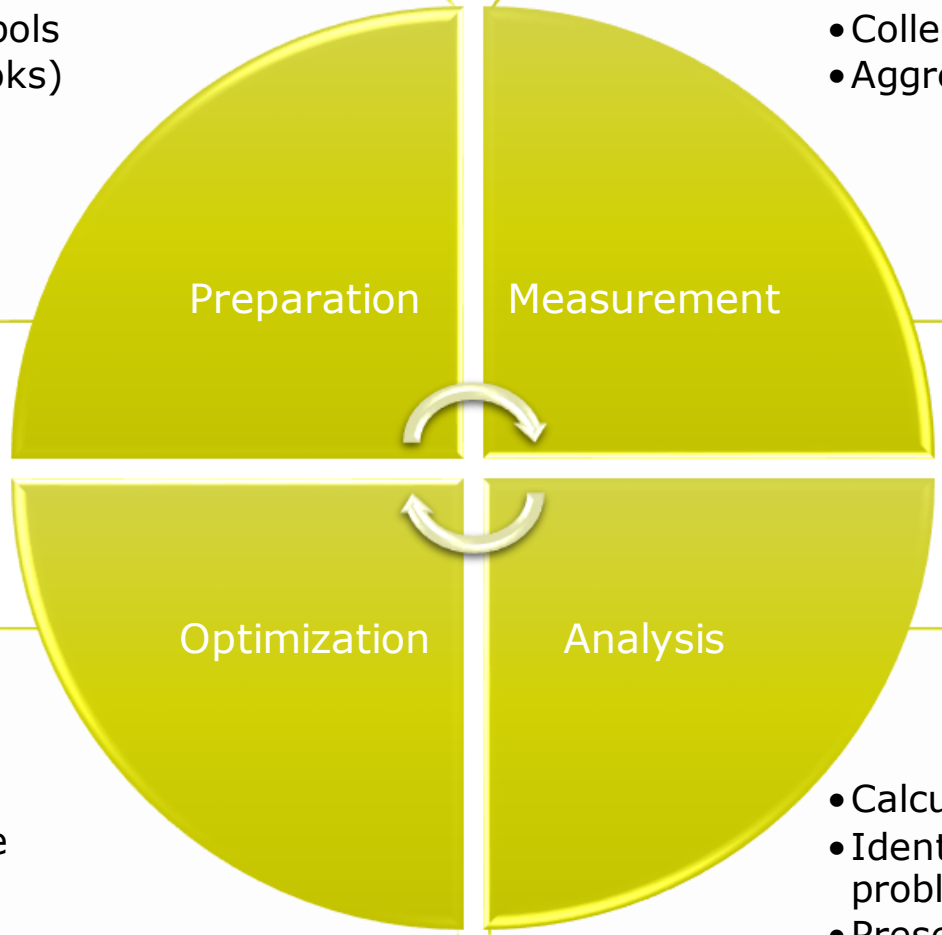
Hint: save the benchmark output (or note the run time) to be able to refer to it later

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

VI-HPS Team

Performance engineering workflow

- Prepare application with symbols
- Insert extra code (probes/hooks)



- Collection of performance data
- Aggregation of performance data

- Modifications intended to eliminate/reduce performance problem

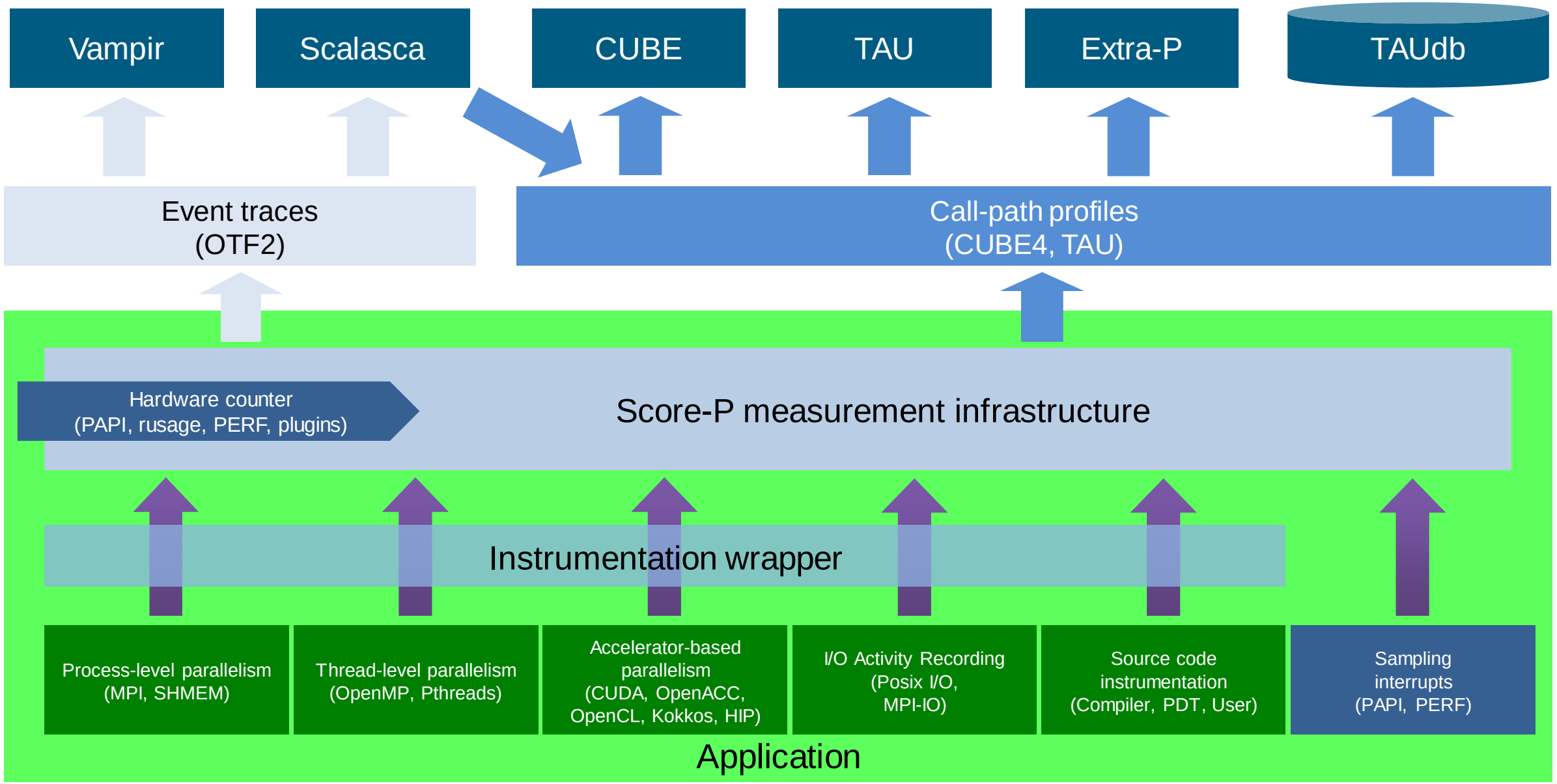
- Calculation of metrics
- Identification of performance problems
- Presentation of results

Score-P

- Infrastructure for instrumentation and performance measurements
- Instrumented application can be used to produce several results:
  - Call-path profiling: CUBE4 data format used for data exchange
  - Event-based tracing: OTF2 data format used for data exchange
  - Online profiling: In conjunction with the Periscope Tuning Framework
- Supported parallel paradigms:
  - Multi-process: MPI, SHMEM
  - Thread-parallel: OpenMP, Pthreads
  - Accelerator-based: CUDA, OpenCL, OpenACC, HIP

- Open Source; portable and scalable to all major HPC systems
- Initial project funded by BMBF
- Close collaboration with PRIMA project funded by DOE

GEFÖRDERT VOM
Bundesministerium für Bildung und Forschung


Vampir
Scalasca
CUBE
TAU
Extra-P
TAUdb
Event traces
(OTF2)
Call-path profiles
(CUBE4, TAU)
Hardware counter
(PAPI, rusage, PERF, plugins)
Score-P measurement infrastructure
Instrumentation wrapper
Process-level parallelism
(MPI, SHMEM)
Thread-level parallelism
(OpenMP, Pthreads)
Accelerator-based
parallelism
(CUDA, OpenACC,
OpenCL, Kokkos, HIP)
I/O Activity Recording
(Posix I/O,
MPI-IO)
Source code
instrumentation
(Compiler, PDT, User)
Sampling
interrupts
(PAPI, PERF)
Application

Hands-on: TeaLeaf_CUDA

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

Local installation (JUWELS Booster)

- Set account and default environment (NVHPC + ParaStationMPI) via helper script:

```
% source /p/project/training2341/setup.sh
```

- Load the modules for the tool environment:

```
% module load Score-P CubeGUI
```

- Copy tutorial sources to your WORK directory (or your personal workspace)
  - Only required if not done already (for opening exercise)

```
% cd $WORK
% tar xf $PROJECT/examples/tea_leaf.tar.gz
% cd TeaLeaf_CUDA
```

TeaLeaf_CUDA: Makefile

```
#Crown Copyright 2014 AWE
#
# This file is part of TeaLeaf.
#
# TeaLeaf is free software...
#
# Agnostic, platform independent Makefile for the TeaLeaf benchmark code.
# It is not meant to be clever in any way, just a simple build script.
#
# this works as well:-
#
# make COMPILER=PGI [OPENMP=1]
#

...

#PREP=scorep --cuda

MPI_COMPILER=$(PREP) mpifort
C_MPI_COMPILER=$(PREP) mpicc
# No preposition for CXX_MPI_COMPILER!
CXX_MPI_COMPILER=mpic++
NVCC=$(PREP) nvcc -ccbin $(CXX_MPI_COMPILER)

...
```

Specify the suite of compilers (and optionally OpenMP)

No instrumentation by default

Uncomment or set PREP to instrumenter preposition

Instrumenting tea_leaf

```
% make COMPILER=PGI PREP="scorep --cuda"

scorep --cuda mpif90 -fastsse -gopt -g -c data.f90 -o data.o
[…]
mpicc -fastsse -gopt -g -c timer_c.c -o timer_c.o
scorep --cuda nvcc -ccbin mpicxx -I/p/software/juwelsbooster/stages/2024/software/CUDA/12/include \
 -std=c++14 -gencode arch=compute_80,code=sm_80 -restrict -Xcompiler "-fastsse -gopt -c -g"  \
 -DNO_ERR_CHK -O3 -c cuda_errors.cu -o cuda_errors.o
[…]
scorep --cuda mpif90 -fastsse -gopt     -g \
data.o definitions.o global_mpi.o tea.o report.o timer.o parse.o read_input.o initialise_chunk.o \
build_field.o update_halo.o start.o generate_chunk.o initialise.o field_summary.o calc_dt.o \
timestep.o set_field.o tea_leaf_common.o tea_leaf_cg.o tea_leaf_cheby.o tea_leaf_ppcg.o \
tea_leaf_jacobi.o tea_solve.o tea_leaf.o diffuse.o  timer_c.o        \
cuda_errors.o cuda_strings.o field_summary_kernel_cuda.o generate_chunk_kernel_cuda.o init_cuda.o \
initialise_chunk_kernel_cuda.o pack_kernel_cuda.o set_field_kernel_cuda.o tea_leaf_kernel_cuda.o \
update_halo_kernel_cuda.o \
-L/p/software/juwelsbooster/stages/2024/software/CUDA/12/lib64 \
-lstdc++ -lcudart \
-o bin.scorep/tea_leaf
```

Measurement configuration: scorep-info

```
% 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
 [...]
SCOREP_CUDA_ENABLE
  Description: CUDA measurement features
 [... More configuration variables ...]
```

- Score-P measurements are configured via environmental variables

Required for CUDA measurements. [yes|default] recommended to start with.

Summary measurement collection

```
% cd bin.scorep
% cp ../jobscripts/juwelsbooster/scorep.sbatch .
% cat scorep.sbatch
...
# Score-P measurement configuration
export SCOREP_CUDA_ENABLE=default
export SCOREP_CUDA_BUFFER=48M

export SCOREP_EXPERIMENT_DIRECTORY=scorep-tea_leaf-8
#export SCOREP_FILTERING_FILE=../config/scorep.filter

#export SCOREP_ENABLE_TRACING=true
#export SCOREP_TOTAL_MEMORY=250M

# Run the application
srun ./tea_leaf

% sbatch scorep.sbatch
```

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

Leave these lines commented out for the moment

- Submit job

TeaLeaf_CUDA Reference Execution

```
% cat TeaLeaf_scorep.<job_id>.out

   Tea Version 1.400
       MPI Version
    OpenMP Version
   Task Count:     8

Input read finished.
Using CUDA Kernels

[…]

Solver to use: PPCG
Preconditioner to use: None
Test problem   6 is within   0.1397839E-05% of the expected solution
 This test is considered PASSED
 First step overhead  -0.5253252983093262
 Wall clock     38.01989197731018
```

- Verify the reported execution configuration and that the test execution passed

Compare to previous reference execution without instrumentation

TeaLeaf summary analysis report examination

```
% ls
tea_leaf tea.in TeaLeaf_scorep.<job_id>.out
scorep.sbatch scorep-tea_leaf-8/

% ls scorep-tea_leaf-8
MANIFEST.md  profile.cubex  scorep.cfg




% cube scorep-tea_leaf-8/profile.cubex

      [CUBE GUI showing summary analysis report]
```

Hint:
Copy ‘profile.cubex’ to local system (laptop) using ‘scp’ to improve responsiveness of GUI

- Creates experiment directory including
  - A brief content overview (MANIFEST.md)
  - A record of the measurement configuration (scorep.cfg)
  - The analysis report that was collated after measurement (profile.cubex)
- Interactive exploration with Cube

Further information

- Community instrumentation & measurement infrastructure
  - Instrumentation (various methods)
  - Basic and advanced profile generation
  - Event trace recording
  - Online access to profiling data
- Available under 3-clause BSD open-source license
- Documentation & Sources:
  - https://www.score-p.org
- User guide also part of installation:
  - <prefix>/share/doc/scorep/{pdf,html}/
- Support and feedback: support@score-p.org
- Subscribe to news@score-p.org, to be up to date

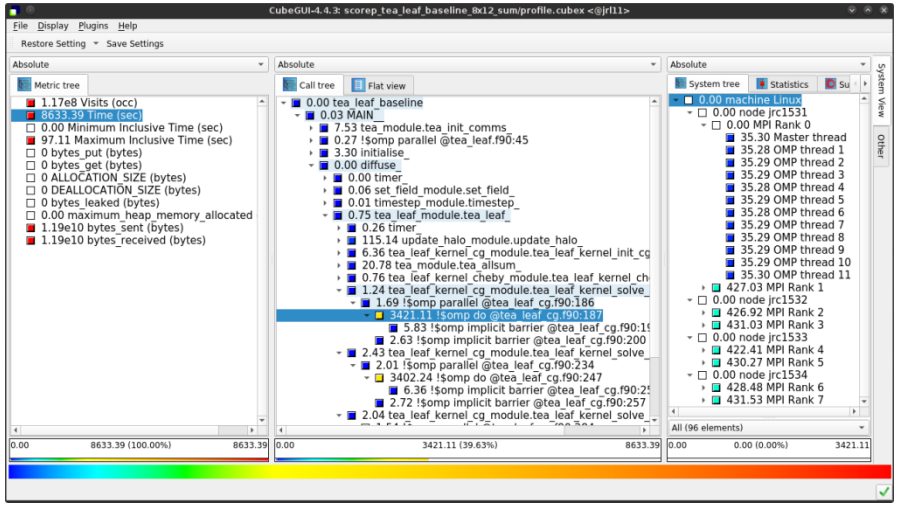
Analysis report examination with Cube

Brian Wylie
Jülich Supercomputing Centre

Cube

CubeLib DOI 10.5281/zenodo.7737408
CubeGUI DOI 10.5281/zenodo.7737411

- Parallel program analysis report exploration tools
  - Libraries for XML+binary report reading & writing
  - Algebra utilities for report processing
  - GUI for interactive analysis exploration
    - Requires Qt ≥ 5



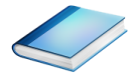
- Originally developed as part of the Scalasca toolset

- Now available as a separate components
  - Can be installed independently of Score-P, e.g., on laptop or desktop
  - Latest release: Cube v4.8.2 (Sept 2023)

Note: source distribution tarballs for Linux, as well as binary packages provided for Windows & MacOS, from **www.scalasca.org** website in software/Cube-4x

Cube GUI

mailto: **scalasca@fz-juelich.de**



- Run ***remote*** ***(Jupyter-JSC)***
  - start Jupyter-JSC and then start Xpra desktop
  - load cube module and start cube

```
[jwlogin~]$ module load CubeGUI
[jwlogin~]$ cube ./scorep-20221114_*/profile.cubex
```

- Run ***remote*** *(ssh)*
  - start X server (e.g., Xming) locally
  - connect to juwels with X forwarding enabled

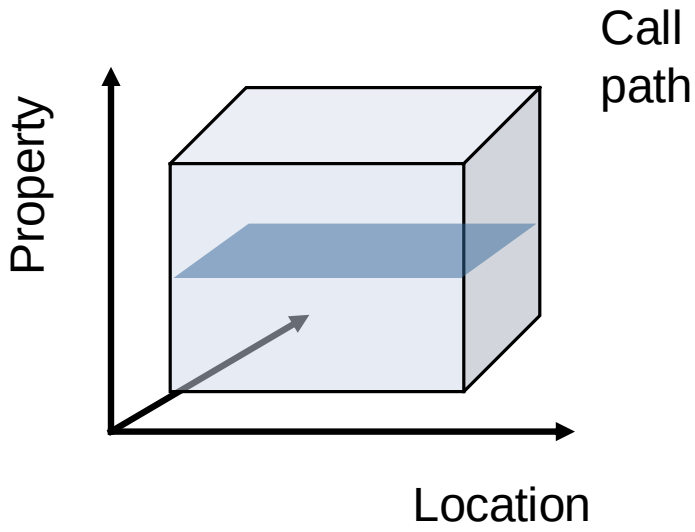
```
desk$ ssh -X <yourid>@juwels-booster.fz-juelich.de
Welcome to …
[jwlogin~]$ module load CubeGUI
[jwlogin~]$ cube ./scorep-20221114_*/profile.cubex
```

- Install & run ***local***
  - install Cube GUI locally on desktop
    - binary packages available for MacOS & Windows and externally provided by OpenHPC and various Linux distributions
    - source package available for Linux, requires Qt
      - configure/build/install manually or use your favourite framework (e.g. Spack or EasyBuild)
  - copy .cubex file (or entire scorep directory) to desktop from remote system
    OR locally mount remote filesystem
  - start cube locally

```
desk$ mkdir $HOME/mnt
desk$ sshfs [user@]remote.sys:[dir] $HOME/mnt
desk$ cd $HOME/mnt
desk$ cube ./scorep-20221114_*/profile.cubex
```

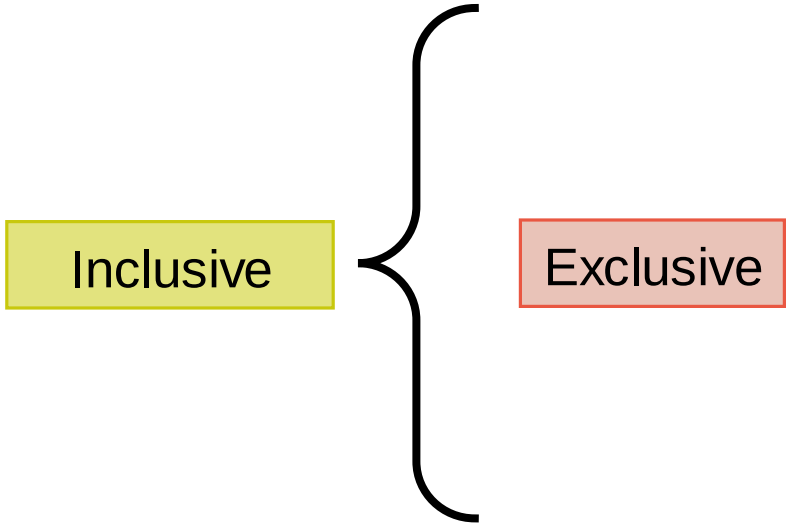
Analysis presentation and exploration

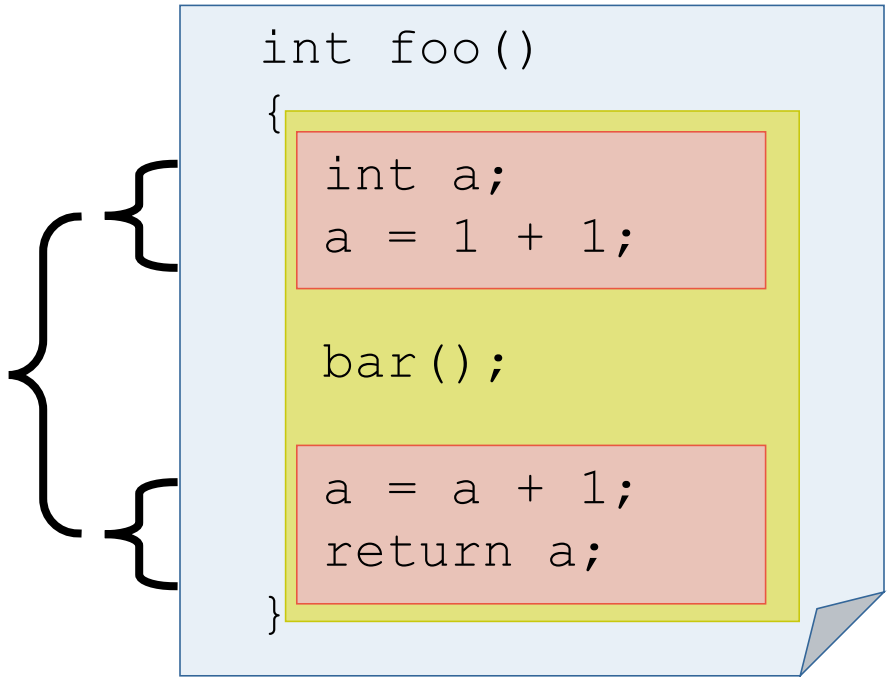
- Representation of values (severity matrix) on three hierarchical axes
  - Performance property (metric)
  - Call path (program location)
  - System location (process/thread)
- Three coupled tree browsers
- Cube displays severities
  - As *value*: for precise comparison
  - As *colour*: for easy identification of hotspots
  - *Inclusive* value when closed & *exclusive* value when expanded
  - Customizable via display *modes*



Inclusive vs. exclusive values

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





Demo:
TeaLeaf case study

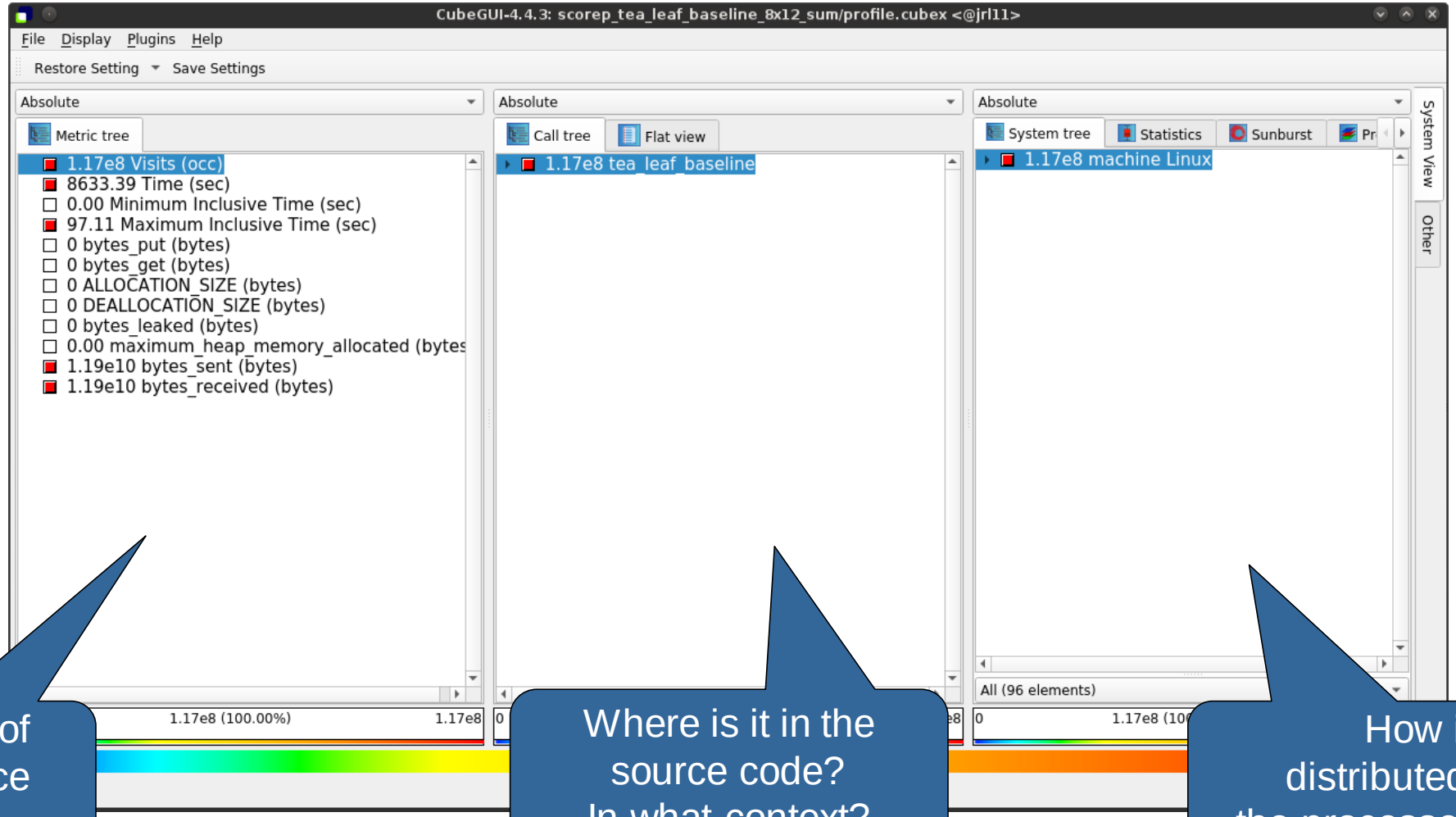
Case study: TeaLeaf

- HPC mini-app developed by the UK Mini-App Consortium
  - Solves the linear 2D heat conduction equation on a spatially decomposed regular grid using a 5 point stencil with implicit solvers
  - Part of the Mantevo 3.0 suite
  - Available on GitHub: http://uk-mac.github.io/TeaLeaf/

- Measurements of TeaLeaf reference v1.0 taken on Jureca cluster @ JSC
  - Using Intel 19.0.3 compilers, Intel MPI 2019.3, and Score-P 5.0
  - Run configuration
    - 8 MPI ranks with 12 OpenMP threads each

```
% cd ~/workshop-vihps/Experiments
% cube scorep_tea_leaf_baseline_8x12_sum/profile.cubex
                                    [GUI showing summary analysis report]
```

Score-P analysis report exploration (opening view)

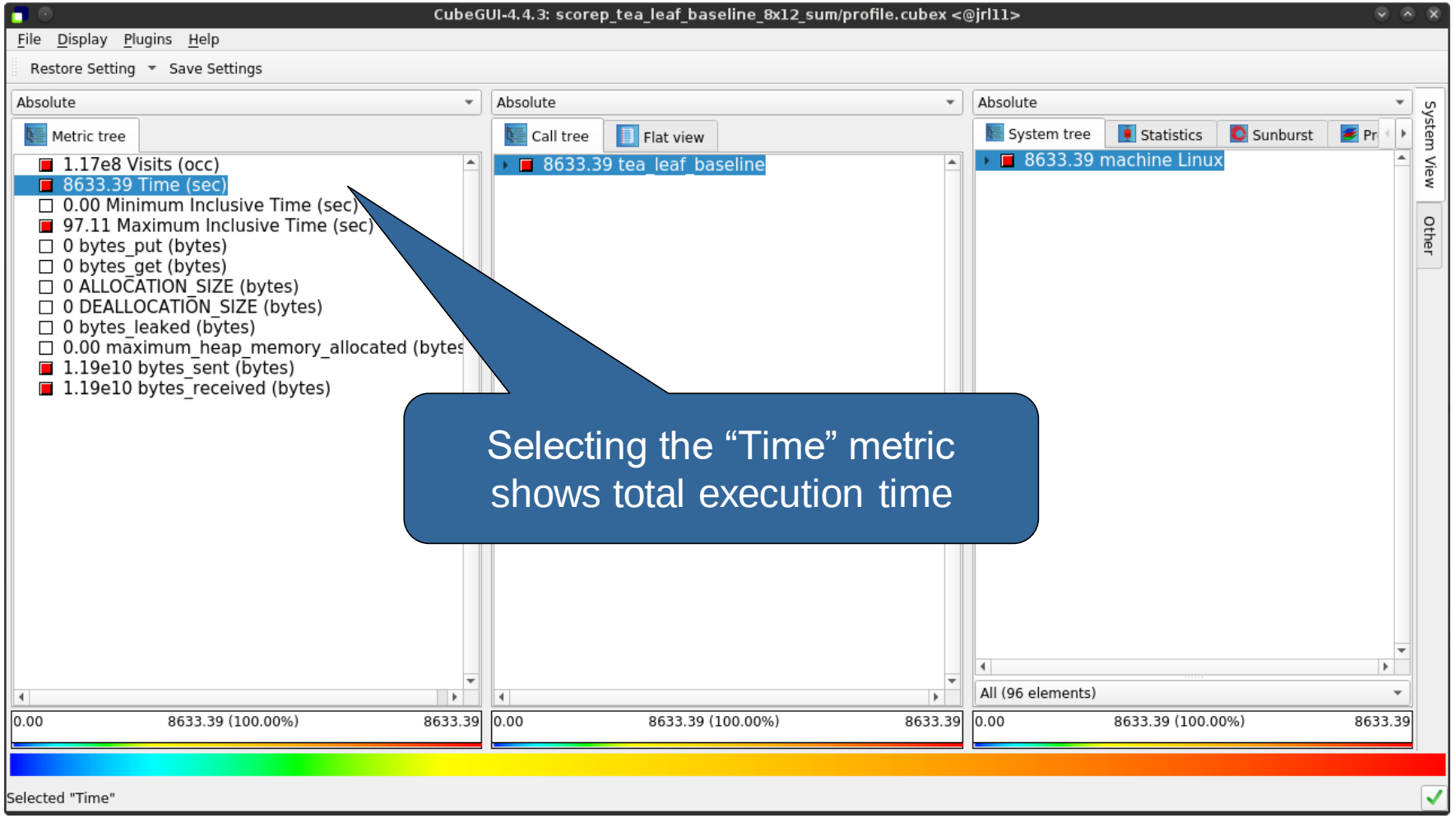


What kind of performance metric?

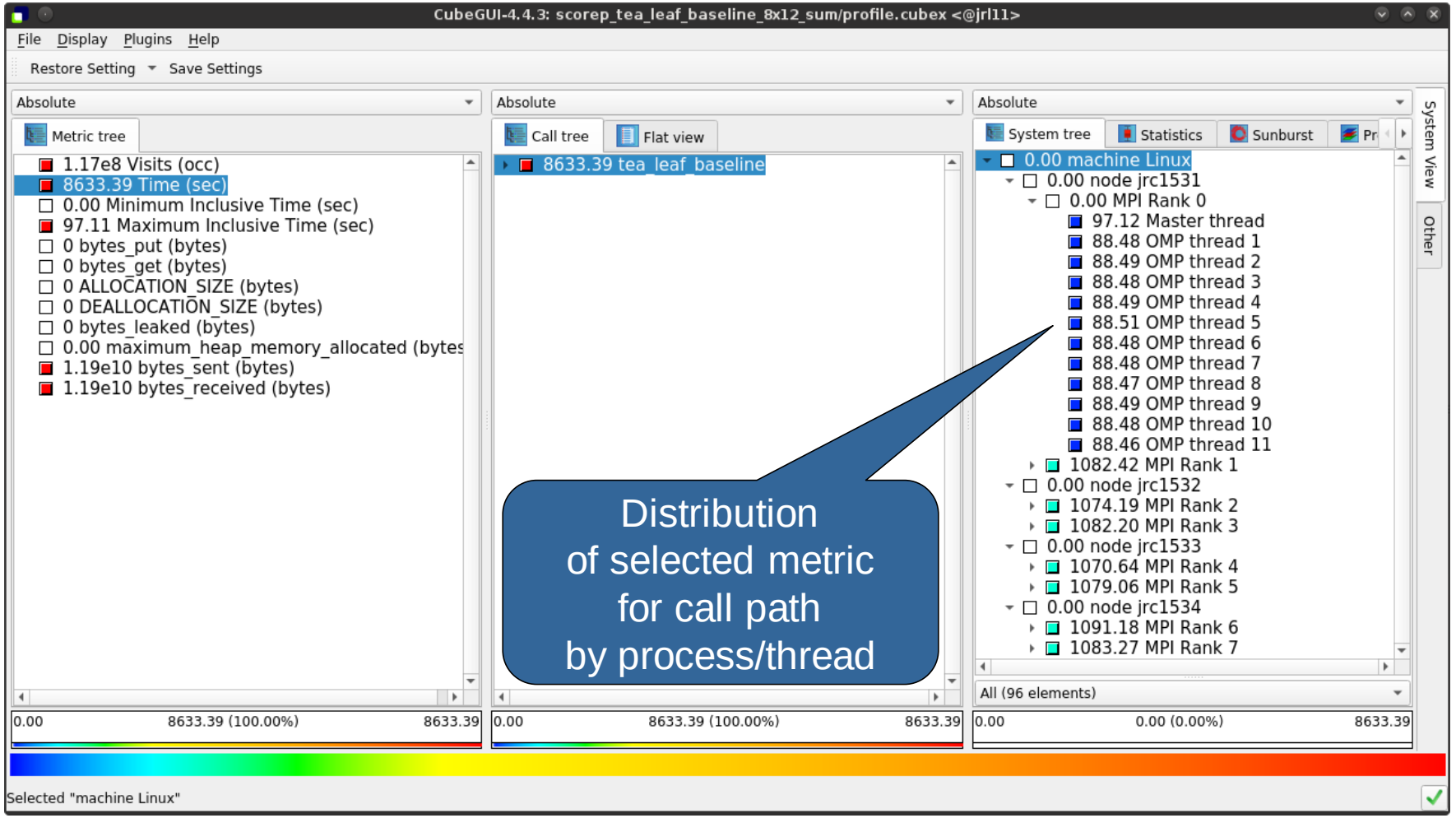
Where is it in the source code? In what context?

How is it distributed across the processes/threads?

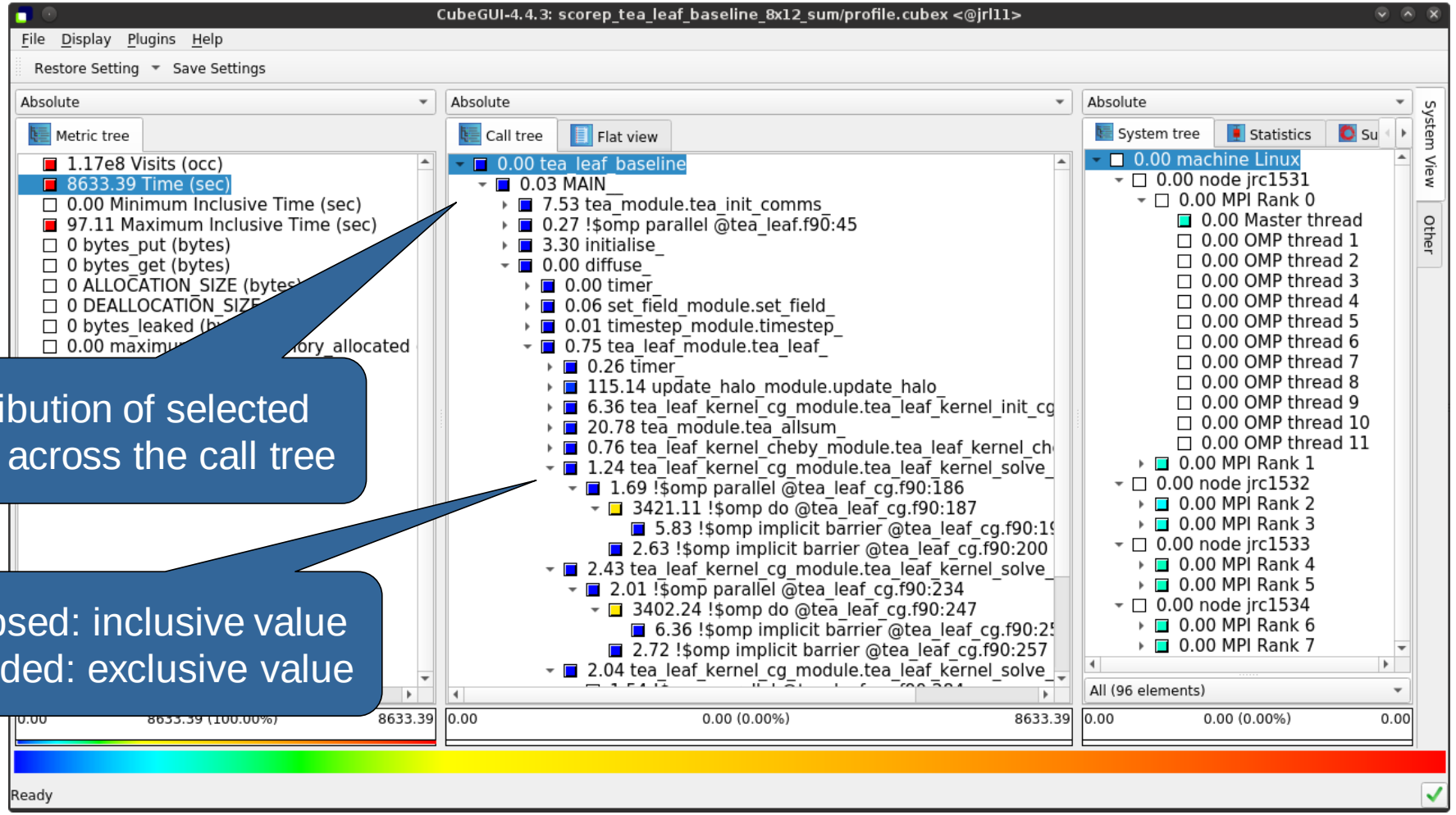
Metric selection



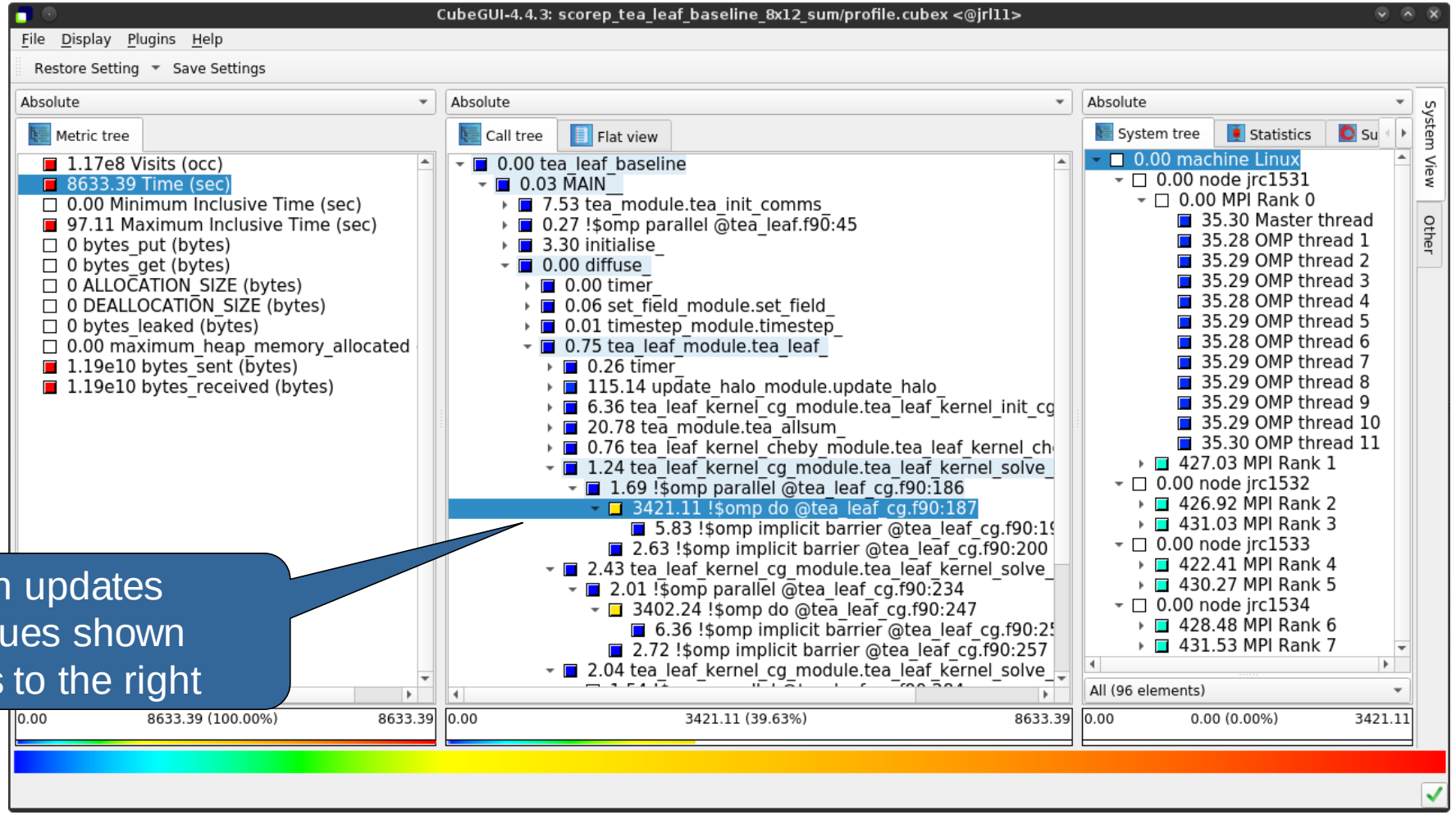
Expanding the system tree



Expanding the call tree

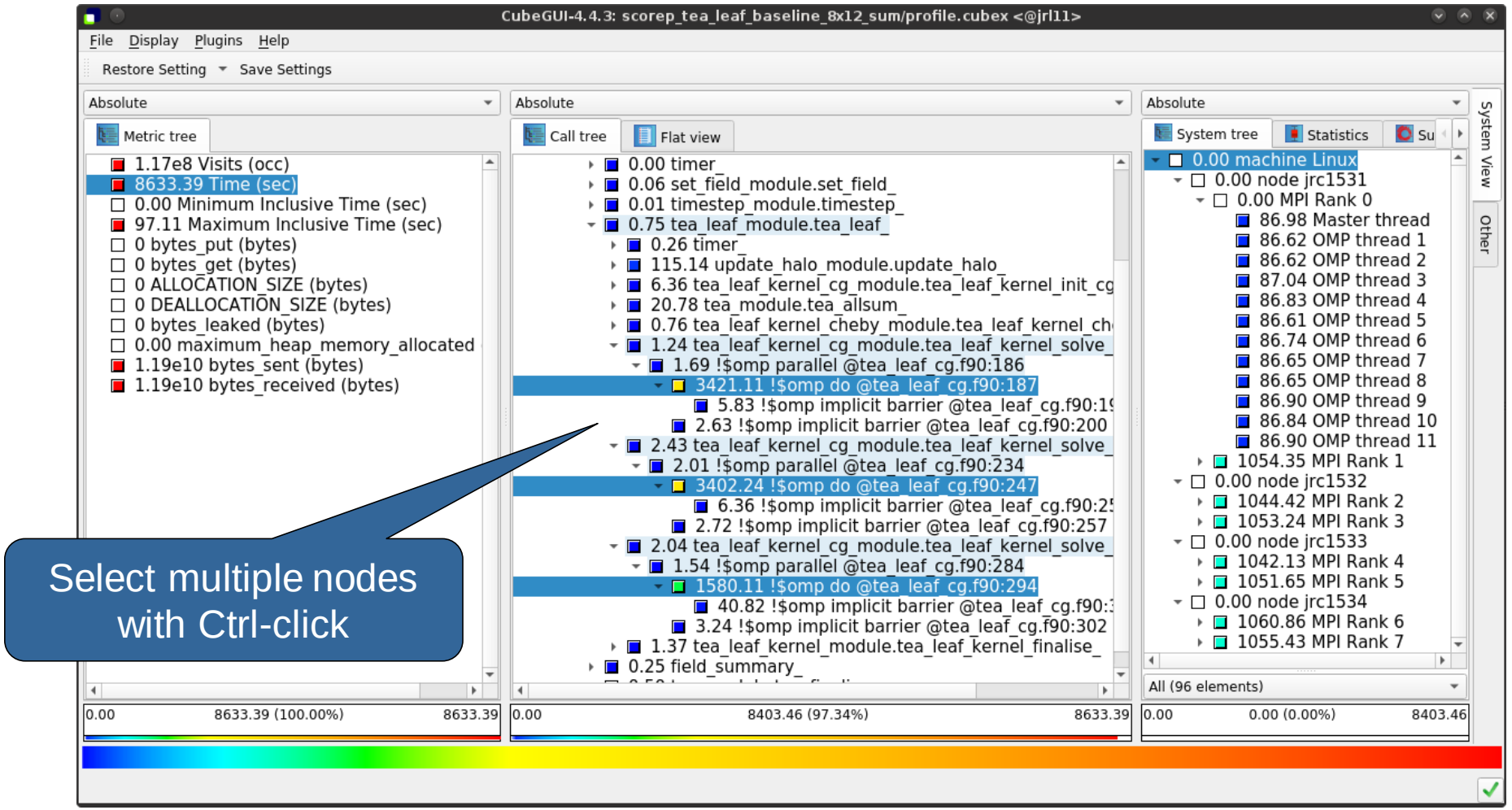


Selecting a call path

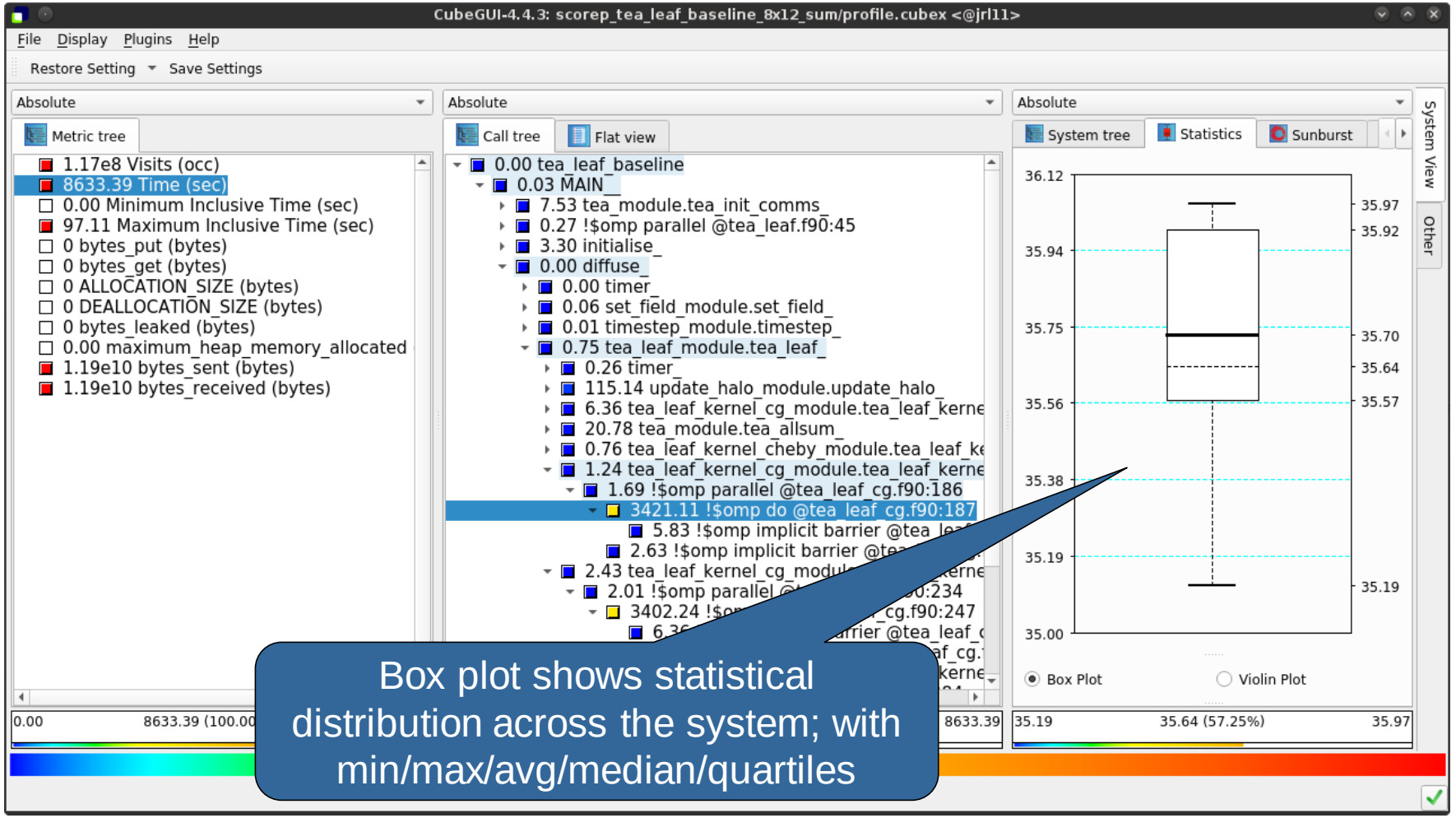


Selection updates metric values shown in columns to the right

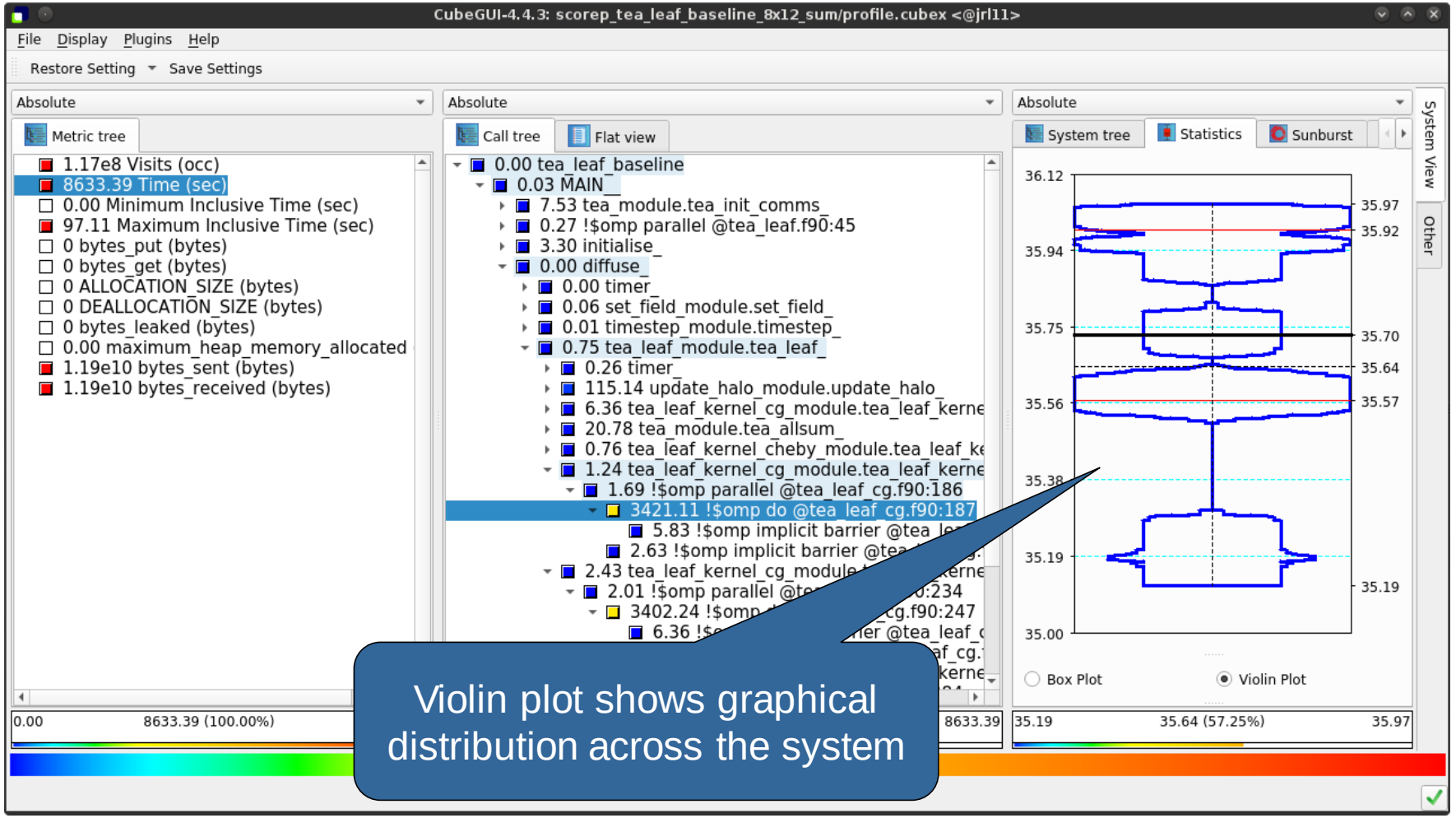
Multiple selection



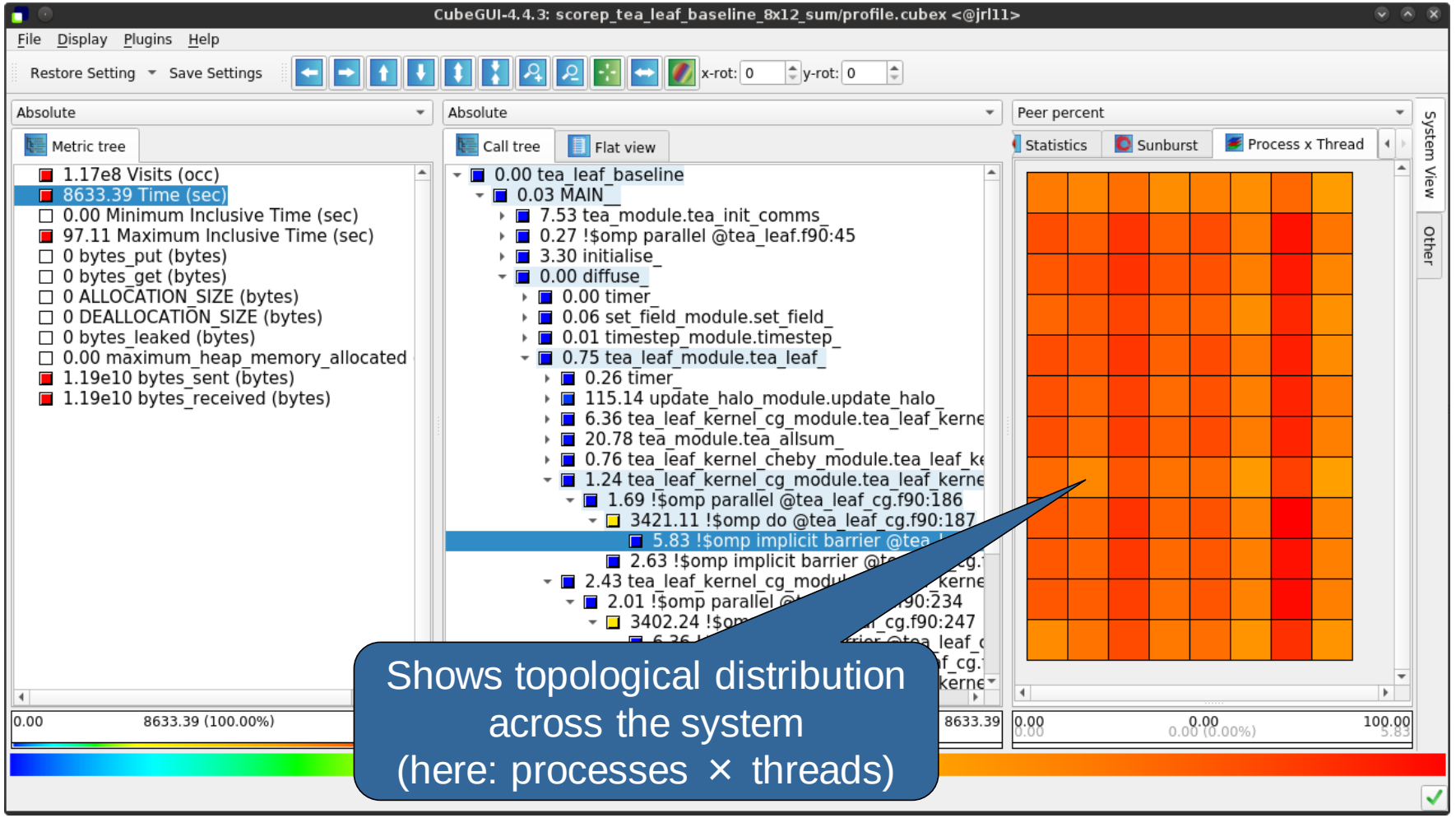
Box plot view



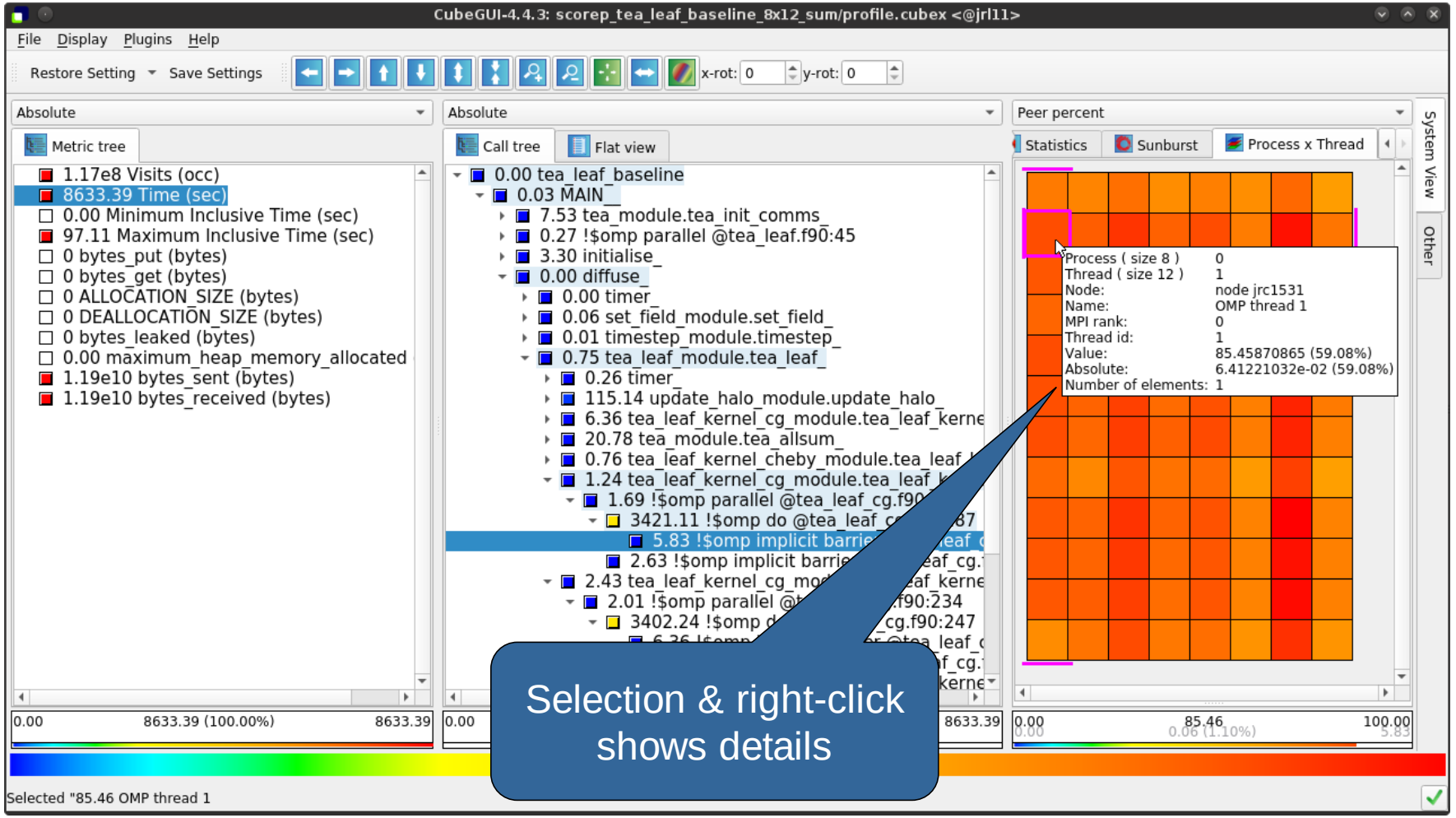
Violin plot view



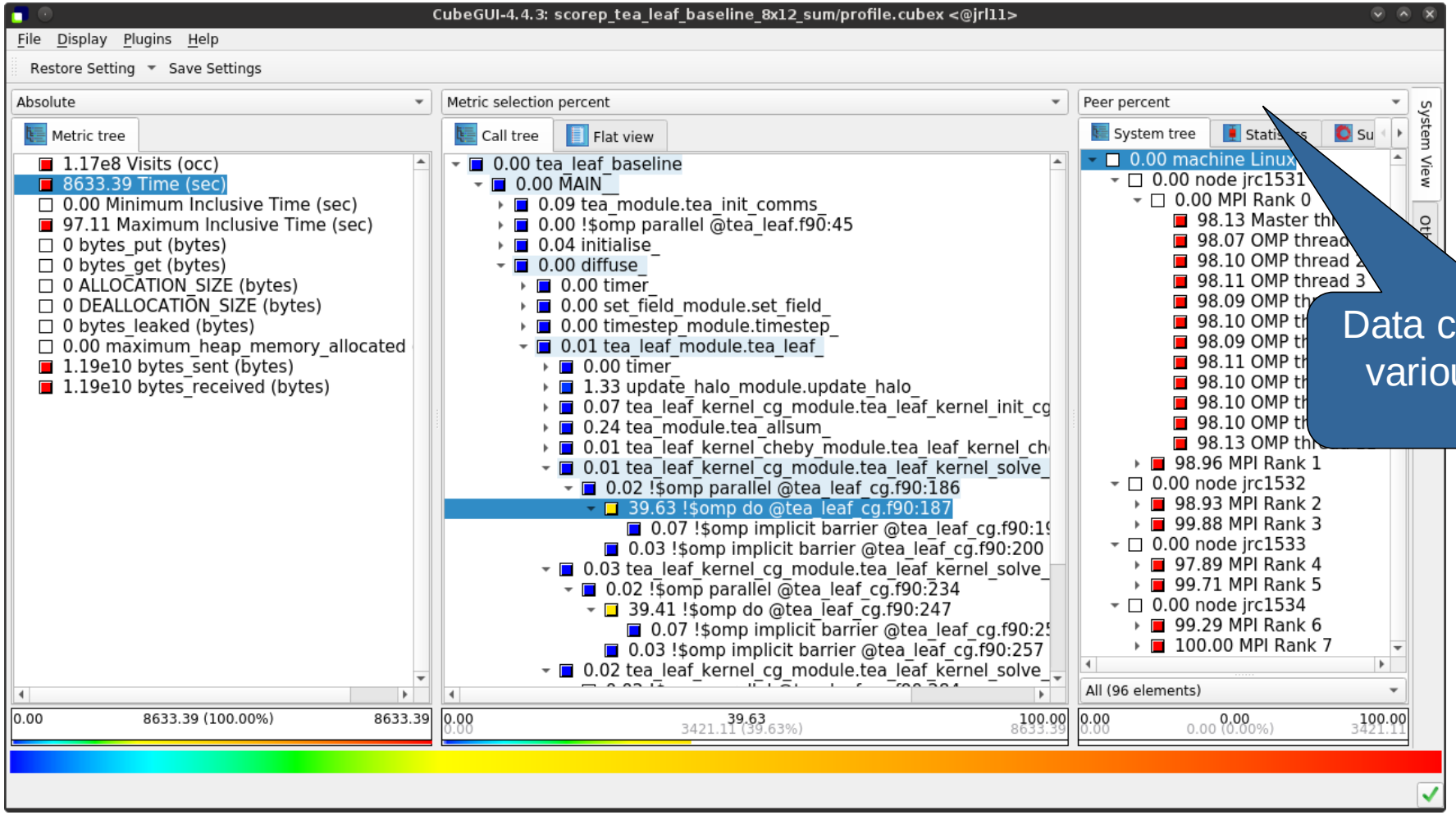
Topology view



Topology view (cont.)



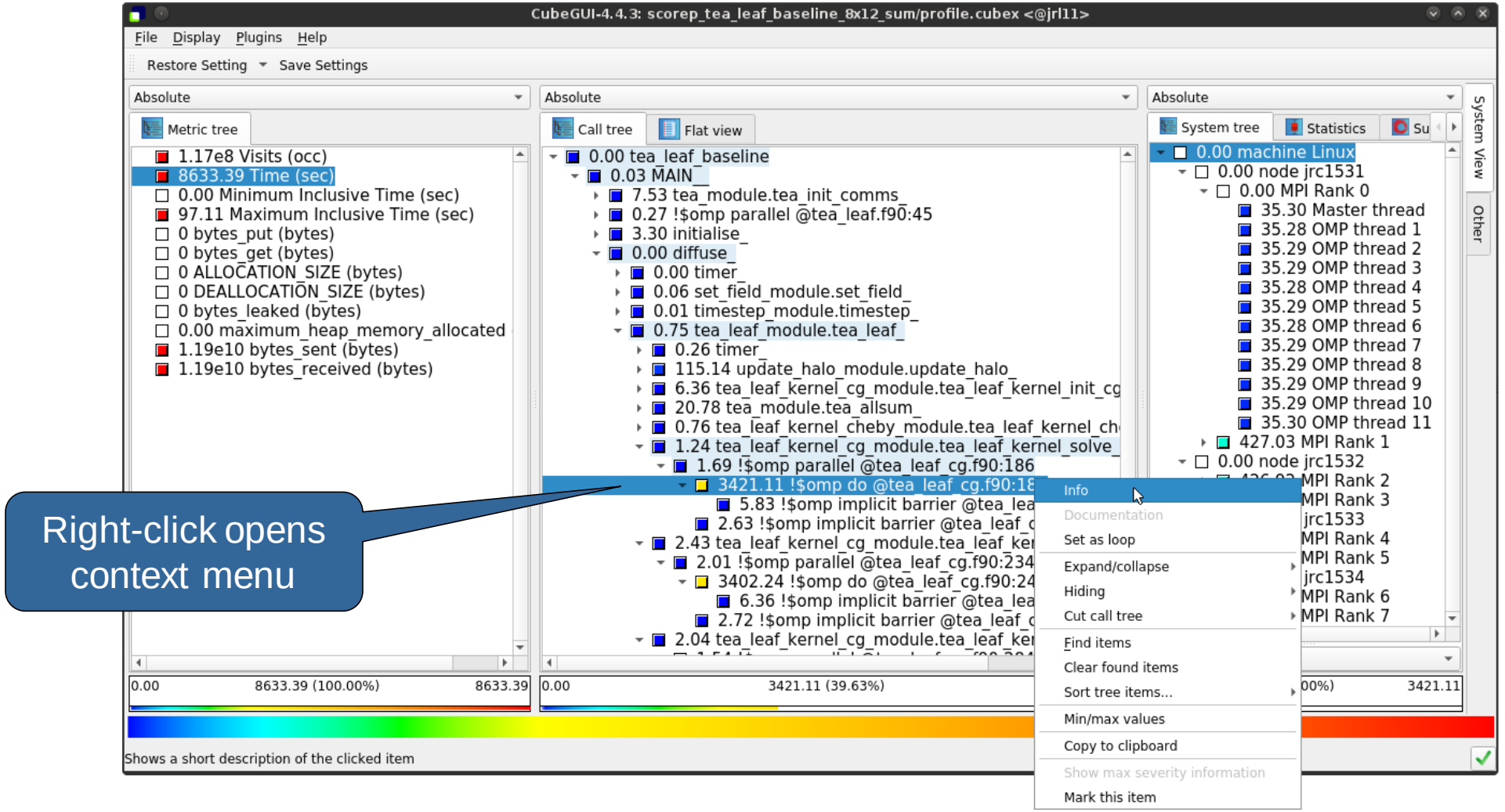
Alternative display modes



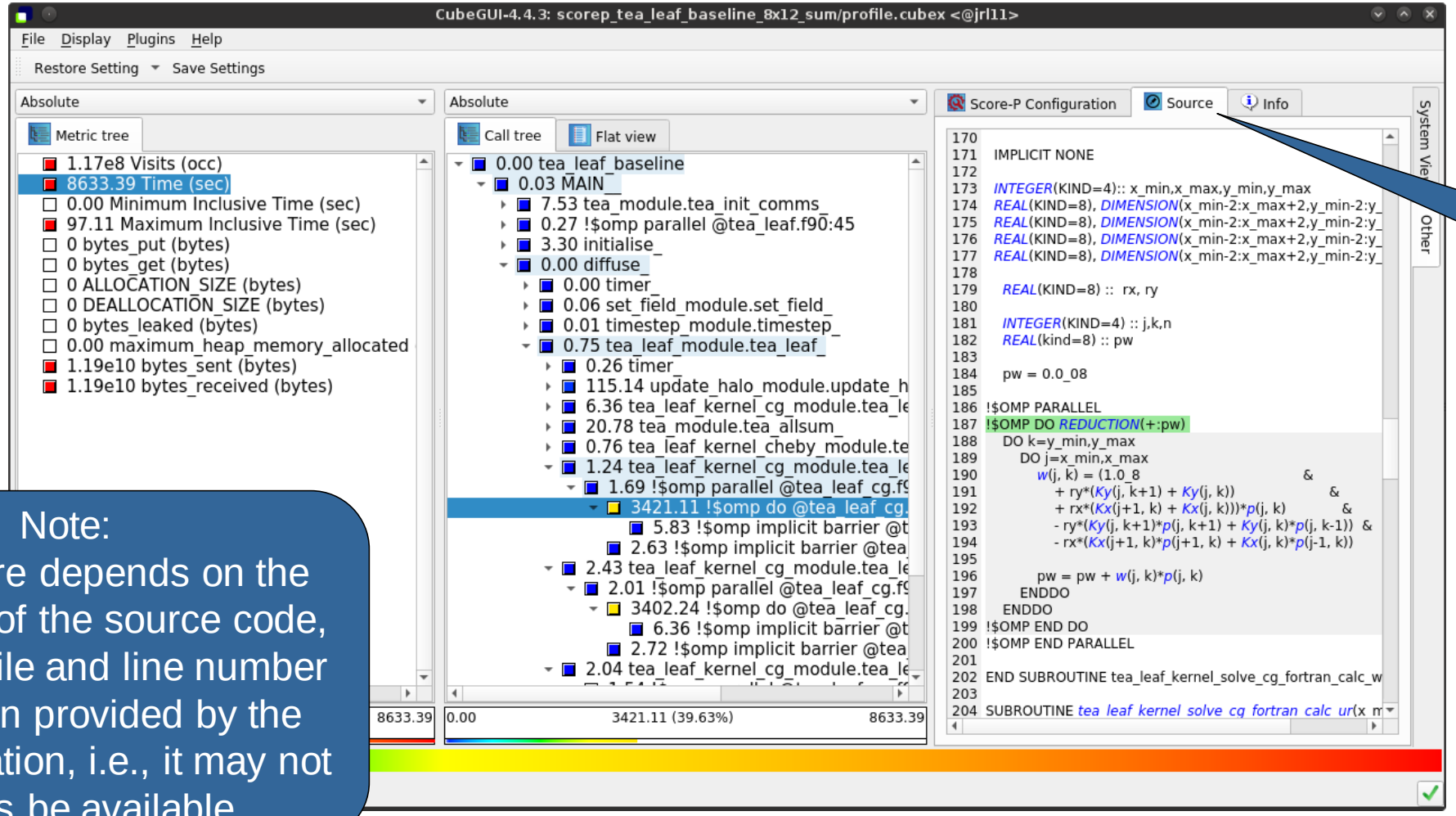
Important display modes

- 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

Source-code view via context menu



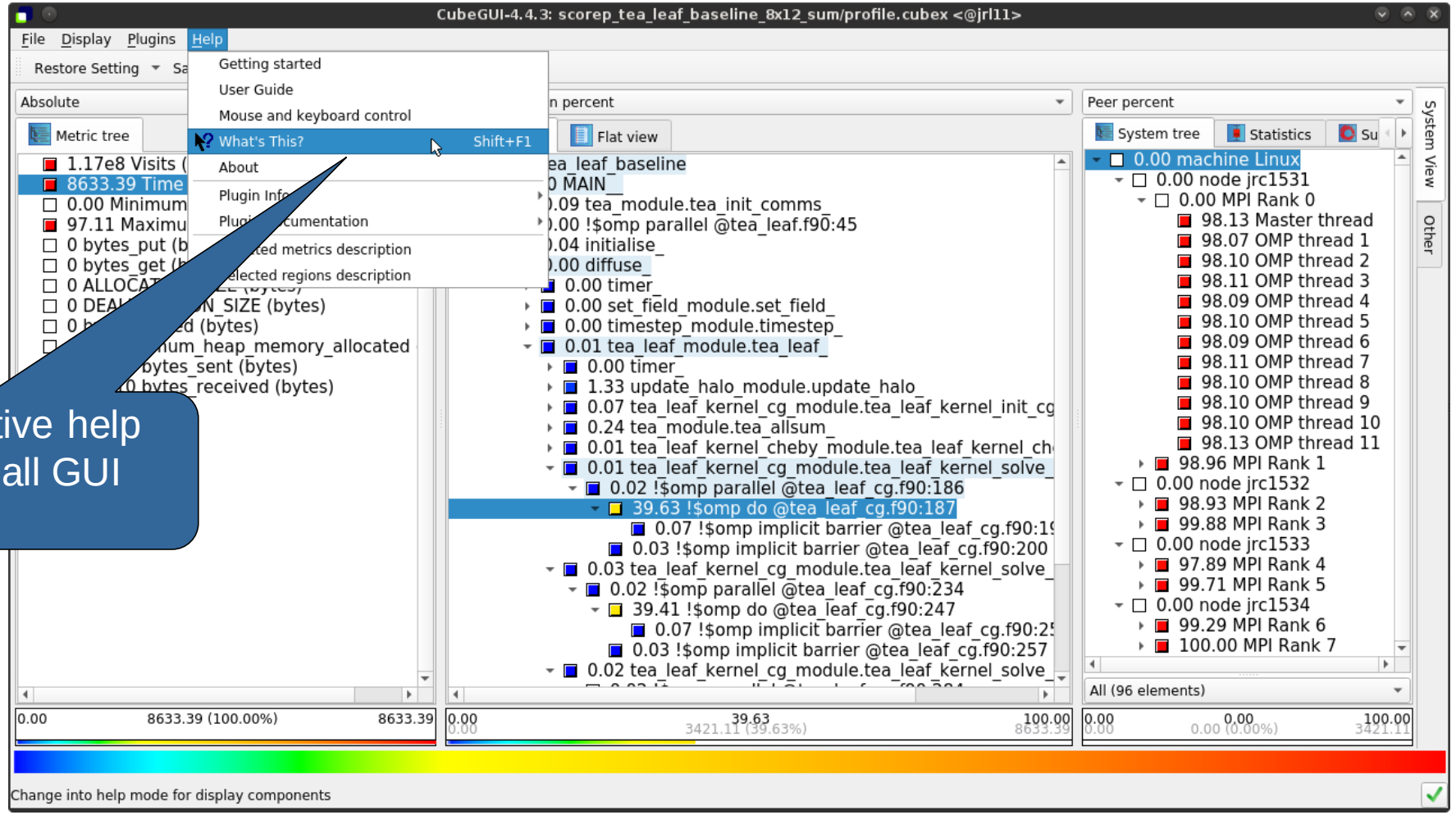
Source-code view



Select “Source” tab

Note:
This feature depends on the availability of the source code, as well as file and line number information provided by the instrumentation, i.e., it may not always be available

Context-sensitive help



Context-sensitive help available for all GUI items

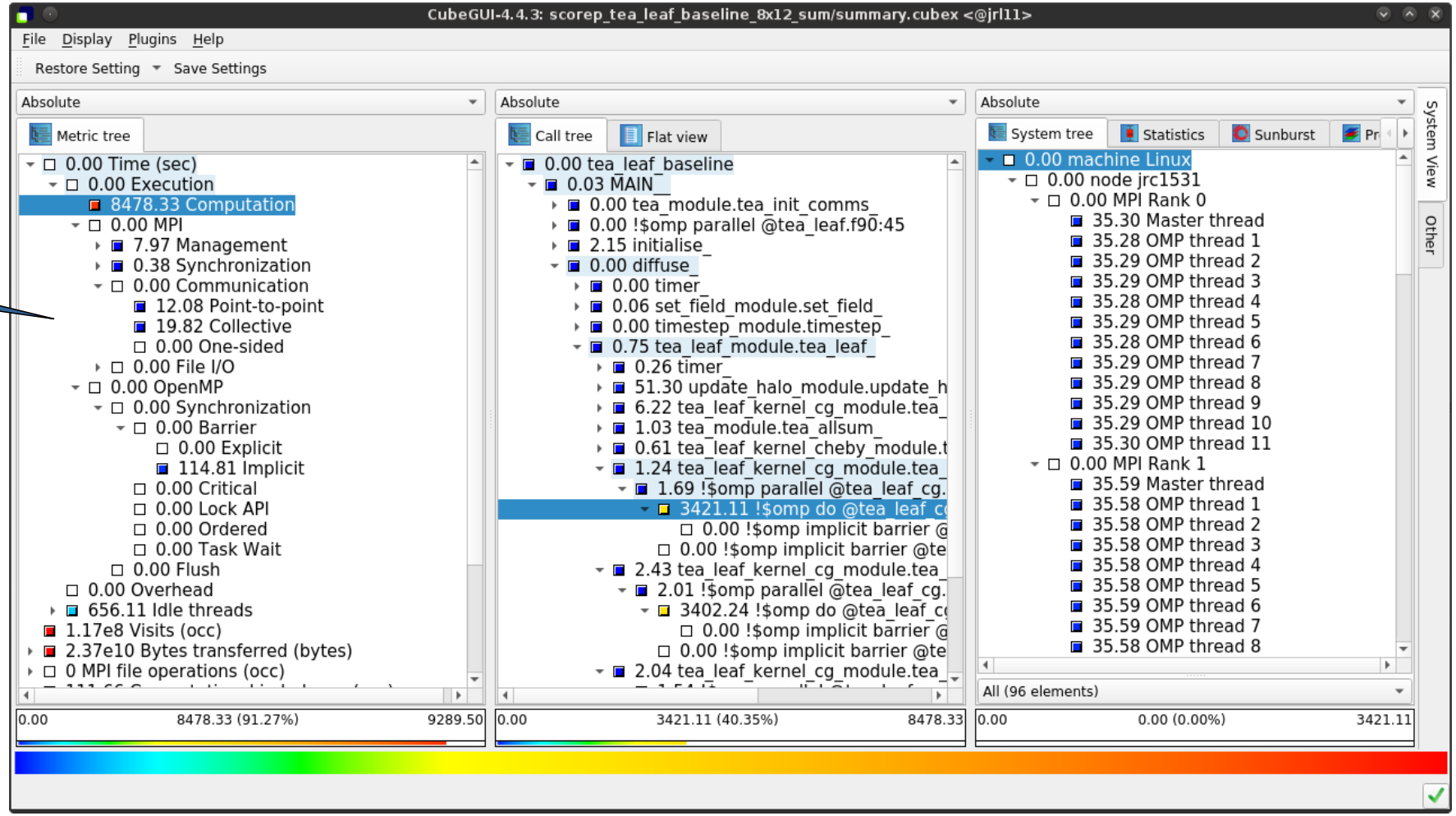
Scalasca report post-processing

- Scalasca's report post-processing derives additional metrics and generates a structured metric hierarchy
- Automatically run (if needed) when using the **square** convenience command:

```
% square scorep_tea_leaf_baseline_8x12_sum
INFO: Post-processing runtime summarization report (profile.cubex)...
INFO: Displaying ./scorep_tea_leaf_baseline_8x12_sum/summary.cubex...

                          [GUI showing post-processed summary analysis report]
```

Post-processed summary analysis report

Split base metrics into more specific metrics, e.g. computation vs parallelization costs

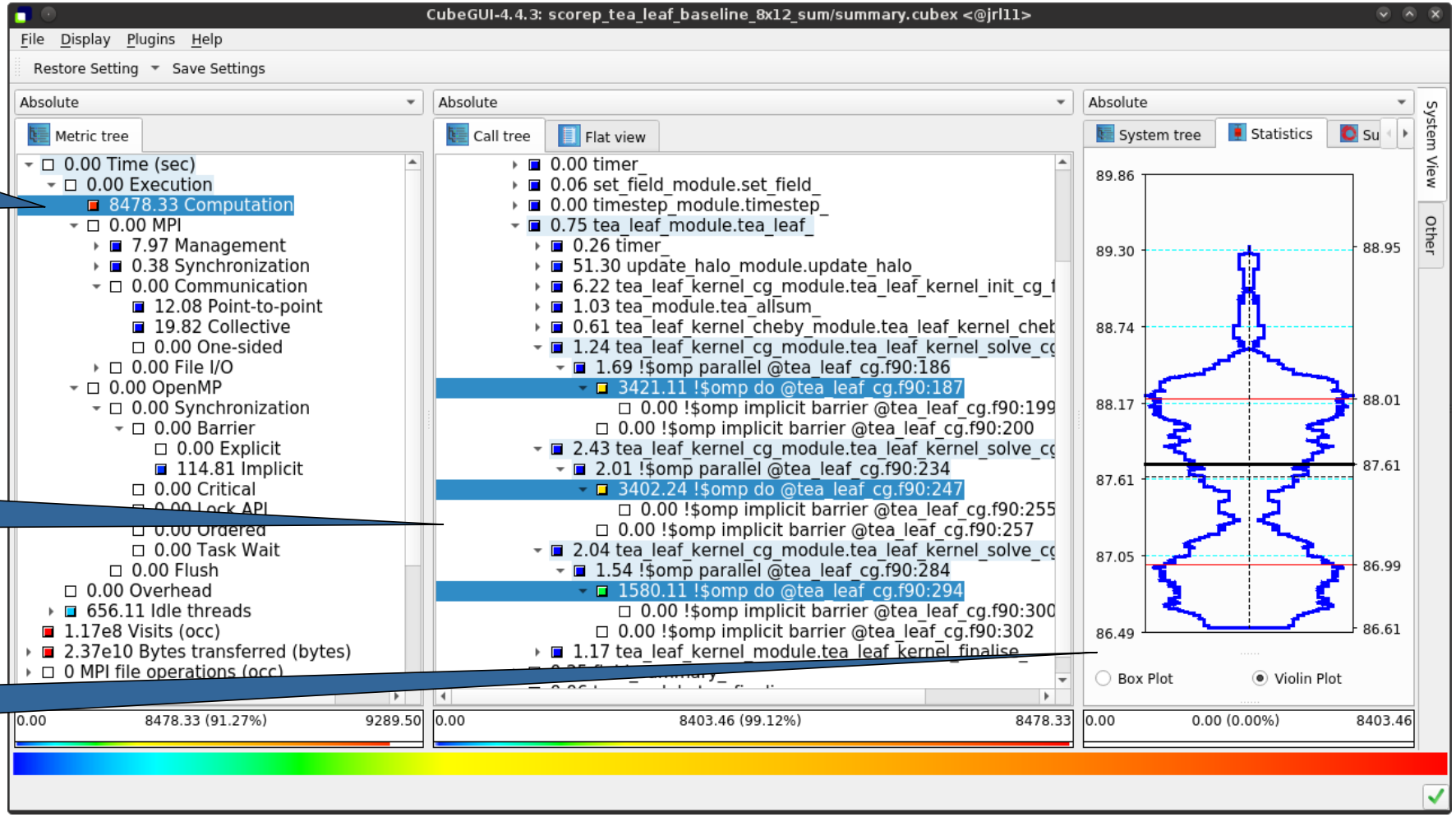


TeaLeaf summary report analysis (I)

91% of the execution time is computation…

…almost entirely spent in 3 OpenMP `do` loops…

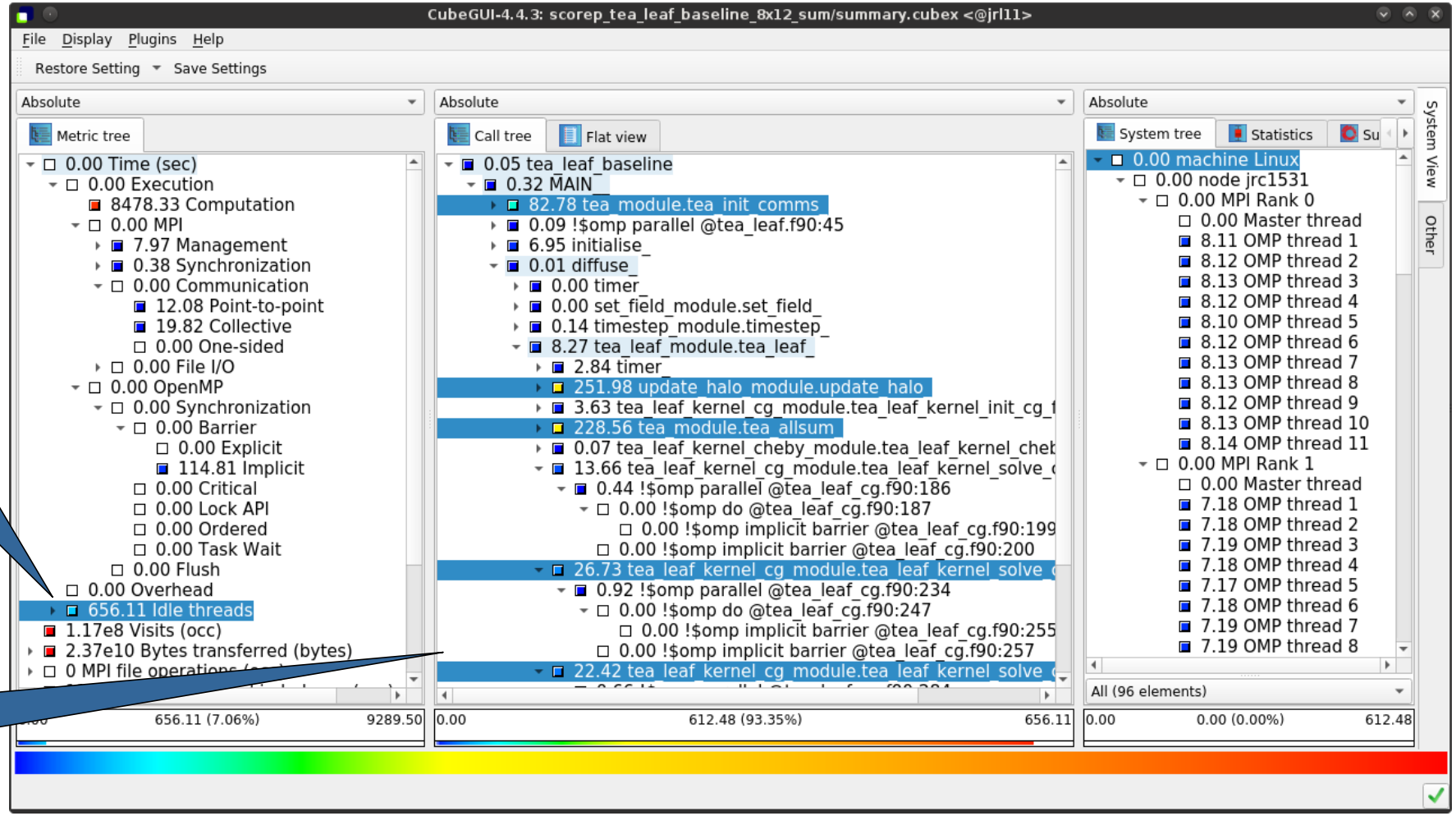
…with a slight imbalance across ranks & threads



TeaLeaf summary report analysis (II)

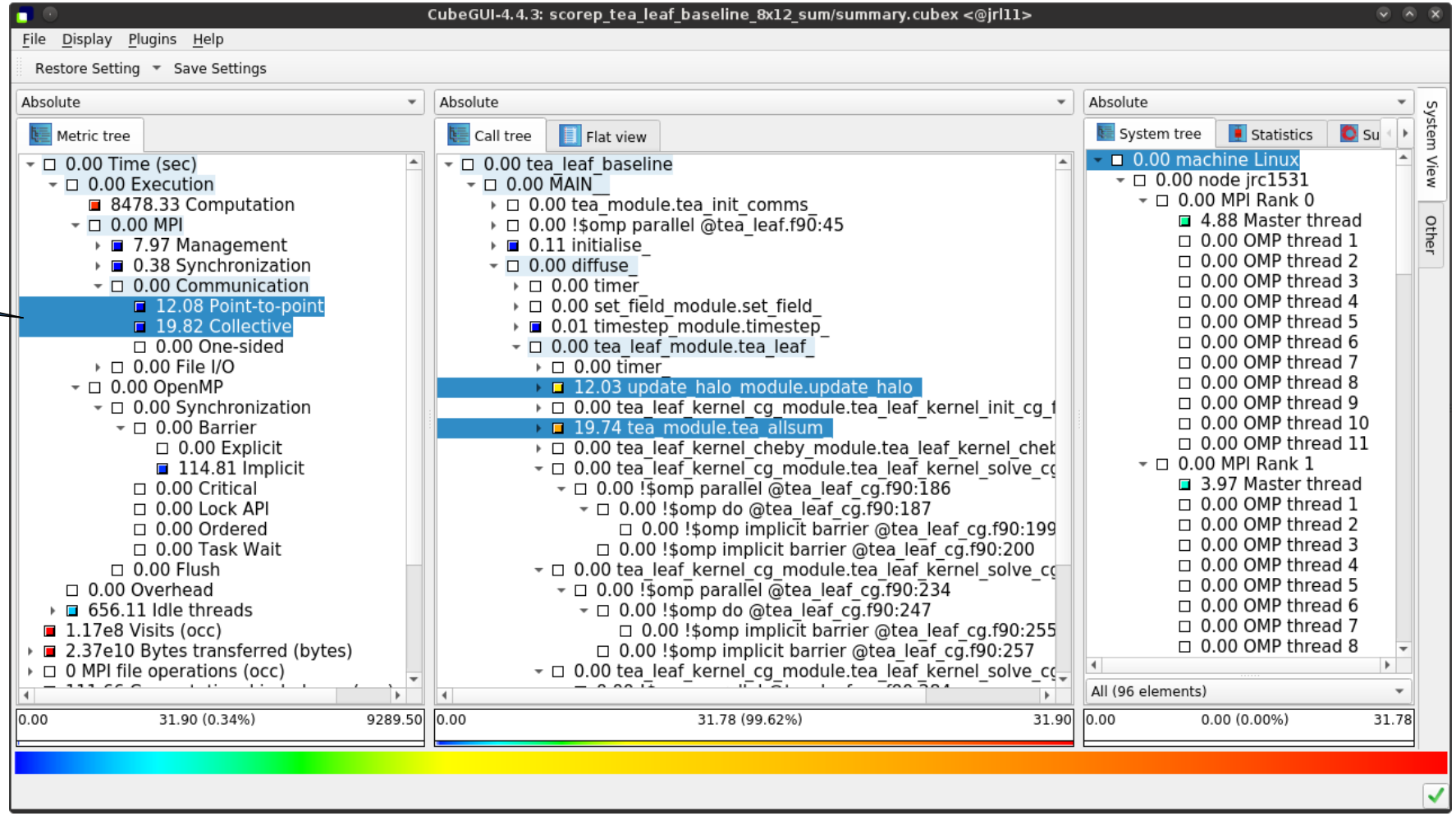
7% of the total CPU execution time is due due to "idle threads"…

… when not within OpenMP-parallelized code regions



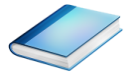
TeaLeaf summary report analysis (III)

MPI communication time is negligible (0.34%); but communication is only on the master threads (MPI_THREAD_FUNNELED)



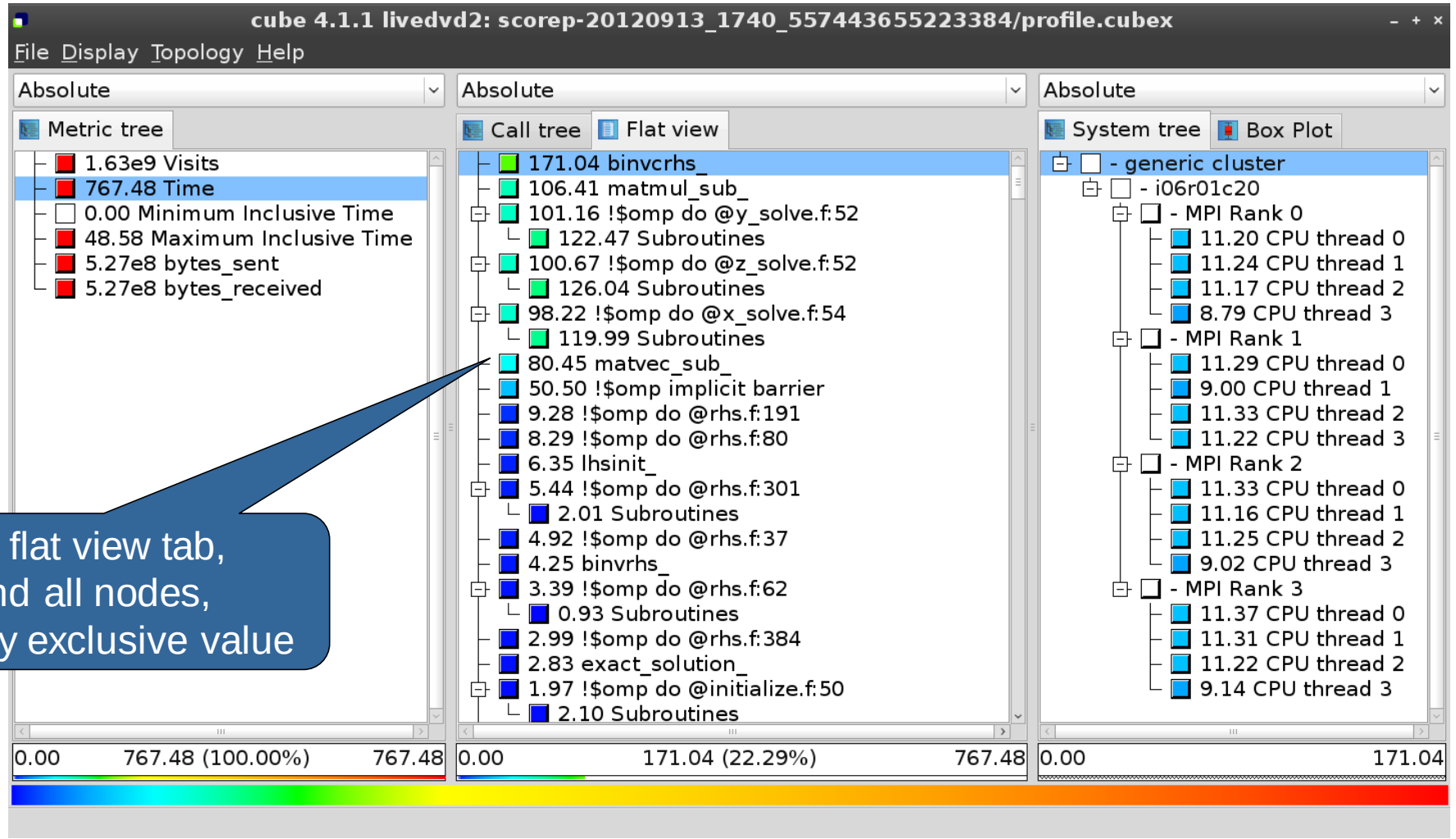
Cube: Further information

- Parallel program analysis report exploration tools
  - Libraries for Cube report reading & writing
  - Algebra utilities for report processing
  - GUI for interactive analysis exploration
- Available under 3-clause BSD open-source license
- Documentation & sources:
  - https://www.scalasca.org
- User guide also part of installation:
  - <prefix>/share/doc/cubegui/CubeUserGuide.pdf
- Contact:
  - mailto: scalasca@fz-juelich.de

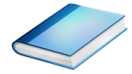
Reference material 

cube scalasca

Flat profile view



Derived metrics



- Derived metrics are defined using CubePL expressions, e.g.:

metric::time(i)/metric::visits(e)

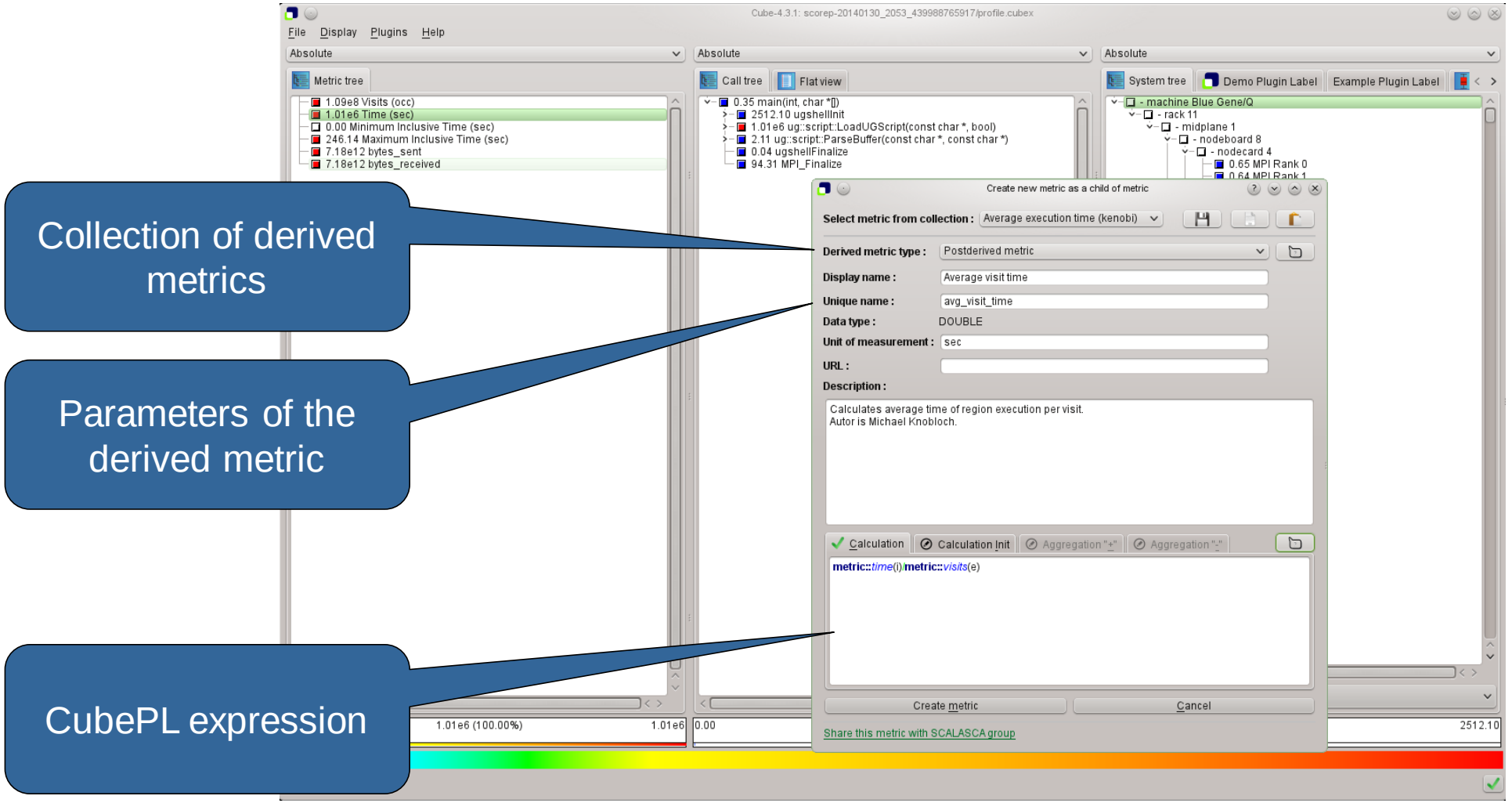
- Values of derived metrics are not stored, but calculated on-the-fly
- Types of derived metrics:
  - Prederived: evaluation of the CubePL expression is performed before aggregation
  - Postderived: evaluation of the CubePL expression is performed after aggregation

- Examples:
  - "Average execution time": Postderived metric with expression

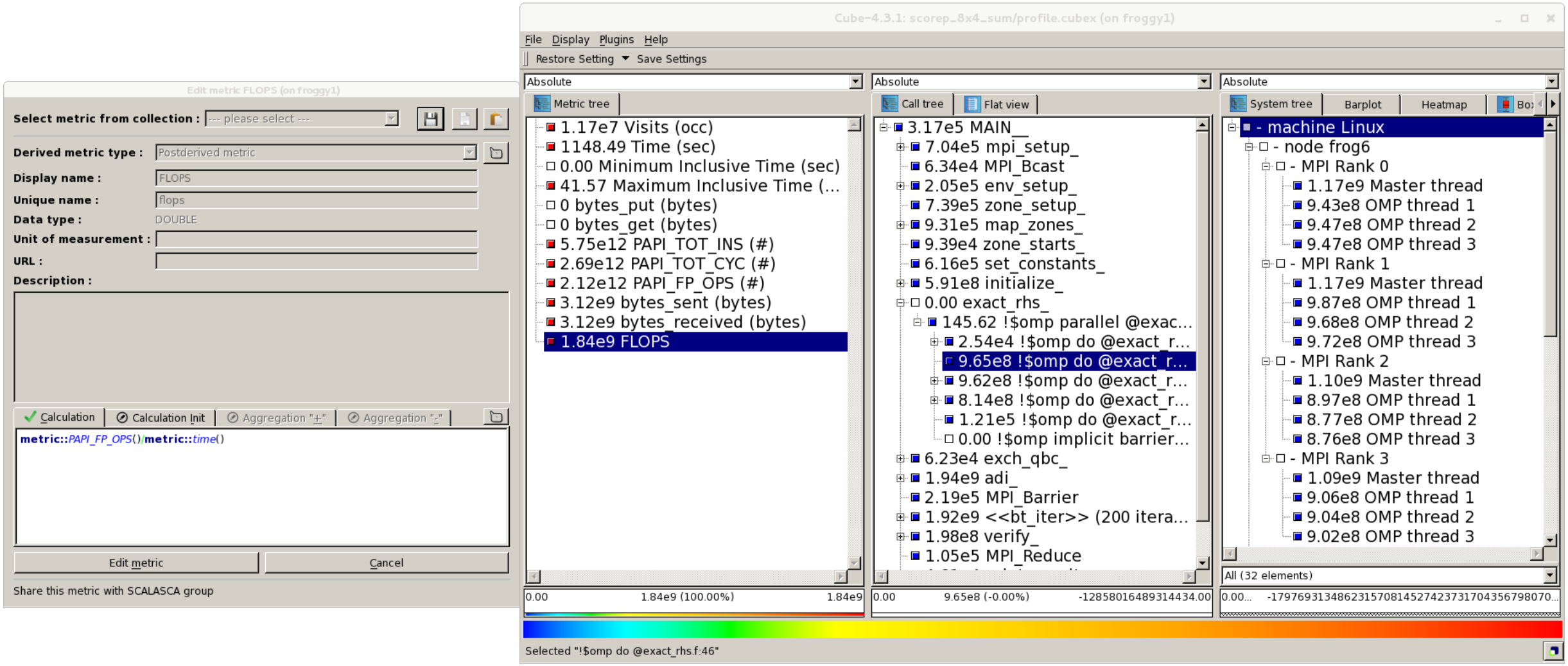
metric::time(i)/metric::visits(e)

  - "Number of FLOP per second": Postderived metric with expression

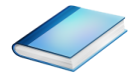
Derived metrics in Cube GUI



Example: FLOPS based on PAPI_FP_OPS and time



CUBE algebra utilities



- Extracting solver sub-tree from analysis report

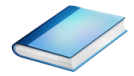
```
% cube_cut  -r '<<ITERATION>>'  scorep_bt-mz_C_32x4_sum/profile.cubex
Writing cut.cubex... done.
```

- Calculating difference of two reports

```
% cube_diff  scorep_bt-mz_C_32x4_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

Iteration profiling



- Show time dependent behavior by “unrolling” iterations
- Preparations:
  - Mark loop body by using Score-P instrumentation API in your source code

```
SCOREP_USER_REGION_DEFINE( scorep_bt_loop )
SCOREP_USER_REGION_BEGIN( scorep_bt_loop, "<<bt_iter>>", SCOREP_USER_REGION_TYPE_DYNAMIC )
SCOREP_USER_REGION_END( scorep_bt_loop )
```

- Result in the Cube profile:
  - Iterations shown as separate call trees

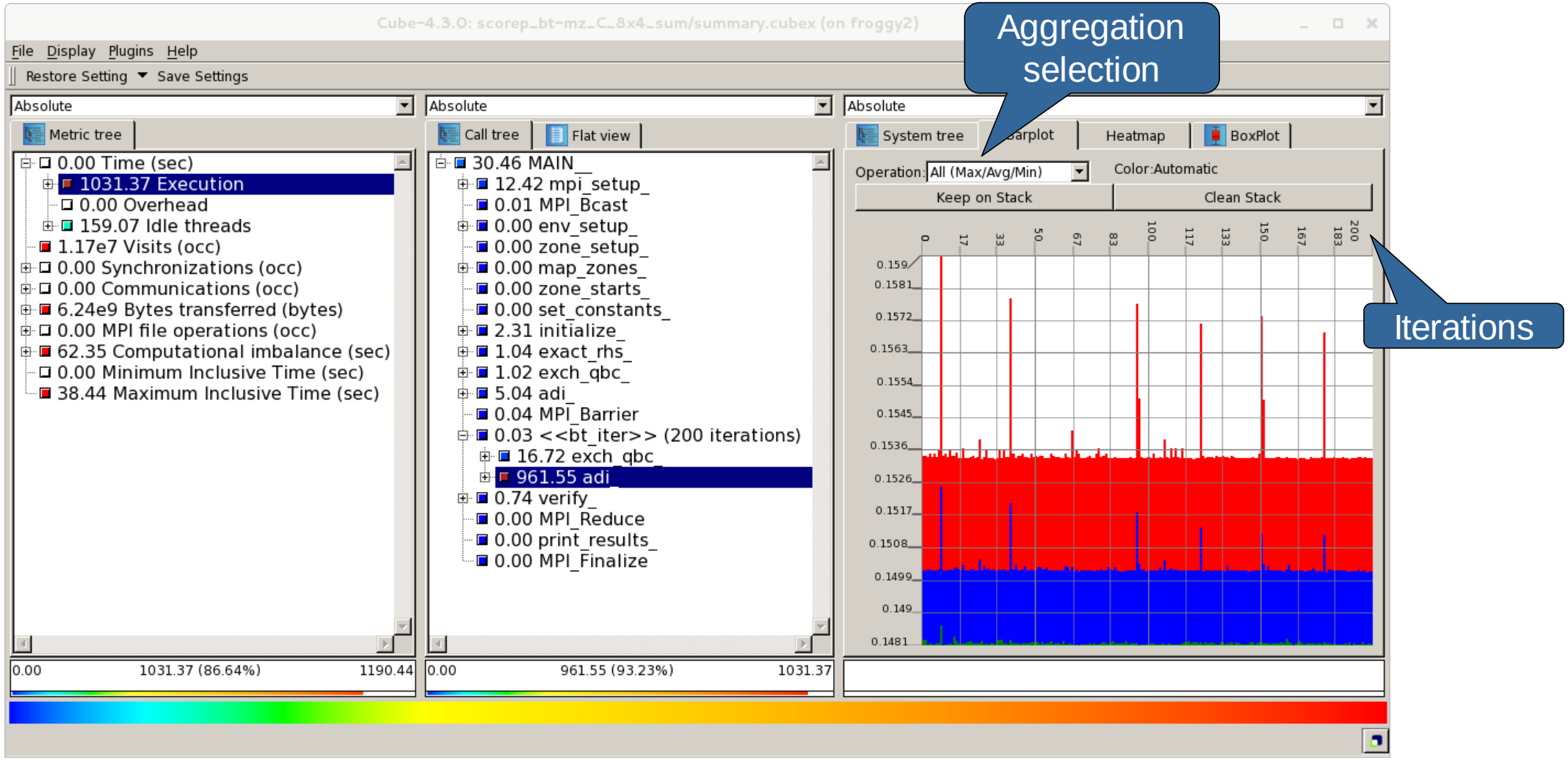
➢ Useful for checking results for specific iterations

or

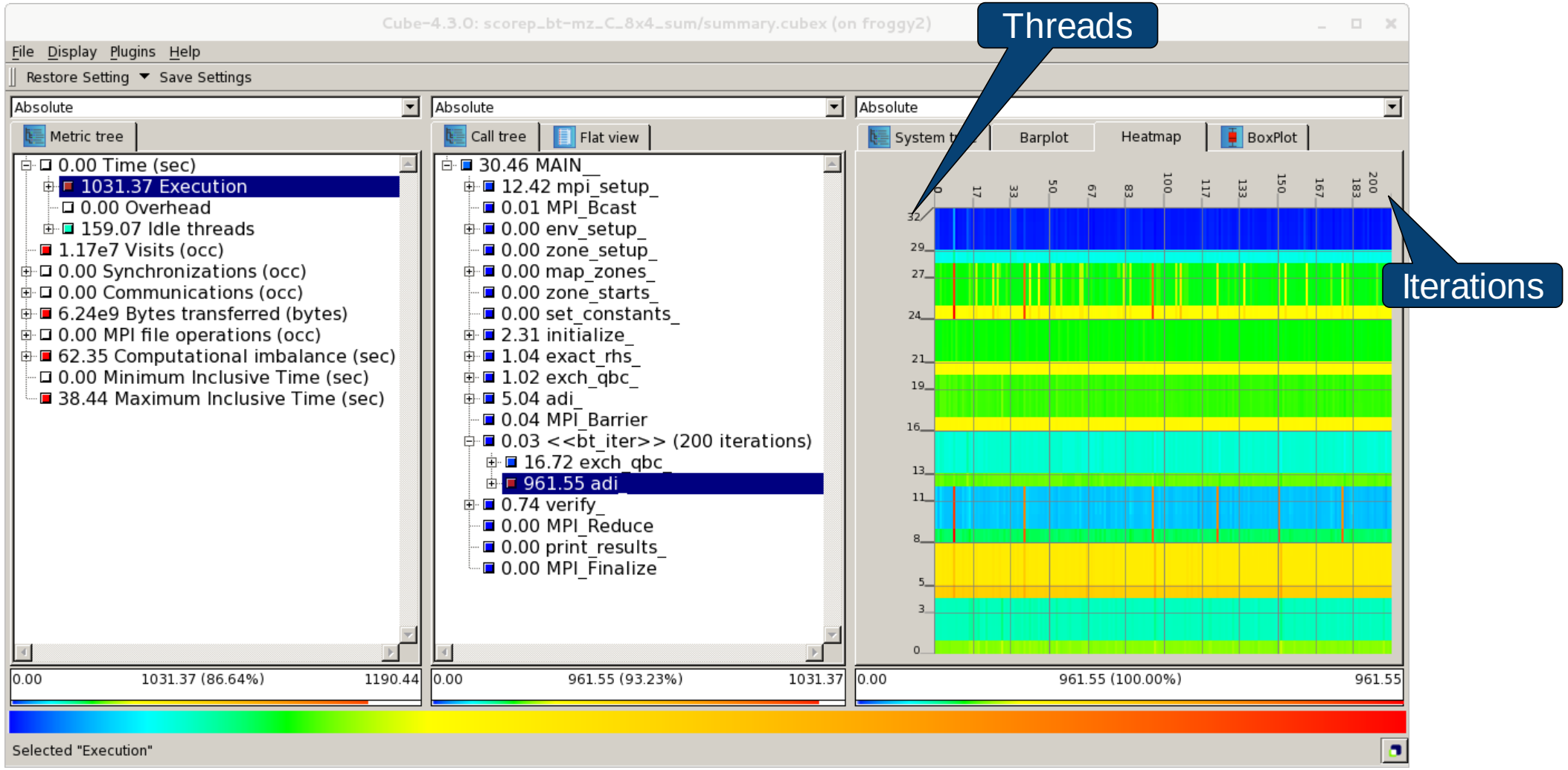
  - Select your user-instrumented region and mark it as loop
  - Choose “Hide iterations”

➢ View the Barplot statistics or the (thread x iterations) Heatmap

Iteration profiling: Barplot



Iteration profiling: Heatmap



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

VI-HPS Team

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 of 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
  - but there wasn't much GPU-related performance information
  - and the execution took rather longer than expected!
    - even when ignoring measurement start-up/completion, therefore
    - it was probably dilated by instrumentation/measurement overhead

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 customisation & 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

Mastering heterogeneous applications

- Record CUDA application events and device activities

```
% export SCOREP_CUDA_ENABLE=default
```

- Record OpenCL application events and device activities

```
% export SCOREP_OPENCL_ENABLE=api,kernel
```

- Record OpenACC application events

```
% export SCOREP_OPENACC_ENABLE=yes
```

  - Can be combined with CUDA if it is a NVIDIA device

```
% export SCOREP_CUDA_ENABLE=kernel
```

Up to now:
using default values

For all available options check:
scorep-info config-vars --full

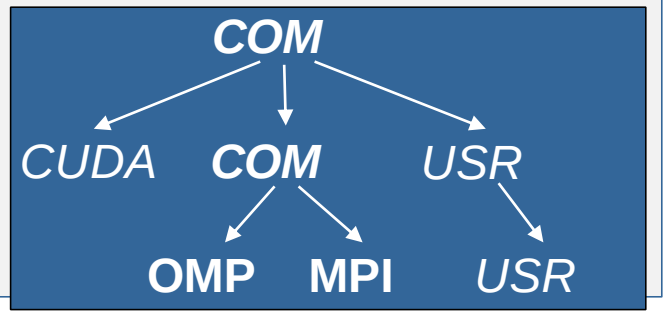
TeaLeaf CUDA extended summary measurement

```
% edit scorep.sbatch
% cat scorep.sbatch

# Score-P measurement configuration
export SCOREP_EXPERIMENT_DIRECTORY=scorep-tea_leaf-8.extended
export SCOREP_CUDA_ENABLE=default,driver,sync
export SCOREP_CUDA_BUFFER=48MB
#export SCOREP_FILTERING_FILE=../config/scorep.filter

# Run the application
srun ./tea_leaf



% sbatch scorep.sbatch
```

- Set new experiment directory and re-run measurement with extended CUDA event configuration
- Submit job

TeaLeaf summary analysis result scoring

```
% scorep-score scorep-tea_leaf-8.extended/profile.cubex

Estimated aggregate size of event trace:                   2132MB
Estimated requirements for largest trace buffer (max_buf): 312MB
Estimated memory requirements (SCOREP_TOTAL_MEMORY):       316MB
(hint: When tracing set SCOREP_TOTAL_MEMORY=316MB to avoid intermediate flushes
 or reduce requirements using USR regions filters.)

flt     type  max_buf[B]     visits time[s] time[%] time/visit[us]  region
         ALL 326,402,801 79,047,257  563.08   100.0           7.12  ALL
         USR 139,104,758 36,515,070    8.34     1.5           0.23  USR
        CUDA 100,904,778 26,426,831  504.82    89.7          19.10  CUDA
         COM  36,088,754  9,637,220    8.70     1.5           0.90  COM
         MPI  32,385,584  3,391,032   41.23     7.3          12.16  MPI
      SCOREP  17,918,927  3,077,104    0.00     0.0           0.00  SCOREP
```



- Report scoring as textual output

2.1 GB total memory
316MB per MPI process

- Region/callpath classification
  - **MPI** pure MPI functions
  - **OMP** pure OpenMP regions
  - **CUDA** API regions & kernels
  - **USR** user-level computation
  - **COM** "combined" USR+OpenMP/MPI
  - **ALL** aggregate of all region types

Score-P filtering: Automatic Generation of Filter Files

- **Basic usage**: `scorep-score -g` default heuristic targets:
  - Buffer usage: relevancy
  - Time per visits: overhead
- Creates annotated filter file:
  - `initial_scorep.filter`
  - Repeated calls create backups
  - Usage with `-f <file>` results in inclusion
- **Objective**:
  - Starting point for filtering
  - Syntax introduction

```
-g [<list>]
Generation of an initial filter file with the name
'initial_scorep.filter'. A valid parameter list has the form
KEY=VALUE[,KEY=VALUE]*. By default, uses the following control
parameters:

    `bufferpercent=1,timepervisit=1`

A region is included in the filter file (i.e., excluded from
measurement) if it matches all of the given conditions, with the
following keys:
- `bufferpercent`       : estimated memory requirements exceed the
                          given threshold in percent of the total
                          estimated trace buffer requirements
- `bufferabsolute`      : estimated memory requirements exceed
                          the given absolute threshold in MB
- `visits`              : number of visits exceeds the given
                          threshold
- `timepervisit`        : time per visit value is below the given
                          threshold in microseconds
- `type`                : region type matches the given value
                          (allowed: 'usr', 'com', 'both')
```

Score-P filtering: Automatic Generation of Filter Files

```
. . .
#
# Generated with the following parameters:
#  - A region has to use at least 1% of the estimated trace buffer.
#  - A region has to have a time/visits value of less than 1 us.
#  - A region that is of type USR.
#
# The file contains comments for each region providing additional information
# regarding the respective region.
# The common prefix for the files is:
# '/'
#
# Please refer to the Score-P user guide for more options on filtering.
SCOREP_REGION_NAMES_BEGIN
  EXCLUDE
    # type=USR max_buf= 63,576,240 visits=16,628,248, time=   2.17s (  0.4%), time/visit=    0.13us
    # name='bool std::operator< <char, std::char_traits<char>, std::allocator<char> >(std::__cxx11:: [...]
    # file='p/software/juwelsbooster/stages/2022/software/GCCcore/11.2.0/include/c++/11.2.0/bits/ [...]
    MANGLED _ZStltIcSt11char_traitsIcESaIcEEbRKNSt7__cxx1112basic_stringIT_T0_T1_EESA_
. . .

SCOREP_REGION_NAMES_END
```

Introductory preamble

Used generation parameters (either default or explicit)

Common path prefix to simplify file paths in entries

Annotated entry with comments containing `scorep-score` stats

Additional entries

TeaLeaf summary analysis report filtering – Preview mode

```
% scorep-score -r -f ../config/scorep.filter \
      scorep-tea_leaf-8.extended/profile.cubex
Estimated aggregate size of event trace:                   1186MB
Estimated requirements for largest trace buffer (max_buf): 173MB
Estimated memory requirements (SCOREP_TOTAL_MEMORY):       177MB
(hint: When tracing set SCOREP_TOTAL_MEMORY=177MB to avoid intermediate flushes
 or reduce requirements using USR regions filters.)

flt     type  max_buf[B]     visits time[s] time[%] time/visit[us]  region
 -       ALL 326,402,801 79,047,257  563.08   100.0           7.12  ALL
 -       USR 139,104,758 36,515,070    8.34     1.5           0.23  USR
 -      CUDA 100,904,778 26,426,831  504.82    89.7          19.10  CUDA
 -       COM  36,088,754  9,637,220    8.70     1.5           0.90  COM
 -       MPI  32,385,584  3,391,032   41.23     7.3          12.16  MPI
 -    SCOREP  17,918,927  3,077,104    0.00     0.0           0.00  SCOREP

 *       ALL 180,590,173 40,887,545  554.22    98.4          13.55  ALL-FLT
 +       FLT 145,812,628 38,159,712    8.86     1.6           0.23  FLT
 *      CUDA 100,904,778 26,426,831  504.82    89.7          19.10  CUDA-FLT
 -       MPI  32,385,584  3,391,032   41.23     7.3          12.16  MPI-FLT
 *       COM  27,912,950  7,540,860    7.40     1.3           0.98  COM-FLT
 -    SCOREP  17,918,927  3,077,104    0.00     0.0           0.00  SCOREP-FLT
 *       USR   1,469,130    451,718    0.77     0.1           1.71  USR-FLT

 +       USR  63,576,240 16,628,248    2.17     0.4           0.13  bool std::operator< < [...]
 -      CUDA  35,739,496  9,310,175    5.47     1.0           0.59  cuPointerGetAttributes
 -    SCOREP  17,918,927  3,077,104    0.00     0.0           0.00  tea_leaf
 +       USR  16,354,962  4,194,104    0.58     0.1           0.14  std::map<int, dim3, [...]
 +       USR  16,354,884  4,194,080    0.61     0.1           0.15  std::_Rb_tree<int, std::pair […]
```

- Score report breakdown by region (w/o additional metrics)

Filtered routines marked with '+'

region names with full signatures

TeaLeaf summary analysis report filtering – Preview incl. Counters

```
% scorep-score -f ../config/scorep.filter -c 2 \
      scorep-tea_leaf-8.extended/profile.cubex

Estimated aggregate size of event trace:                   3486MB
Estimated requirements for largest trace buffer (max_buf): 506MB
Estimated memory requirements (SCOREP_TOTAL_MEMORY):       510MB
(hint: When tracing set SCOREP_TOTAL_MEMORY=510MB to avoid
intermediate flushes or reduce requirements using USR regions filters.)

flt     type     max_buf[B]      visits time[s] time[%] time/visit[us]  region
 -       ALL 1,006,819,003 79,047,257  563.08   100.0           7.12  ALL
 -       USR   454,765,555 36,515,070    8.34     1.5           0.23  USR
 -      CUDA   329,881,005 26,426,831  504.82    89.7          19.10  CUDA
 -       COM   117,982,465  9,637,220    8.70     1.5           0.90  COM
 -       MPI    60,485,278  3,391,032   41.23     7.3          12.16  MPI
 -    SCOREP    43,704,700  3,077,104    0.00     0.0           0.00  SCOREP

 *       ALL   530,123,873 40,887,545  554.22    98.4          13.55  ALL-FLT
 +       FLT   476,695,130 38,159,712    8.86     1.6           0.23  FLT
 *      CUDA   329,881,005 26,426,831  504.82    89.7          19.10  CUDA-FLT
 *       COM    91,253,875  7,540,860    7.40     1.3           0.98  COM-FLT
 -       MPI    60,485,278  3,391,032   41.23     7.3          12.16  MPI-FLT
 -    SCOREP    43,704,700  3,077,104    0.00     0.0           0.00  SCOREP-FLT
 *       USR     4,802,925    451,718    0.77     0.1           1.71  USR-FLT
```

- Report scoring with prospective filter

3.5 GB of memory in total, 506 MB per rank!

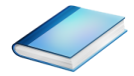
(Including 2 metric values)

TeaLeaf filtered summary measurement

```
% edit scorep.sbatch
% cat scorep.sbatch

# Score-P measurement configuration
export SCOREP_EXPERIMENT_DIRECTORY=scorep_tea_leaf_sum.filtered
export SCOREP_CUDA_ENABLE=default,driver,sync
export SCOREP_CUDA_BUFFER=48MB
export SCOREP_FILTERING_FILE=../config/scorep.filter

# Run the application
srun ./tea_leaf



% sbatch scorep.sbatch
```

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

Score-P filtering



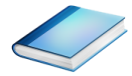
```
% cat ../config/scorep.filt
SCOREP_REGION_NAMES_BEGIN
  EXCLUDE
    binvcrhs*
    matmul_sub*
    matvec_sub*
    exact_solution*
    binvrhs*
    lhs*init*
    timer_*
SCOREP_REGION_NAMES_END

% export SCOREP_FILTERING_FILE=\
../config/scorep.filt
```

Region name filter block using wildcards

Apply filter

- Filtering by source file name
  - All regions in files that are excluded by the filter are ignored
- Filtering by region name
  - All regions that are excluded by the filter are ignored
  - Overruled by source file filter for excluded files
- Apply filter by
  - exporting `SCOREP_FILTERING_FILE` environment variable
- Apply filter at
  - Run-time
  - Compile-time (GCC-plugin, Intel compiler)
    - Add cmd-line option `--instrument-filter`
    - No overhead for filtered regions but recompilation

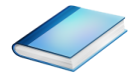
Source file name filter block



- Keywords
  - Case-sensitive
  - `SCOREP_FILE_NAMES_BEGIN`, `SCOREP_FILE_NAMES_END`
    - Define the source file name filter block
    - Block contains `EXCLUDE`, `INCLUDE` rules
  - `EXCLUDE`, `INCLUDE` rules
    - Followed by one or multiple white-space separated source file names
    - Names can contain bash-like wildcards `*`, `?`, `[]`
    - Unlike bash, `*` may match a string that contains slashes
- `EXCLUDE`, `INCLUDE` rules are applied in sequential order
- Regions in source files that are excluded after all rules are evaluated, get filtered

```
# This is a comment
SCOREP_FILE_NAMES_BEGIN
  # by default, everything is included
  EXCLUDE */foo/bar*
  INCLUDE */filter_test.c
SCOREP_FILE_NAMES_END
```

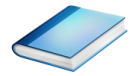
Region name filter block



- Keywords
  - Case-sensitive
  - `SCOREP_REGION_NAMES_BEGIN`, `SCOREP_REGION_NAMES_END`
    - Define the region name filter block
    - Block contains `EXCLUDE`, `INCLUDE` rules
  - `EXCLUDE`, `INCLUDE` rules
    - Followed by one or multiple white-space separated region names
    - Names can contain bash-like wildcards `*`, `?`, `[]`
- `EXCLUDE`, `INCLUDE` rules are applied in sequential order
- Regions that are excluded after all rules are evaluated, get filtered

```
# This is a comment
SCOREP_REGION_NAMES_BEGIN
  # by default, everything is included
  EXCLUDE *
  INCLUDE bar foo
          baz
          main
SCOREP_REGION_NAMES_END
```

Region name filter block, mangling



- Name mangling
  - Filtering based on names seen by the measurement system
    - Dependent on compiler
    - Actual name may be mangled
- `scorep-score` names as starting point (e.g. `matvec_sub_`)
  - Use `*` for Fortran trailing underscore(s) for portability
  - Use `?` and `*` as needed for full signatures or overloading

```
void bar(int* a) {
    *a++;
}
int main() {
    int i = 42;
    bar(&i);
    return 0;
}
```

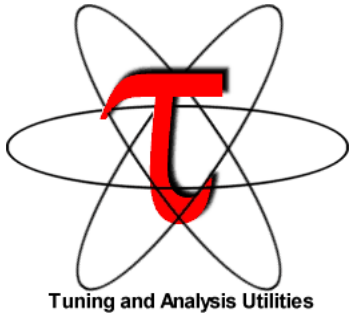
```
# filter bar:
# for gcc-plugin, scorep-score
# displays 'void bar(int*)',
# other compilers may differ

SCOREP_REGION_NAMES_BEGIN
  EXCLUDE void?bar(int?)
SCOREP_REGION_NAMES_END
```

Further information

- Community instrumentation & measurement infrastructure
  - Instrumentation (various methods)
  - Basic and advanced profile generation
  - Event trace recording
  - Online access to profiling data
- Available under 3-clause BSD open-source license
- Documentation & Sources:
  - https://www.score-p.org
- User guide also part of installation:
  - <prefix>/share/doc/scorep/{pdf,html}/
- Support and feedback: support@score-p.org
- Subscribe to news@score-p.org, to be up to date

VI-HPS

Examination of profiles with TAU ParaProf



Sameer Shende
sameer@cs.uoregon.edu
University of Oregon

http://tau.uoregon.edu/TAU_ParaProf_SC23.pdf

Application Performance Engineering using TAU

- How much time is spent in each application routine and outer *loops*? Within loops, what is the contribution of each *statement*? What is the time spent in OpenMP loops?
- How many instructions are executed in these code regions? Using Likwid or PAPI, TAU measures floating point, Level 1 and 2 *data cache misses*, hits, branches taken.
- What is the time taken in OS routines for thread scheduling? How much time is wasted?
- What is the memory usage of the code? When and where is memory allocated/de-allocated? Are there any memory leaks? What is the memory footprint of the application? What is the memory high water mark?
- What are the I/O characteristics of the code? What is the peak read and write *bandwidth* of individual calls, total volume?
- What is the contribution of each *phase* of the program? What is the time wasted/spent waiting for collectives, and I/O operations in Initialization, Computation, I/O phases?
- How does the application *scale*? What is the efficiency, runtime breakdown of performance across different core counts?

TAU: Quickstart Guide

Profiling:

```
MPI: % mpirun -np 16 tau_exec -ebs ./a.out
```

- MPI+OpenMP:
```
% export TAU_OMPT_SUPPORT_LEVEL=full;
% mpirun -np 16 tau_exec -T ompt -ompt ./a.out
```
- Pthread:
```
% mpirun -np 16 tau_exec -T mpi,pthread -ebs ./a.out
```
- CUDA:
```
% mpirun -np 4 tau_exec -T cupti -cupti -ebs ./a.out
```

```
Analysis: % pprof -a -m | more;  % paraprof (GUI)
```

Tracing:

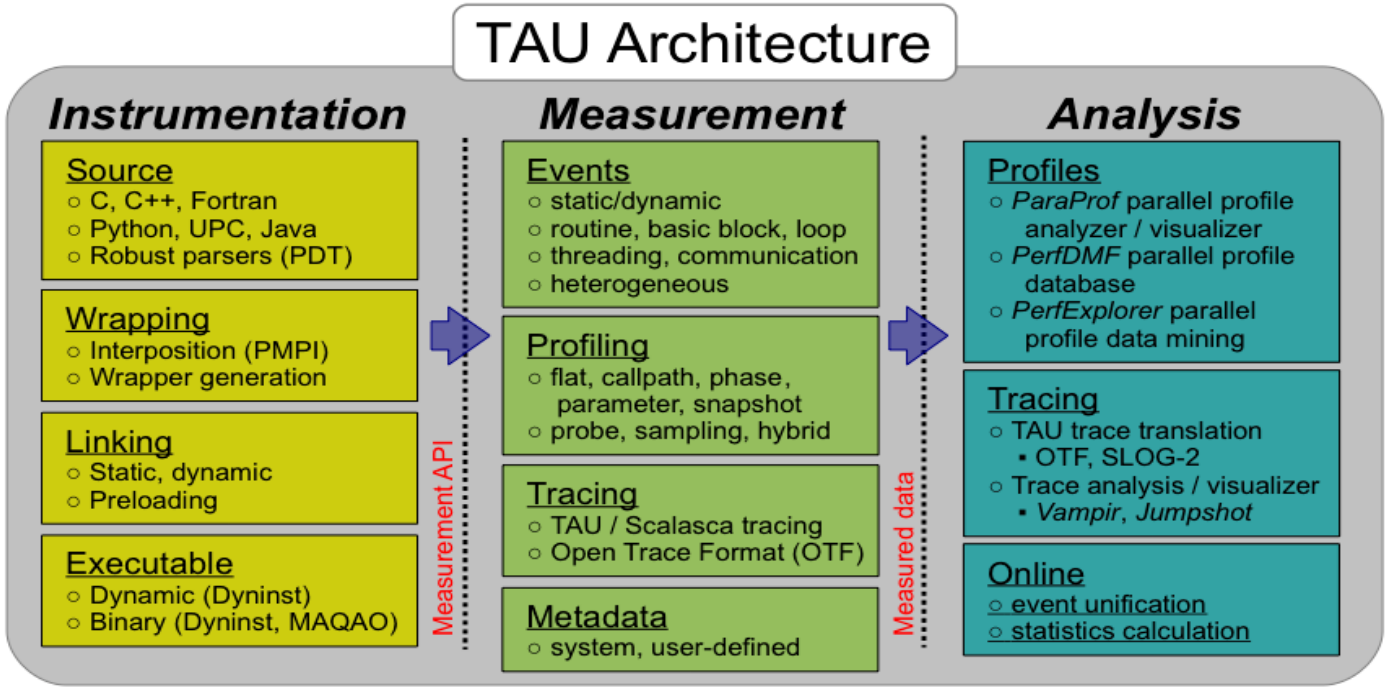
- Vampir: MPI:
```
% export TAU_TRACE=1; export TAU_TRACE_FORMAT=otf2
% mpirun -np 16 tau_exec ./a.out; vampir traces.otf2 &
```
- Chrome/Jumpshot:
```
% export TAU_TRACE=1; mpirun -np 64 tau_exec ./a.out
% tau_treemerge.pl;
```

```
Chrome: % tau_trace2json tau.trc tau.edf -chrome -ignoreatomic -o app.json
   Chrome browser: chrome://tracing   (Load -> app.json) or Perfetto.dev
```

- Jumpshot:
```
tau2slog2 tau.trc tau.edf -o app.slog2; jumpshot app.slog2
```

TAU Performance System®

- 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, OMPT interface, hybrid, other thread models
  - GPU, CUDA, OpenCL, OpenACC, ROCm, HIP
  - 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 (TAUdb)
  - 3D profile browser

TAU's Support for Runtime Systems

- *MPI*
  - PMPI profiling interface
  - MPI_T tools interface using performance and control variables
- *Pthread*
  - Captures time spent in routines per thread of execution
- *OpenMP*
  - OMPT tools interface to track salient OpenMP runtime events
  - Opari source rewriter
  - Preloading wrapper OpenMP runtime library when OMPT is not supported
- *OpenACC*
  - OpenACC instrumentation API
  - Track data transfers between host and device (per-variable)
  - Track time spent in kernels

TAU's Support for Runtime Systems (contd.)

- *OpenCL*
  - OpenCL profiling interface
  - Track timings of kernels
- *Intel® OneAPI*
  - Level Zero
  - Track time spent in kernels executing on GPU
  - Track time spent in OneAPI runtime calls
- *CUDA*
  - Cuda Profiling Tools Interface (CUPTI)
  - Track data transfers between host and GPU
  - Track access to uniform shared memory between host and GPU
- *ROCm*
  - Rocprofiler and Roctracer instrumentation interfaces
  - Track data transfers and kernel execution between host and GPU
- *Kokkos*
  - Kokkos profiling API
  - Push/pop interface for region, kernel execution interface
- *Python*
  - Python interpreter instrumentation API
  - Tracks Python routine transitions as well as Python to C transitions

Examples of Multi-Level Instrumentation

- *MPI + OpenMP*
  - MPI_T + PMPI + OMPT may be used to track MPI and OpenMP
- *MPI + CUDA*
  - PMPI + CUPTI interfaces
- *Kokkos + OpenMP*
  - Kokkos profiling API + OMPT to transparently track events
- *Kokkos + pthread + MPI*
  - Kokkos + pthread wrapper interposition library + PMPI layer
- *Python + CUDA + MPI*
  - Python + CUPTI + pthread profiling interfaces (e.g., Tensorflow, PyTorch) + MPI
- *MPI + OpenCL*
  - PMPI + OpenCL profiling interfaces

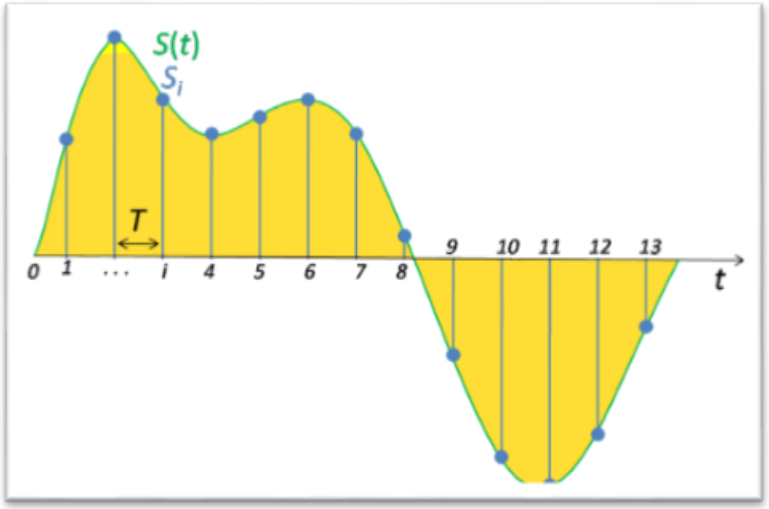
Performance Data Measurement

Direct via Probes

```
Call START('potential')
// code
Call STOP('potential')
```

- Exact measurement
- Fine-grain control
- Calls inserted into code

Indirect via Sampling



- No code modification
- Minimal effort
- Relies on debug symbols (**-g**)

Types of Performance Profiles

- *Flat* profiles
  - Metric (e.g., time) spent in an event
  - Exclusive/inclusive, # of calls, child calls, …
- *Callpath* profiles
  - Time spent along a calling path (edges in callgraph)
  - "*main=> f1 => f2 => MPI_Send*"
  - Set the TAU_CALLPATH and TAU_CALLPATH_DEPTH environment variables
- *Callsite* profiles
  - Time spent along in an event at a given source location
  - Set the TAU_CALLSITE environment variable
- *Phase* profiles
  - Flat profiles under a phase (nested phases allowed)
  - Default "main" phase
  - Supports static or dynamic (e.g. per-iteration) phases

Using TAU's Runtime Preloading Tool: tau_exec

- Preload a wrapper that intercepts the runtime system call and substitutes with another
  - **MPI**
  - **OpenMP**
  - **POSIX I/O**
  - **Memory allocation/deallocation routines**
  - **Wrapper library for an external package**
- No modification to the binary executable!
- Enable other TAU options (communication matrix, OTF2, event-based sampling)
- Add tau_exec before the name of the binary
  - mpirun –np 64 tau_exec ./a.out
  - mpirun tau_exec –T ompt,v5,mpi,papi -ompt ./a.out

tau_exec

```
$ tau_exec

Usage: tau_exec [options] [--] <exe> <exe options>

Options:
        -v              Verbose mode
        -s              Show what will be done but don't actually do anything (dryrun)
        -qsub           Use qsub mode (BG/P only, see below)
        -io             Track I/O
        -memory         Track memory allocation/deallocation
        -memory_debug Enable memory debugger
        -cuda           Track GPU events via CUDA
        -cupti          Track GPU events via CUPTI (Also see env. variable TAU_CUPTI_API)
        -opencl         Track GPU events via OpenCL
        -openacc        Track GPU events via OpenACC (currently PGI only)
        -ompt           Track OpenMP events via OMPT interface
        -armci          Track ARMCI events via PARMCI
        -ebs            Enable event-based sampling
        -ebs_period=<count> Sampling period (default 1000)
        -ebs_source=<counter> Counter (default itimer)
        -um           Enable Unified Memory events via CUPTI
        -T <DISABLE,GNU,ICPC,MPI,OMPT,OPENMP,PAPI,PDT,PROFILE,PTHREAD,SCOREP,SERIAL> : Specify TAU tags
        -loadlib=<file.so>   : Specify additional load library
        -XrunTAUsh-<options> : Specify TAU library directly
        -gdb            Run program in the gdb debugger

Notes:
           Defaults if unspecified: -T MPI
           MPI is assumed unless SERIAL is specified
```

- Tau_exec preloads the TAU wrapper libraries and performs measurements.

No need to recompile the application!

tau_exec Example (continued)

```
Example:
    mpirun -np 2 tau_exec -T icpc,ompt,mpi  -ompt ./a.out
    mpirun -np 2 tau_exec -io ./a.out
Example - event-based sampling with samples taken every 1,000,000 FP instructions
    mpirun -np 8 tau_exec -ebs -ebs_period=1000000 -ebs_source=PAPI_FP_INS ./ring
Examples - GPU:
    tau_exec -T serial,cupti -cupti ./matmult (Preferred for CUDA 4.1 or later)
    tau_exec -openacc ./a.out
    tau_exec -T serial -opencl ./a.out (OPENCL)
    mpirun -np 2 tau_exec -T mpi,cupti,papi -cupti -um ./a.out (Unified Virtual Memory in CUDA 6.0+)

qsub mode (IBM BG/Q only):
    Original:
      qsub -n 1 --mode smp -t 10 ./a.out
    With TAU:
      tau_exec -qsub -io -memory -- qsub -n 1 … -t 10 ./a.out

Memory Debugging:
    -memory option:
      Tracks heap allocation/deallocation and memory leaks.
    -memory_debug option:
      Detects memory leaks, checks for invalid alignment, and checks for
      array overflow.  This is exactly like setting TAU_TRACK_MEMORY_LEAKS=1
      and TAU_MEMDBG_PROTECT_ABOVE=1 and running with -memory
```

- tau_exec can enable event based sampling while launching the executable using the –ebs flag!

Simplifying TAU's usage (tau_exec)

- Uninstrumented execution linked with –dynamic (dynamic executables only!)
  % mpirun -np 16 ./a.out
- Track MPI performance
  % mpirun -np 16 tau_exec ./a.out
- Track OpenMP, and MPI performance (MPI enabled by default; OMPT in Clang 9+, Intel 19+)
  % export TAU_OMPT_SUPPORT_LEVEL=full;
  % mpirun -np 16 tau_exec –T mpi,pdt,ompt,papi –ompt ./a.out
- Track memory operations
  % export TAU_TRACK_MEMORY_LEAKS=1
  % mpirun -np 16 tau_exec –memory_debug ./a.out (bounds check)
- Use event based sampling (compile with –g)
  % mpirun -np 16 tau_exec –ebs ./a.out
  Also –ebs_source=<PAPI_COUNTER> -ebs_period=<overflow_count> -ebs_resolution=<file|function|line>
- Load wrapper interposition library
  % mpirun -np 16 tau_exec –loadlib=<path/libwrapper.so> ./a.out
- Track GPGPU operations (-rocm, -l0, -opencl, -cupti, -cupti –um, -openacc):
  % mpirun -np 16 tau_exec –cupti ./a.out

Configuration tags for tau_exec

```
% ./configure -pdt=<dir> -mpi -papi=<dir>; make install
```
Creates in $TAU:
Makefile.tau-papi-mpi-pdt(Configuration parameters in stub makefile)
shared-papi-mpi-pdt/libTAU.so

```
% ./configure -pdt=<dir> -mpi; make install
```
creates
Makefile.tau-mpi-pdt
shared-mpi-pdt/libTAU.so

To explicitly choose preloading of shared-<options>/libTAU.so change:
```
% mpirun -np 256 ./a.out      to
% mpirun -np 256  tau_exec -T <comma_separated_options> ./a.out

% mpirun -np 256  tau_exec -T papi,mpi,pdt ./a.out
```
Preloads $TAU/shared-papi-mpi-pdt/libTAU.so
```
% mpirun -np 256  tau_exec -T papi ./a.out
```
Preloads $TAU/shared-papi-mpi-pdt/libTAU.so by matching.
```
% aprun -n 256 tau_exec -T papi,mpi,pdt -s ./a.out
```
Does not execute the program. Just displays the library that it will preload if executed without the –s option.
NOTE: -mpi configuration is selected by default. Use –T serial for
Sequential programs.

TAU Execution Command (tau_exec)

- Uninstrumented execution
  - % mpirun -np 256 ./a.out
- Track GPU operations
  - % mpirun –np 256 tau_exec –rocm ./a.out
  - % mpirun –np 256 tau_exec –cupti ./a.out
  - % mpirun –np 256 tau_exec –opencl ./a.out
  - % mpirun –np 256 tau_exec –openacc ./a.out
  - % mpirun –np 256 tau_exec –l0 ./a.out
- Track MPI performance
  - % mpirun -np 256 tau_exec ./a.out
- Track I/O, and MPI performance (MPI enabled by default)
  - % mpirun -np 256 tau_exec -io ./a.out
- Track OpenMP and MPI execution (using OMPT for Intel v19+ or Clang 8+)
  - % export TAU_OMPT_SUPPORT_LEVEL=full;
  - % mpirun –np 256 tau_exec –T ompt,mpi -ompt ./a.out
- Track memory operations
  - % export TAU_TRACK_MEMORY_LEAKS=1
  - % mpirun –np 256 tau_exec –memory_debug ./a.out (bounds check)
- Use event based sampling (compile with –g)
  - % mpirun –np 256 tau_exec –ebs ./a.out
  - Also -ebs_source=<PAPI_COUNTER> -ebs_period=<overflow_count> -ebs_resolution=<file | function | line>

Hands-On Exercises for ParaProf

```
% source /p/project/training2341/setup.sh
% wget http://tau.uoregon.edu/demo.ppk
% paraprof demo.ppk &

% wget http://tau.uoregon.edu/data.tgz
% tar zxf data.tgz; cd data/tau;
% paraprof *.ppk &
```

TAU's Runtime Environment Variables

Environment Variable	Default	Description
TAU_TRACE	0	Setting to 1 turns on tracing
TAU_CALLPATH	0	Setting to 1 turns on callpath profiling
TAU_TRACK_MEMORY_FOOTPRINT	0	Setting to 1 turns on tracking memory usage by sampling periodically the resident set size and high water mark of memory usage
TAU_TRACK_POWER	0	Tracks power usage by sampling periodically.
TAU_CALLPATH_DEPTH	2	Specifies depth of callpath. Setting to 0 generates no callpath or routine information, setting to 1 generates flat profile and context events have just parent information (e.g., Heap Entry: foo)
TAU_SAMPLING	1	Setting to 1 enables event-based sampling.
TAU_TRACK_SIGNALS	0	Setting to 1 generate debugging callstack info when a program crashes
TAU_COMM_MATRIX	0	Setting to 1 generates communication matrix display using context events
TAU_THROTTLE	1	Setting to 0 turns off throttling. Throttles instrumentation in lightweight routines that are called frequently
TAU_THROTTLE_NUMCALLS	100000	Specifies the number of calls before testing for throttling
TAU_THROTTLE_PERCALL	10	Specifies value in microseconds. Throttle a routine if it is called over 100000 times and takes less than 10 usec of inclusive time per call
TAU_CALLSITE	0	Setting to 1 enables callsite profiling that shows where an instrumented function was called. Also compatible with tracing.
TAU_PROFILE_FORMAT	Profile	Setting to "merged" generates a single file. "snapshot" generates xml format

Runtime Environment Variables

Environment Variable	Default	Description
TAU_METRICS	TIME	Setting to a comma separated list generates other metrics. (e.g., ENERGY,TIME,P_VIRTUAL_TIME,PAPI_FP_INS,PAPI_NATIVE_<event>:<subevent>)
TAU_TRACE	0	Setting to 1 turns on tracing
TAU_TRACE_FORMAT	Default	Setting to “otf2” turns on TAU’s native OTF2 trace generation (configure with –otf=download)
TAU_EBS_UNWIND	0	Setting to 1 turns on unwinding the callstack during sampling (use with tau_exec –ebs or TAU_SAMPLING=1)
TAU_EBS_RESOLUTION	line	Setting to “function” or “file” changes the sampling resolution to function or file level respectively.
TAU_TRACK_LOAD	0	Setting to 1 tracks system load on the node
TAU_SELECT_FILE	Default	Setting to a file name, enables selective instrumentation based on exclude/include lists specified in the file.
TAU_OMPT_SUPPORT_LEVEL	basic	Setting to “full” improves resolution of OMPT TR6 regions on threads 1.. N-1. Also, “lowoverhead” option is available.
TAU_OMPT_RESOLVE_ADDRESS_EAGERLY	1	Setting to 1 is necessary for event based sampling to resolve addresses with OMPT. Setting to 0 allows the user to do offline address translation.
TAU_EVENT_THRESHOLD	0.5	Define a threshold value (e.g., .25 is 25%) to trigger marker events for min/max

Runtime Environment Variables

Environment Variable	Default	Description
TAU_TRACK_MEMORY_LEAKS	0	Tracks allocates that were not de-allocated (needs –optMemDbg or tau_exec –memory)
TAU_EBS_SOURCE	TIME	Allows using PAPI hardware counters for periodic interrupts for EBS (e.g., TAU_EBS_SOURCE=PAPI_TOT_INS when TAU_SAMPLING=1)
TAU_EBS_PERIOD	100000	Specifies the overflow count for interrupts
TAU_MEMDBG_ALLOC_MIN/MAX	0	Byte size minimum and maximum subject to bounds checking (used with TAU_MEMDBG_PROTECT_*)
TAU_MEMDBG_OVERHEAD	0	Specifies the number of bytes for TAU's memory overhead for memory debugging.
TAU_MEMDBG_PROTECT_BELOW/ABOVE	0	Setting to 1 enables tracking runtime bounds checking below or above the array bounds (requires –optMemDbg while building or tau_exec –memory)
TAU_MEMDBG_ZERO_MALLOC	0	Setting to 1 enables tracking zero byte allocations as invalid memory allocations.
TAU_MEMDBG_PROTECT_FREE	0	Setting to 1 detects invalid accesses to deallocated memory that should not be referenced until it is reallocated (requires –optMemDbg or tau_exec –memory)
TAU_MEMDBG_ATTEMPT_CONTINUE	0	Setting to 1 allows TAU to record and continue execution when a memory error occurs at runtime.
TAU_MEMDBG_FILL_GAP	Undefined	Initial value for gap bytes
TAU_MEMDBG_ALINGMENT	Sizeof(int)	Byte alignment for memory allocations

Installing and Configuring TAU

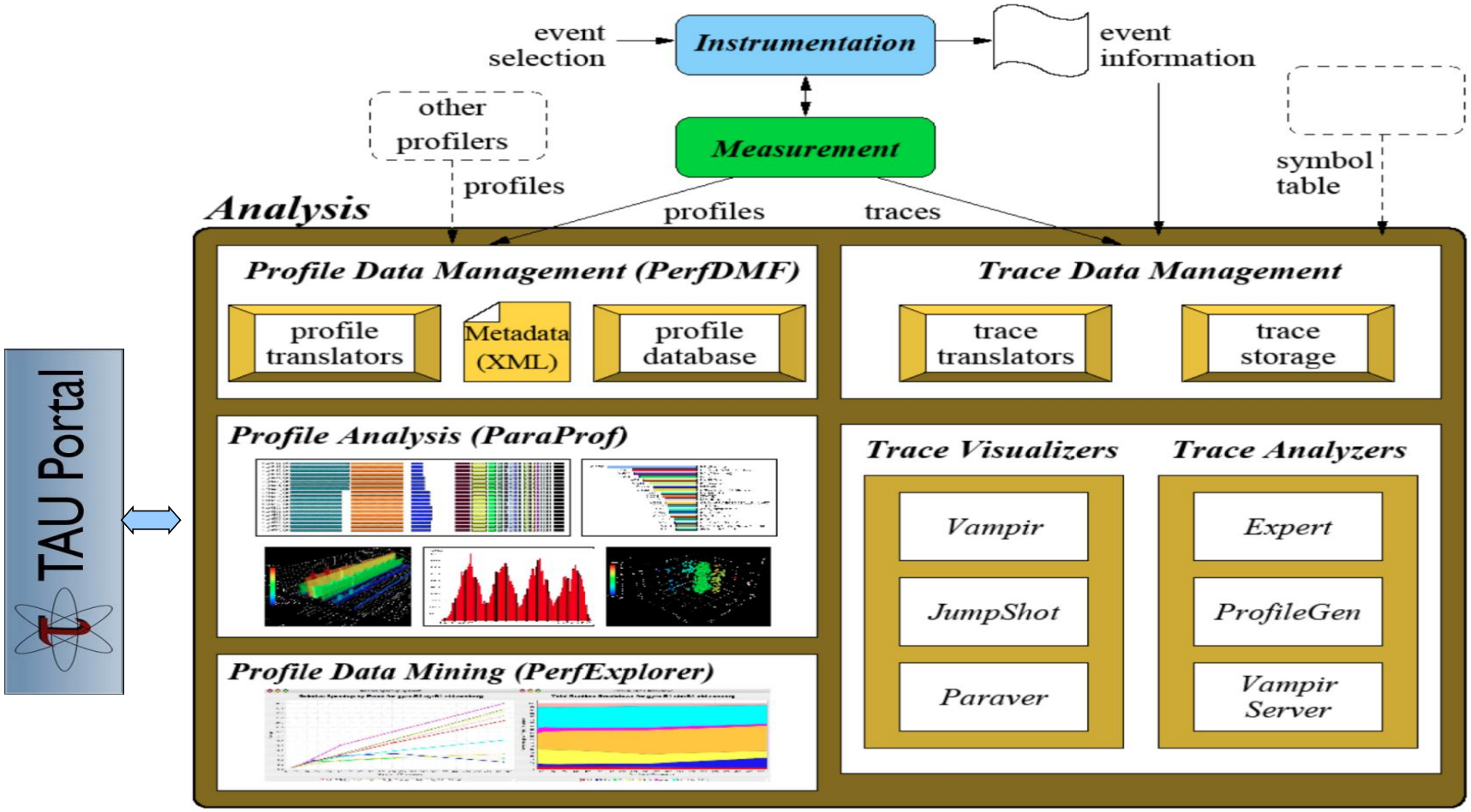
- Installing PDT:
  - wget http://tau.uoregon.edu/pdt.tgz
  - ./configure; make ; make install
- Installing TAU :
  - wget http://tau.uoregon.edu/tau.tgz
  - ./configure –ompt –c++=mpiicpc –cc=mpiicc –fortran=mpiifort -mpi -bfd=download -pdt=<dir> -papi=<dir> ...
  - make install; export PATH=<taudir>/x86_64/bin:$PATH
  - All configurations are stored in <taudir>/.all_configs if you wish to see how TAU was configured!
- Using TAU for source instrumentation:
  - export TAU_MAKEFILE=<taudir>/x86_64/lib/Makefile.tau-<TAGS>
  - make CC=tau_cc.sh CXX=tau_cxx.sh F90=tau_f90.sh
  - Use tau_exec with uninstrumented binaries instead of recompiling the source code.

Installing TAU on your laptop for paraprof (GUI)

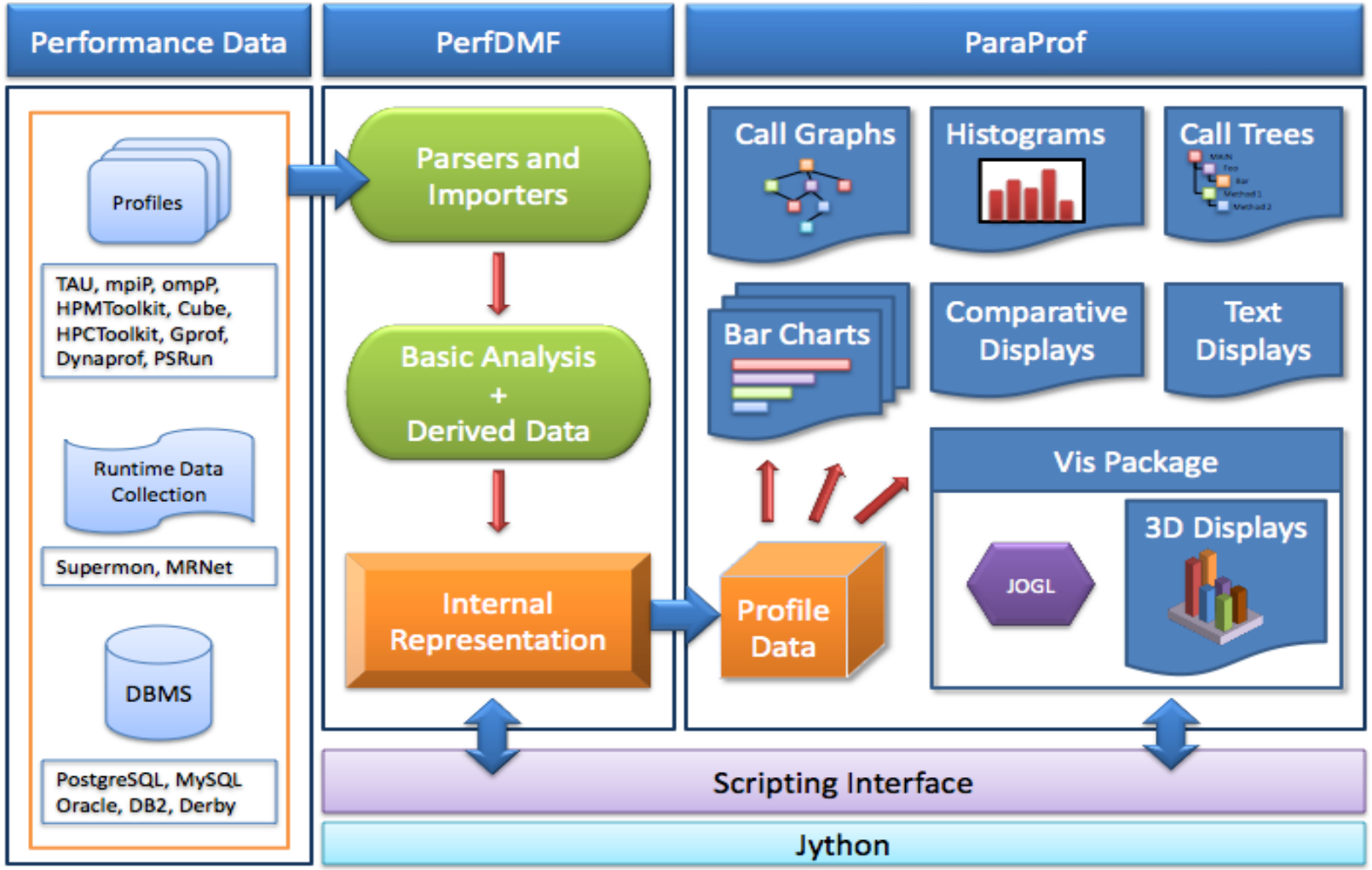
- Microsoft Windows
  - Install Java from Oracle.com
  - http://tau.uoregon.edu/tau.exe
  - Install, click on a ppk file to launch paraprof
- macOS
  - Install Java 11.0.3:
    - Download http://tau.uoregon.edu/java.dmg
    - If you have multiple Java installations, add to your ~/.zshrc (or ~/.bashrc as appropriate):
    - export PATH=/Library/Java/JavaVirtualMachines/jdk-11.0.3.jdk/Contents/Home/bin:$PATH
    - java -version
  - Download and install TAU (copy to /Applications from dmg):
    - http://tau.uoregon.edu/tau.dmg
    - export PATH=/Applications/TAU/tau/apple/bin:$PATH
    - paraprof app.ppk &
- macOS (arm64, M1/M2)
  - http://tau.uoregon.edu/java_arm64.dmg
  - http://tau.uoregon.edu/tau_arm64.dmg
- Linux (http://tau.uoregon.edu/tau.tgz)
  - ./configure; make install; export PATH=<taudir>/x86_64/bin:$PATH
  - paraprof app.ppk &

TAU's Analysis Tools: ParaProf

TAU Analysis



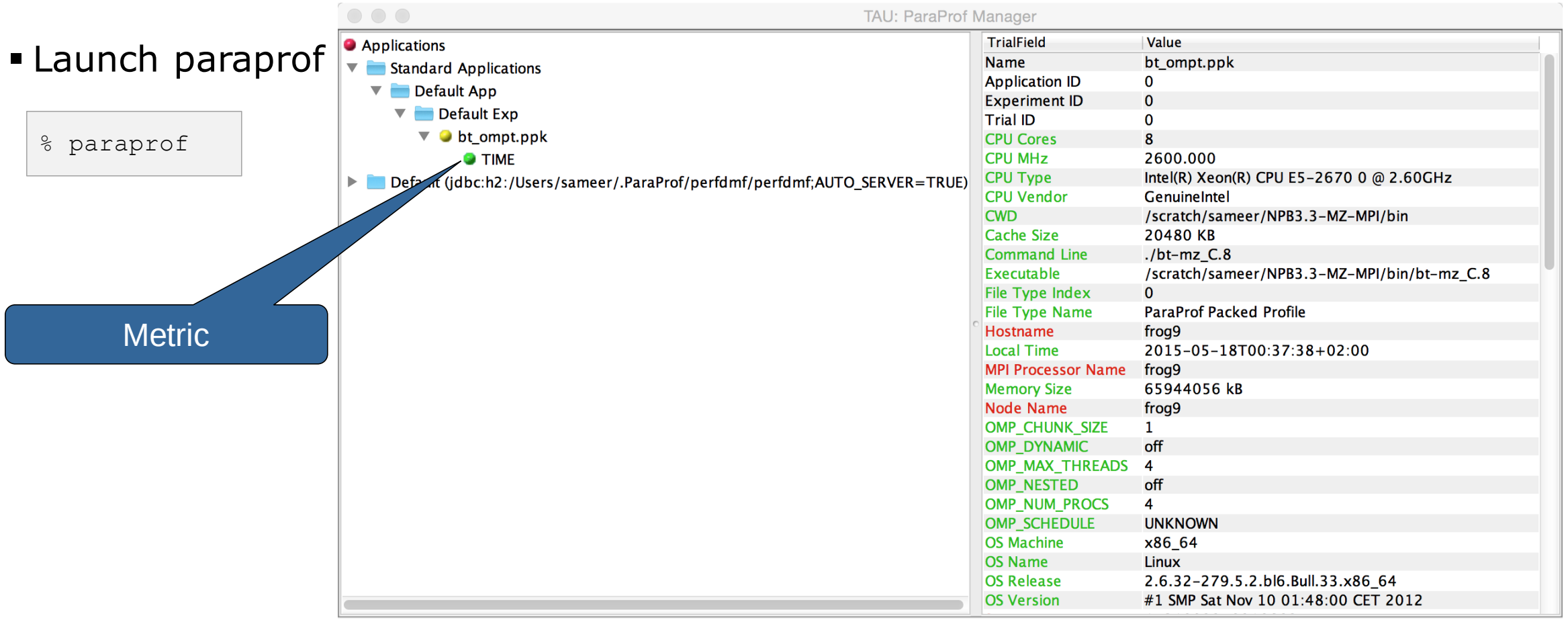
ParaProf Profile Analysis Framework



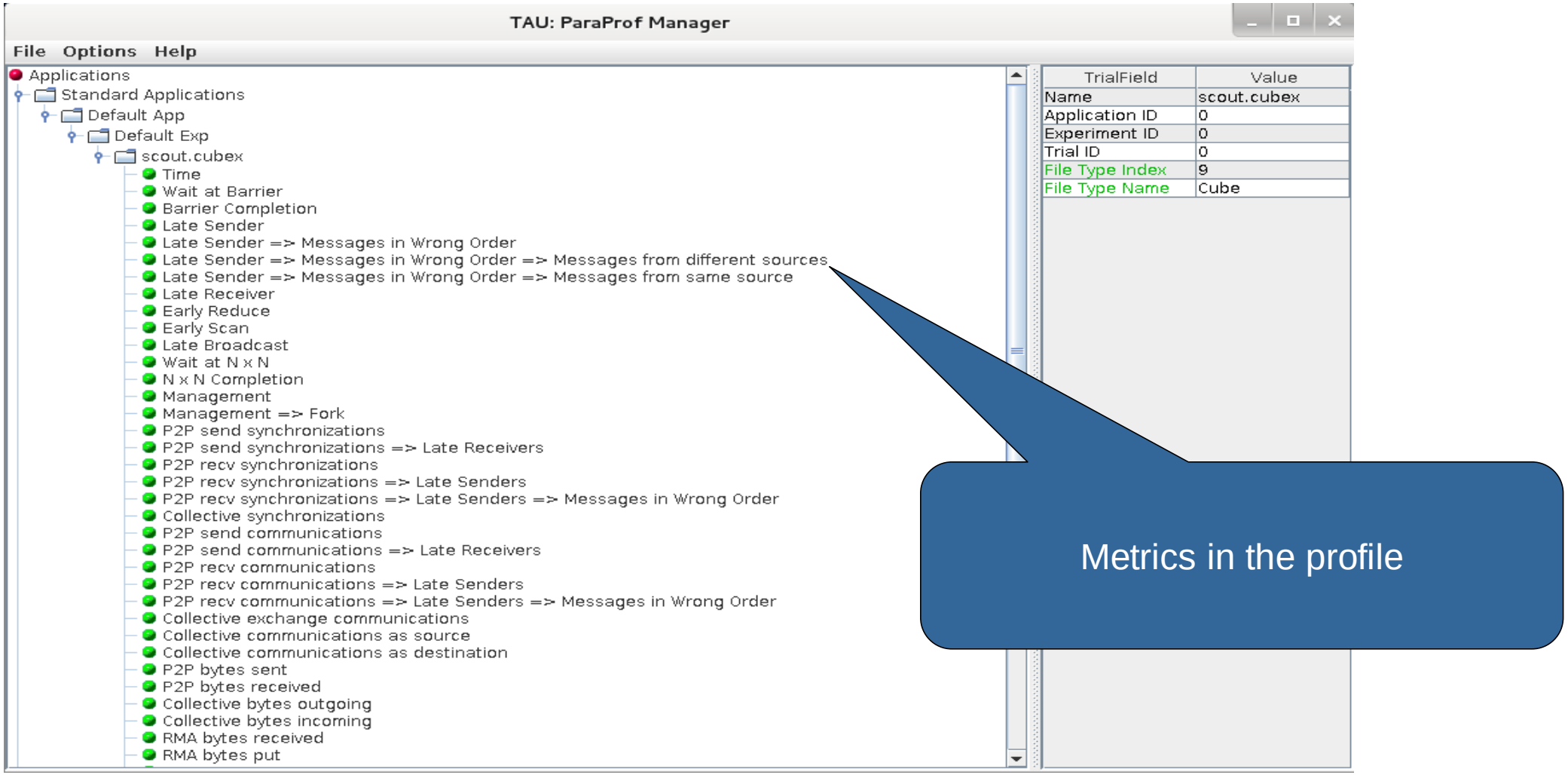
TAU Analysis Tools: paraprof

- Launch paraprof

```
% paraprof
```



ParaProf Manager Widow: scout.cubex

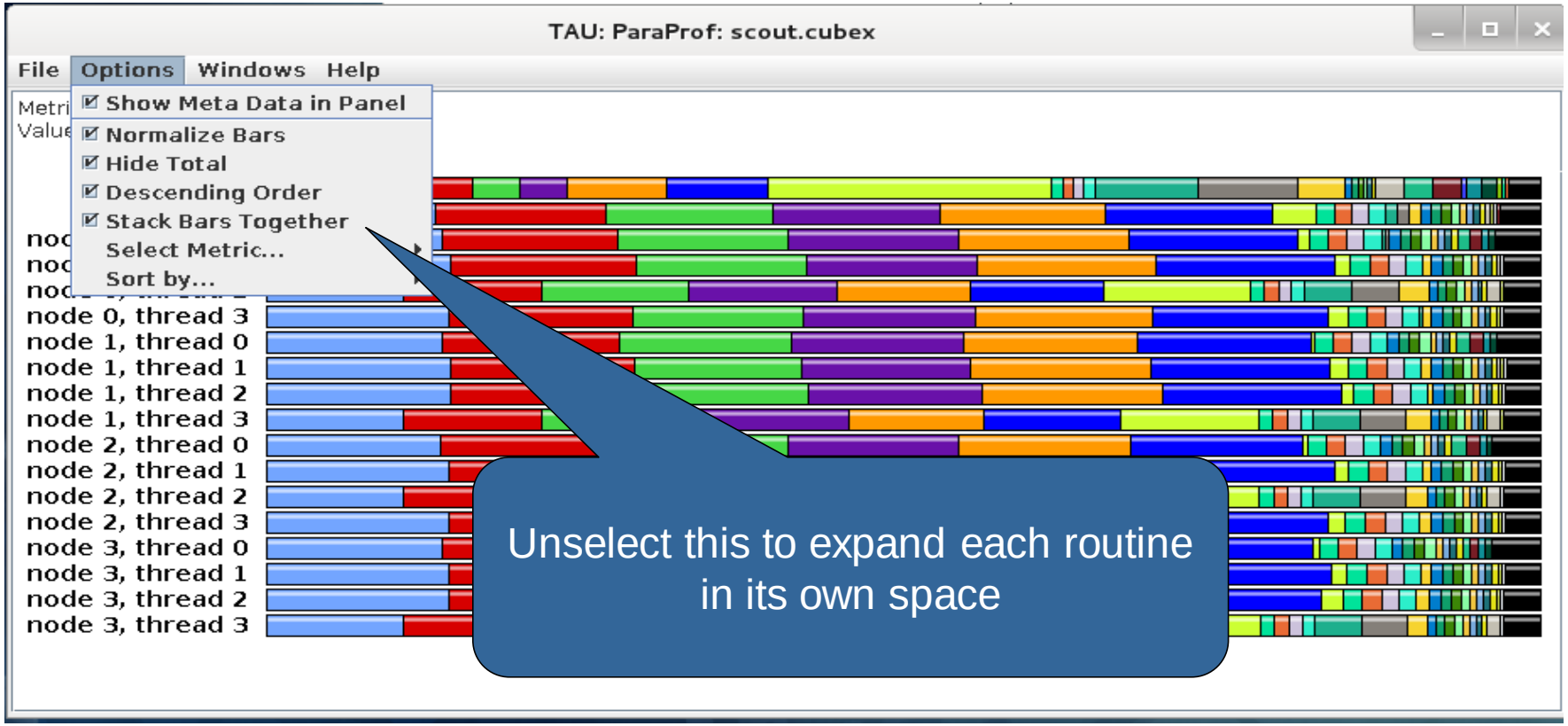


Paraprof main window

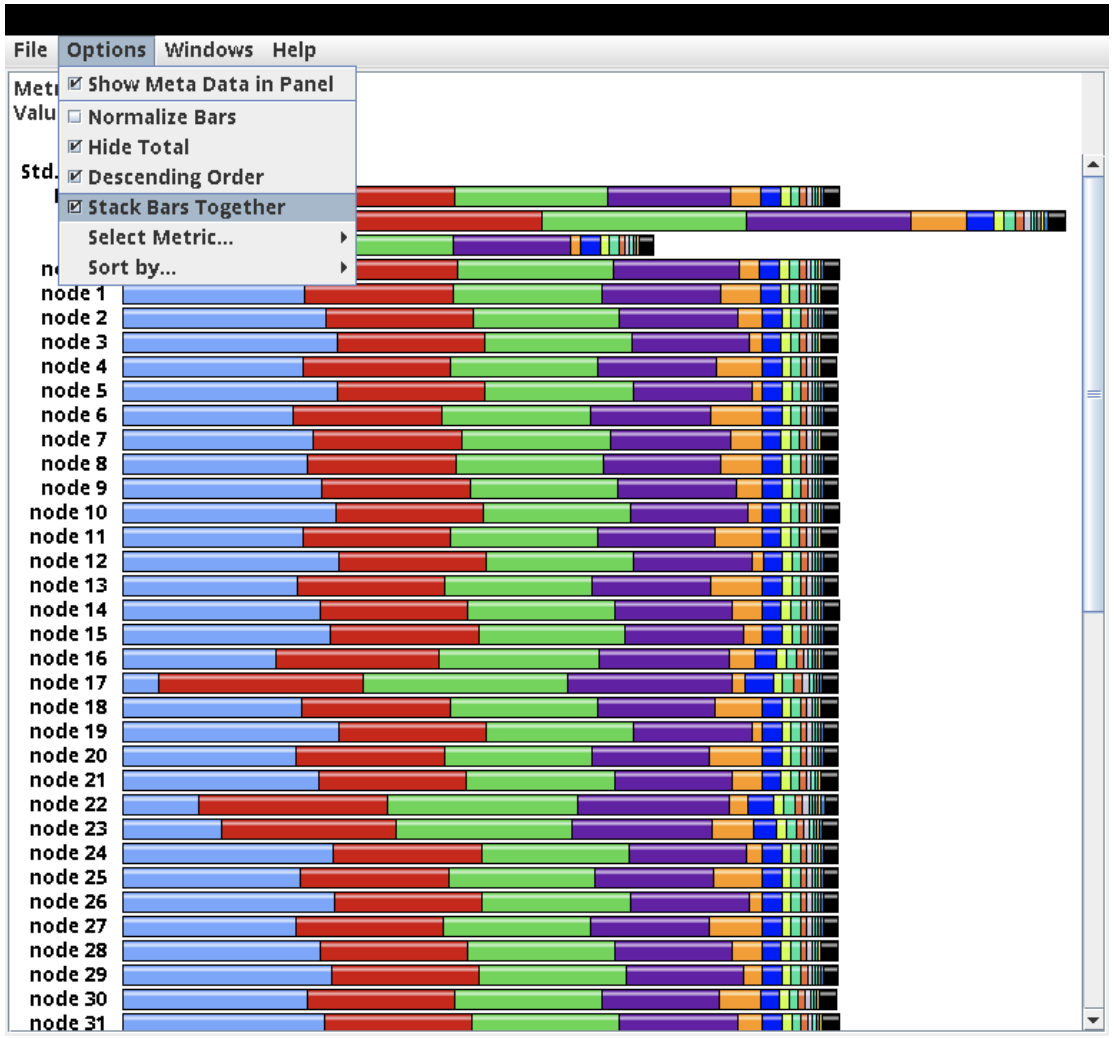
Colors represent code regions

Options -> uncheck Stack Bars Together

Paraprof main window



ParaProf Profile Browser



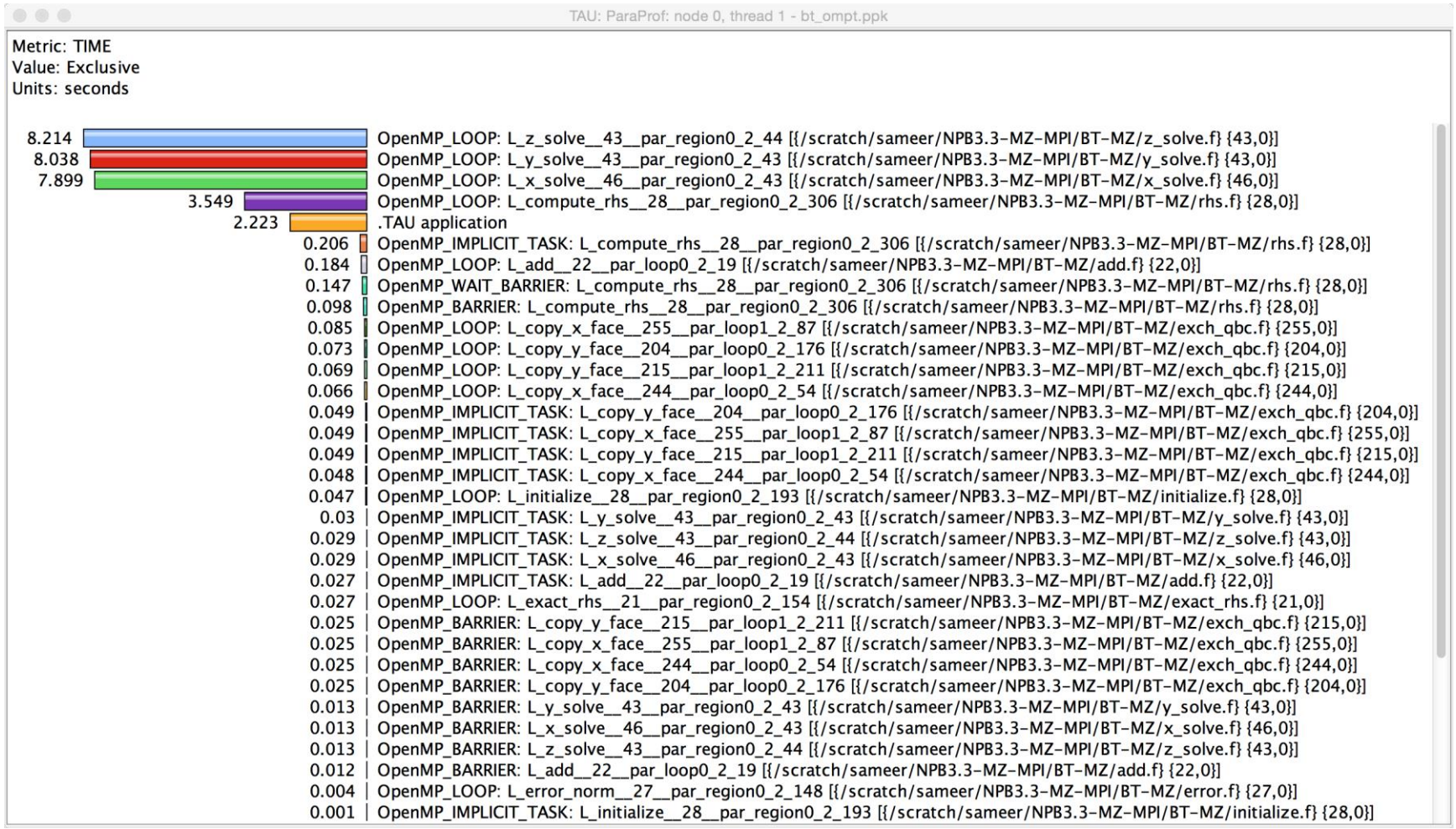
Paraprof main window

Left/right click here

Each routine occupies its own space. Can see the extent of imbalance across all threads.

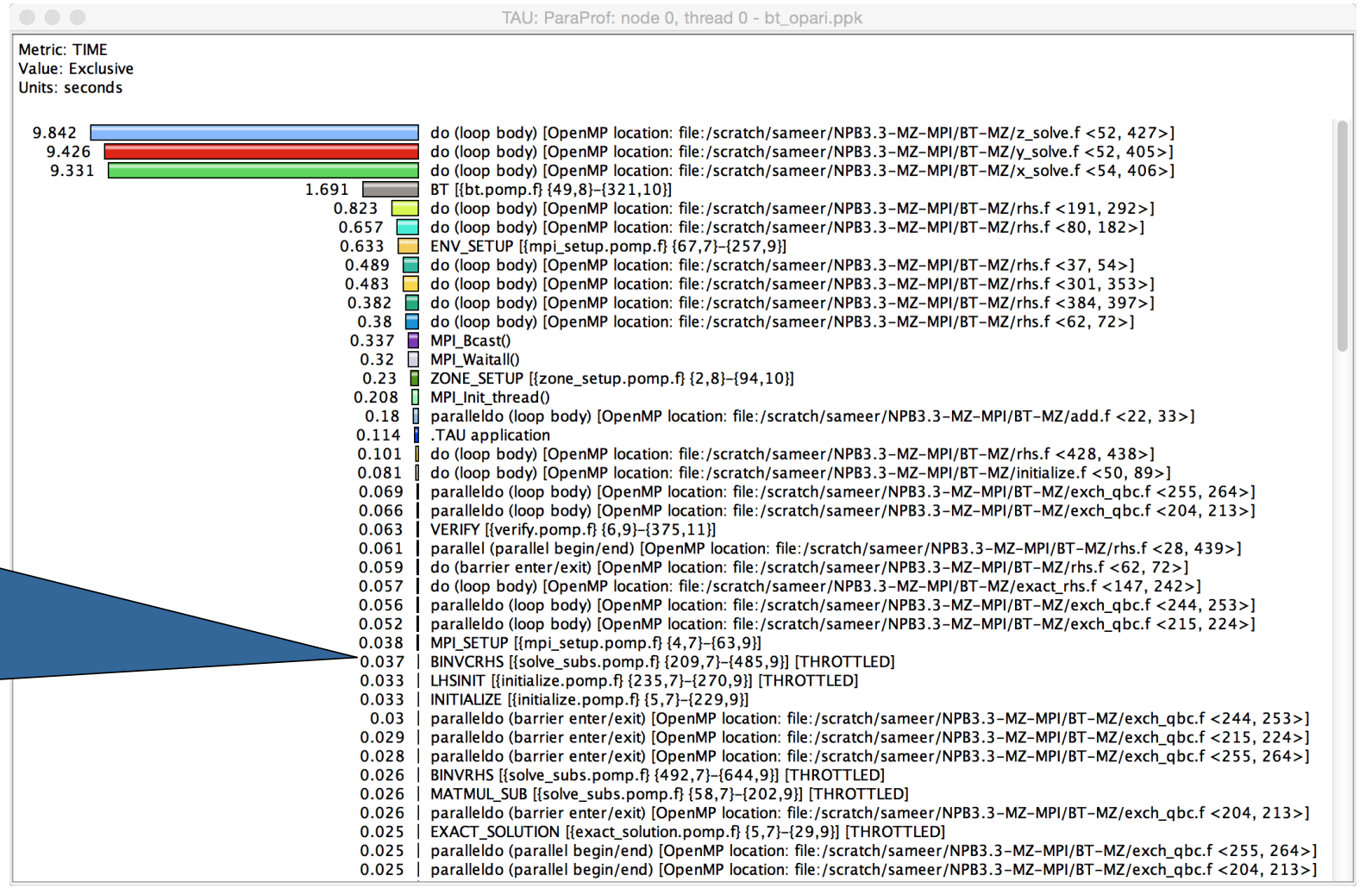
Paraprof node window (function barchart window)

Exclusive time spent in each code region (OpenMP loop) is shown here for MPI rank 0 thread 1

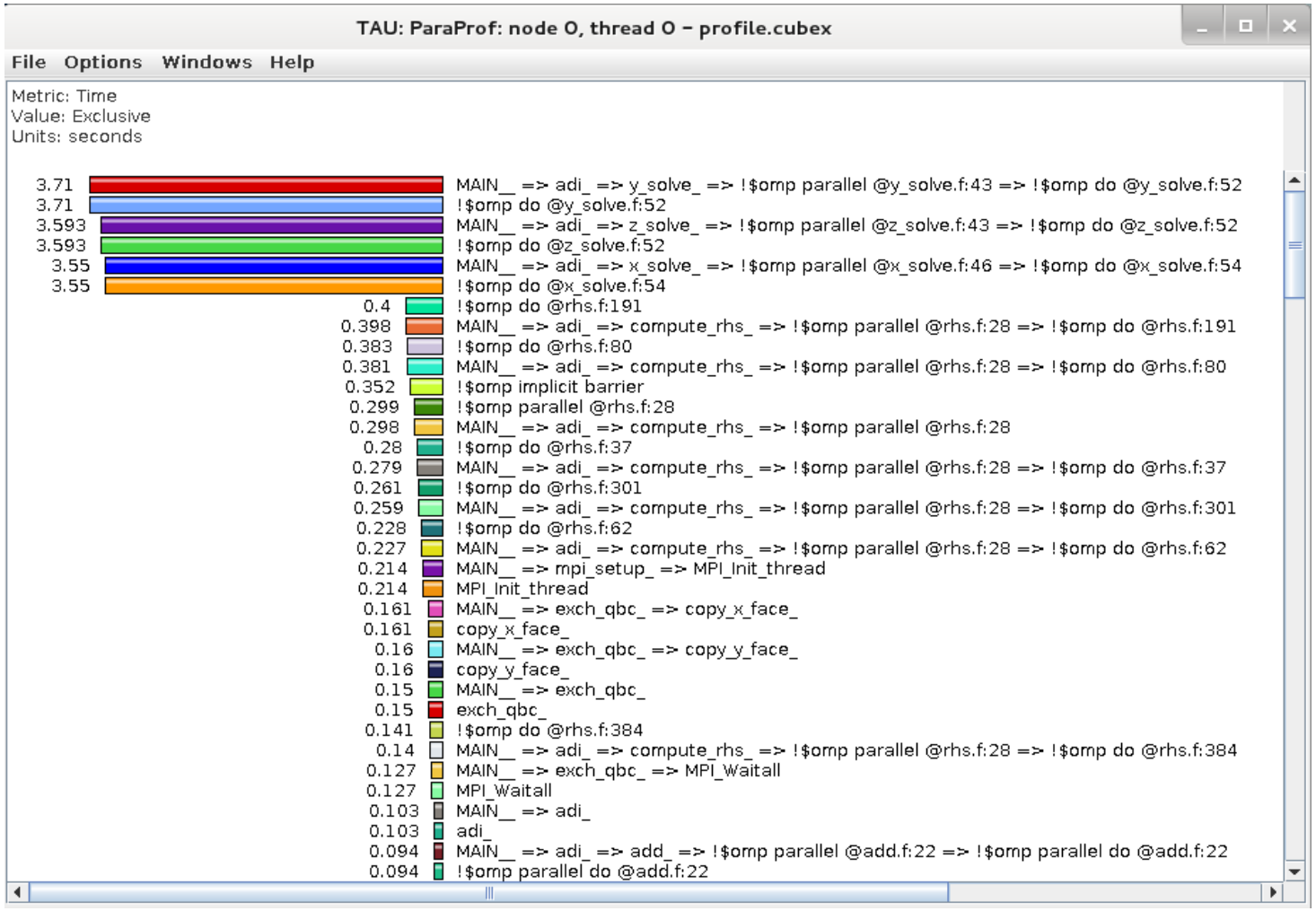


Instrumenting Source Code with PDT and Opari

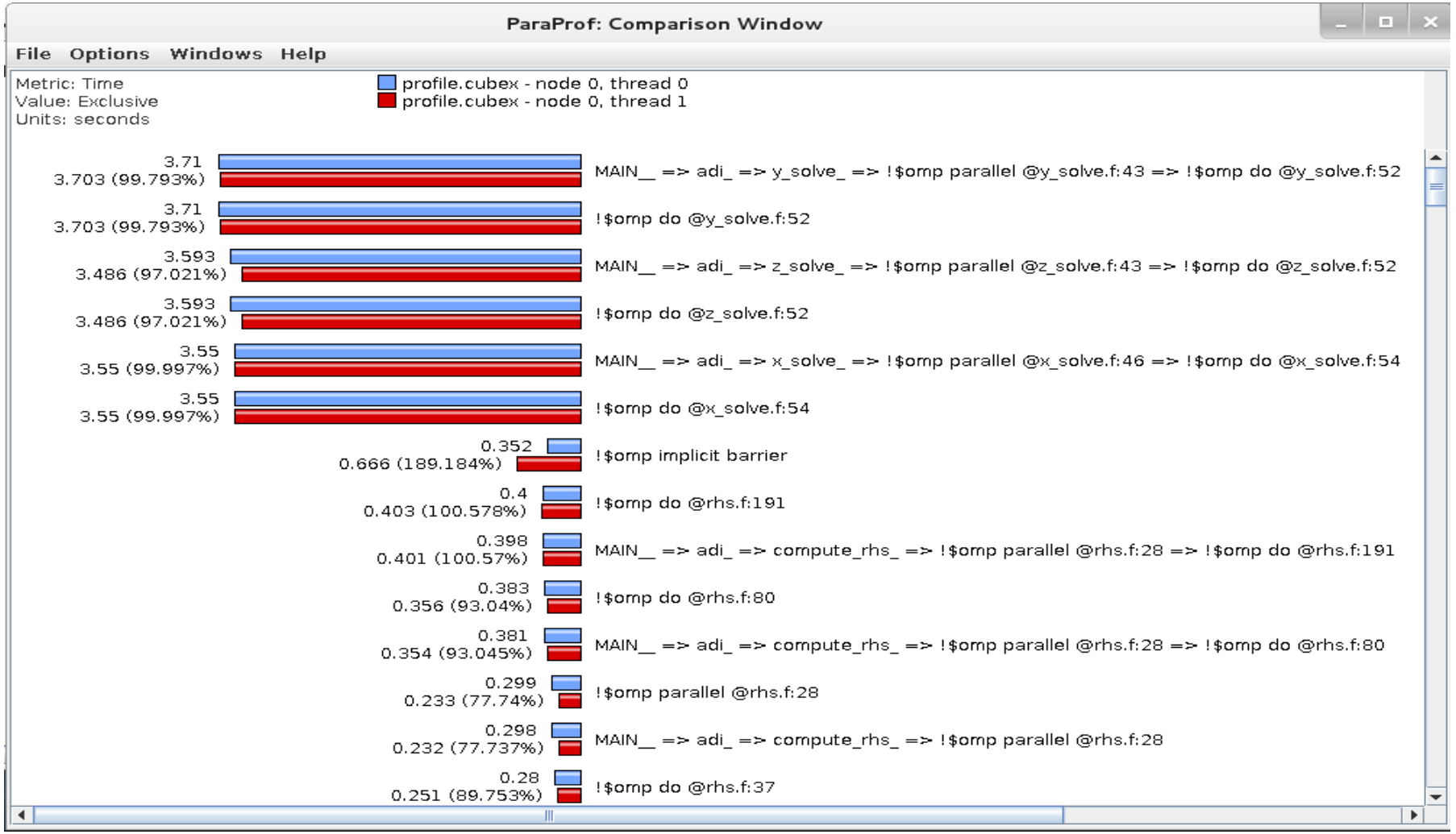
Frequently executing lightweight routines are automatically throttled at runtime. Reduces runtime dilation.



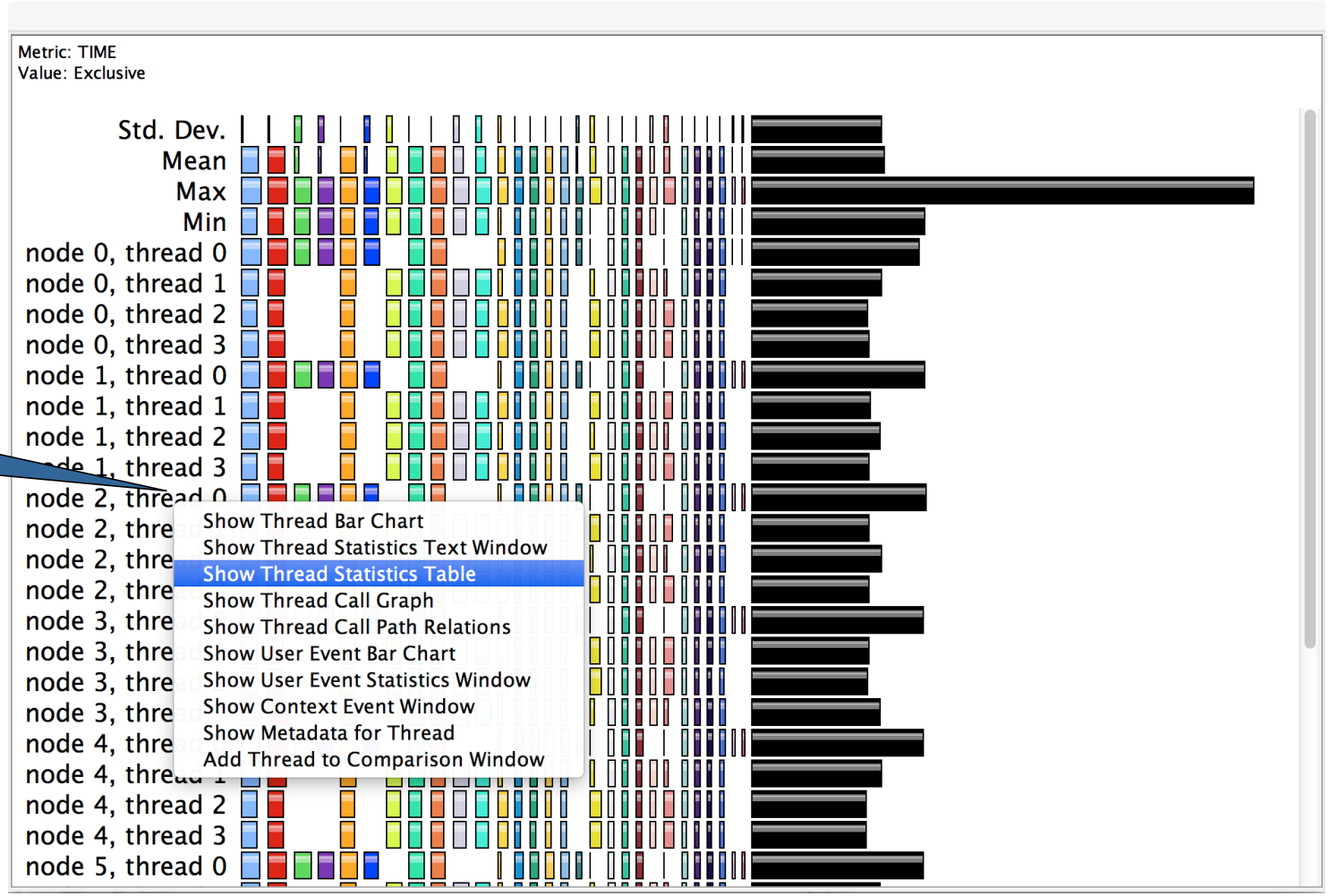
ParaProf: Node view in a callpath profile



ParaProf: Add thread to comparison window

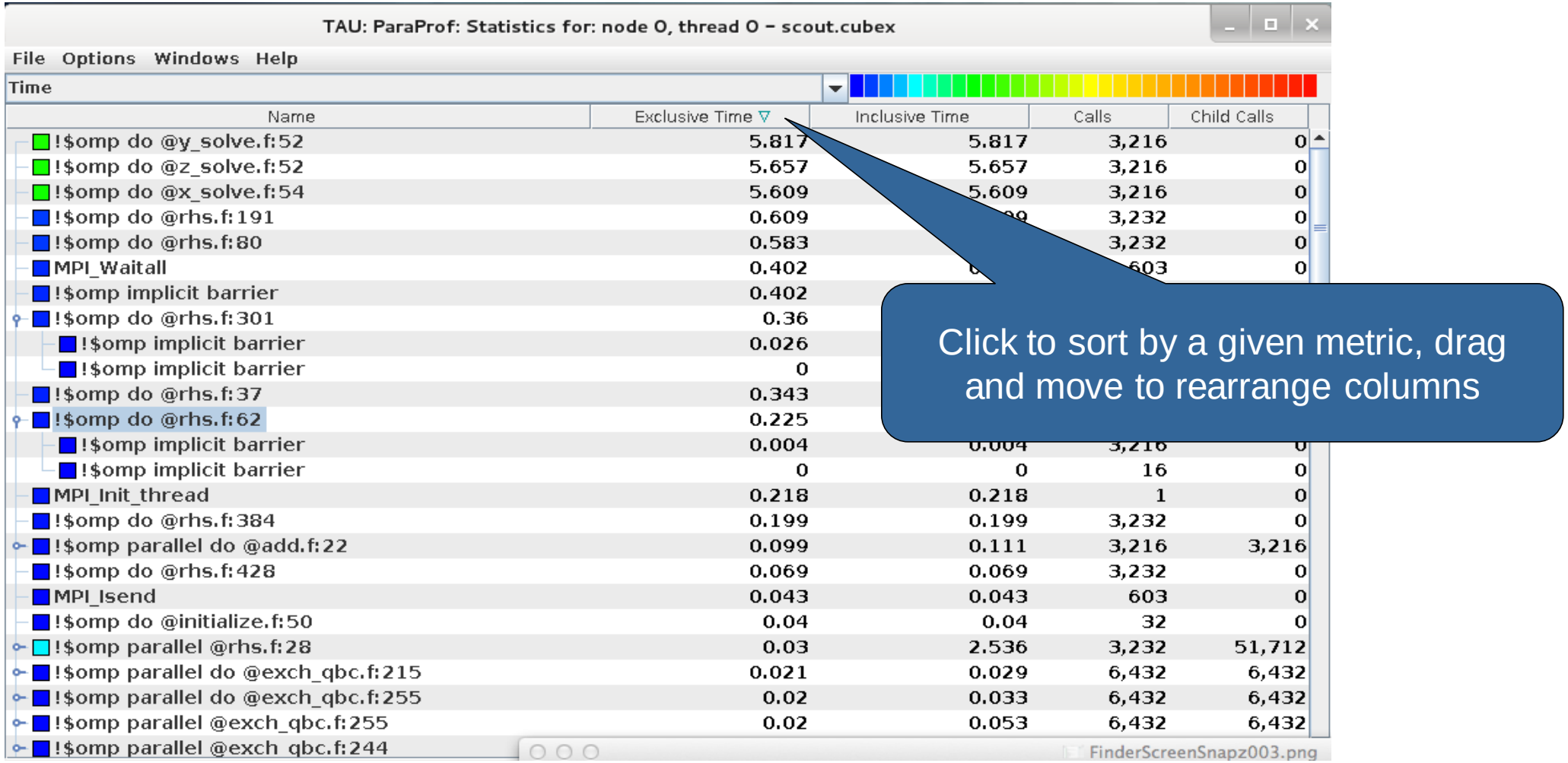


Paraprof Thread Statistics Table with TAU_SAMPLING=1



Right click here

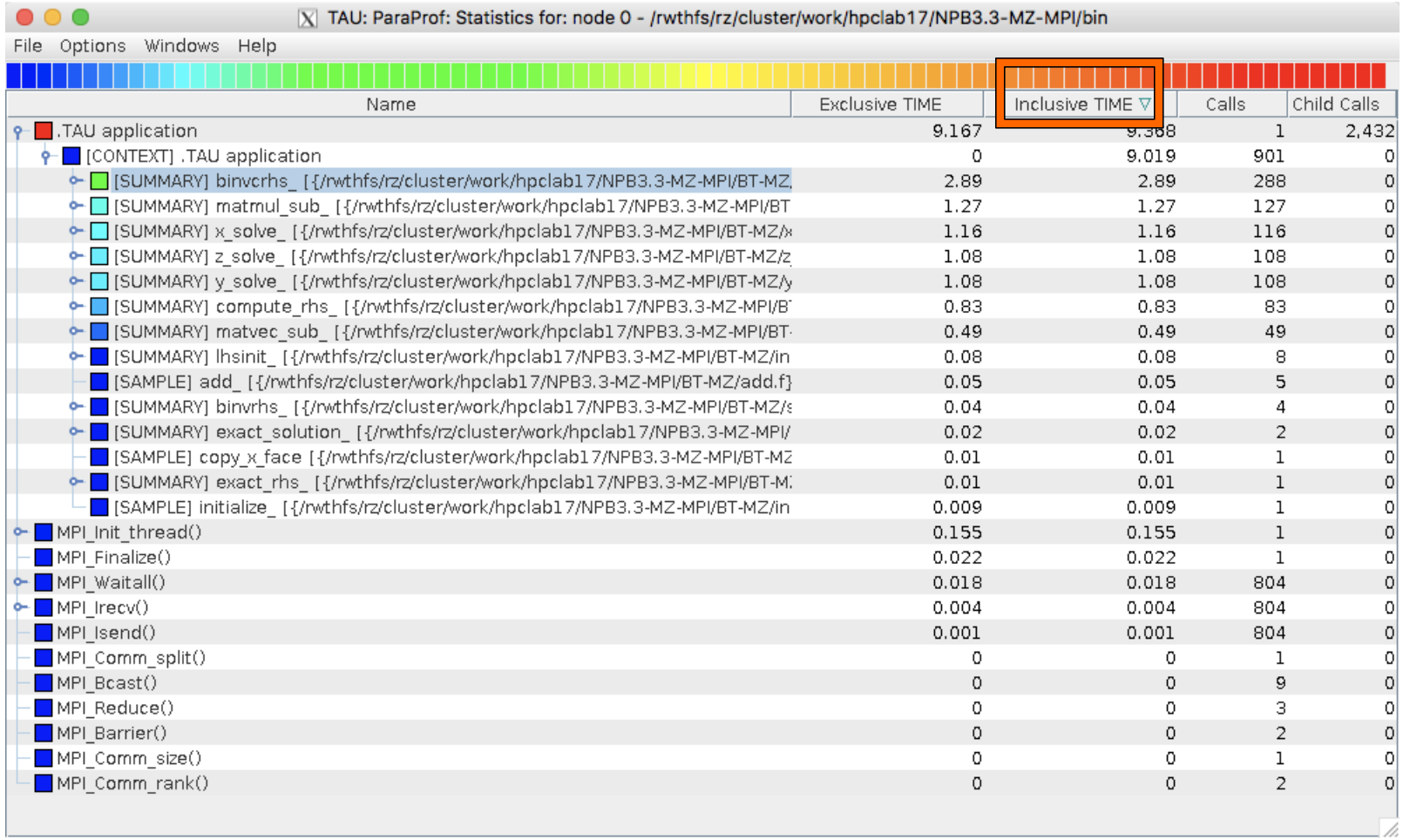
ParaProf: Thread Statistics Table



ParaProf

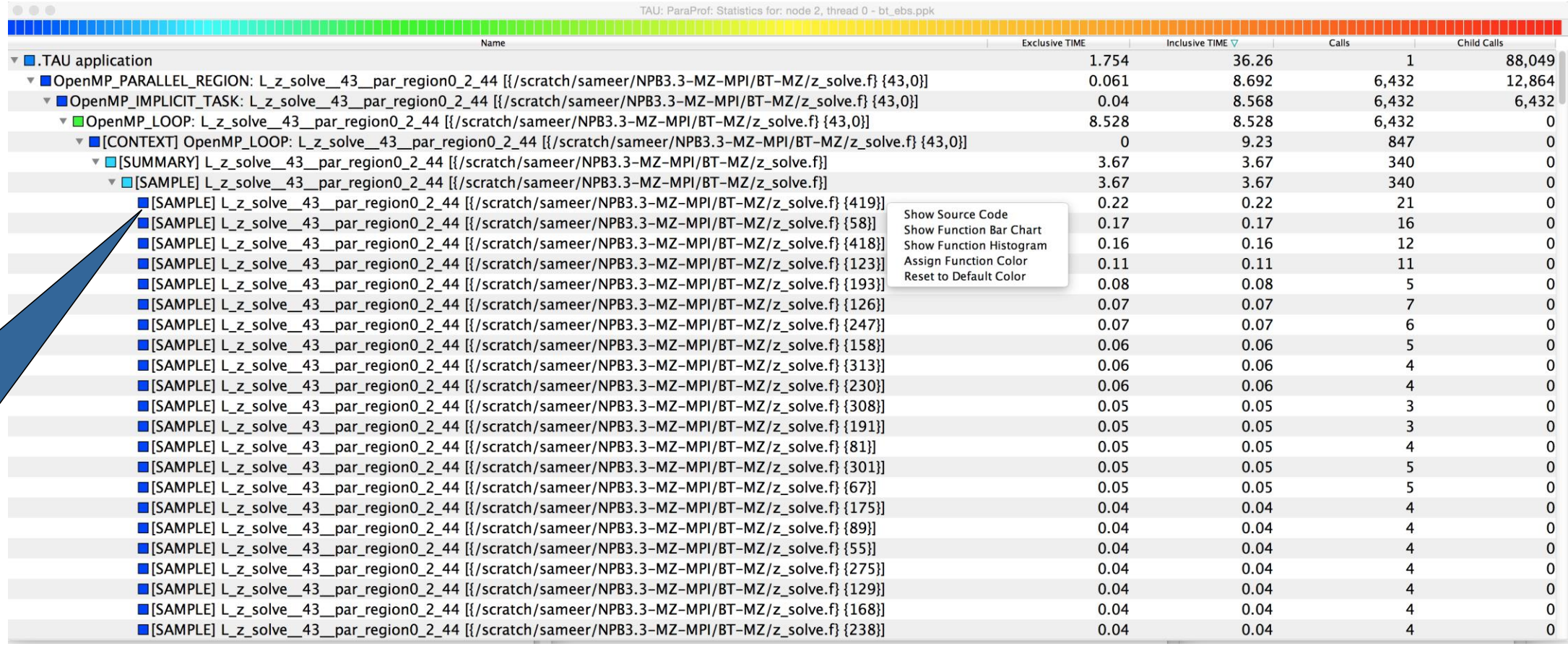
- Click on Columns:
- to sort by incl time

- Open binvcrhs
- Click on Sample



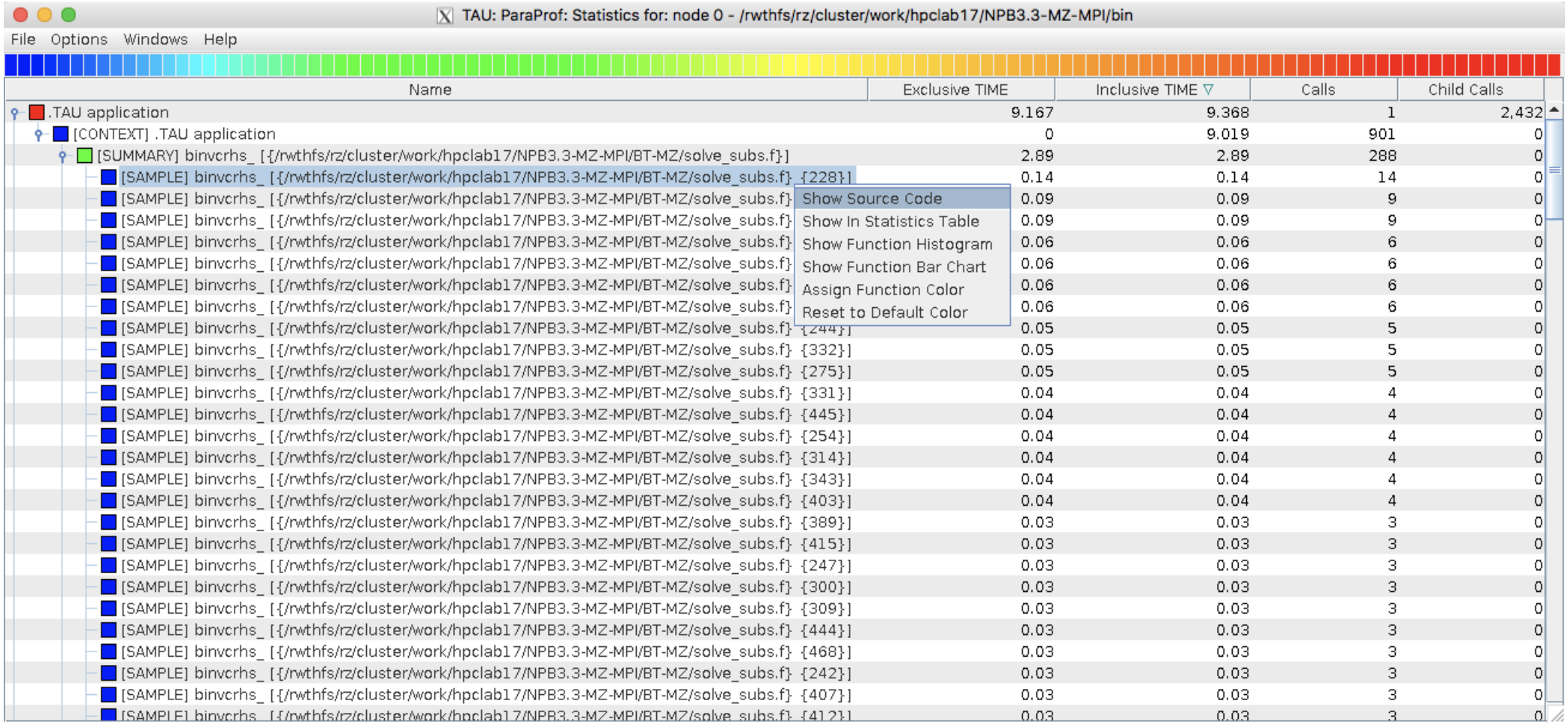
Name	Exclusive TIME	Inclusive TIME ▽	Calls	Child Calls
.TAU application	9.167	9.388	1	2,432
[CONTEXT] .TAU application	0	9.019	901	0
[SUMMARY] binvcrhs_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ	2.89	2.89	288	0
[SUMMARY] matmul_sub_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT	1.27	1.27	127	0
[SUMMARY] x_solve_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/x	1.16	1.16	116	0
[SUMMARY] z_solve_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/z	1.08	1.08	108	0
[SUMMARY] y_solve_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/y	1.08	1.08	108	0
[SUMMARY] compute_rhs_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/B	0.83	0.83	83	0
[SUMMARY] matvec_sub_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT	0.49	0.49	49	0
[SUMMARY] lhsinit_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/in	0.08	0.08	8	0
[SAMPLE] add_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/add.f}	0.05	0.05	5	0
[SUMMARY] binvrhs_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/s	0.04	0.04	4	0
[SUMMARY] exact_solution_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/	0.02	0.02	2	0
[SAMPLE] copy_x_face [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ	0.01	0.01	1	0
[SUMMARY] exact_rhs_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-M	0.01	0.01	1	0
[SAMPLE] initialize_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/in	0.009	0.009	1	0
MPI_Init_thread()	0.155	0.155	1	0
MPI_Finalize()	0.022	0.022	1	0
MPI_Waitall()	0.018	0.018	804	0
MPI_Irecv()	0.004	0.004	804	0
MPI_Isend()	0.001	0.001	804	0
MPI_Comm_split()	0	0	1	0
MPI_Bcast()	0	0	9	0
MPI_Reduce()	0	0	3	0
MPI_Barrier()	0	0	2	0
MPI_Comm_size()	0	0	1	0
MPI_Comm_rank()	0	0	2	0

Paraprof Thread Statistics Table

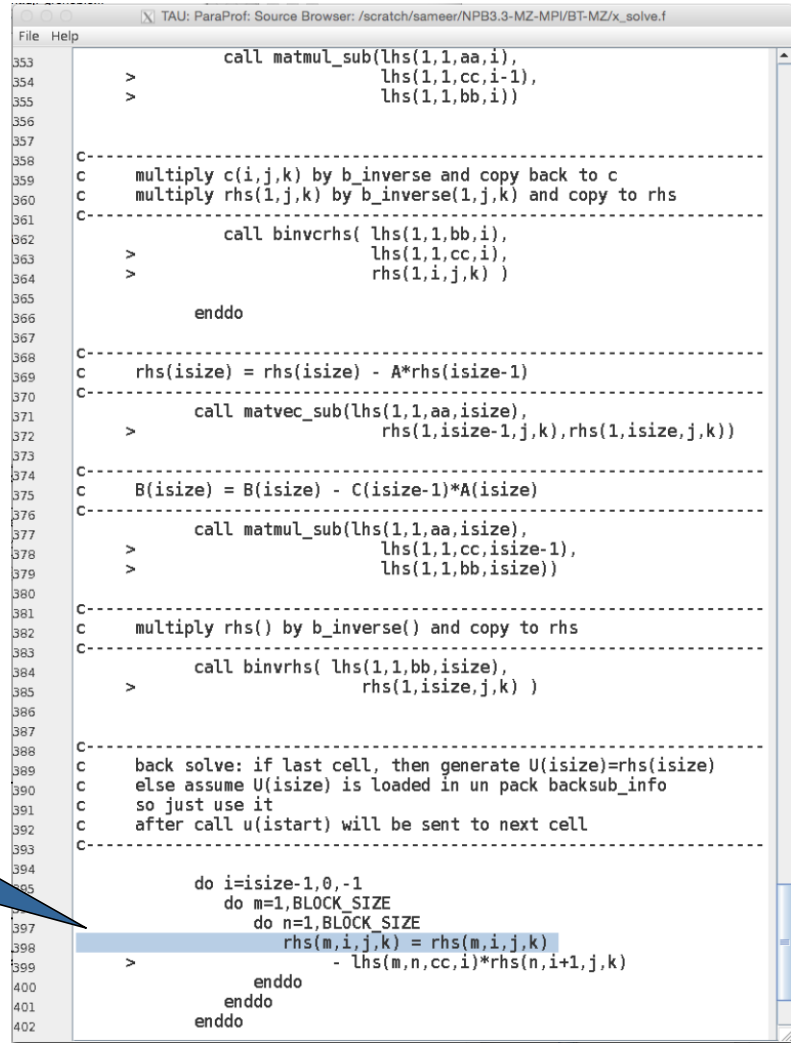


Right click here and choose “Show Source Code” for a sample

ParaProf

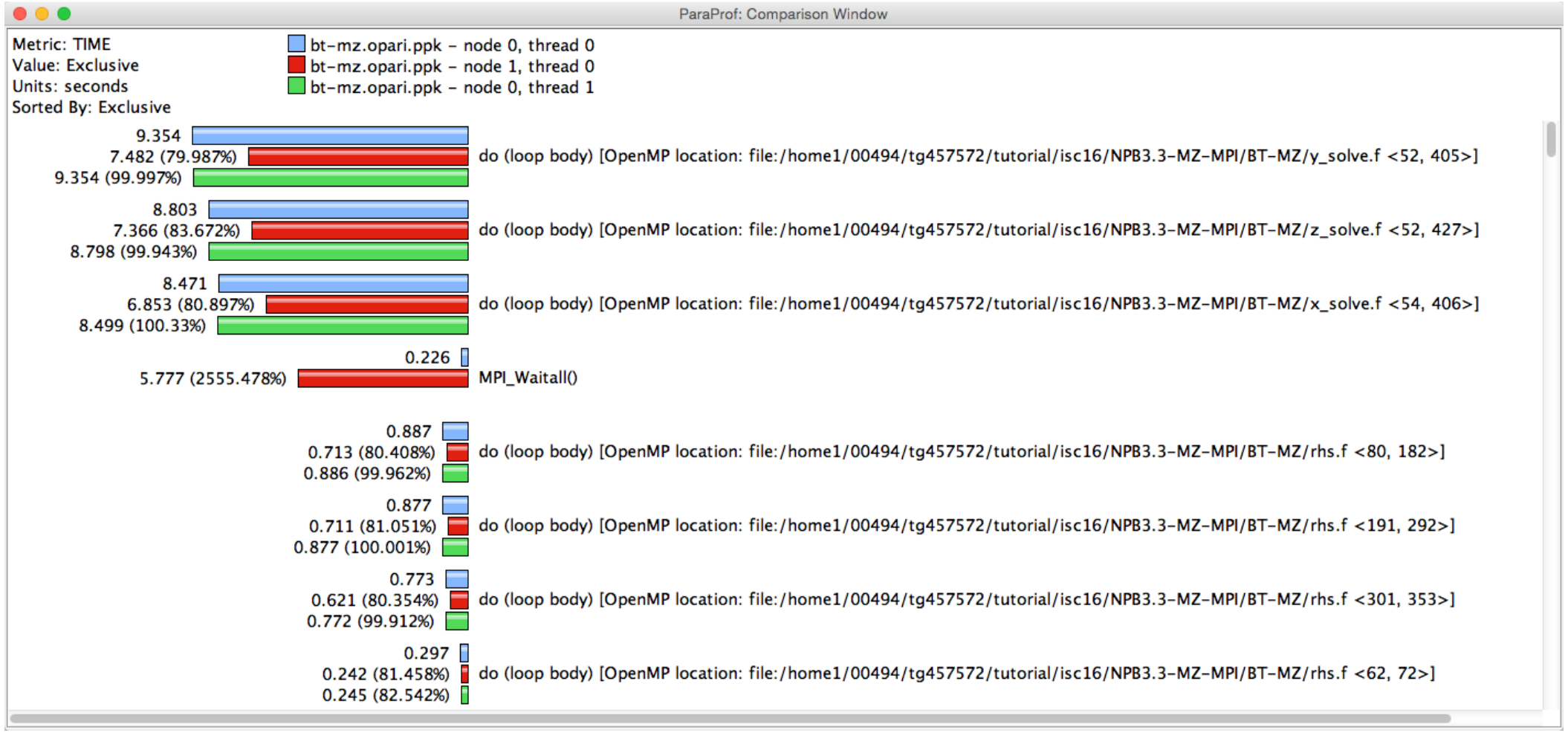

TAU: ParaProf: Statistics for: node 0 - /rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/bin
File Options Windows Help
Name
Exclusive TIME
Inclusive TIME
Calls
Child Calls
.TAU application 9.167 9.368 1 2,432
[CONTEXT] .TAU application 0 9.019 901 0
[SUMMARY] binvcrhs_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f}] 2.89 2.89 288 0
[SAMPLE] binvcrhs_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} {228}] 0.14 0.14 14 0
[SAMPLE] binvcrhs_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} 0.09 0.09 9 0
[SAMPLE] binvcrhs_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} 0.09 0.09 9 0
[SAMPLE] binvcrhs_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} 0.06 0.06 6 0
[SAMPLE] binvcrhs_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} 0.06 0.06 6 0
[SAMPLE] binvcrhs_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} 0.06 0.06 6 0
[SAMPLE] binvcrhs_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} 0.06 0.06 6 0
[SAMPLE] binvcrhs_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} 0.05 0.05 5 0
[SAMPLE] binvcrhs_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} {332}] 0.05 0.05 5 0
[SAMPLE] binvcrhs_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} {275}] 0.05 0.05 5 0
[SAMPLE] binvcrhs_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} {331}] 0.04 0.04 4 0
[SAMPLE] binvcrhs_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} {445}] 0.04 0.04 4 0
[SAMPLE] binvcrhs_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} {254}] 0.04 0.04 4 0
[SAMPLE] binvcrhs_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} {314}] 0.04 0.04 4 0
[SAMPLE] binvcrhs_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} {343}] 0.04 0.04 4 0
[SAMPLE] binvcrhs_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} {403}] 0.04 0.04 4 0
[SAMPLE] binvcrhs_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} {389}] 0.03 0.03 3 0
[SAMPLE] binvcrhs_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} {415}] 0.03 0.03 3 0
[SAMPLE] binvcrhs_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} {247}] 0.03 0.03 3 0
[SAMPLE] binvcrhs_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} {300}] 0.03 0.03 3 0
[SAMPLE] binvcrhs_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} {309}] 0.03 0.03 3 0
[SAMPLE] binvcrhs_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} {444}] 0.03 0.03 3 0
[SAMPLE] binvcrhs_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} {468}] 0.03 0.03 3 0
[SAMPLE] binvcrhs_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} {242}] 0.03 0.03 3 0
[SAMPLE] binvcrhs_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} {407}] 0.03 0.03 3 0
Show Source Code
Show In Statistics Table
Show Function Histogram
Show Function Bar Chart
Assign Function Color
Reset to Default Color

Statement Level Profiling with TAU

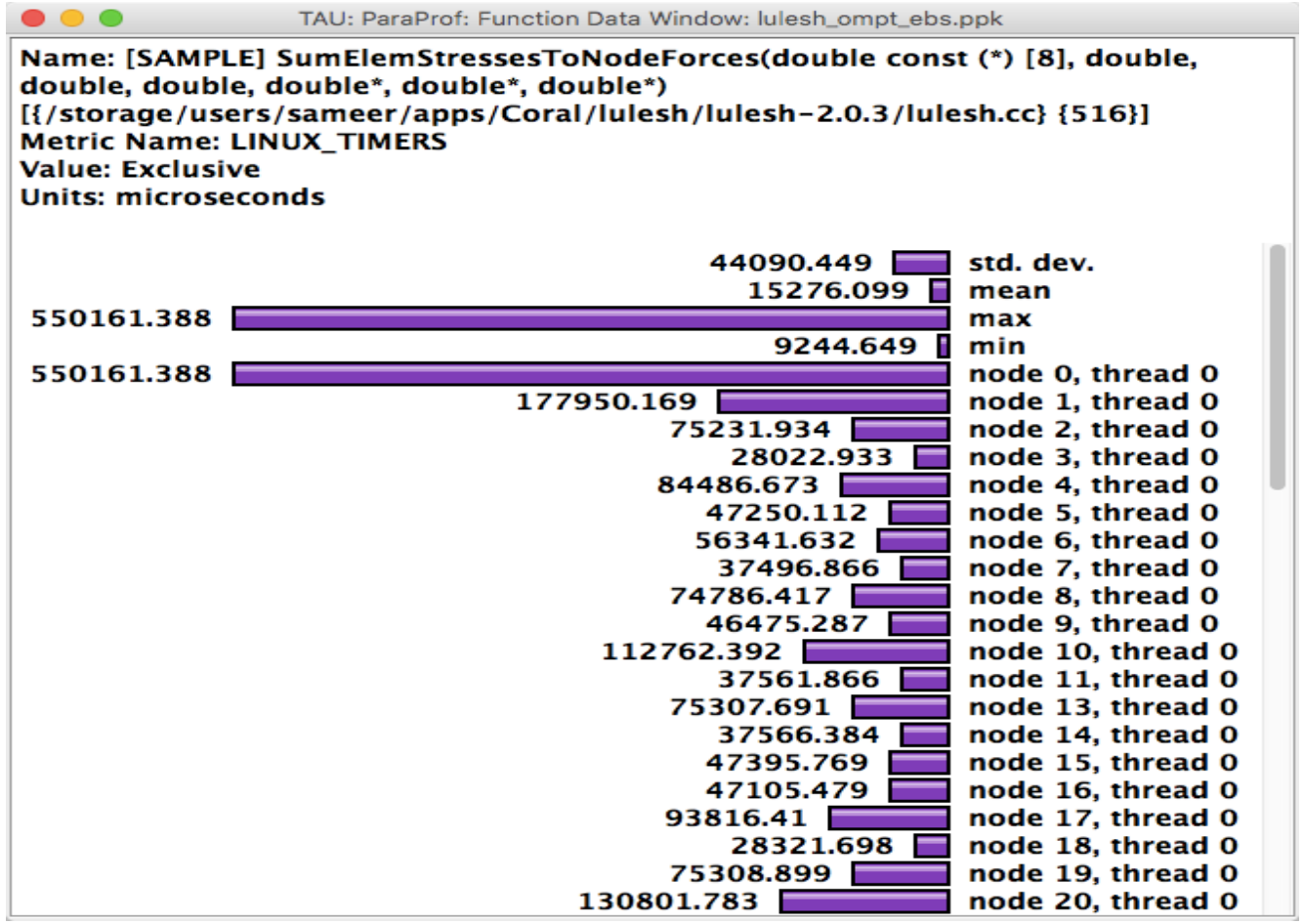


Source location where samples are taken. Compute intensive region.

ParaProf Comparison Window



TAU – Event Based Sampling (EBS)



% export TAU_SAMPLING=1

Examples: Callstack Sampling in TAU

TAU: ParaProf: Statistics for: n,c,t 2,0,0 - gamess_unw_call_ebs.ppk

Name	Inclusive TIME	Calls
.TAU application	79.592	1
MPI_Recv()	75.607	6,870
[CONTEXT] MPI_Recv()	74.848	1,497
[UNWIND] /gpfs/mira-home/sameer/gamess-theta-tau/object/unport.f.410 [@] MAIN__ [{/gpfs/mira-home/sameer/gamess-theta-t	26.196	524
[UNWIND] /gpfs/mira-home/yuri/dist/Github/gamess-theta-tau/ddi/src/ddi_fortran.c.67 [@] beging_ [{/gpfs/mira-home/sameer/g	21.7	434
[UNWIND] /gpfs/mira-home/sameer/gamess-theta-tau/object/gamess.f.538 [@] main [{/gpfs/mira-home/sameer/gamess-theta-ta	11.85	237
[UNWIND] /gpfs/mira-home/yuri/dist/Github/gamess-theta-tau/ddi/src/ddi_init.c.113 [@] ddi_init_ [{/gpfs/mira-home/yuri/dist/Gi	8.701	174
[UNWIND] /gpfs/mira-home/yuri/dist/Github/gamess-theta-tau/ddi/src/ddi_server.c.99 [@] DDI_Init [{/gpfs/mira-home/yuri/dist/C	5.75	115
[UNWIND] /lib64/libc-2.22.so.0 [@] _start [{/home/abuild/rpmbuild/BUILD/glibc-2.22/csu/../sysdeps/x86_64/start.S} {118}]	0.2	4
[SAMPLE] GNII_DlaProgress [{/opt/cray/ugni/6.0.14-6.0.4.0_14.1__ge7db4a2.ari/lib64/libugni.so.0.6.0} {0}]	0.2	4
[UNWIND] [/opt/cray/ugni/6.0.14-6.0.4.0_14.1__ge7db4a2.ari/lib64/libugni.so.0.6.0.0] [@] UNRESOLVED UNKNOWN	0.15	3
[SAMPLE] GNI_CqGetEvent [{/opt/cray/ugni/6.0.14-6.0.4.0_14.1__ge7db4a2.ari/lib64/libugni.so.0.6.0} {0}]	0.051	1
[UNWIND] /opt/cray/pe/mpt/7.6.3/gni/mpich-intel/16.0/lib/libmpich_intel.so.3.0.1.0 [@] MPIDI_CH3I_Progress [{/opt/cray/pe/mpt/7	0.05	1
MPI_Finalize()	3.601	1
MPI_Send()	0.122	6,866
MPI_Init_thread()	0.112	1
[CONTEXT] .TAU application	0.05	1
MPI_Bcast()	0.014	6
MPI_Allgather()	0.004	3
MPI_Barrier()	0.003	7
MPI_Comm_create()	0.002	4
MPI_Gather()	0.002	1
MPI_Comm_split()	0.002	1
MPI_Group_intersection()	0.001	1
MPI_Comm_group()	0.001	1
MPI_Group_incl()	0	3
MPI_Comm_rank()	0	6
MPI_Comm_size()	0	2

% export TAU_SAMPLING=1; export TAU_EBS_UNWIND=1

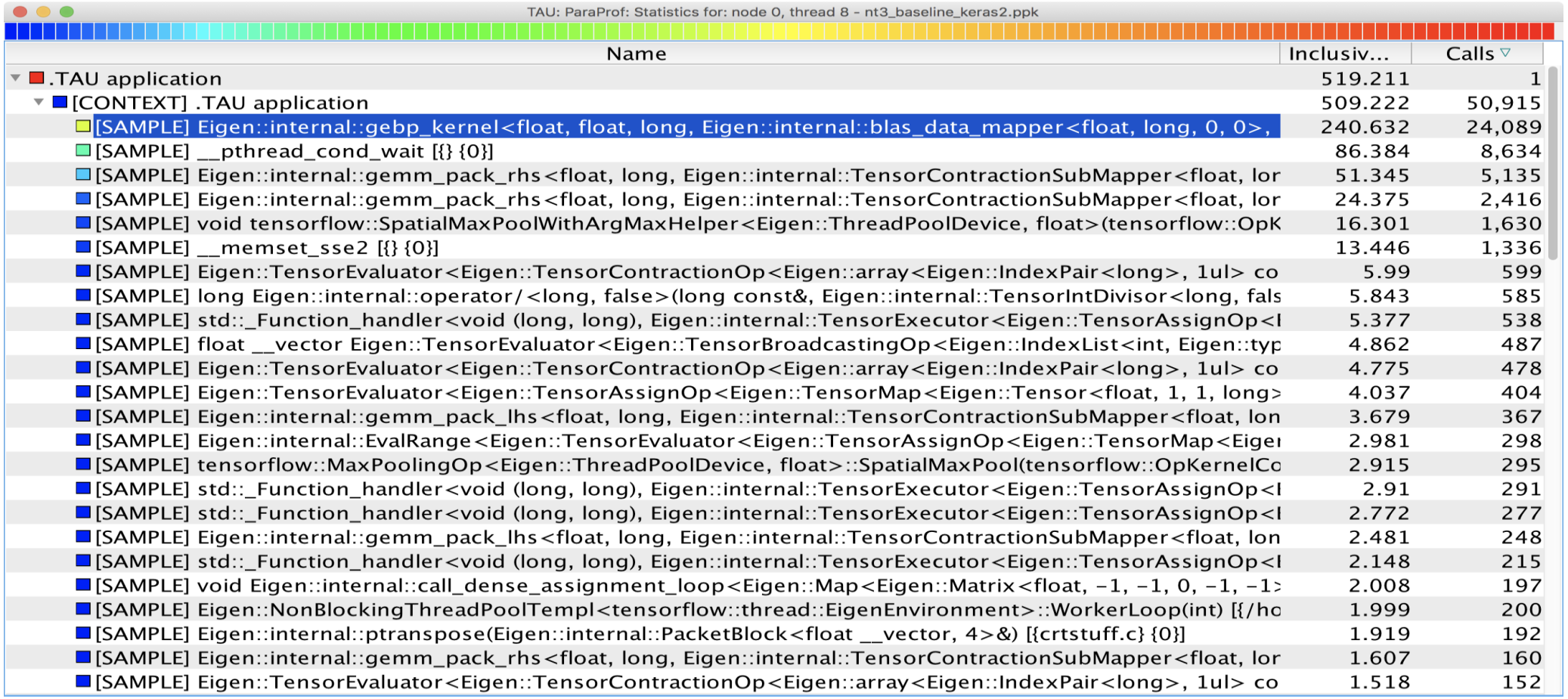
UNWINDING CALLSTACKS

% export TAU_SAMPLING=1; export TAU_EBS_UNWIND=1

UNWINDING CALLSTACKS

Name	Inclusive TIME	Calls
.TAU application	79.592	1
MPI_Recv()	75.607	6,870
[CONTEXT] MPI_Recv()	74.848	1,497
[UNWIND] /gpfs/mira-home/sameer/gamess-theta-tau/object/unport.f.410 [@] MAIN__ [{/gpfs/mira-home/sameer/gamess-theta-	26.196	524
[UNWIND] /gpfs/mira-home/yuri/dist/Github/gamess-theta-tau/ddi/src/ddi_fortran.c.67 [@] beging_ [{/gpfs/mira-home/sameer/g	21.7	434
[UNWIND] /gpfs/mira-home/sameer/gamess-theta-tau/object/gamess.f.538 [@] main [{/gpfs/mira-home/sameer/gamess-theta-ta	11.85	237
[UNWIND] /gpfs/mira-home/sameer/gamess-theta-tau/object/unport.f.410 [@] MAIN__ [{/gpfs/mira-home/sameer/gamess-thet	11.85	237
[UNWIND] /gpfs/mira-home/yuri/dist/Github/gamess-theta-tau/ddi/src/ddi_fortran.c.67 [@] beging_ [{/gpfs/mira-home/sam	11.85	237
[UNWIND] /gpfs/mira-home/yuri/dist/Github/gamess-theta-tau/ddi/src/ddi_init.c.113 [@] ddi_init_ [{/gpfs/mira-home/yur	11.85	237
[UNWIND] /gpfs/mira-home/yuri/dist/Github/gamess-theta-tau/ddi/src/ddi_server.c.99 [@] DDI_Init [{/gpfs/mira-home/	11.85	237
[UNWIND] /gpfs/mira-home/yuri/dist/Github/gamess-theta-tau/ddi/src/ddi_recv.c.65 [@] DDI_Server [{/gpfs/mira-ho	11.85	237
[UNWIND] /lus/theta-fs0/software/perftools/tau/tau-2.26.3/src/Profile/TauMpi.c.2371 [@] DDI_Recv_request [{/gpfs	11.85	237
[UNWIND] /opt/cray/pe/mpt/7.6.3/gni/mpich-intel/16.0/lib/libmpich_intel.so.3.0.1.0 [@] MPI_Recv [{/lus/theta-fs(	11.85	237
[UNWIND] /opt/cray/pe/mpt/7.6.3/gni/mpich-intel/16.0/lib/libmpich_intel.so.3.0.1.0 [@] PMPI_Recv [{/opt/cray/	11.7	234
[SAMPLE] MPIDI_CH3I_Progress [{/opt/cray/pe/mpt/7.6.3/gni/mpich-intel/16.0/lib/libmpich_intel.so.3.0.1} {0	11.3	226
[SAMPLE] MPIDU_Sched_are_pending [{/opt/cray/pe/mpt/7.6.3/gni/mpich-intel/16.0/lib/libmpich_intel.so.3.0	0.2	4
[SAMPLE] MPID_nem_gni_poll [{/opt/cray/pe/mpt/7.6.3/gni/mpich-intel/16.0/lib/libmpich_intel.so.3.0.1} {0}]	0.15	3
[SAMPLE] MPID_nem_network_poll [{/opt/cray/pe/mpt/7.6.3/gni/mpich-intel/16.0/lib/libmpich_intel.so.3.0.1	0.05	1
[UNWIND] ch3_progress.c.0 [@] PMPI_Recv [{/opt/cray/pe/mpt/7.6.3/gni/mpich-intel/16.0/lib/libmpich_intel.so.	0.15	3
[UNWIND] /gpfs/mira-home/yuri/dist/Github/gamess-theta-tau/ddi/src/ddi_init.c.113 [@] ddi_init_ [{/gpfs/mira-home/yuri/dist/G	8.701	174
[UNWIND] /gpfs/mira-home/yuri/dist/Github/gamess-theta-tau/ddi/src/ddi_server.c.99 [@] DDI_Init [{/gpfs/mira-home/yuri/dist/(	5.75	115
[UNWIND] /lib64/libc-2.22.so.0 [@] _start [{/home/abuild/rpmbuild/BUILD/glibc-2.22/csu/../sysdeps/x86_64/start.S} {118}]	0.2	4
[SAMPLE] GNII_DlaProgress [{/opt/cray/ugni/6.0.14-6.0.4.0_14.1__ge7db4a2.ari/lib64/libugni.so.0.6.0} {0}]	0.2	4
[UNWIND] [/opt/cray/ugni/6.0.14-6.0.4.0_14.1__ge7db4a2.ari/lib64/libugni.so.0.6.0.0] [@] UNRESOLVED UNKNOWN	0.15	3
[SAMPLE] GNI_CqGetEvent [{/opt/cray/ugni/6.0.14-6.0.4.0_14.1__ge7db4a2.ari/lib64/libugni.so.0.6.0} {0}]	0.051	1
[UNWIND] /opt/cray/pe/mpt/7.6.3/gni/mpich-intel/16.0/lib/libmpich_intel.so.3.0.1.0 [@] MPIDI_CH3I_Progress [{/opt/cray/pe/mpt/	0.05	1
MPI_Finalize()	3.601	1
MPI_Send()	0.122	6,866
MPI_Init_thread()	0.112	1
[CONTEXT] .TAU application	0.05	1

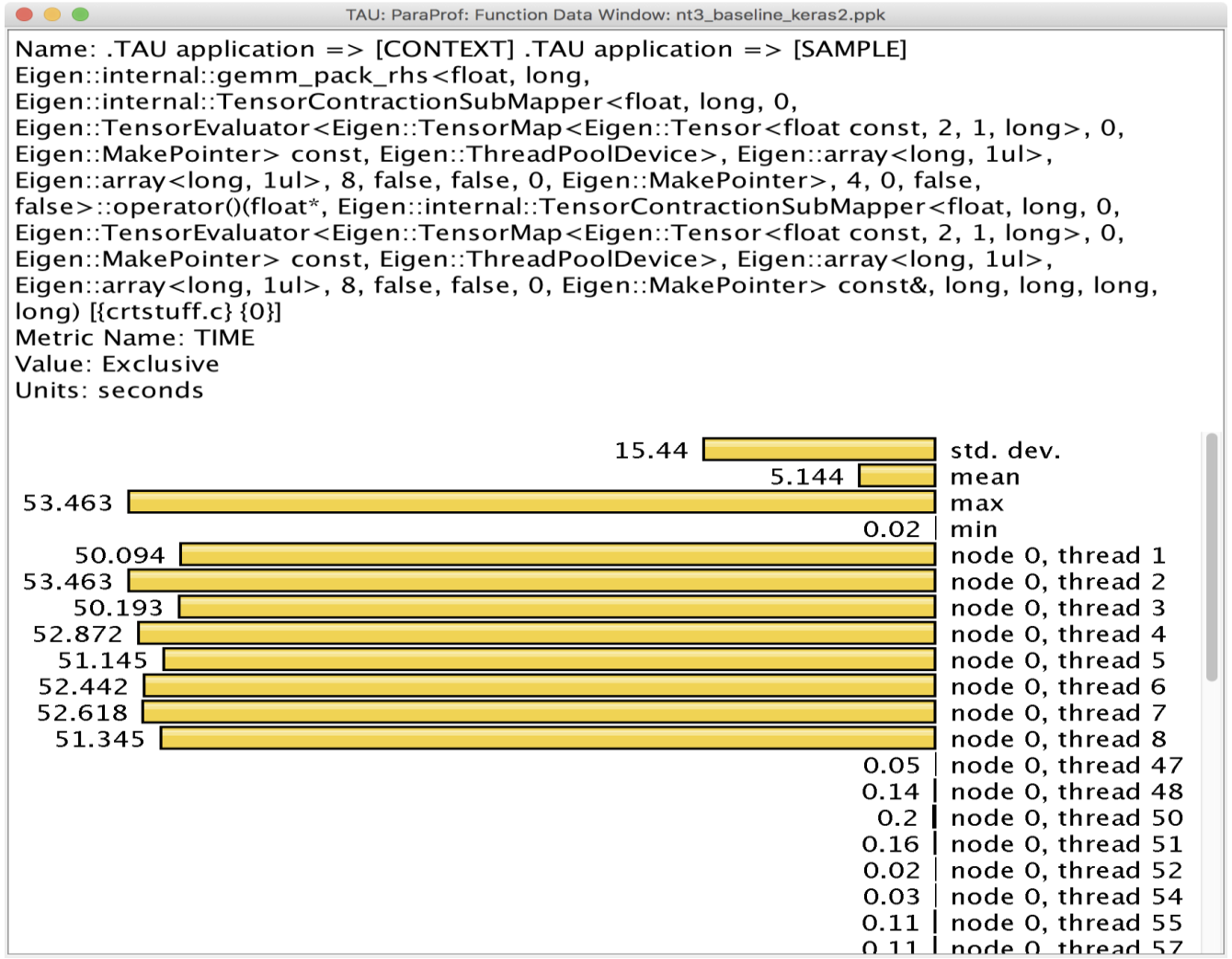
Deep Learning: Tensorflow



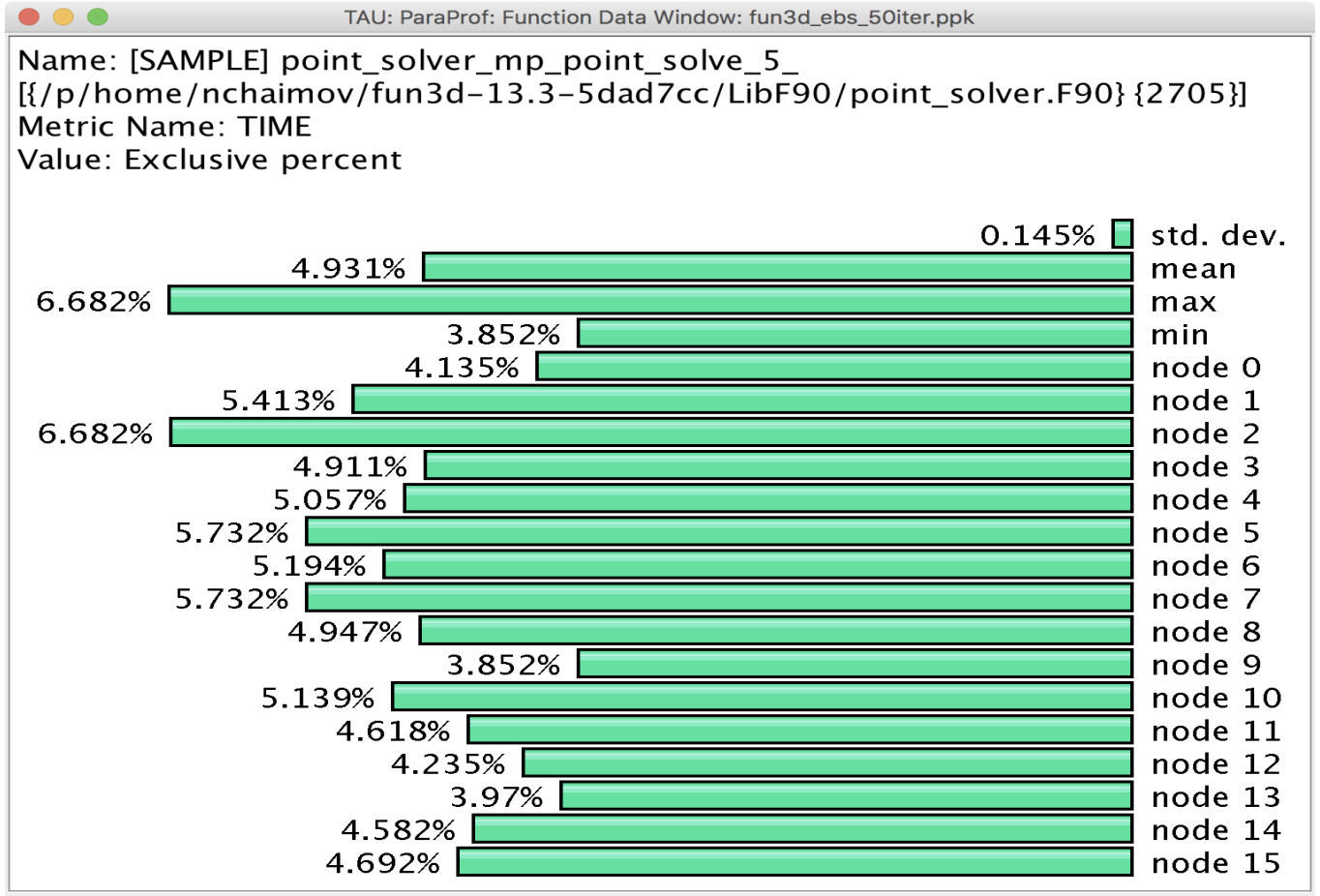
Name	Inclusiv...	Calls
.TAU application	519.211	1
[CONTEXT] .TAU application	509.222	50,915
[SAMPLE] Eigen::internal::gebp_kernel<float, float, long, Eigen::internal::blas_data_mapper<float, long, 0, 0>,	240.632	24,089
[SAMPLE] __pthread_cond_wait [{} {0}]	86.384	8,634
[SAMPLE] Eigen::internal::gemm_pack_rhs<float, long, Eigen::internal::TensorContractionSubMapper<float, lor	51.345	5,135
[SAMPLE] Eigen::internal::gemm_pack_rhs<float, long, Eigen::internal::TensorContractionSubMapper<float, lor	24.375	2,416
[SAMPLE] void tensorflow::SpatialMaxPoolWithArgMaxHelper<Eigen::ThreadPoolDevice, float>(tensorflow::OpK	16.301	1,630
[SAMPLE] __memset_sse2 [{} {0}]	13.446	1,336
[SAMPLE] Eigen::TensorEvaluator<Eigen::TensorContractionOp<Eigen::array<Eigen::IndexPair<long>, 1ul> co	5.99	599
[SAMPLE] long Eigen::internal::operator/<long, false>(long const&, Eigen::internal::TensorIntDivisor<long, fals	5.843	585
[SAMPLE] std::_Function_handler<void (long, long), Eigen::internal::TensorExecutor<Eigen::TensorAssignOp<I	5.377	538
[SAMPLE] float __vector Eigen::TensorEvaluator<Eigen::TensorBroadcastingOp<Eigen::IndexList<int, Eigen::typ	4.862	487
[SAMPLE] Eigen::TensorEvaluator<Eigen::TensorContractionOp<Eigen::array<Eigen::IndexPair<long>, 1ul> co	4.775	478
[SAMPLE] Eigen::TensorEvaluator<Eigen::TensorAssignOp<Eigen::TensorMap<Eigen::Tensor<float, 1, 1, long>	4.037	404
[SAMPLE] Eigen::internal::gemm_pack_lhs<float, long, Eigen::internal::TensorContractionSubMapper<float, lon	3.679	367
[SAMPLE] Eigen::internal::EvalRange<Eigen::TensorEvaluator<Eigen::TensorAssignOp<Eigen::TensorMap<Eiger	2.981	298
[SAMPLE] tensorflow::MaxPoolingOp<Eigen::ThreadPoolDevice, float>::SpatialMaxPool(tensorflow::OpKernelCo	2.915	295
[SAMPLE] std::_Function_handler<void (long, long), Eigen::internal::TensorExecutor<Eigen::TensorAssignOp<I	2.91	291
[SAMPLE] std::_Function_handler<void (long, long), Eigen::internal::TensorExecutor<Eigen::TensorAssignOp<I	2.772	277
[SAMPLE] Eigen::internal::gemm_pack_lhs<float, long, Eigen::internal::TensorContractionSubMapper<float, lon	2.481	248
[SAMPLE] std::_Function_handler<void (long, long), Eigen::internal::TensorExecutor<Eigen::TensorAssignOp<I	2.148	215
[SAMPLE] void Eigen::internal::call_dense_assignment_loop<Eigen::Map<Eigen::Matrix<float, -1, -1, 0, -1, -1>	2.008	197
[SAMPLE] Eigen::NonBlockingThreadPoolTempl<tensorflow::thread::EigenEnvironment>::WorkerLoop(int) [{/ho	1.999	200
[SAMPLE] Eigen::internal::ptranspose(Eigen::internal::PacketBlock<float __vector, 4>&) [{crtstuff.c} {0}]	1.919	192
[SAMPLE] Eigen::internal::gemm_pack_rhs<float, long, Eigen::internal::TensorContractionSubMapper<float, lor	1.607	160
[SAMPLE] Eigen::TensorEvaluator<Eigen::TensorContractionOp<Eigen::array<Eigen::IndexPair<long>, 1ul> co	1.518	152

```
% tau_python -ebs nt3_baseline_keras2.py (CANDLE)
```

Sampling Tensorflow



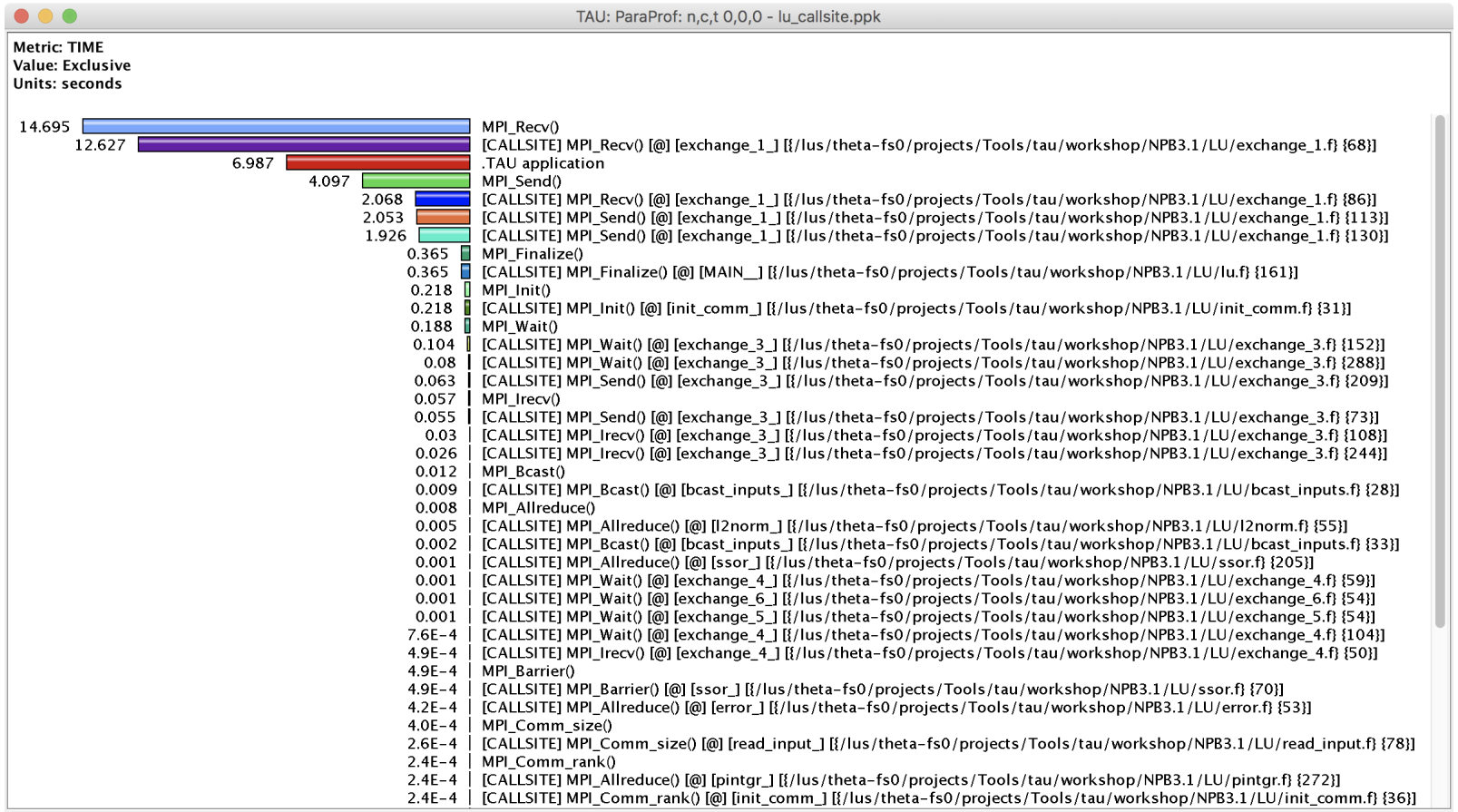
Event Based Sampling (EBS)



Uninstrumented!

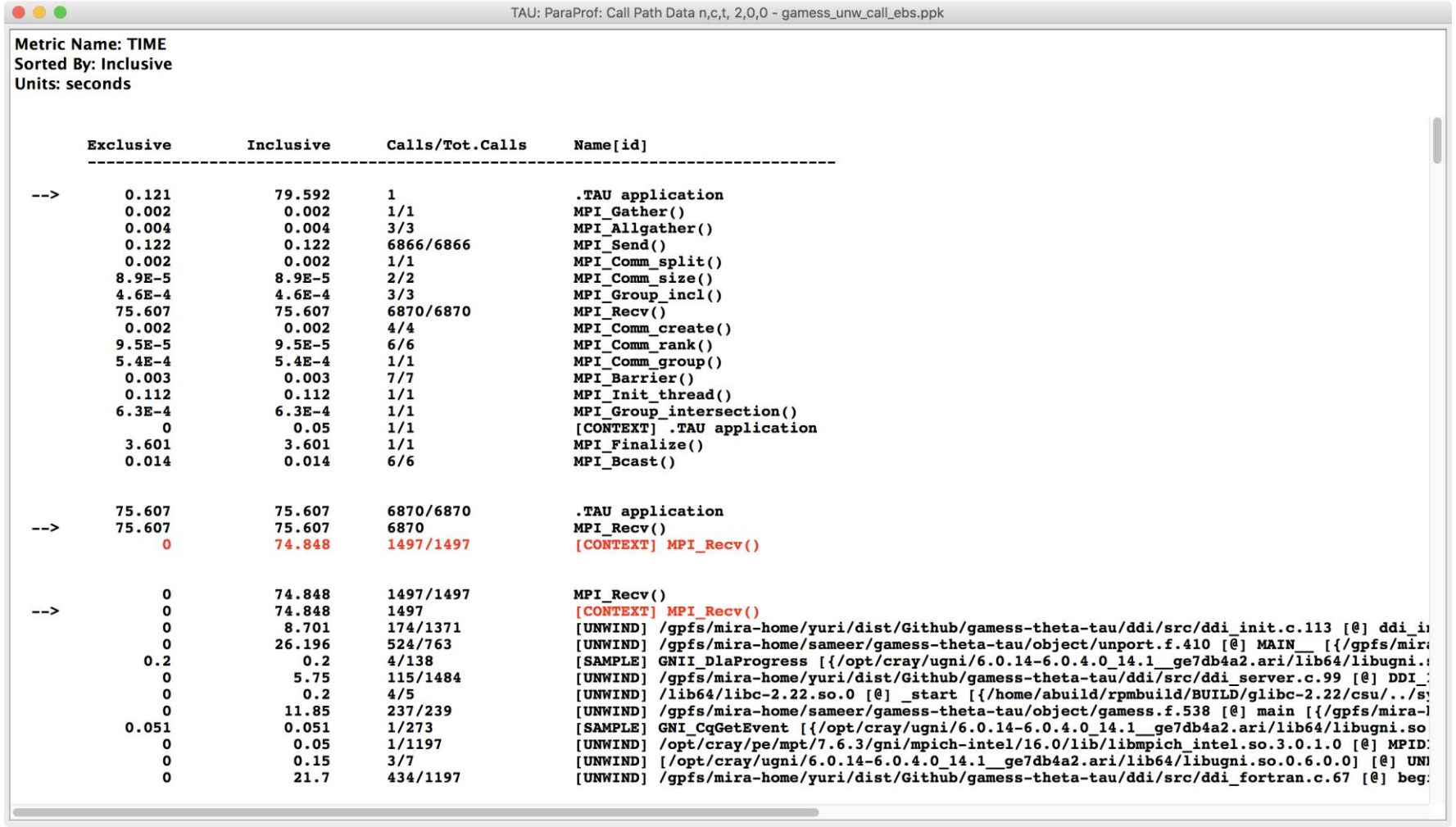
```
% mpirun -np 16 tau_exec -ebs a.out
```

Callsite Profiling and Tracing



% export TAU_CALLSITE=1

CALLPATH THREAD RELATIONS WINDOW



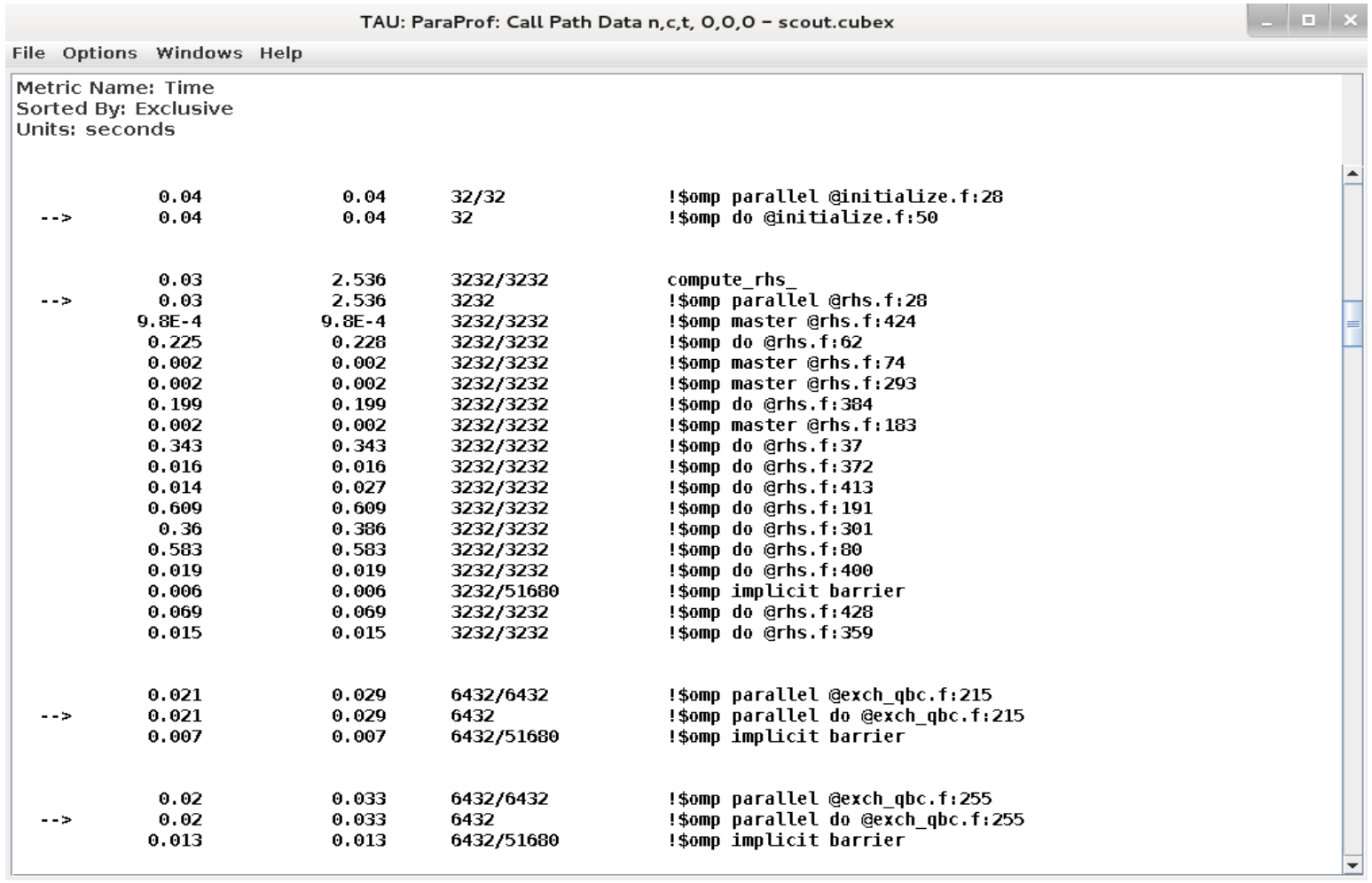
CALLPATH THREAD RELATIONS WINDOW

TAU: ParaProf: Call Path Data n,c,t, 2,0,0 - gamess_unw_call_ebs.ppk

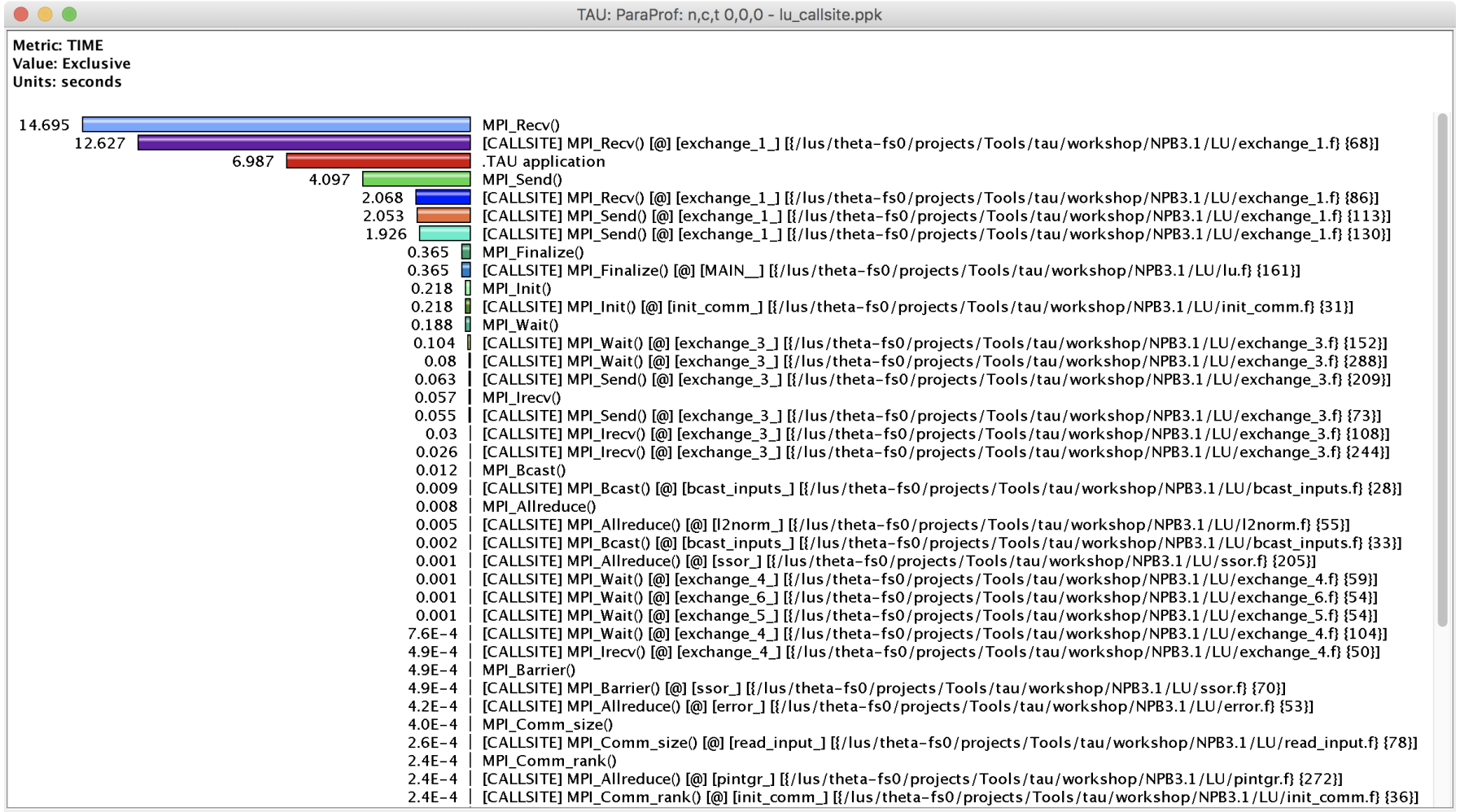
Metric Name: TIME
Sorted By: Exclusive
Units: seconds

	Exclusive	Inclusive	Calls/Tot.Calls	Name[id]
	75.607	75.607	6870/6870	.TAU application
-->	75.607	75.607	6870	MPI_Recv()
	0	74.848	1497/1497	[CONTEXT] MPI_Recv()
	0.15	0.15	3/444	[UNWIND] /opt/cray/pe/mpt/7.6.3/gni/mpich-intel/16.0/lib/libmpich_intel.so.3.0.1.0 [@] PMPI_Recv
	22.046	22.046	441/444	[UNWIND] /opt/cray/pe/mpt/7.6.3/gni/mpich-intel/16.0/lib/libmpich_intel.so.3.0.1.0 [@] MPIDI_CH3
-->	22.196	22.196	444	[SAMPLE] MPID_nem_gni_poll [{/opt/cray/pe/mpt/7.6.3/gni/mpich-intel/16.0/lib/libmpich_intel.so.3
	5.6	5.6	112/273	[UNWIND] /opt/cray/ugni/6.0.14-6.0.4.0_14.1__ge7db4a2.ari/lib64/libugni.so.0.6.0.0 [@] MPID_nem_
	0.051	0.051	1/273	[CONTEXT] MPI_Recv()
	7.651	7.651	153/273	[UNWIND] /opt/cray/ugni/6.0.14-6.0.4.0_14.1__ge7db4a2.ari/lib64/libugni.so.0.6.0.0 [@] MPID_nem_
	0.35	0.35	7/273	[UNWIND] [/opt/cray/ugni/6.0.14-6.0.4.0_14.1__ge7db4a2.ari/lib64/libugni.so.0.6.0.0] [@] UNRESOL
-->	13.652	13.652	273	[SAMPLE] GNI_CqGetEvent [{/opt/cray/ugni/6.0.14-6.0.4.0_14.1__ge7db4a2.ari/lib64/libugni.so.0.6.
	11.3	11.3	226/226	[UNWIND] /opt/cray/pe/mpt/7.6.3/gni/mpich-intel/16.0/lib/libmpich_intel.so.3.0.1.0 [@] PMPI_Recv
-->	11.3	11.3	226	[SAMPLE] MPIDI_CH3I_Progress [{/opt/cray/pe/mpt/7.6.3/gni/mpich-intel/16.0/lib/libmpich_intel.so
	10.349	10.349	207/207	[UNWIND] /opt/cray/ugni/6.0.14-6.0.4.0_14.1__ge7db4a2.ari/lib64/libugni.so.0.6.0.0 [@] MPID_nem_
-->	10.349	10.349	207	[SAMPLE] GNI_SmsgGetNextWTag [{/opt/cray/ugni/6.0.14-6.0.4.0_14.1__ge7db4a2.ari/lib64/libugni.so
	0.2	0.2	4/138	[CONTEXT] MPI_Recv()
	6.701	6.701	134/138	[UNWIND] /opt/cray/ugni/6.0.14-6.0.4.0_14.1__ge7db4a2.ari/lib64/libugni.so.0.6.0.0 [@] GNI_CqGet
-->	6.901	6.901	138	[SAMPLE] GNII_DlaProgress [{/opt/cray/ugni/6.0.14-6.0.4.0_14.1__ge7db4a2.ari/lib64/libugni.so.0.
	5.25	5.25	105/109	[UNWIND] gni_poll.c.0 [@] MPID_nem_gni_poll [{/opt/cray/pe/mpt/7.6.3/gni/mpich-intel/16.0/lib/li
	0.2	0.2	4/109	[UNWIND] gni_poll.c.0 [@] MPIDI_CH3I_Progress [{/opt/cray/pe/mpt/7.6.3/gni/mpich-intel/16.0/lib/
-->	5.45	5.45	109	[SAMPLE] MPID_nem_gni_check_localCQ [{gni_poll.c} {0}]
	3.601	3.601	1/1	.TAU application
-->	3.601	3.601	1	MPI_Finalize()

ParaProf: Callpath Thread Relations Window



Callsite Profiling and Tracing (TAU_CALLSITE=1)



Identifying MPI Collective Sync Wait in Thread Callpath Relations

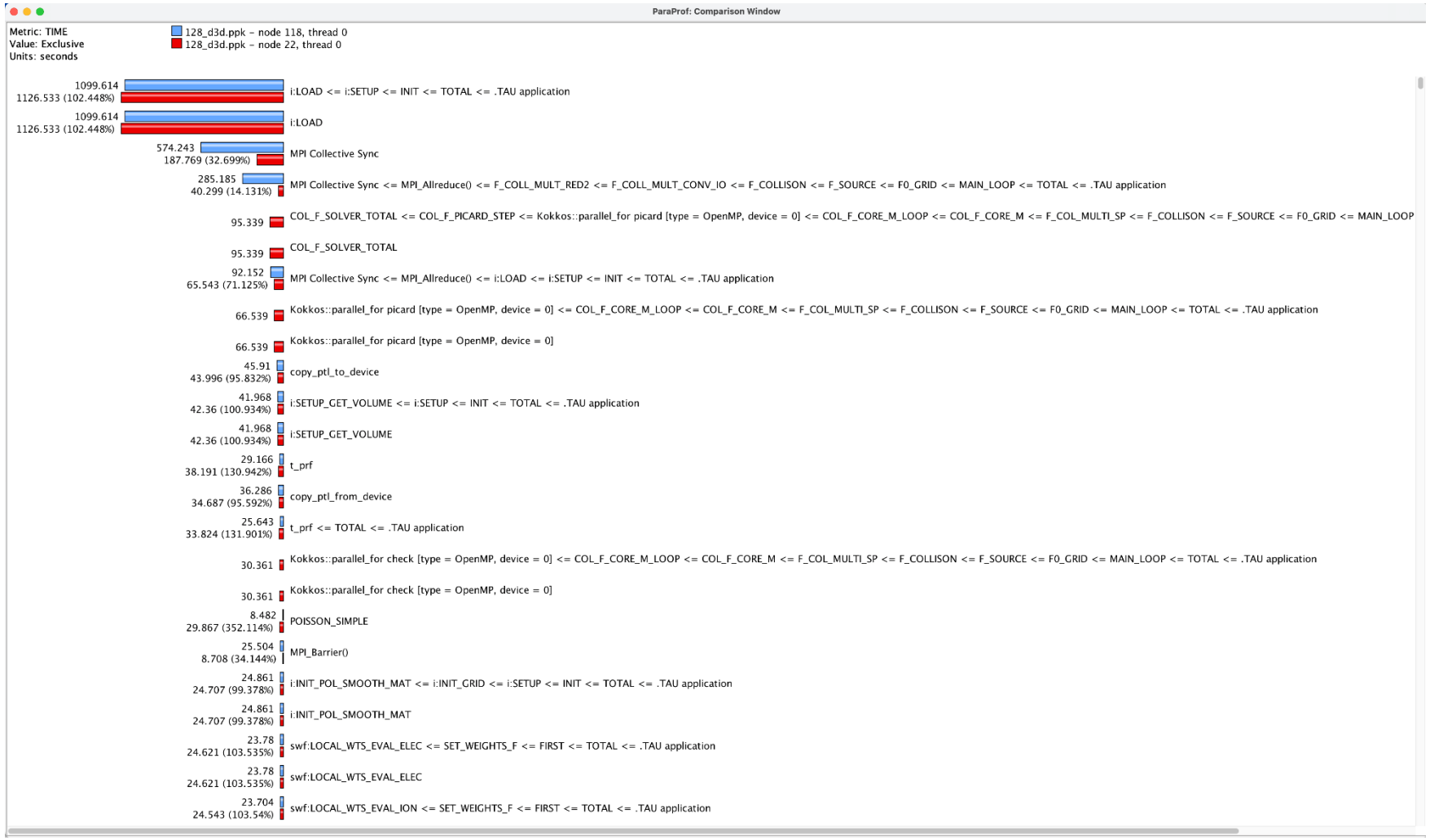
TAU: ParaProf: Call Path Data n,c,t, 118,0,0 - 128_d3d.ppk

Metric Name: TIME
Sorted By: Exclusive
Units: seconds

	Exclusive	Inclusive	Calls/Tot.Calls	Name[id]
	1099.614	1191.772	1/1	i:SETUP
-->	1099.614	1191.772	1	i:LOAD
	0.006	92.158	3/9543	MPI_Allreduce()
	9.8E-4	9.8E-4	11/15177	MPI_Gatherv()
	1.448	1.448	43/15177	MPI_Gather()
	15.353	15.353	46/15177	MPI_Alltoall()
	89.821	89.821	4311/15177	MPI_Bcast()
	6.777	6.777	195/15177	MPI_Allgather()
	68.678	68.678	991/15177	MPI_Reduce()
	9.179	9.179	12/15177	MPI_Comm_dup()
	0.125	0.125	25/15177	MPI_Allgatherv()
	382.861	382.861	9543/15177	MPI_Allreduce()
-->	574.243	574.243	15177	MPI Collective Sync
	2.507	2.508	10/186	DISTRIBUTE_F0G
	2.433	2.434	10/186	F_UPD_F0_SP
	5.156	5.158	20/186	F0_CHARGE_SEARCH_INDEX
	5.505	5.507	22/186	PULLBACK_WEIGHT
	24.86	24.872	102/186	UPDATE_PTL_WEIGHT
	0.473	0.473	2/186	MAIN_LOOP
	4.975	4.977	20/186	DIAG_f0_PORT1_PTL
-->	45.91	45.93	186	copy_ptl_to_device
	0.02	0.02	186/272	Kokkos::parallel_for set_buffer_particles_d [type = Cuda, device = 0]

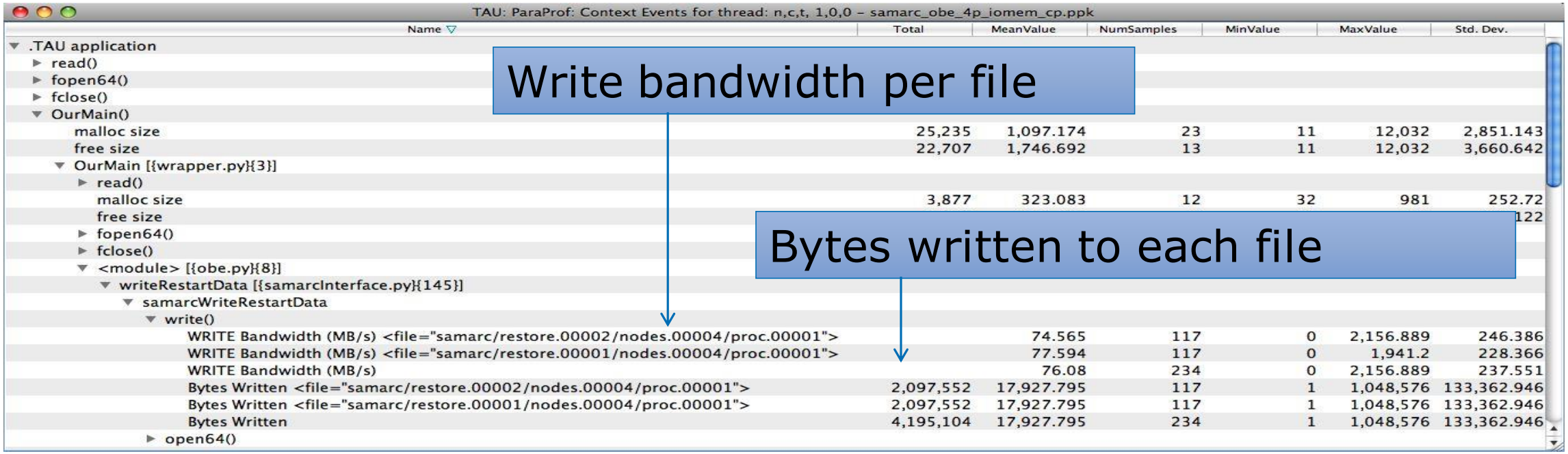
MPI Collective Sync is the time spent in a barrier operation inside a collective

Thread Comparison Window



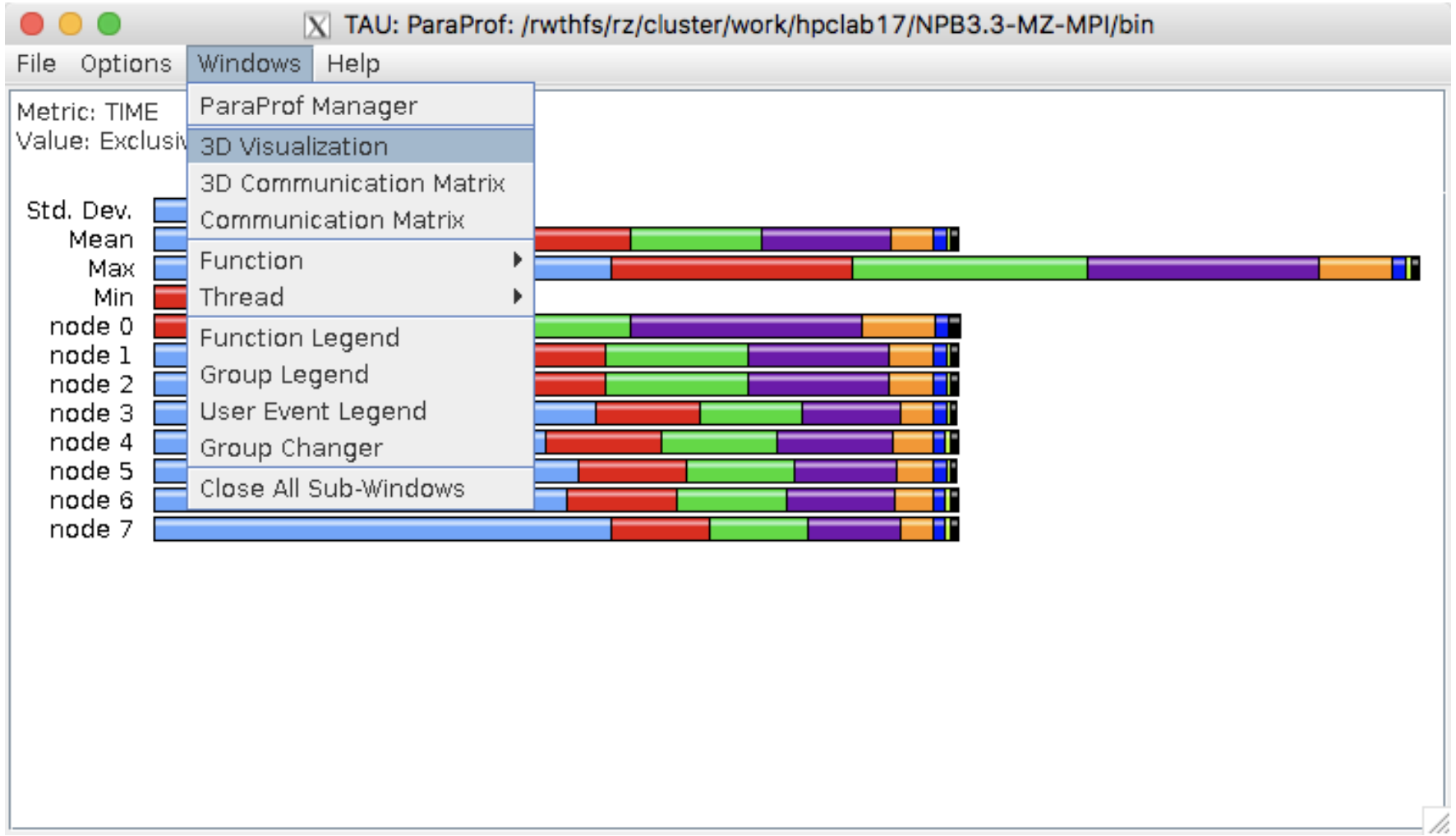
Comparing Rank 118 with 22.
Right click on "node 118" ->
Add node to comparison window

TAU – Context Events

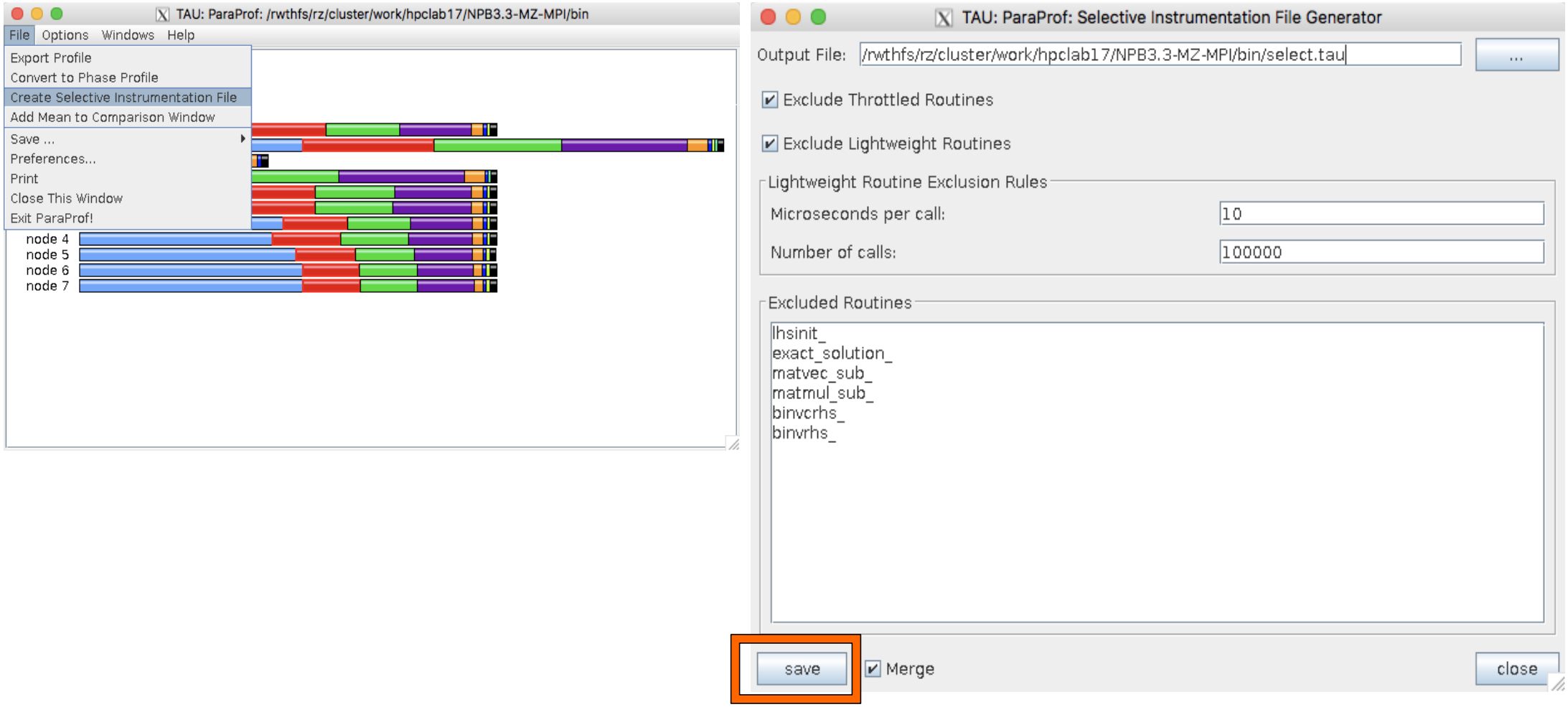


Name	Total	MeanValue	NumSamples	MinValue	MaxValue	Std. Dev.
.TAU application						
read()						
fopen64()						
fclose()						
OurMain()						
malloc size	25,235	1,097.174	23	11	12,032	2,851.143
free size	22,707	1,746.692	13	11	12,032	3,660.642
OurMain [{wrapper.py}{3}]						
read()						
malloc size	3,877	323.083	12	32	981	252.72
free size						122
fopen64()						
fclose()						
<module> [{obe.py}{8}]						
writeRestartData [{samarcInterface.py}{145}]						
samarcWriteRestartData						
write()						
WRITE Bandwidth (MB/s) <file="samarc/restore.00002/nodes.00004/proc.00001">		74.565	117	0	2,156.889	246.386
WRITE Bandwidth (MB/s) <file="samarc/restore.00001/nodes.00004/proc.00001">		77.594	117	0	1,941.2	228.366
WRITE Bandwidth (MB/s)		76.08	234	0	2,156.889	237.551
Bytes Written <file="samarc/restore.00002/nodes.00004/proc.00001">	2,097,552	17,927.795	117	1	1,048,576	133,362.946
Bytes Written <file="samarc/restore.00001/nodes.00004/proc.00001">	2,097,552	17,927.795	117	1	1,048,576	133,362.946
Bytes Written	4,195,104	17,927.795	234	1	1,048,576	133,362.946
open64()						

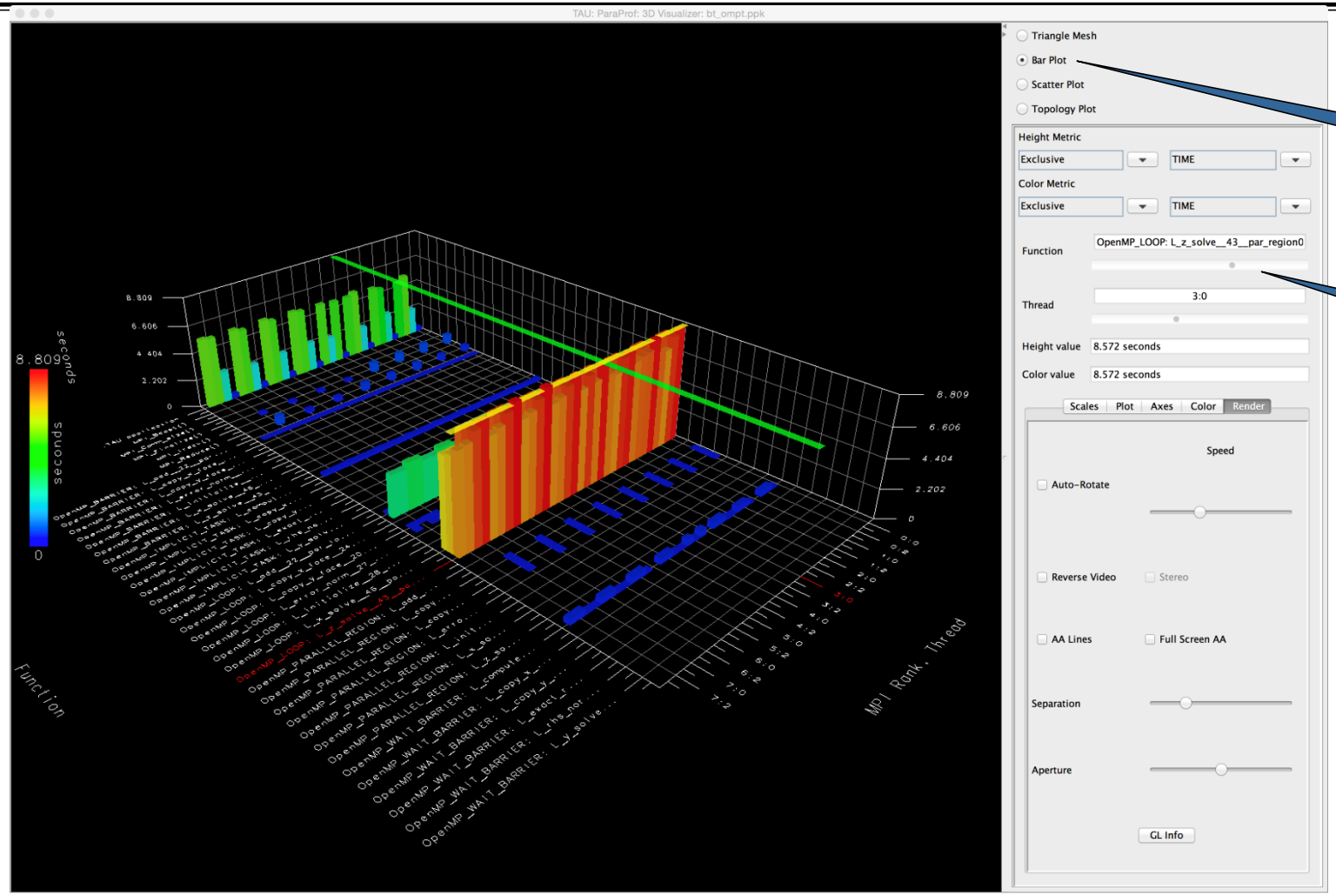
ParaProf with Optimized Instrumentation



Create a Selective Instrumentation File, Re-instrument, Re-run



Paraprof 3D visualization window

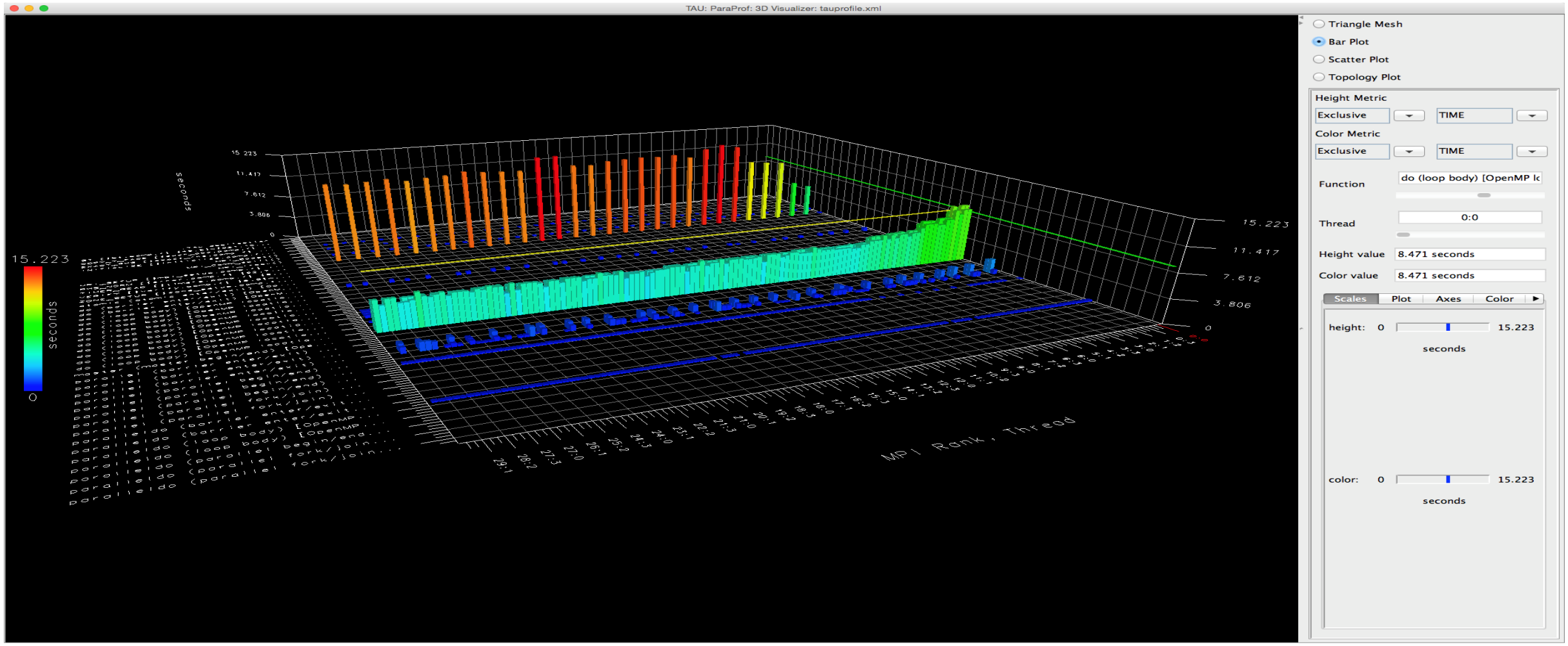


Click Bar Plot

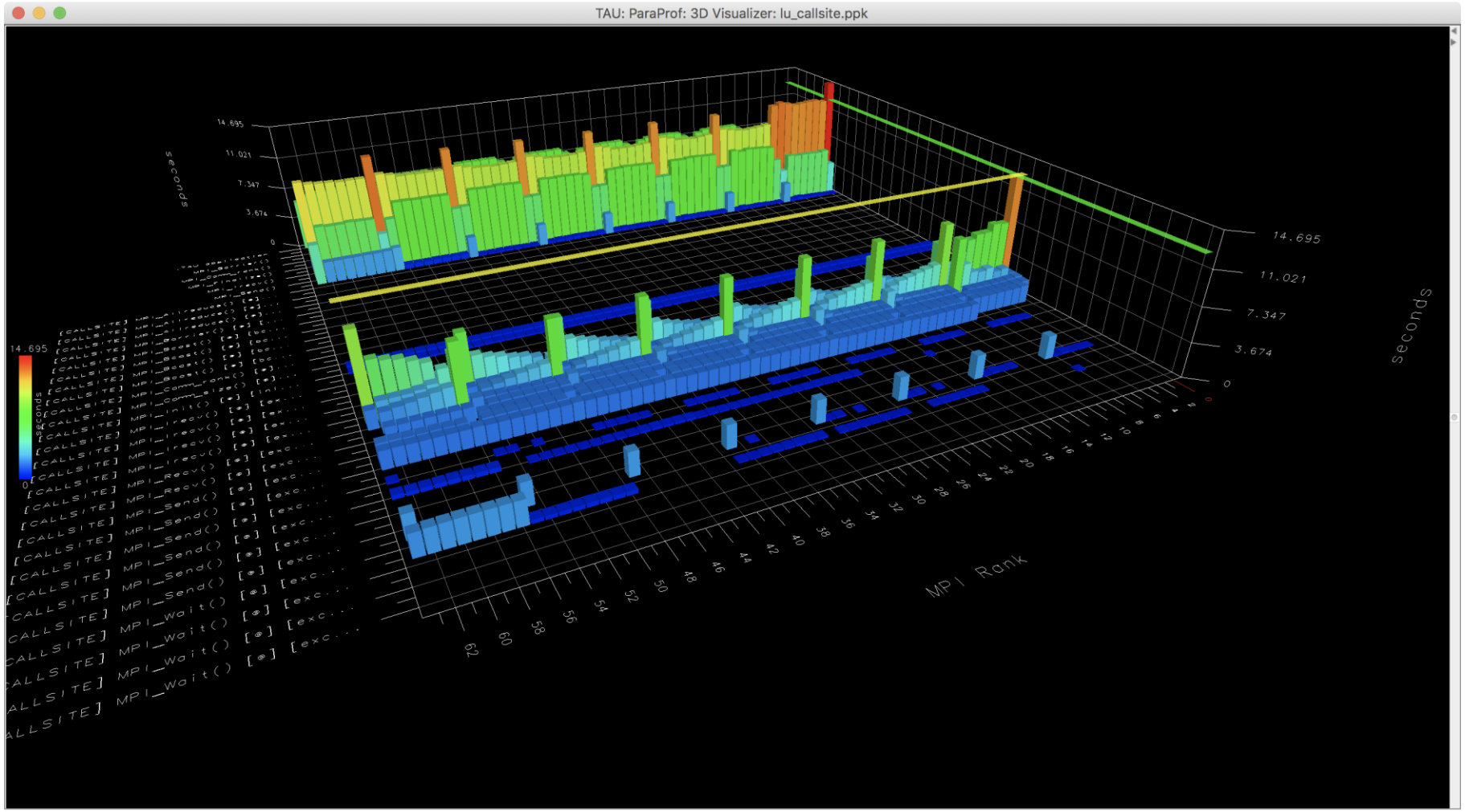
Move Function and Thread Sliders

Windows -> 3D visualization

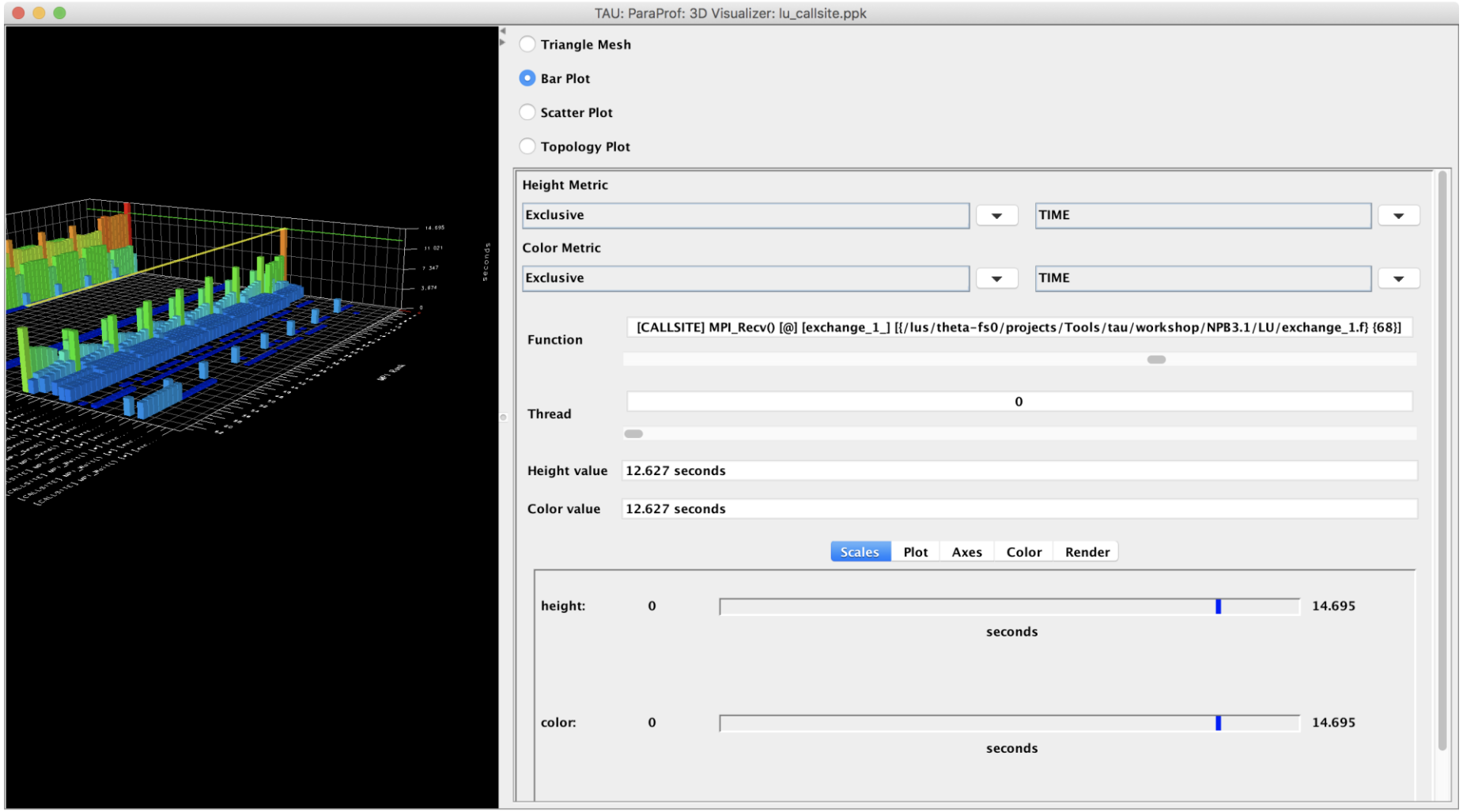
ParaProf: 3D Visualization Window Showing Entire Profile



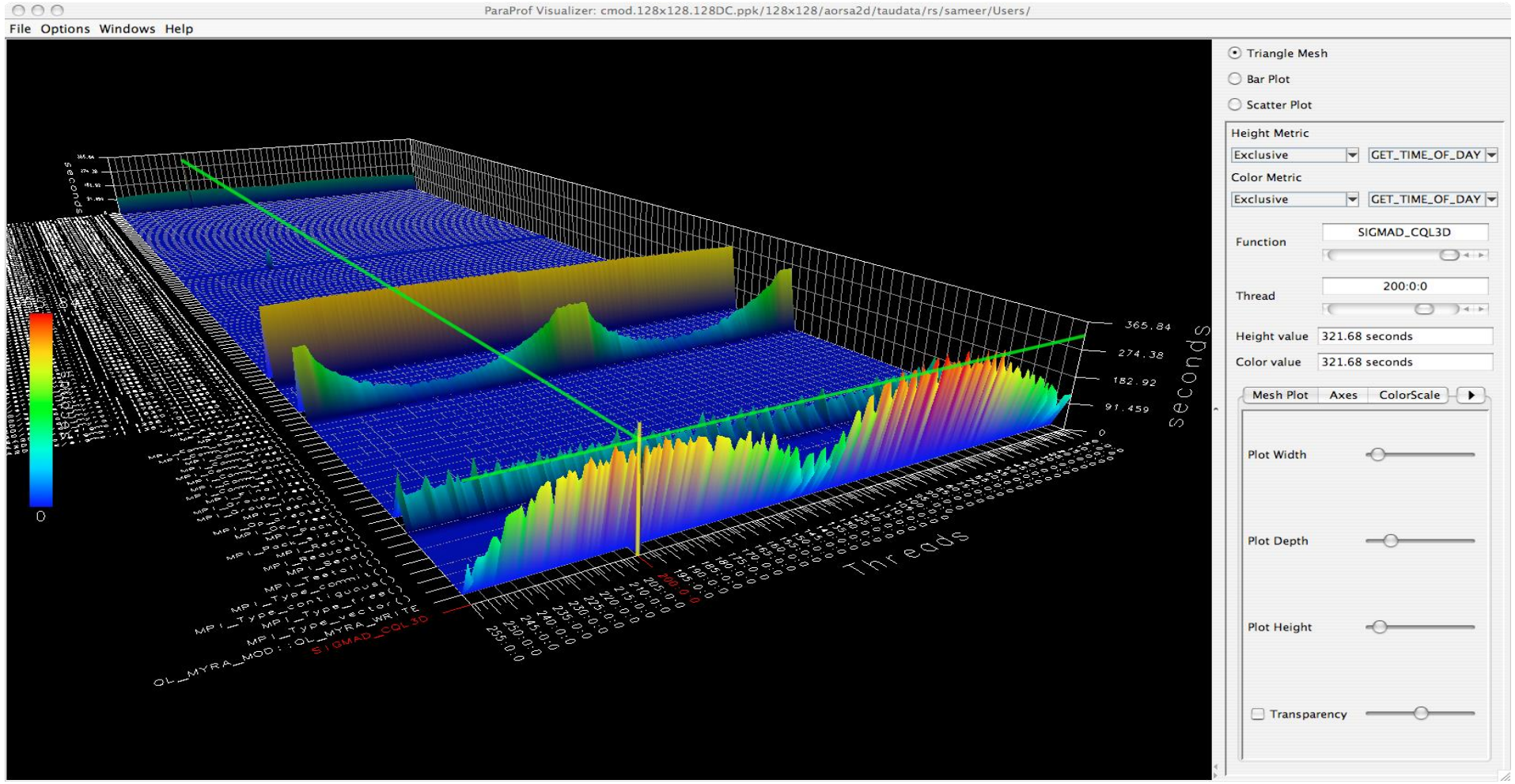
Callsite Profiling and Tracing



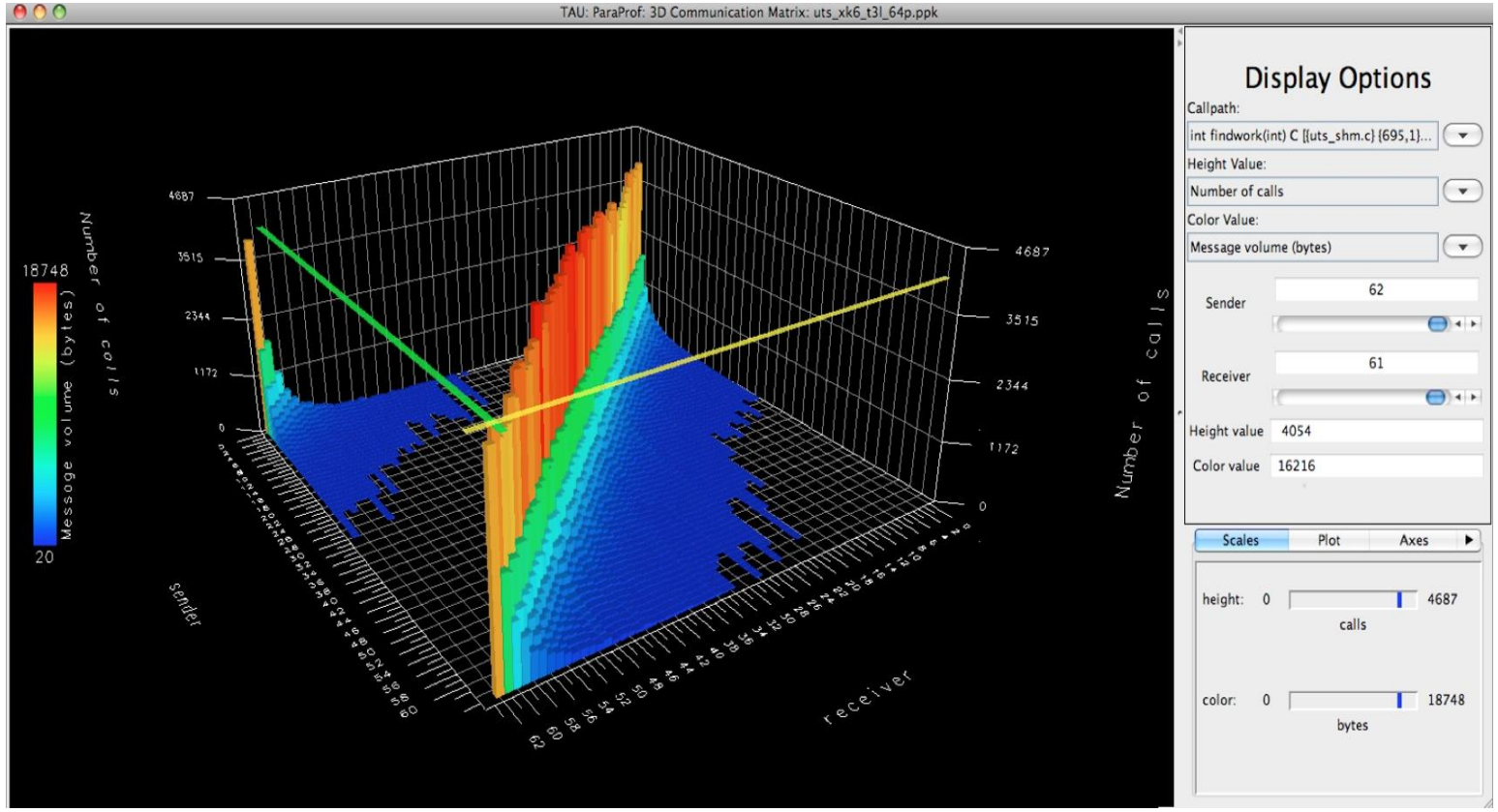
Callsite Profiling and Tracing



Parallel Profile Visualization: ParaProf

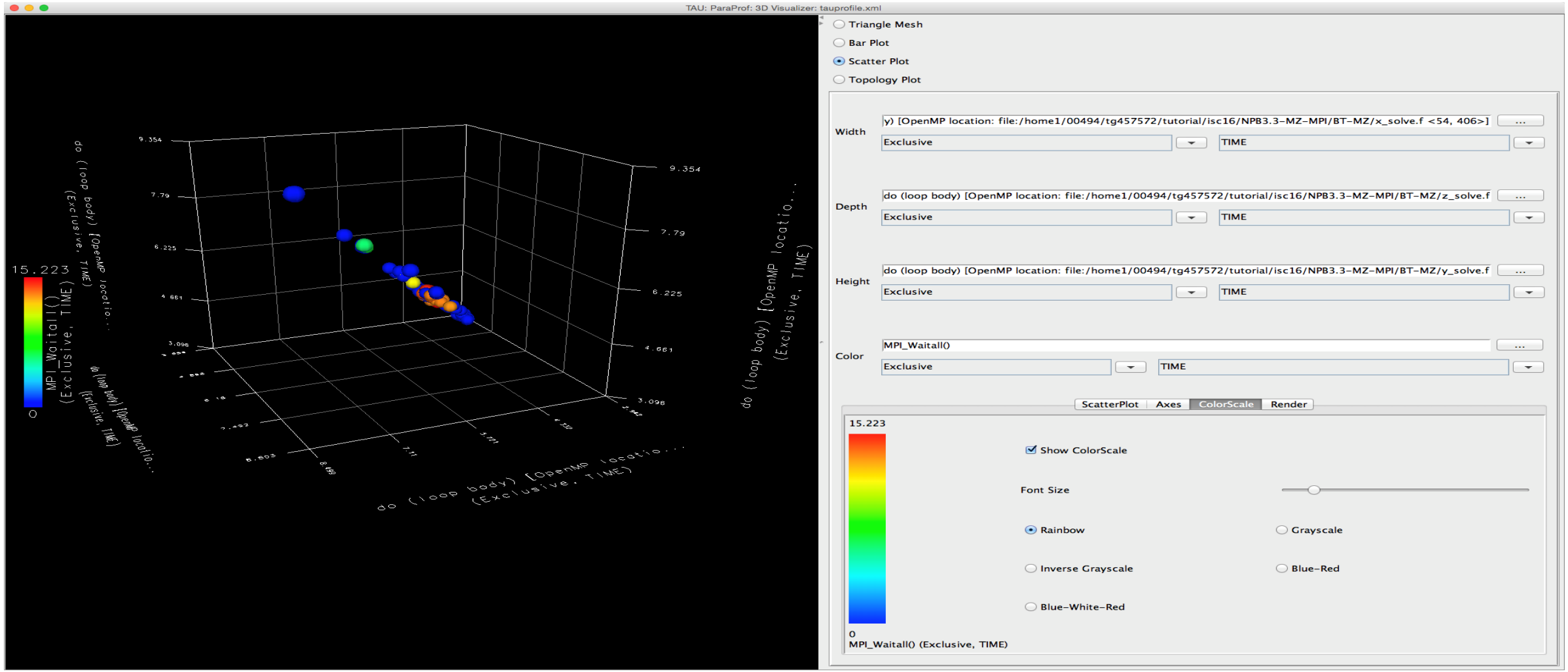


ParaProf 3D Communication Matrix

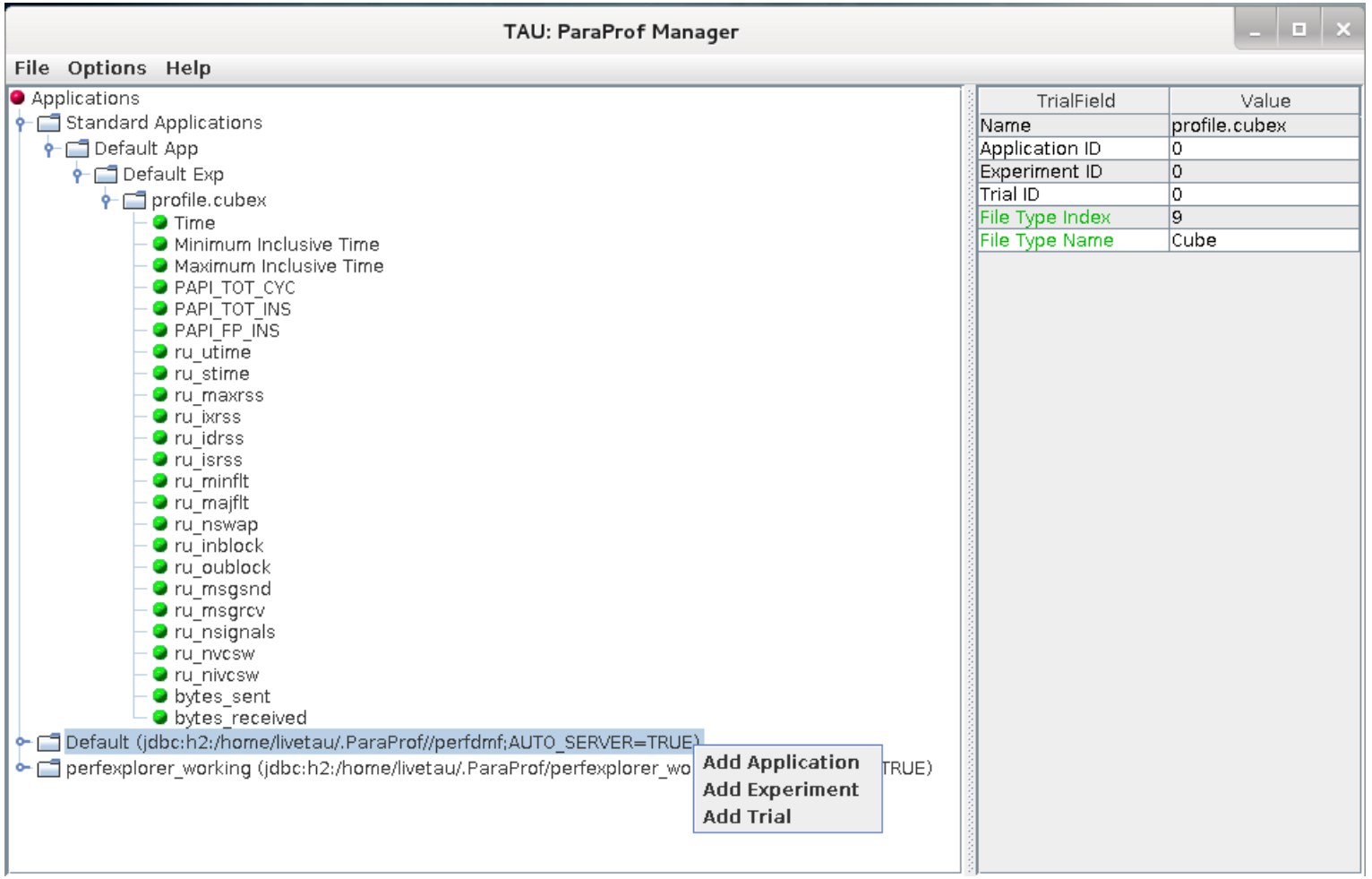


% export TAU_COMM_MATRIX=1

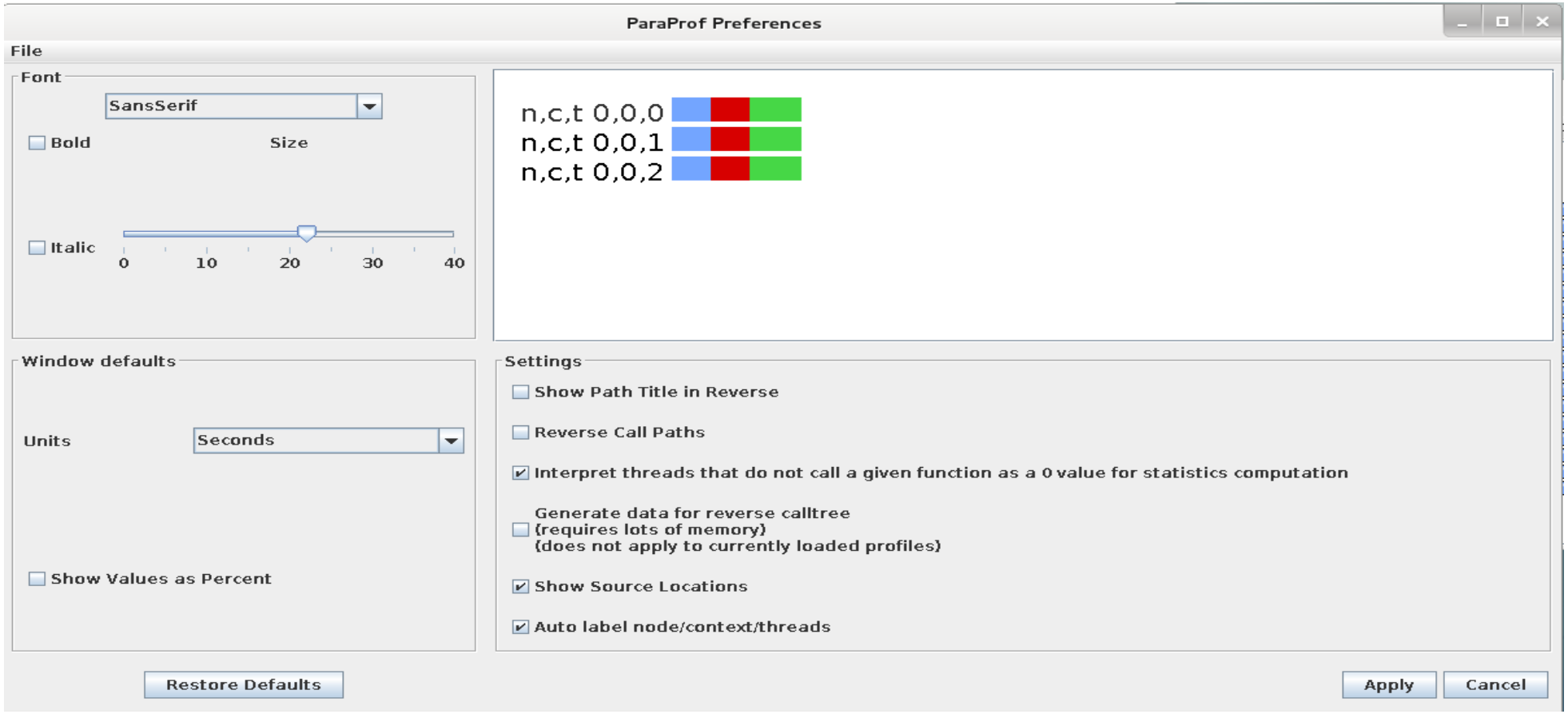
ParaProf: 3D Scatter Plot



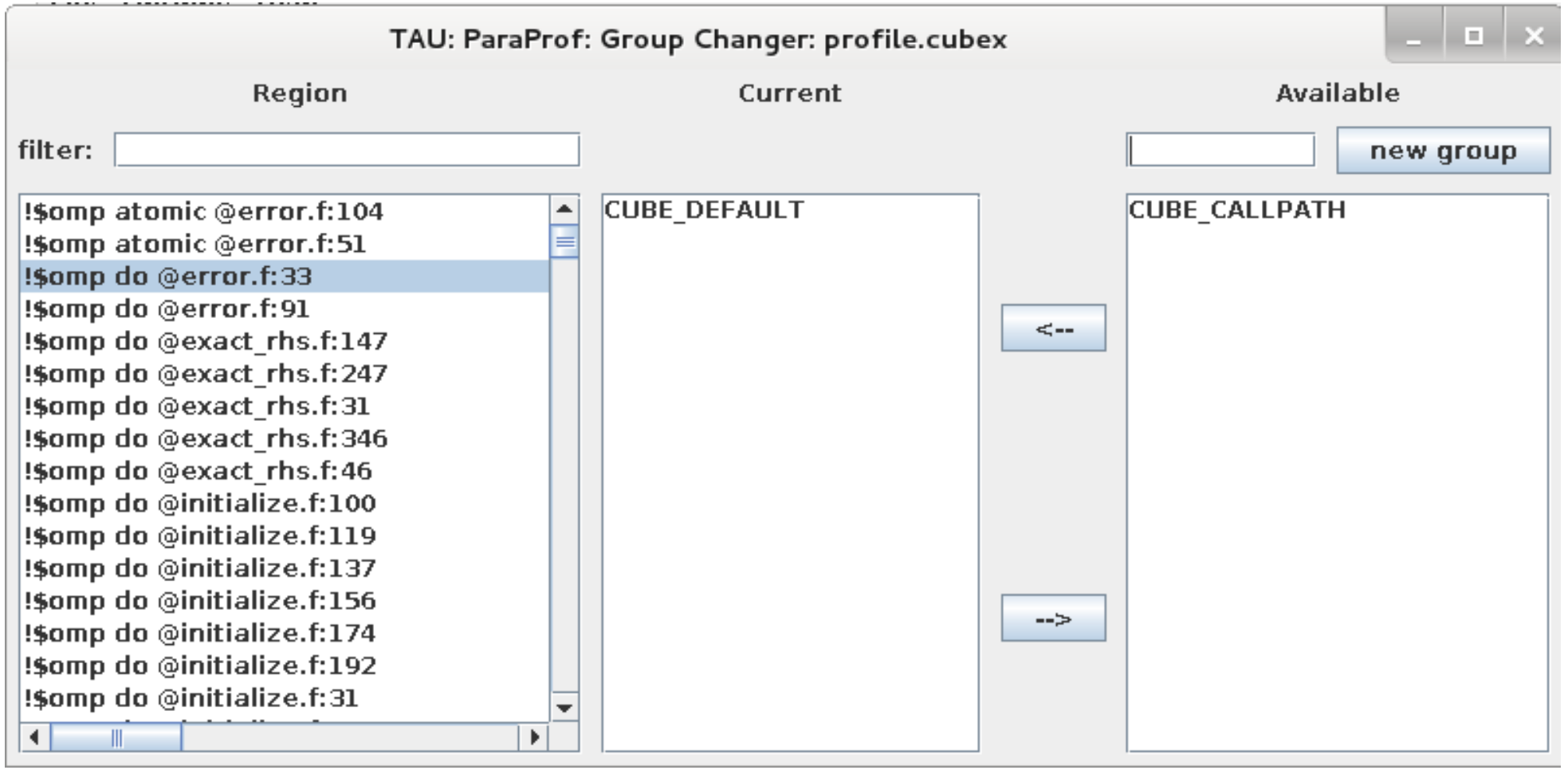
ParaProf: Score-P Profile Files, Database



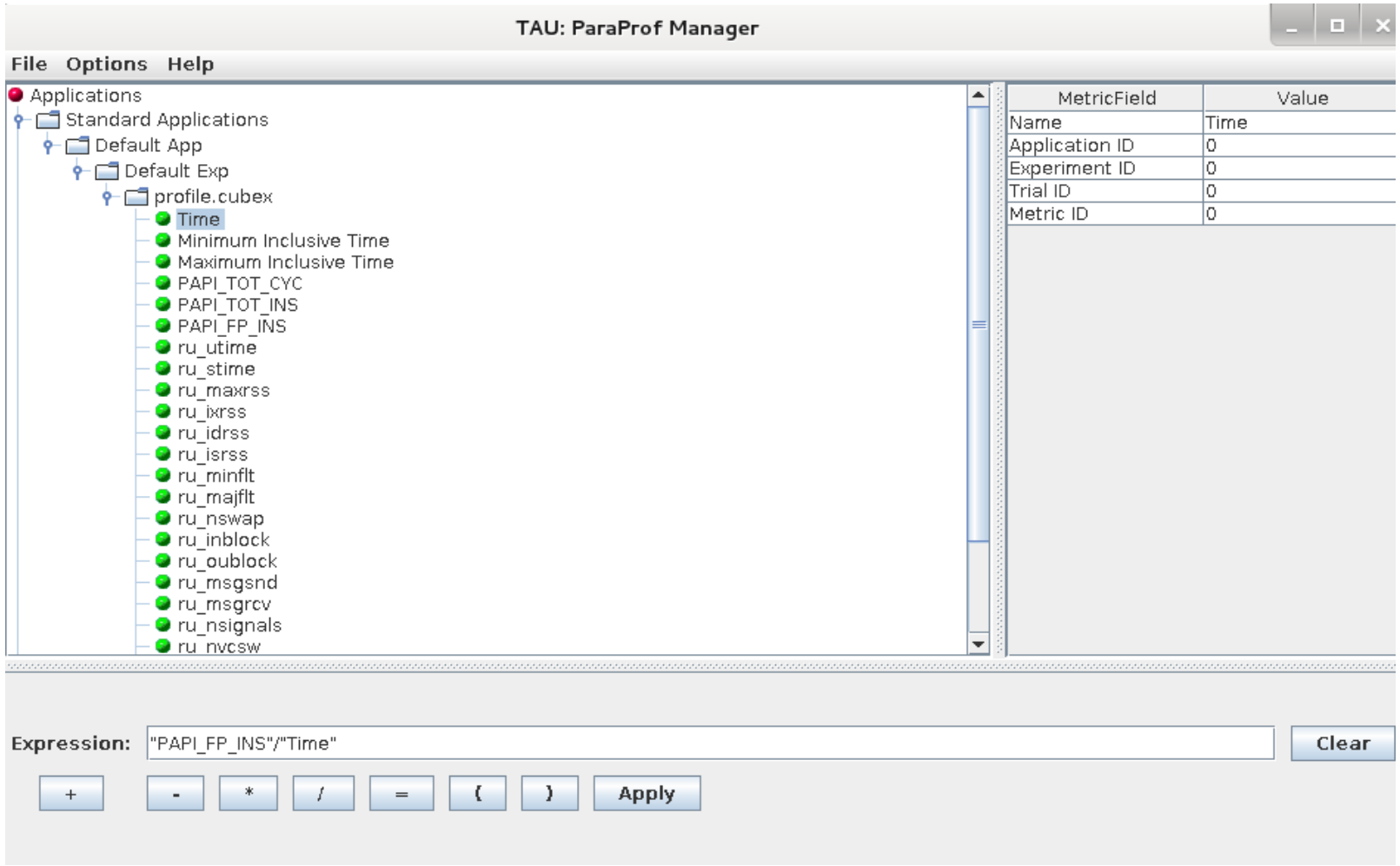
ParaProf: File Preferences Window



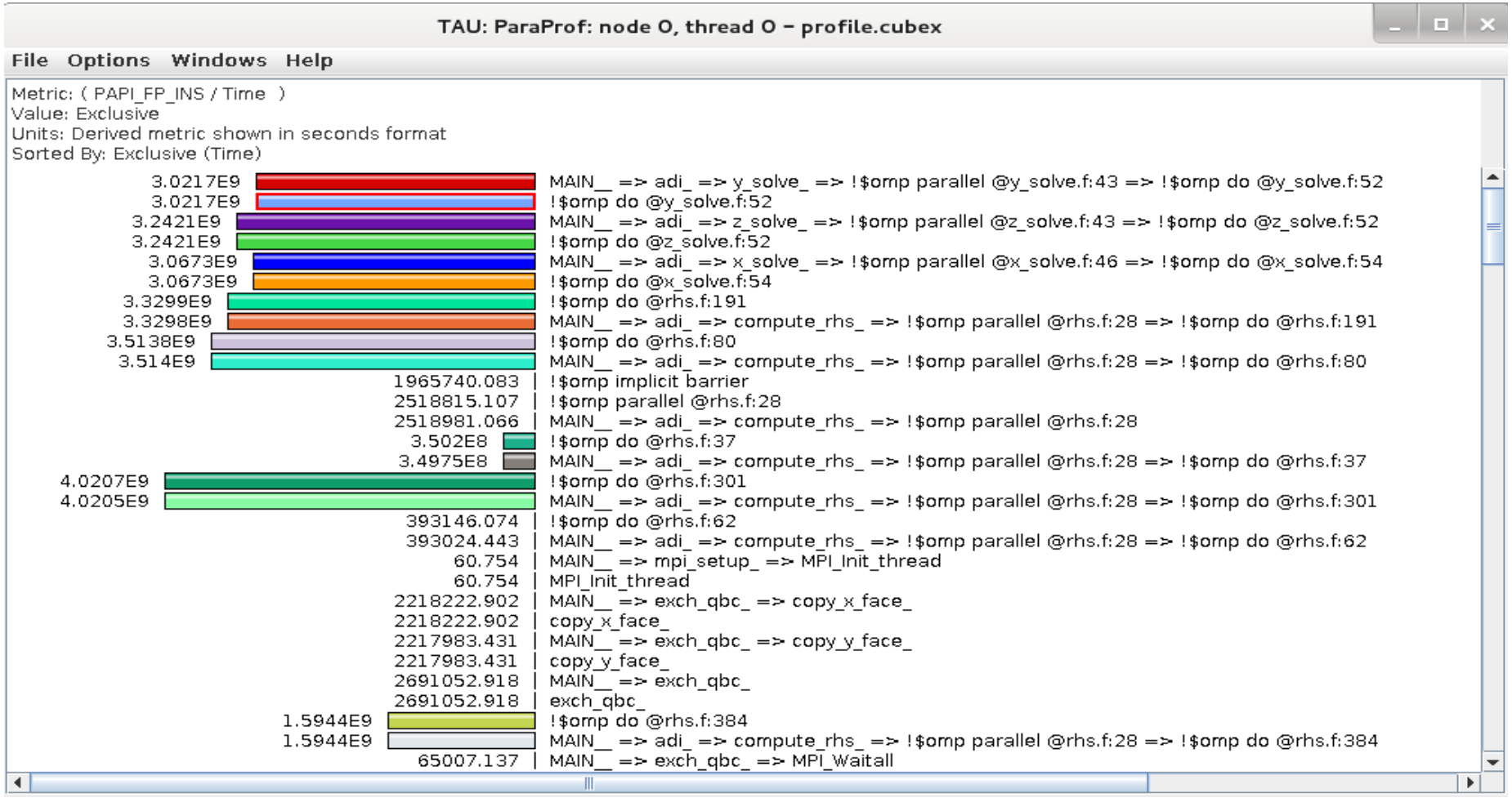
ParaProf: Group Changer Window



ParaProf: Derived Metric Panel in Manager Window



Sorting Derived FLOPS metric by Exclusive Time

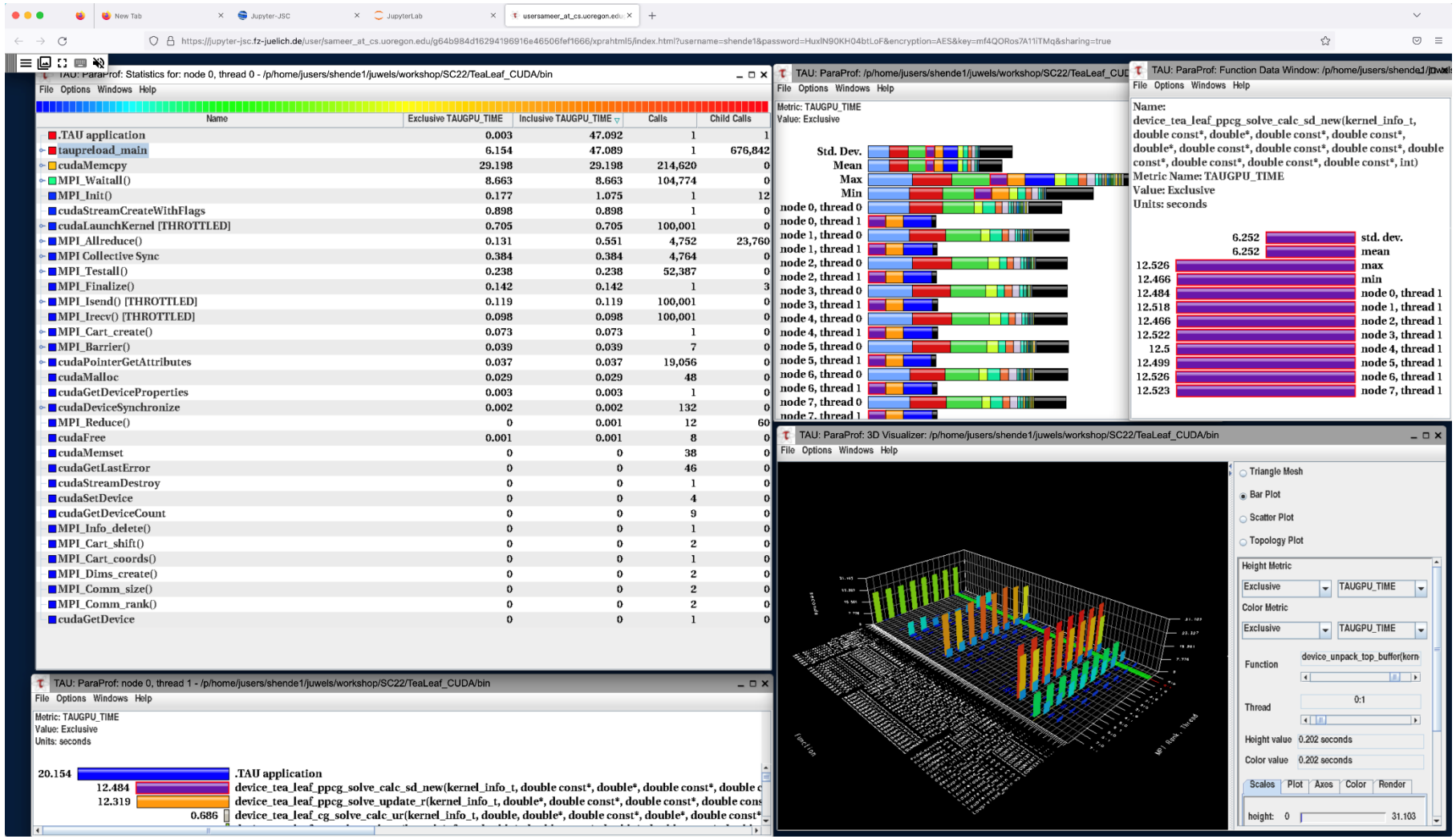


ParaProf Hands-On

Login to Xpra Desktop via https://jupyter-jsc.fz-juelich.de

```
$ source /p/project/training2341/setup.sh
$ tar xf $PROJECT/examples/tea_leaf.tar.gz
$ cd TeaLeaf_CUDA
$ make
$ cd bin
$ cp ../jobscripts/juwelsbooster/tau.sbatch .
$ cat tau.sbatch
….
srun tau_exec -T cupti,mpi -cupti -ebs ./tea_leaf

$ sbatch tau.sbatch
$ pprof –a | more
$ paraprof &
```

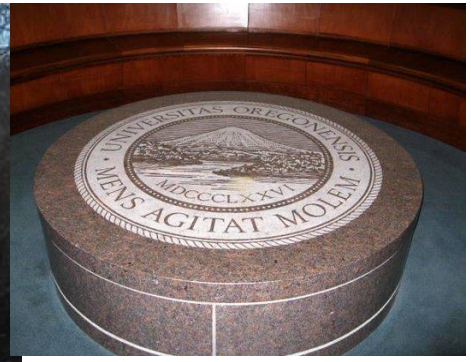
ParaProf: TeaLeaf_CUDA



Login to Xpra Desktop via https://jupyter-jsc.fz-juelich.de

```
$ source /p/project/training2341/setup.sh
$ wget http://tau.uoregon.edu/demo.ppk
$ paraprof demo.ppk &

Windows -> 3D Visualization -> Bar Chart -> Function and Thread slider
```

Performance Research Lab, University of Oregon, Eugene, USA



Support Acknowledgments

- US Department of Energy (DOE)
  - Office of Science contracts, ECP
  - SciDAC, LBL contracts
  - LLNL-LANL-SNL ASC/NNSA contract
  - Battelle, PNNL contract
  - ANL, ORNL contract
- Department of Defense (DoD)
  - PETTT, HPCMP
- National Science Foundation (NSF)
  - Glassbox, SI-2
- NASA
- CEA, France
- Partners:
  - University of Oregon
  - ParaTools, Inc., ParaTools, SAS
  - The Ohio State University
  - University of Tennessee, Knoxville
  - T.U. Dresden, GWT
  - Juelich Supercomputing Center















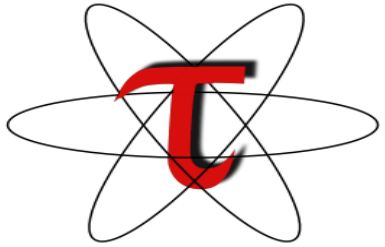




Acknowledgement

This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of two U.S. Department of Energy organizations (Office of Science and the National Nuclear Security Administration) responsible for the planning and preparation of a capable exascale ecosystem, including software, applications, hardware, advanced system engineering, and early testbed platforms, in support of the nation's exascale computing imperative.

Download TAU from U. Oregon



http://tau.uoregon.edu

http://www.hpclinux.com [LiveDVD, OVA]

https://e4s.io [Containers for Extreme-Scale Scientific Software Stack]

Free download, open source, BSD license

Hands-on resume: TeaLeaf MPI+CUDA

Recap: Setup for exercises

- Connect to your training account on JUWELS Booster (with X11-forwarding)

```
% ssh -X <yourid>@juwels-booster.fz-juelich.de
```

Use Jupyter-JSC/Xpra instead!

- Set account and default environment (NVHPC + ParaStationMPI) via helper script

```
% source /p/project/training2341/setup.sh
```

- Change to WORK directory containing TeaLeaf_CUDA sources & bin.scorep directory
  - Existing executable instrumented by Score-P can be reused

```
% cd $WORK
% cd TeaLeaf_CUDA
% cd bin.scorep
```

- Load Score-P module

```
% module load  Score-P CubeGUI
```

TeaLeaf_CUDA trace measurement collection for Vampir...

```
% edit scorep.sbatch
% cat scorep.sbatch

# Score-P measurement configuration
export SCOREP_CUDA_ENABLE=default
export SCOREP_CUDA_BUFFER=48M
export SCOREP_EXPERIMENT_DIRECTORY=scorep-tea_leaf-8_vampir
export SCOREP_FILTERING_FILE=../config/scorep.filter

export SCOREP_ENABLE_TRACING=true
export SCOREP_TOTAL_MEMORY=250M

# Run the application
srun ./tea_leaf
```

- Change to directory with the Score-P instrumented executable and edit the job script

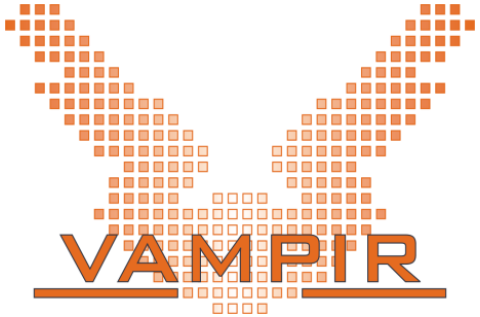
Uncomment lines to enable Score-P trace collection

```
% sbatch scorep.sbatch
```

- Submit the job

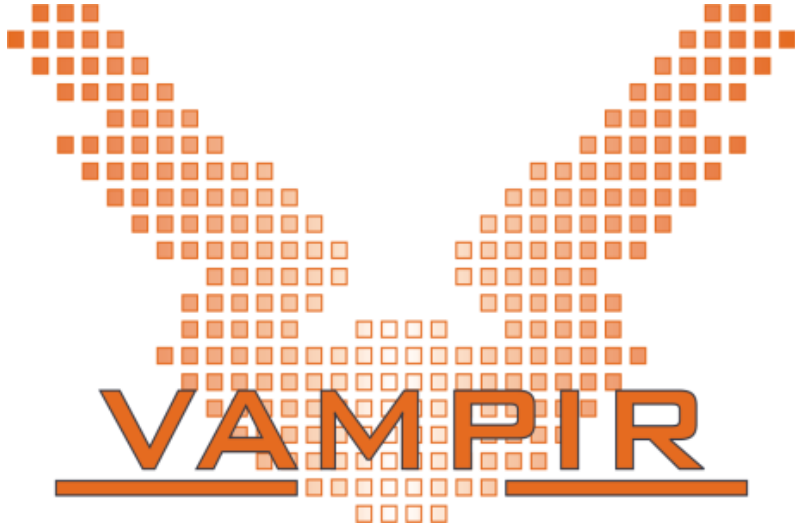
Interactive visualization and time-interval statistics with Vampir

Bert Wesarg
Technische Universität Dresden



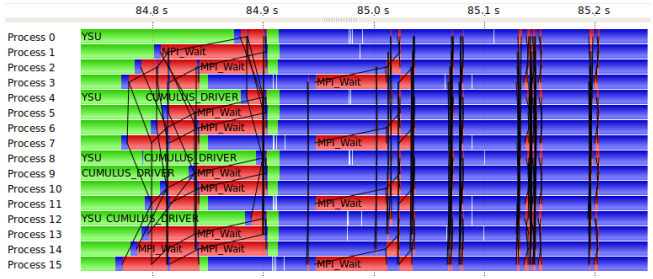
Outline

- **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



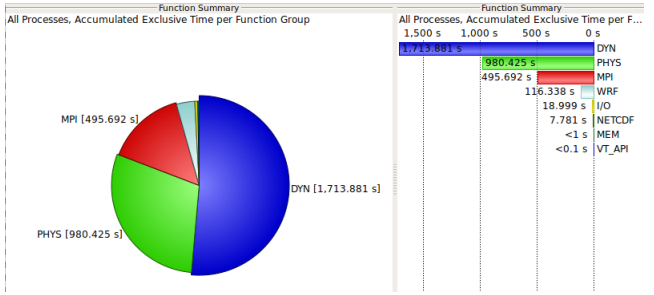
Event Trace Visualization with Vampir

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



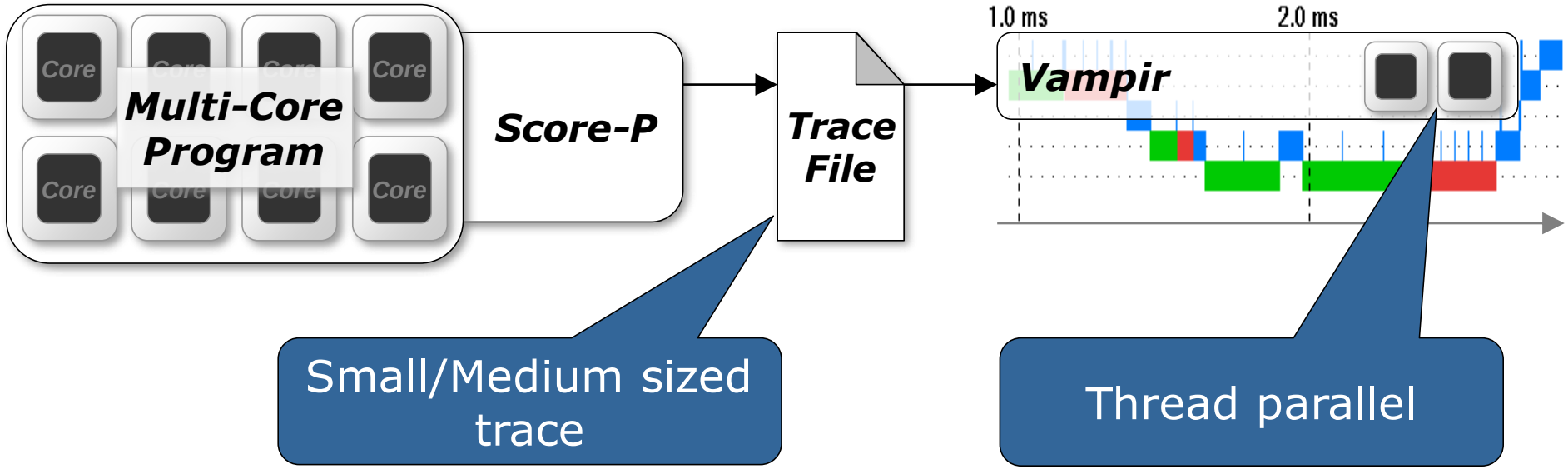
- **Timeline charts**
  - Show application activities and communication along a time axis

- **Summary charts**
  - Provide quantitative results for the currently selected time interval



Visualization Modes (1)
Directly on front end or local machine

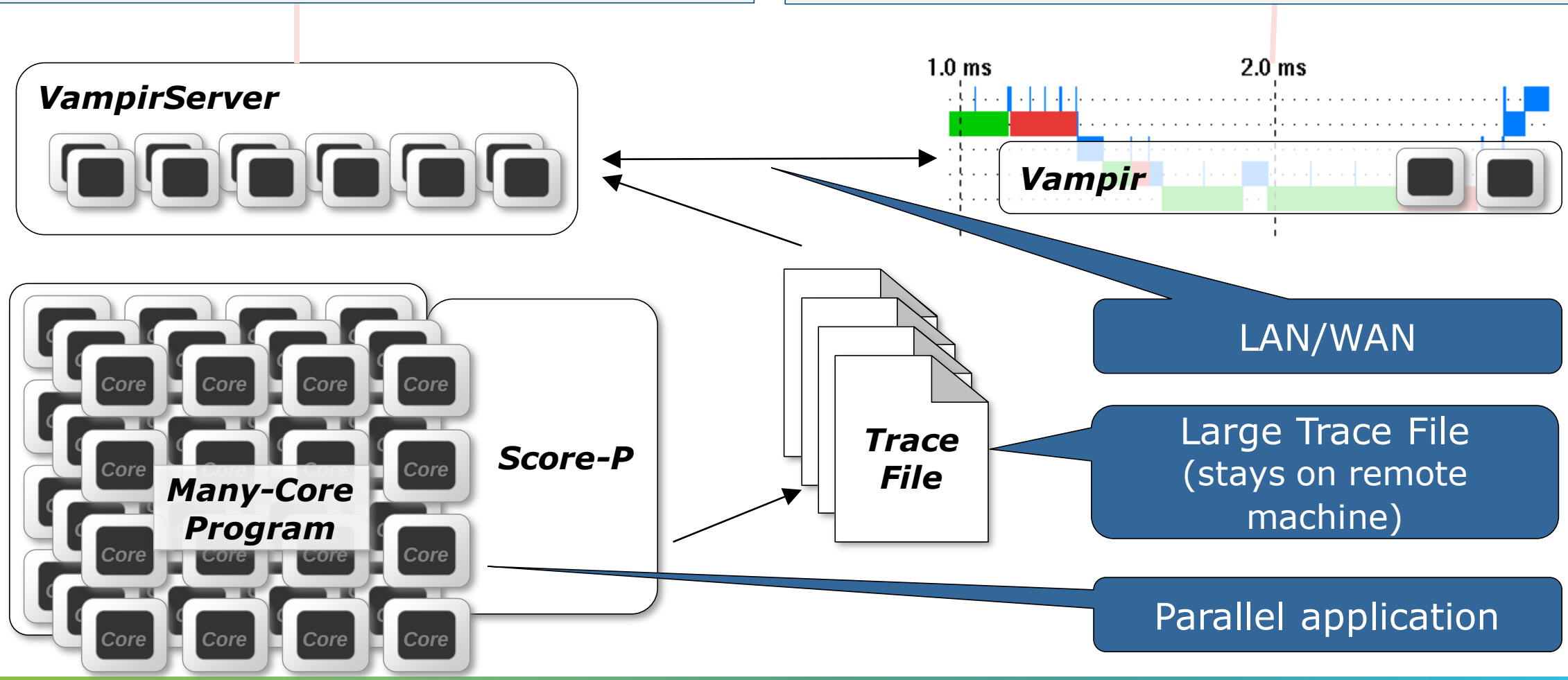
```
% vampir
```



Visualization Modes (2)
On local machine with remote VampirServer

```
% vampirserver start
```

```
% vampir
```

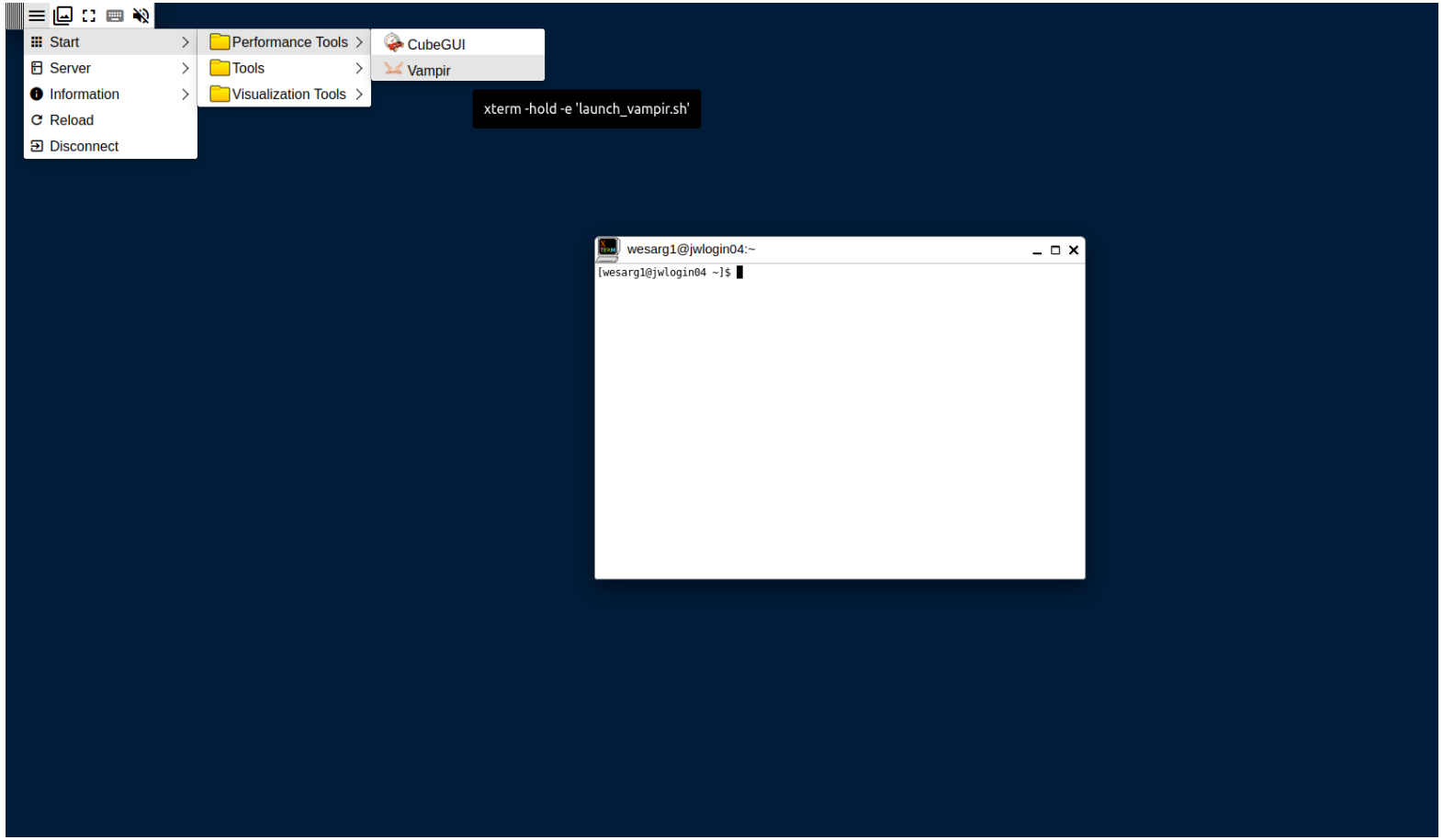


The main displays of Vampir

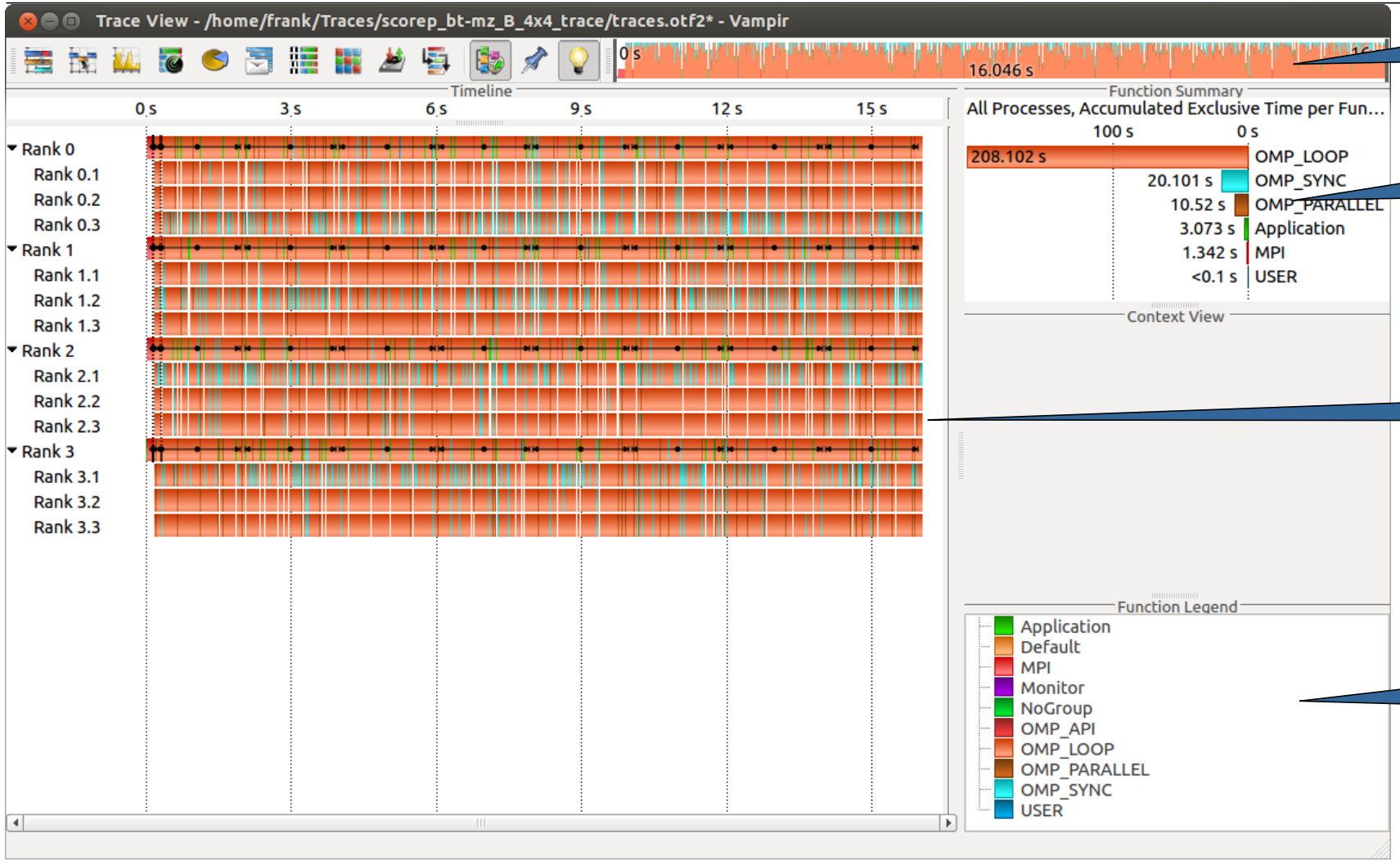
- Timeline Charts:
  -  Master Timeline
  -  Process Timeline
  -  Counter Data Timeline
  -  Performance Radar
- Summary Charts:
  -  Function Summary
  -  Message Summary
  -  Process Summary
  -  Communication Matrix View

Hands-on:
Visualizing and analyzing NPB-MZ-MPI / BT

Start Vampir inside Xpra Desktop



Visualization of the NPB-MZ-MPI / BT trace



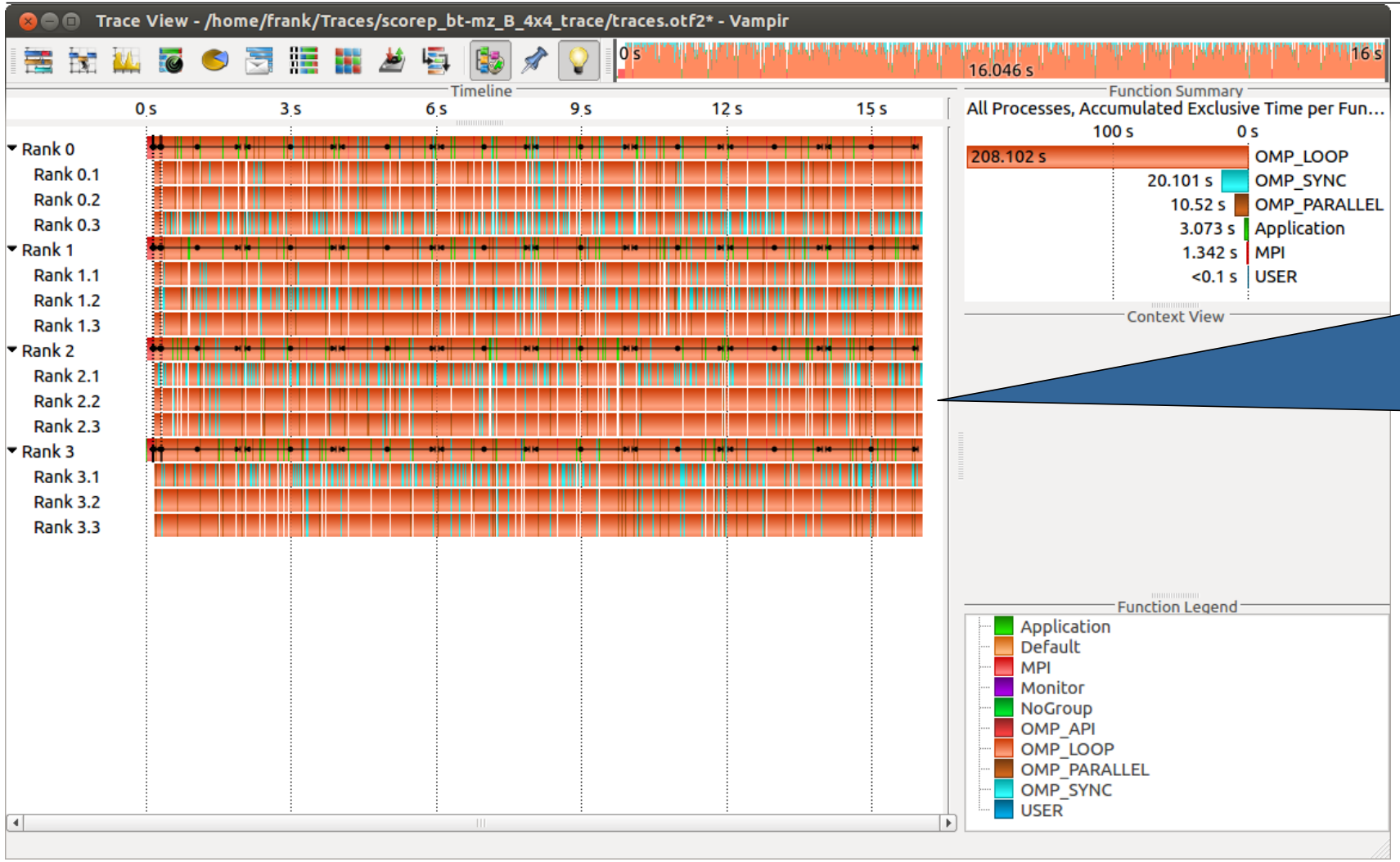
Navigation Toolbar

Function Summary

Master Timeline

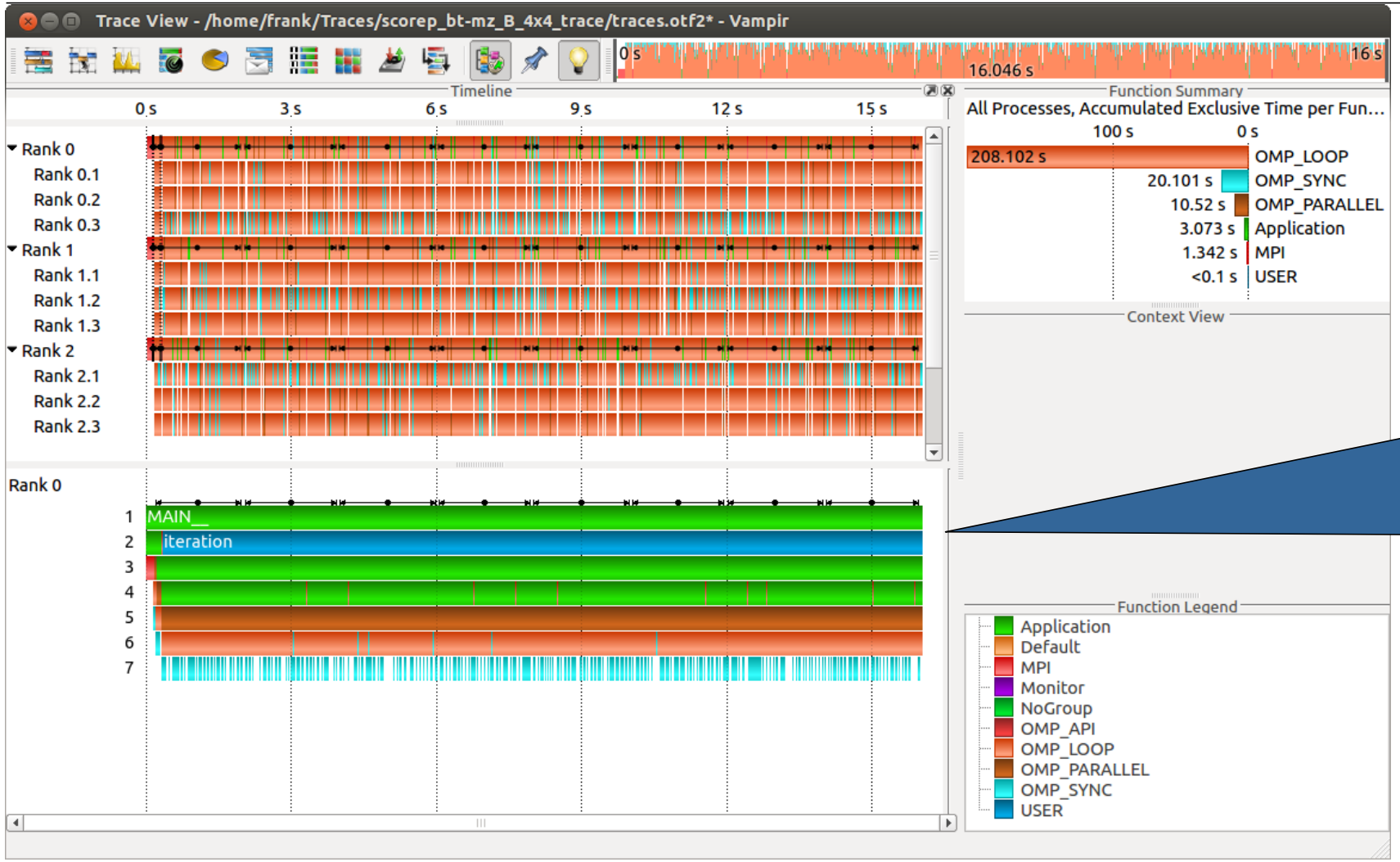
Function Legend

Visualization of the NPB-MZ-MPI / BT trace
Master Timeline



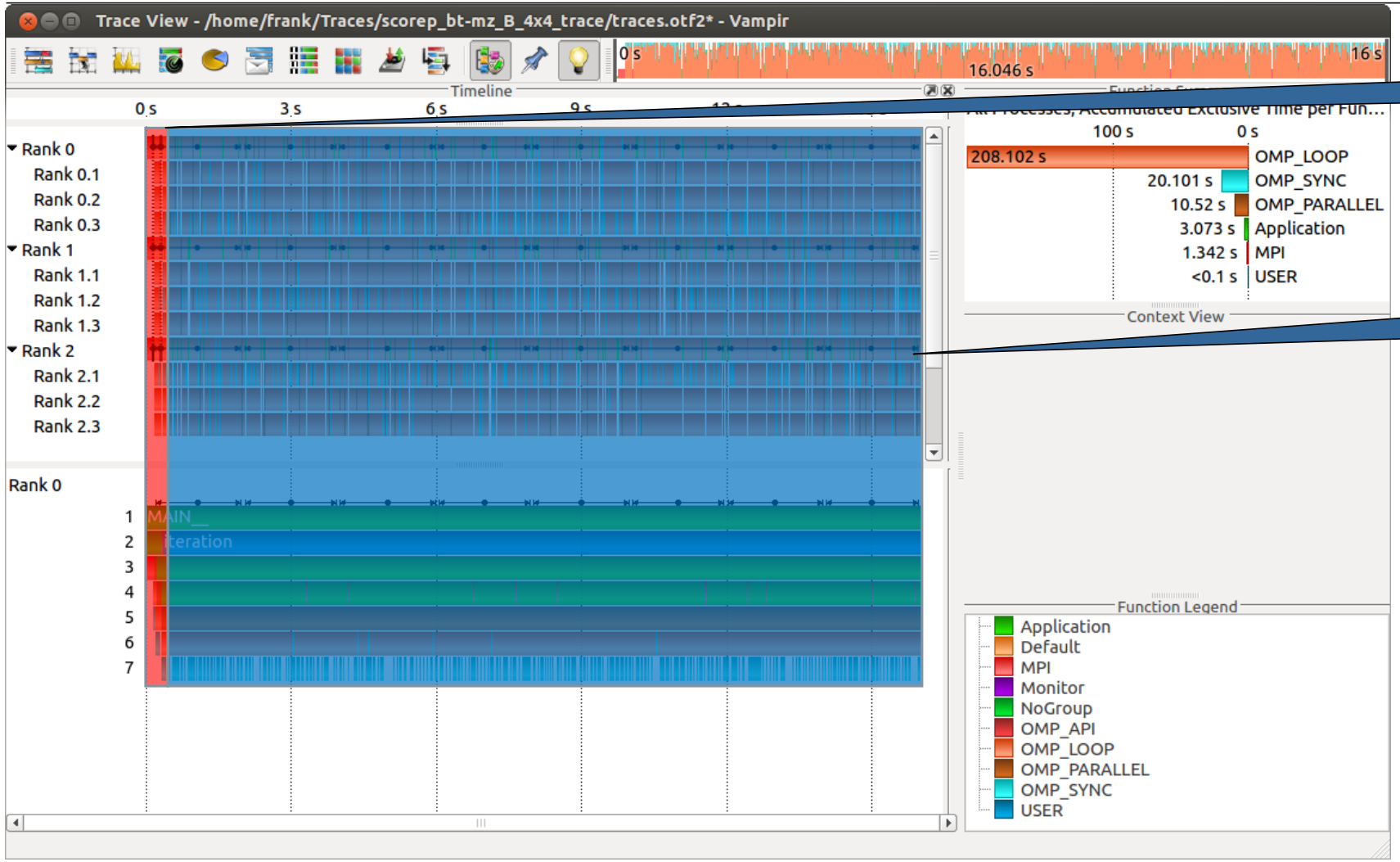
Detailed information about functions, communication and synchronization events for collection of processes.

Visualization of the NPB-MZ-MPI / BT trace
Process Timeline



Detailed information about different levels of function calls in a stacked bar chart for an individual process.

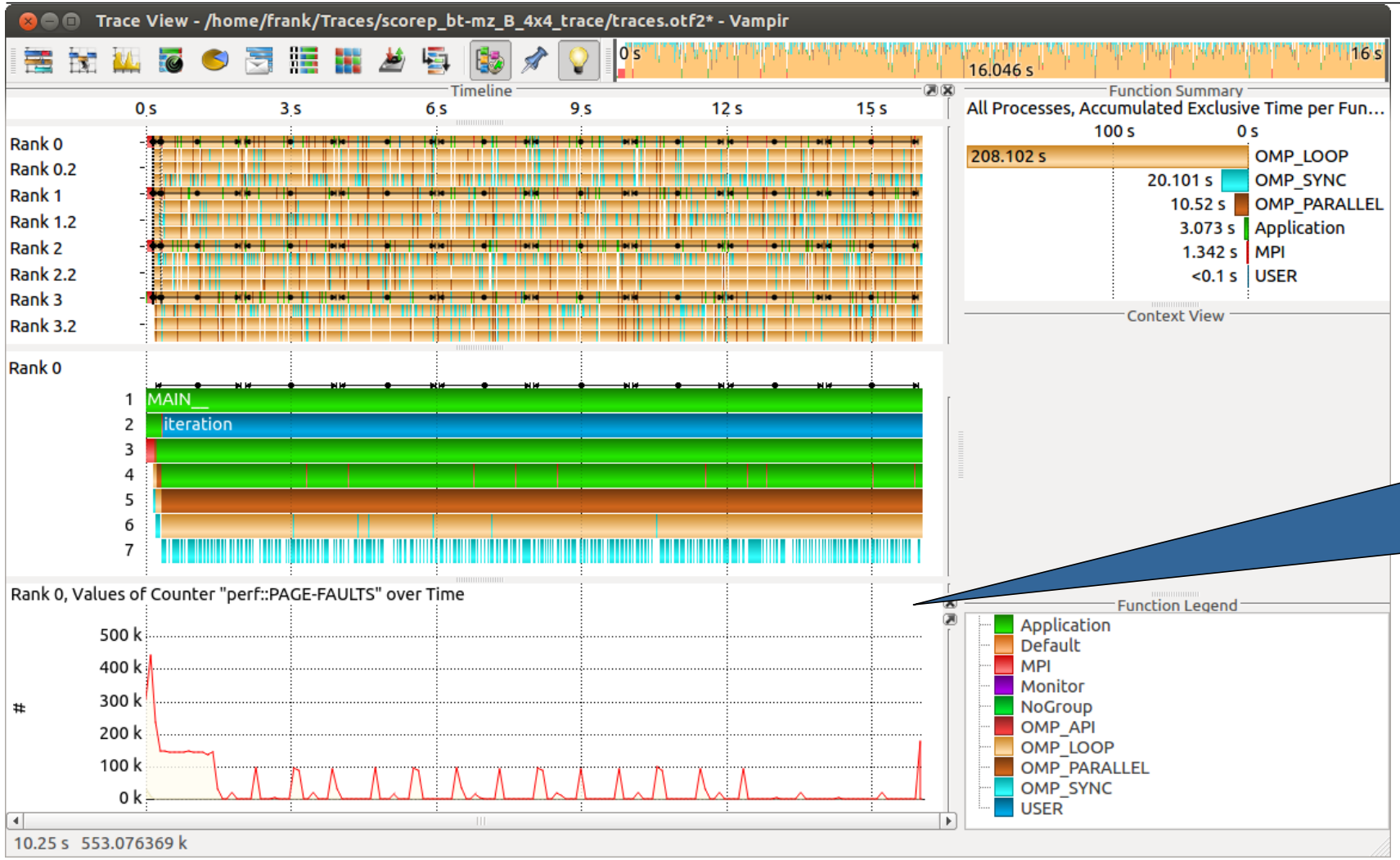
Visualization of the NPB-MZ-MPI / BT trace
Typical program phases



Initialisation Phase

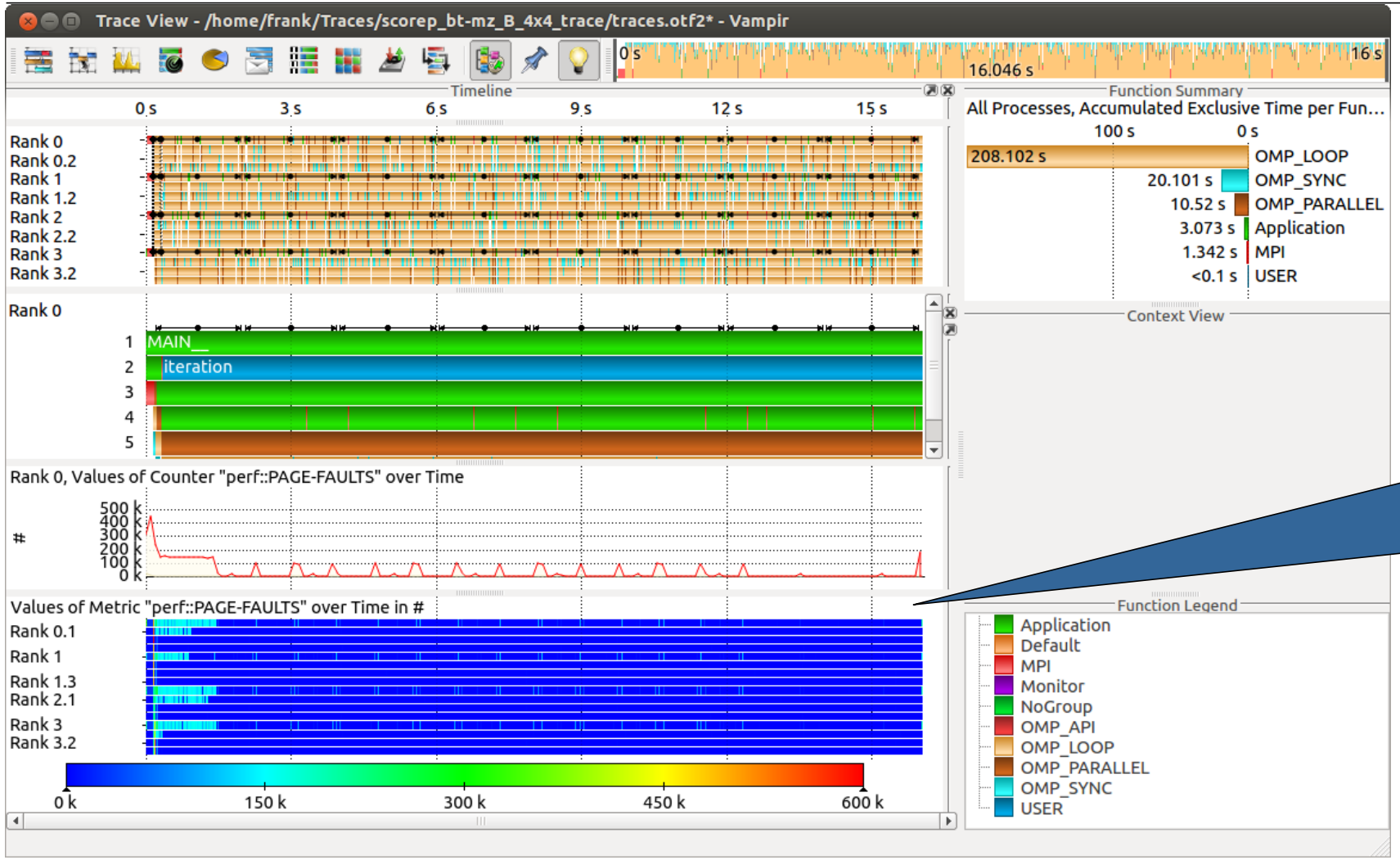
Computation Phase

Visualization of the NPB-MZ-MPI / BT trace
Counter Data Timeline



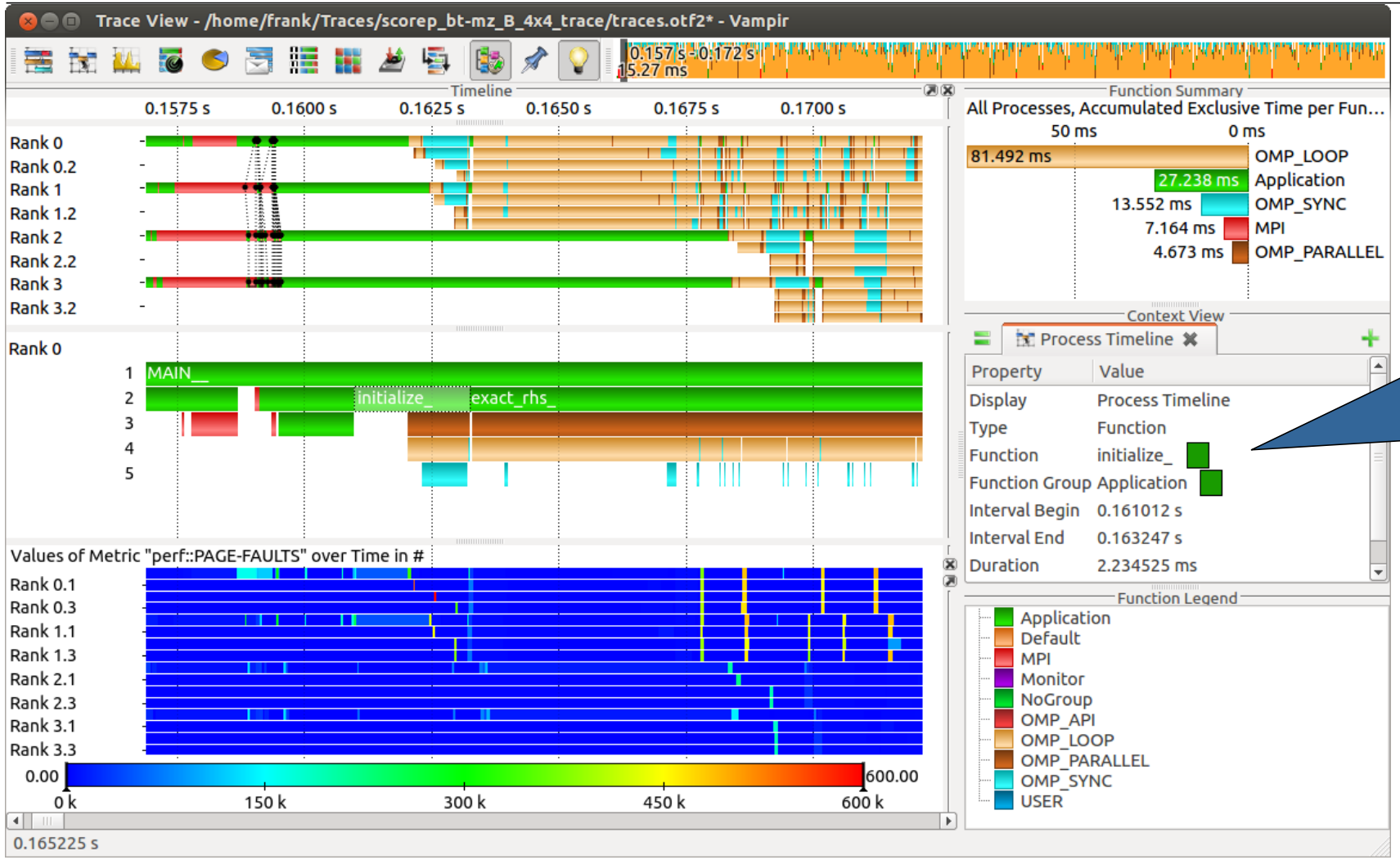
Detailed counter information over time for an individual process.

Visualization of the NPB-MZ-MPI / BT trace
Performance Radar



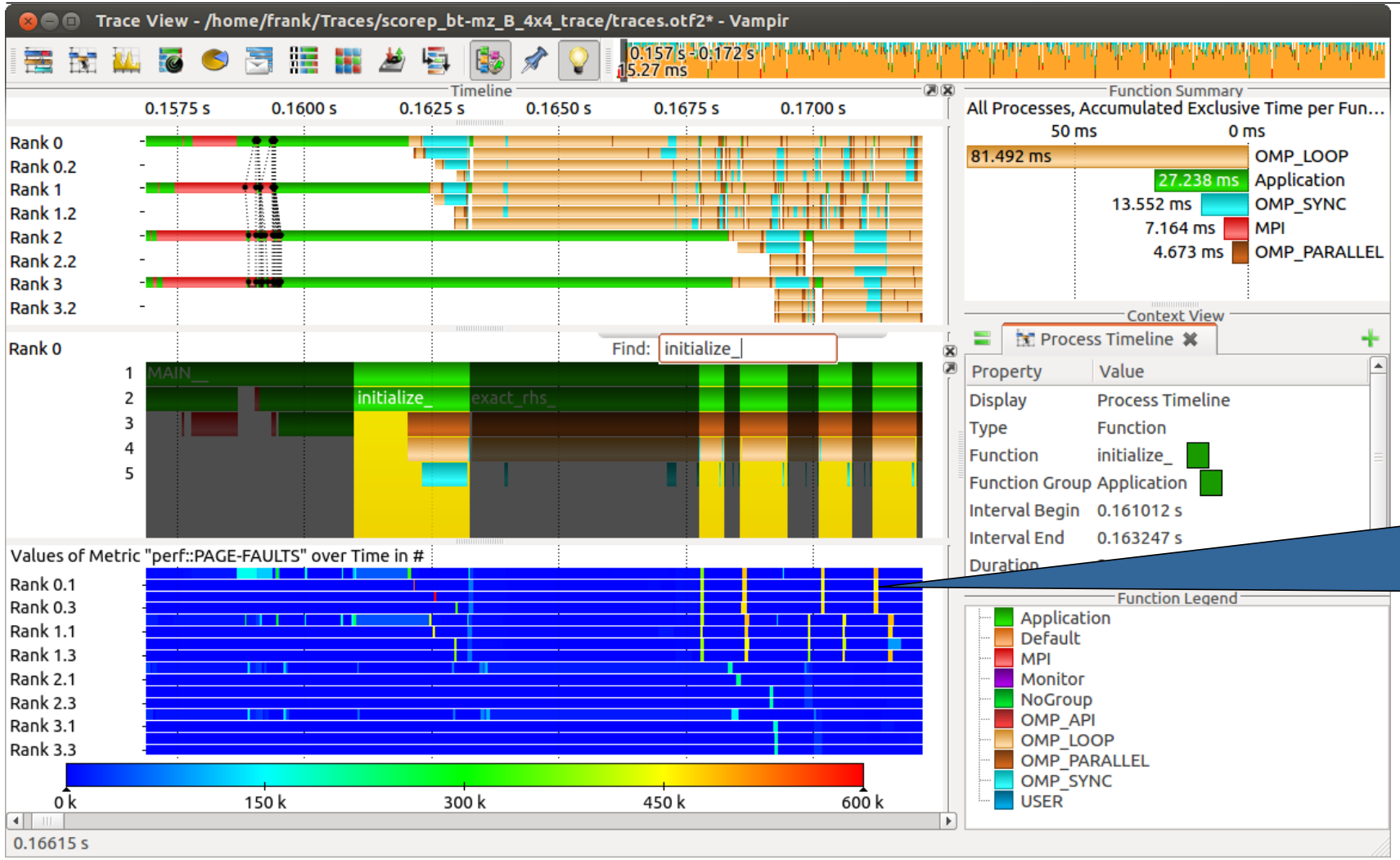
Detailed counter information over time for a collection of processes.

Visualization of the NPB-MZ-MPI / BT trace
Zoom in: Inititialisation Phase



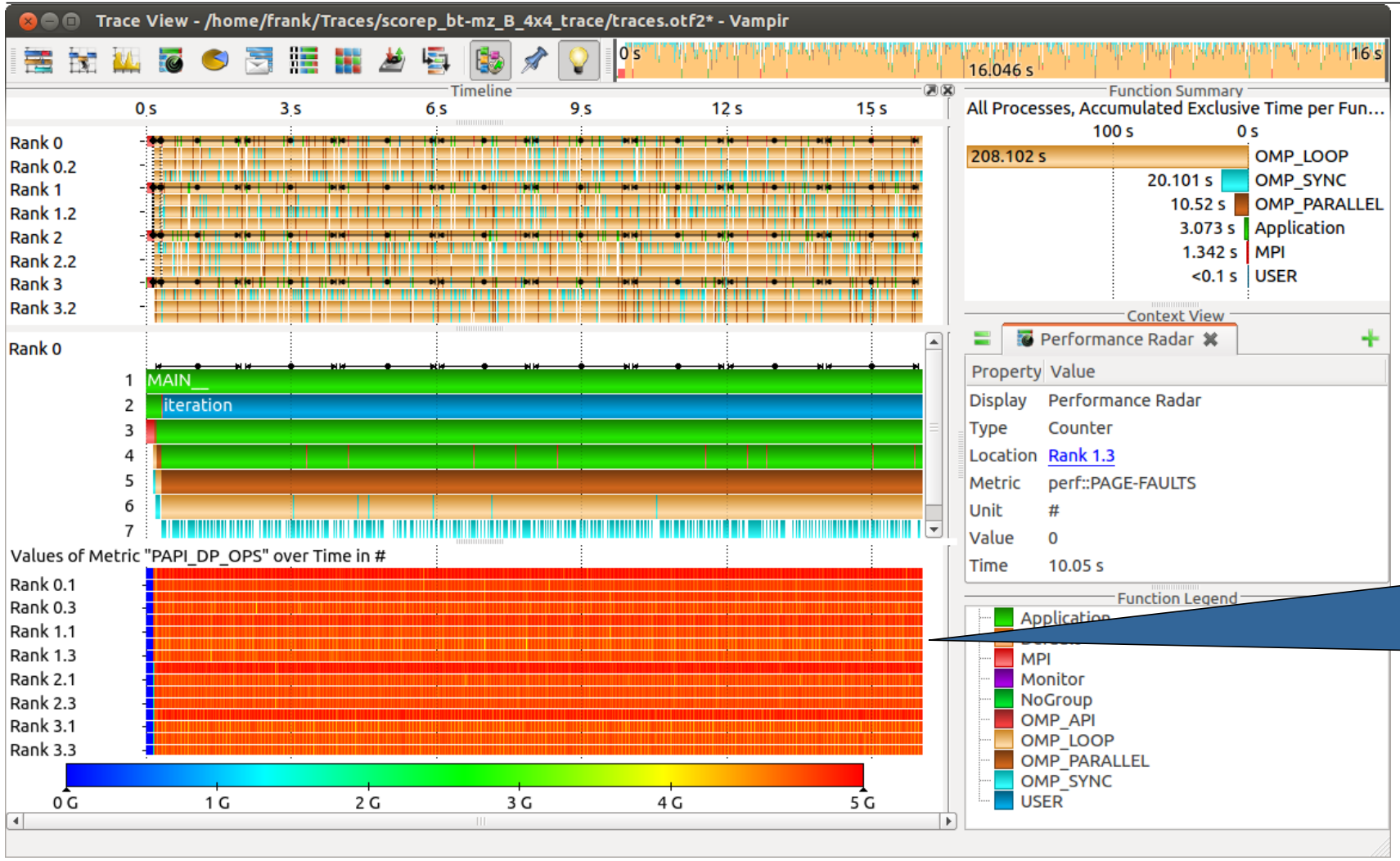
Context View: Detailed information about function “initialize_”.

Visualization of the NPB-MZ-MPI / BT trace
Find Function



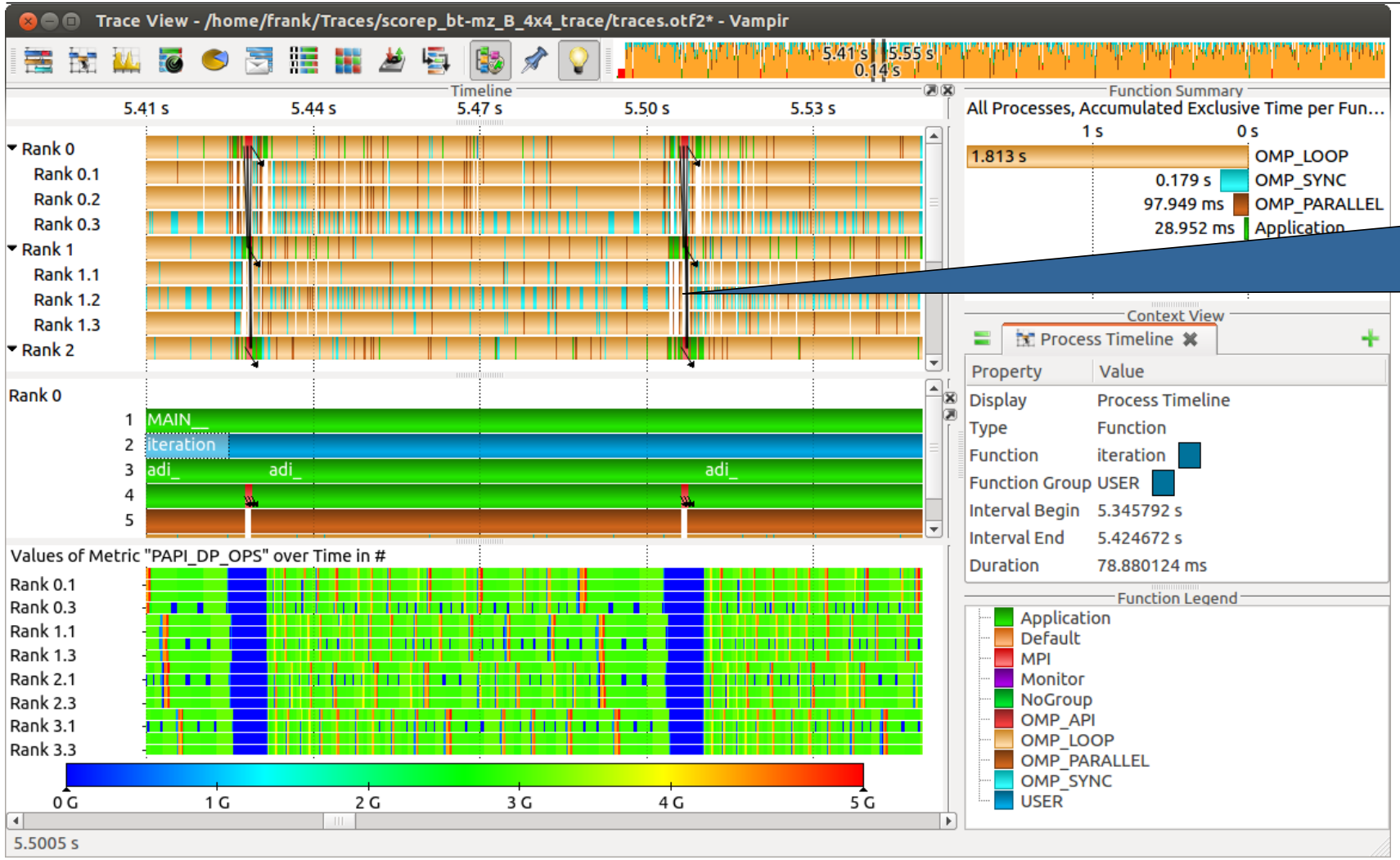
Execution of function "initialize_" results in higher page fault rates.

Visualization of the NPB-MZ-MPI / BT trace
Computation Phase



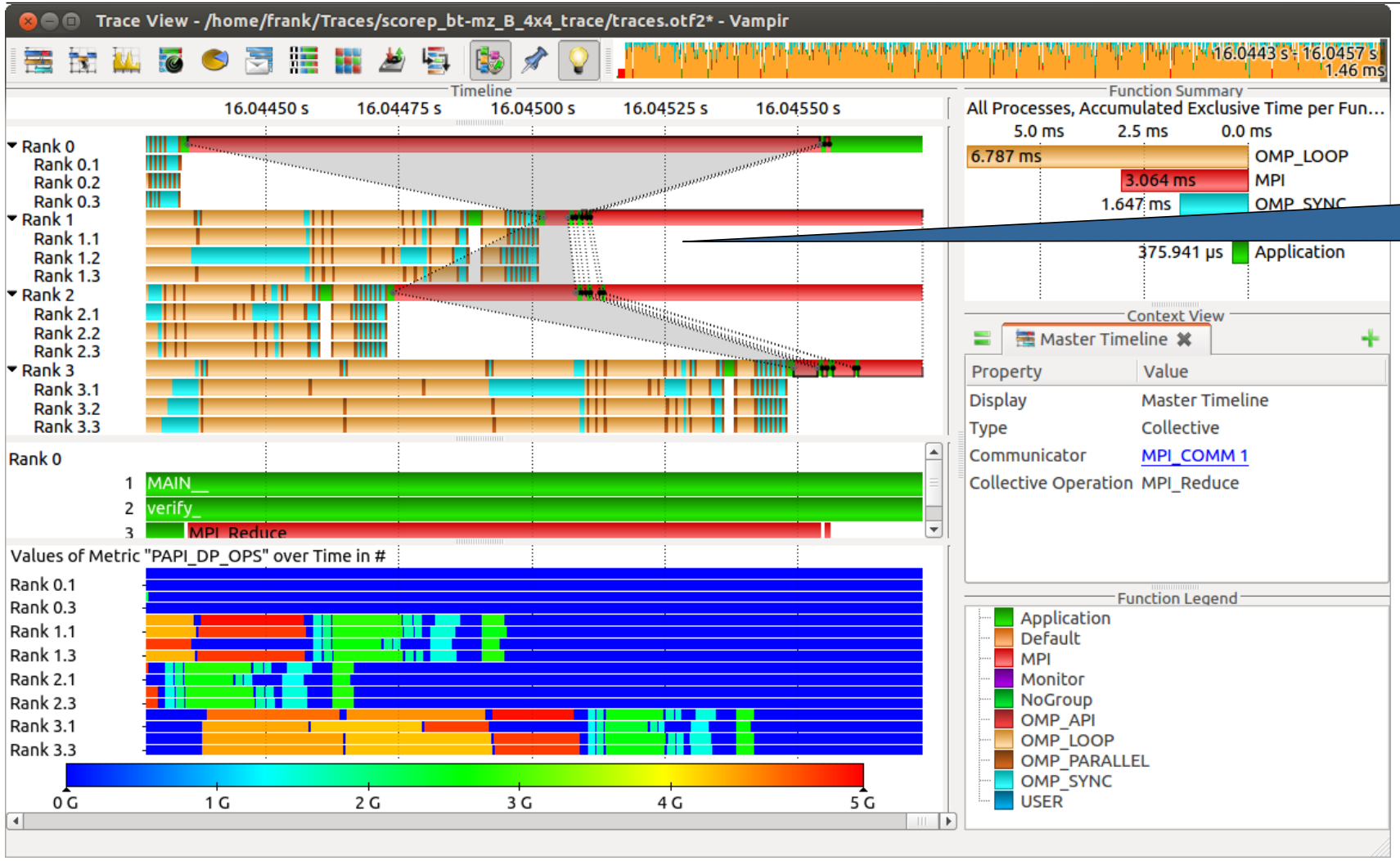
Computation phase results in higher floating point operations.

Visualization of the NPB-MZ-MPI / BT trace
Zoom in: Computation Phase



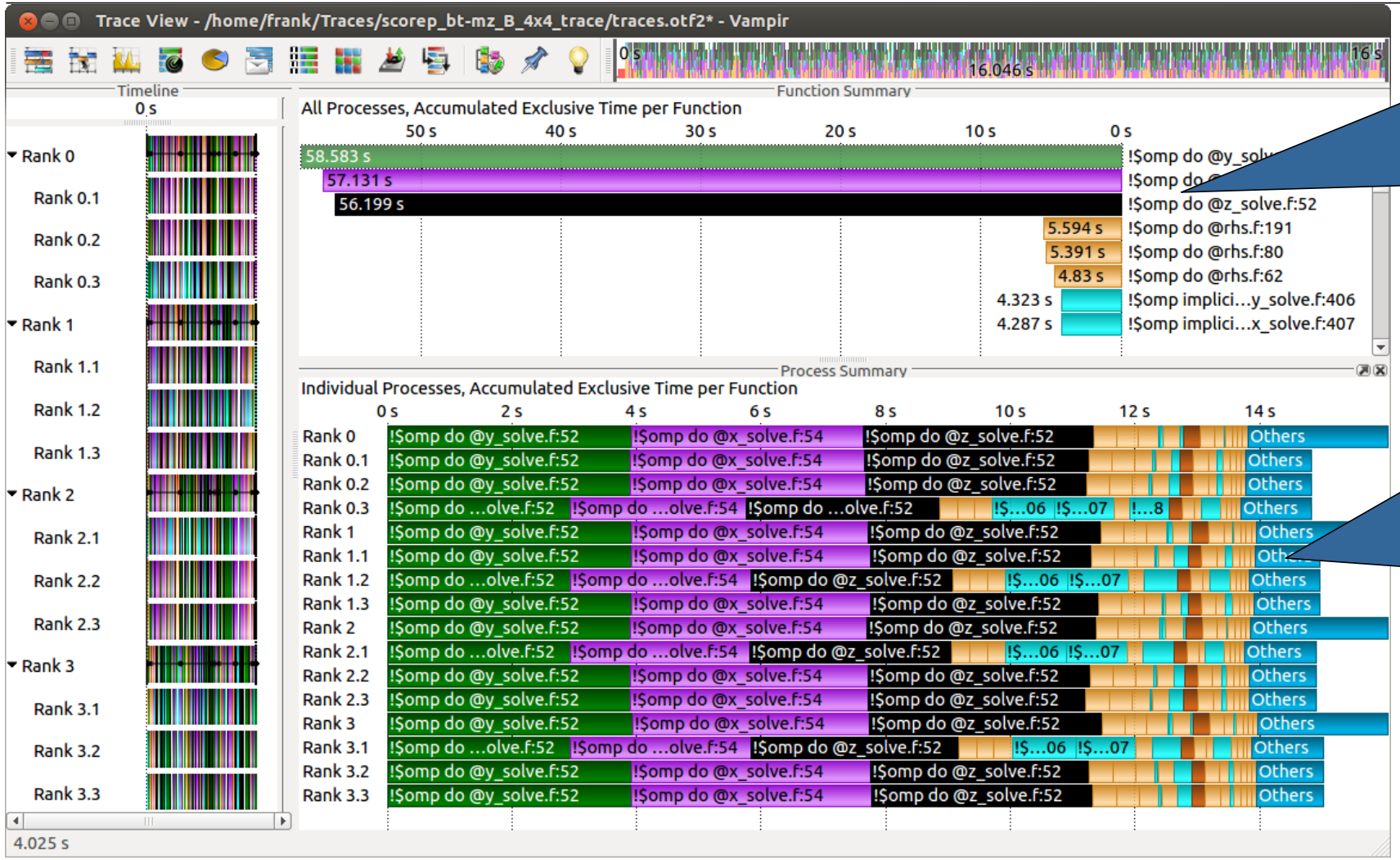
MPI communication results in lower floating point operations.

Visualization of the NPB-MZ-MPI / BT trace
Zoom in: Finalisation Phase



"Early reduce" bottleneck.

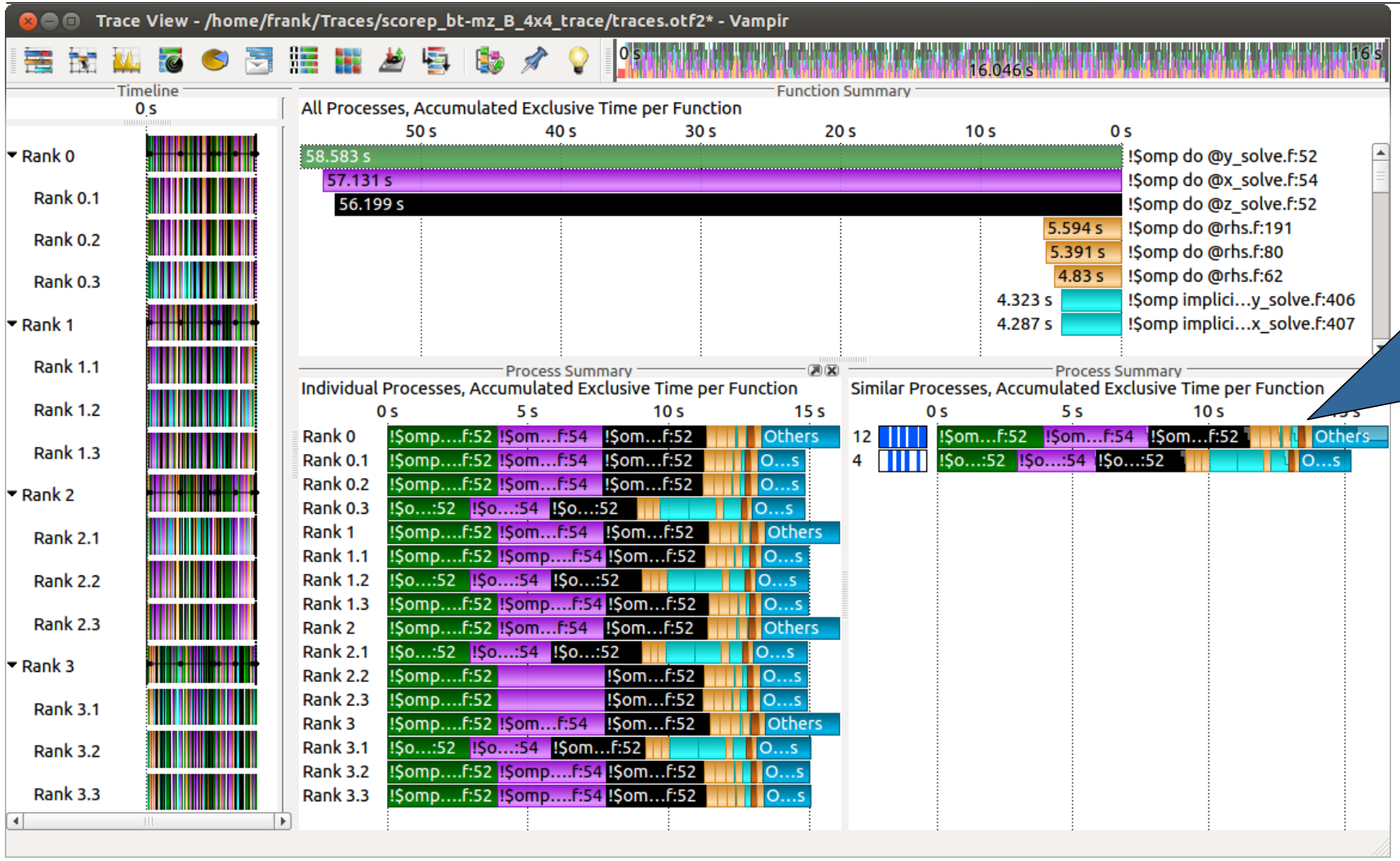
Visualization of the NPB-MZ-MPI / BT trace
Process Summary



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.

Visualization of the NPB-MZ-MPI / BT trace
Process Summary



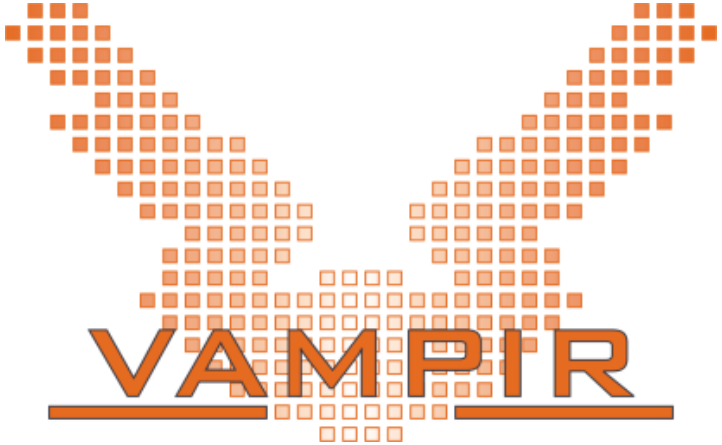
Find groups of similar processes and threads by using summarized function information.

Summary and Conclusion

Summary

- Vampir & VampirServer
  - Interactive trace visualization and analysis
  - Intuitive browsing and zooming
  - Scalable to large trace data sizes (20 TiByte)
  - Scalable to high parallelism (200,000 processes)
- Vampir for Linux, Windows, and Mac OS X

GWT Gesellschaft für Wissens- und Technologietransfer



Visit us at ZIH booth #**1533**

https://vampir.eu

service@vampir.eu

Automatic trace analysis with the Scalasca Trace Tools

Markus Geimer
Jülich Supercomputing Centre

trace tools
scalasca

Scalasca Trace Tools

DOI 10.5281/zenodo.4103922

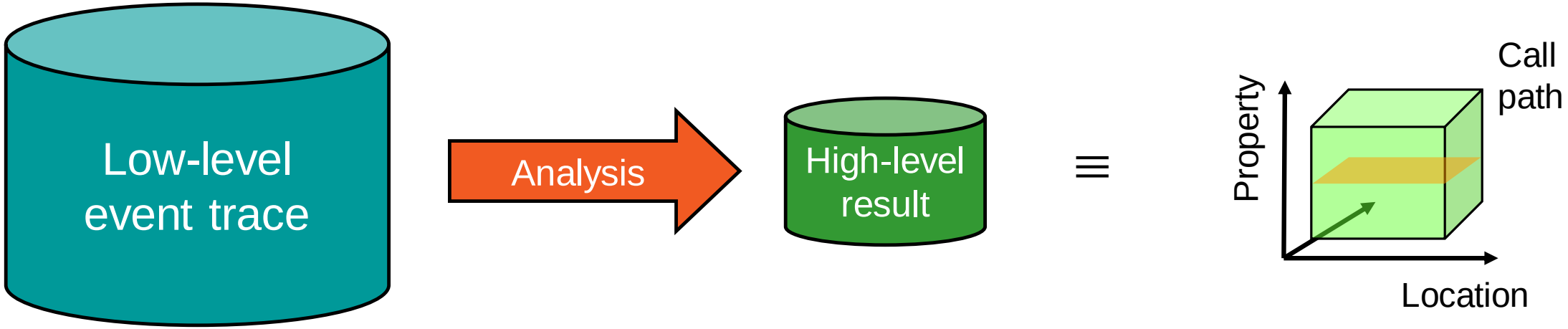
- **Scalable trace-based** performance analysis toolset for the most popular parallel programming paradigms
  - Current focus: MPI, OpenMP, and (to a limited extend) POSIX threads
  - Analysis of traces including *only host-side events* from applications using CUDA, OpenCL, or OpenACC (also in combination with MPI and/or OpenMP) is possible, but results need to be interpreted with some care

- Specifically targeting large-scale parallel applications
  - Demonstrated scalability up to 1.8 million parallel threads
  - Of course also works at small/medium scale

- Latest release:
  - Scalasca Trace Tools v2.6.1 (Dec 2022)

Automatic trace analysis

- Idea
  - Automatic search for patterns of inefficient behaviour
  - Classification of behaviour & quantification of significance
  - Identification of delays as root causes of inefficiencies



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

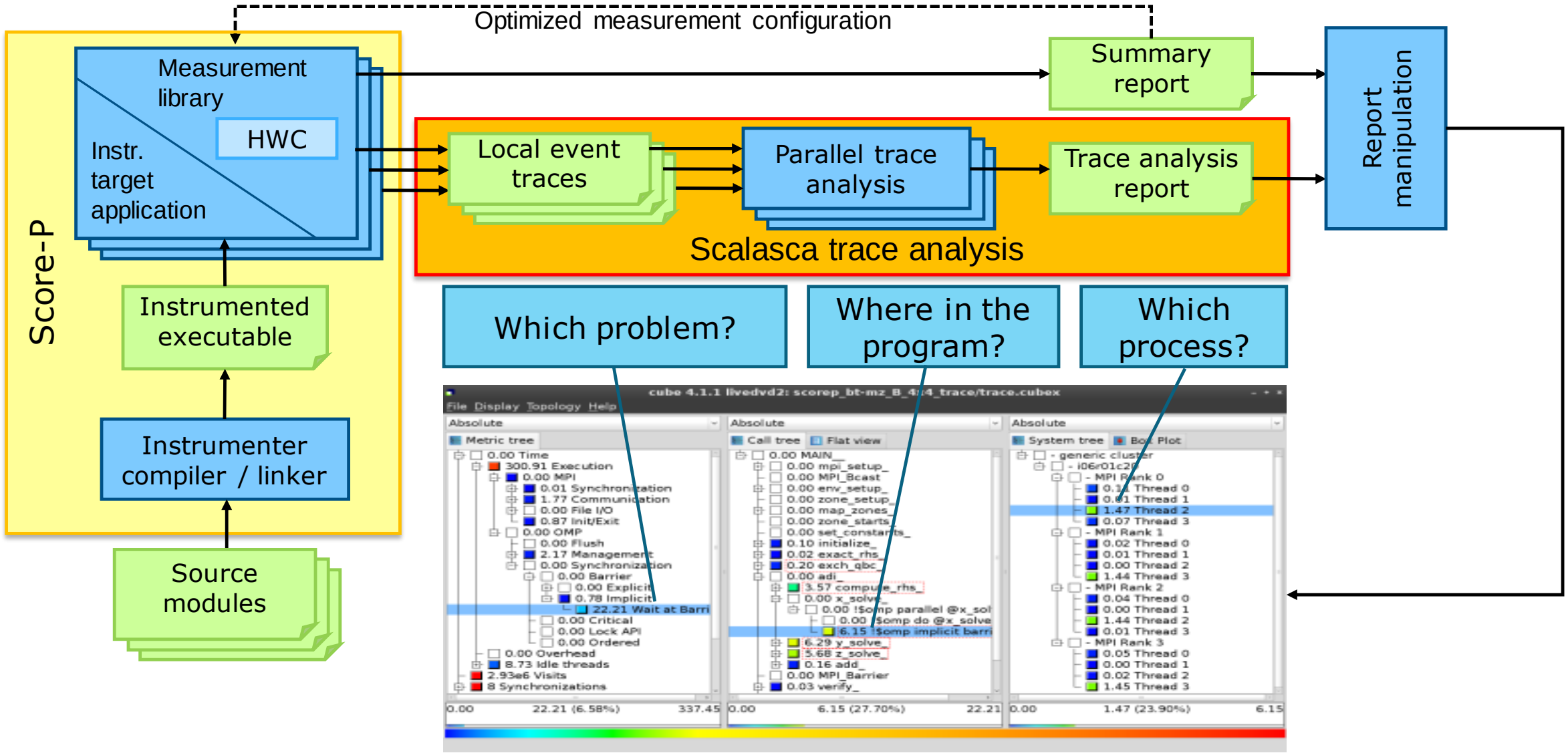
Scalasca Trace Tools: Features

- Open source, 3-clause BSD license
- Supports all major HPC platforms

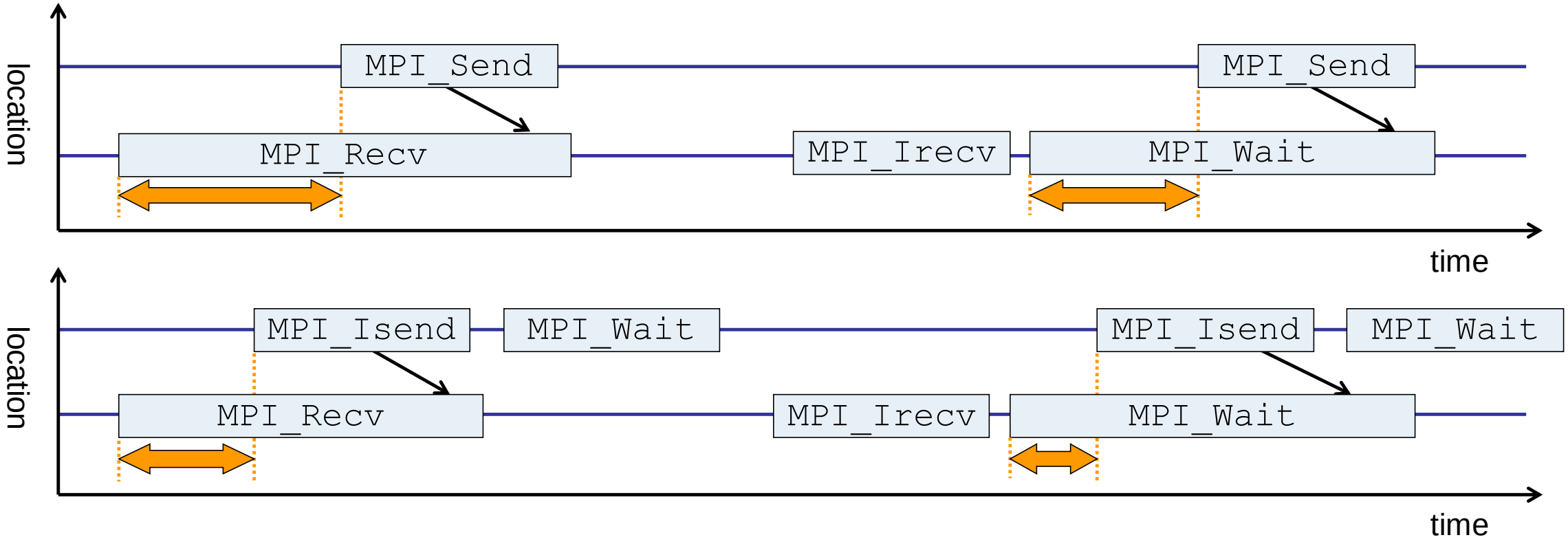
- Uses Score-P instrumenter & measurement libraries
  - Scalasca v2 core package focuses on trace-based analyses
  - Provides convenience commands for measurement, analysis, and post-processing
  - Supports common data formats
    - Reads event traces in OTF2 format
    - Writes analysis reports in CUBE4 format

- Current limitations:
  - Unable to handle traces …
    - with MPI thread level exceeding MPI_THREAD_FUNNELED
    - containing memory events, CUDA/OpenCL device events (kernel, memcpy), SHMEM, or OpenMP nested parallelism
  - PAPI/rusage metrics for trace events are ignored

Scalasca workflow

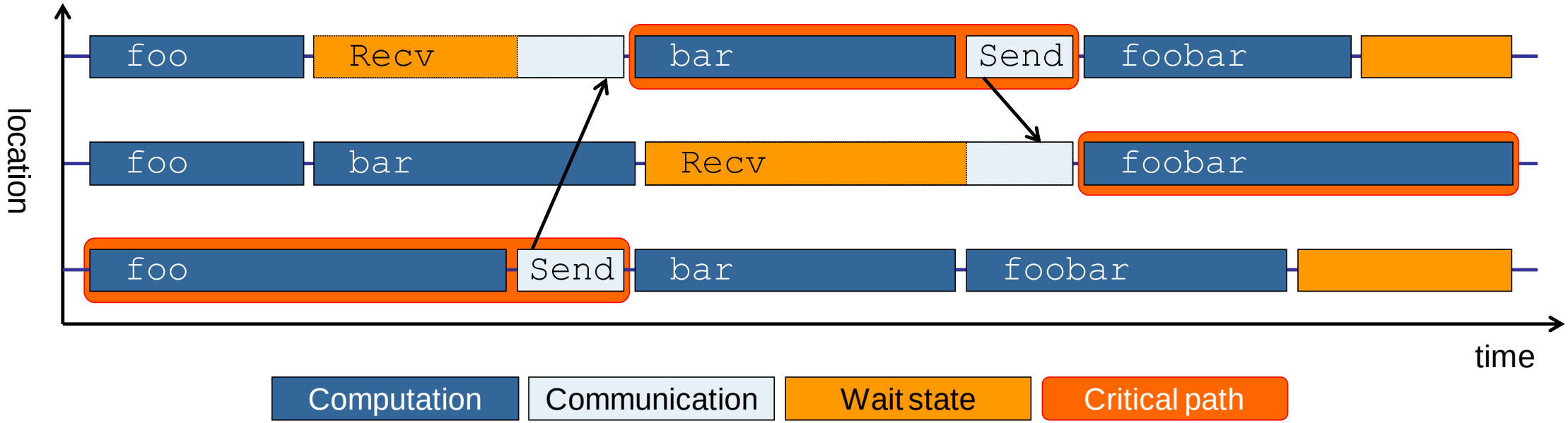


Example: "*Late Sender*" wait state



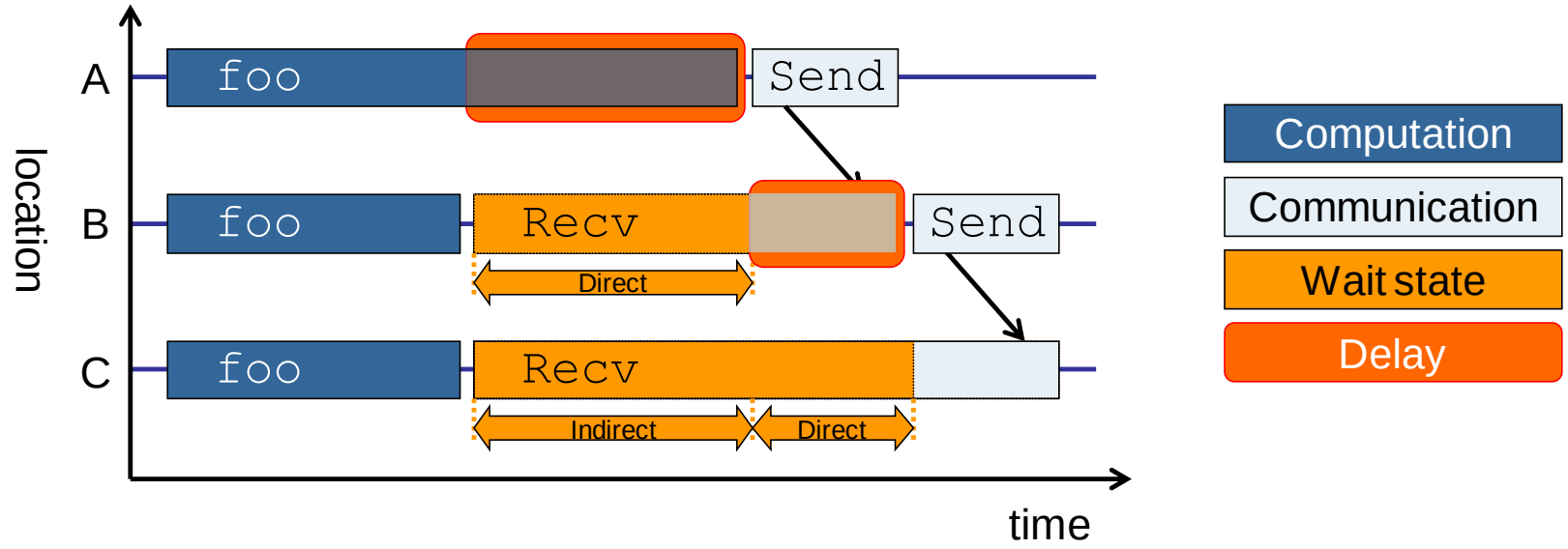
- Waiting time caused by a blocking receive operation posted earlier than the corresponding send
- Applies to blocking as well as non-blocking communication

Example: Critical path



- Shows call paths and processes/threads that are responsible for the program's wall-clock runtime
- Identifies good optimization candidates and parallelization bottlenecks

Example: Root-cause analysis



- Classifies wait states into direct and indirect (i.e., caused by other wait states)
- Identifies *delays* (excess computation/communication) as root causes of wait states
- Attributes wait states as *delay costs*

Hands-on:
TeaLeaf MPI+CUDA

Recap: Setup for exercises

- Connect to your training account on JUWELS Booster (with X11-forwarding)

```
% ssh -X <yourid>@juwels-booster.fz-juelich.de
```

- Set account and default environment (NVHPC + ParaStationMPI) via helper script

```
% source /p/project/training2341/setup.sh
```

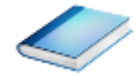
- Change to directory containing TeaLeaf_CUDA sources
  - Existing instrumented executable can be reused

```
% cd $WORK
% cd TeaLeaf_CUDA
```

- Load Scalasca module
  - Depends on (i.e., implicitly loads) Score-P & CubeGUI

```
% module load Scalasca
```

TeaLeaf_CUDA summary measurement collection...



```
% cd bin.scorep
% cp ../jobscripts/juwelsbooster/scalasca.sbatch .
% cat scalasca.sbatch

# Score-P measurement configuration
export SCOREP_CUDA_ENABLE=default
export SCOREP_CUDA_BUFFER=48M
#export SCOREP_EXPERIMENT_DIRECTORY=scorep-tea_leaf-8
export SCOREP_FILTERING_FILE=../config/scorep.filt
#export SCOREP_ENABLE_TRACING=true
#export SCOREP_TOTAL_MEMORY=250M

# Scalasca configuration
export SCAN_ANALYZE_OPTS="--time-correct"

# Run the application
scan -s  srun ./tea_leaf
```

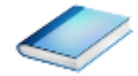
- Change to directory with the Score-P instrumented executable and edit the job script

Hint:
scan = scalasca -analyze
-s = profile/summary (default)

```
% sbatch scalasca.sbatch
```

- Submit the job

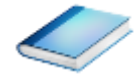
TeaLeaf_CUDA summary measurement



```
S=C=A=N: Scalasca 2.6.1 runtime summarization
S=C=A=N: ./scorep_tea_leaf_8_sum experiment archive
S=C=A=N: Wed Oct 25 14:51:20 2023: Collect start
srun ./tea_leaf

   Tea version    1.400

 [... More application output ...]

S=C=A=N: Wed Oct 25 14:52:09 2023: Collect done (status=0) 49s
S=C=A=N: ./scorep_tea_leaf_8_sum complete.
```

- Run the application using the Scalasca measurement collection & analysis nexus prefixed to launch command
- Creates experiment directory:
  scorep_tea_leaf_8_sum

TeaLeaf_CUDA summary analysis report examination



- Score summary analysis report

```
% square -s  scorep_tea_leaf_8_sum
INFO: Post-processing runtime summarization result (profile.cubex)...
INFO: Score report written to ./scorep_tea_leaf_8_sum/scorep.score
```

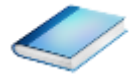
- Post-processing and interactive exploration with Cube

```
% square  scorep_tea_leaf_8_sum
INFO: Displaying ./scorep_tea_leaf_8_sum/summary.cubex...
                    [GUI showing summary analysis report]
```

Hint:
Copy ‘summary.cubex’ to local system (laptop) using ‘scp’ to improve responsiveness of GUI

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

TeaLeaf_CUDA trace measurement collection...



```
% cp ../jobscripts/juwelsbooster/scalasca.sbatch .
% cat scalasca.sbatch

# Score-P measurement configuration
export SCOREP_CUDA_ENABLE=runtime
export SCOREP_CUDA_BUFFER=48M
#export SCOREP_EXPERIMENT_DIRECTORY=scorep-tea_leaf-8
export SCOREP_FILTERING_FILE=../config/scorep.filt
#export SCOREP_ENABLE_TRACING=true
export SCOREP_TOTAL_MEMORY=250M

# Scalasca configuration
export SCAN_ANALYZE_OPTS="--time-correct"

# Run the application
scan -t  srun ./tea_leaf
```

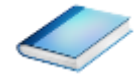
- Change to directory with the Score-P instrumented executable and edit the job script

Hint:
scan = scalasca -analyze
-t = trace collection & analysis

```
% sbatch scalasca.sbatch
```

- Submit the job

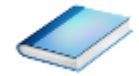
TeaLeaf_CUDA trace measurement ... collection



```
S=C=A=N: Scalasca 2.6.1 trace collection and analysis
S=C=A=N: Wed Oct 25 14:58:52 2023: Collect start
srun ./tea_leaf


   Tea version    1.400

 [... More application output ...]

S=C=A=N: Wed Oct 25 14:59:33 2023: Collect done (status=0) 41s
```

- Starts measurement with collection of trace files …

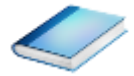
TeaLeaf_CUDA trace measurement ... analysis



```
…
S=C=A=N: Wed Oct 25 14:59:33 2023: Analyze start
srun scout.mpi --time-correct ./scorep_tea_leaf_8_trace/traces.otf2

SCOUT   (Scalasca 2.6.1)

Analyzing experiment archive ./scorep_tea_leaf_8_trace/traces.otf2

Opening experiment archive ... done (0.009s).
Reading definition data    ... done (0.008s).
Reading event trace data   ... done (0.695s).
Preprocessing              ... done (0.721s).
Timestamp correction       ... done (0.711s).
Analyzing trace data       ... done (10.108s).
Writing analysis report    ... done (0.137s).

Max. memory usage         : 426.422MB

        # passes         : 1
        # violated       : 0

Total processing time     : 12.489s
S=C=A=N: Wed Oct 25 14:59:49 2023: Analyze done (status=0) 16s
```

- Continues with automatic (parallel) analysis of trace files

TeaLeaf CUDA trace analysis report exploration



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

```
% square  scorep_tea_leaf_8_trace
INFO: Post-processing runtime summarization report (profile.cubex)...
INFO: Post-processing trace analysis report (scout.cubex)...
INFO: Displaying ./scorep_tea_leaf_8_trace/trace.cubex...

                [GUI showing trace analysis report]
```

Hint:
Run ‘square -s’ first and then copy ‘trace.cubex’ to local system (laptop) using ‘scp’ to improve responsiveness of GUI

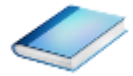
Demo:
TeaLeaf MPI+OpenMP case study

Case study: TeaLeaf MPI+OpenMP

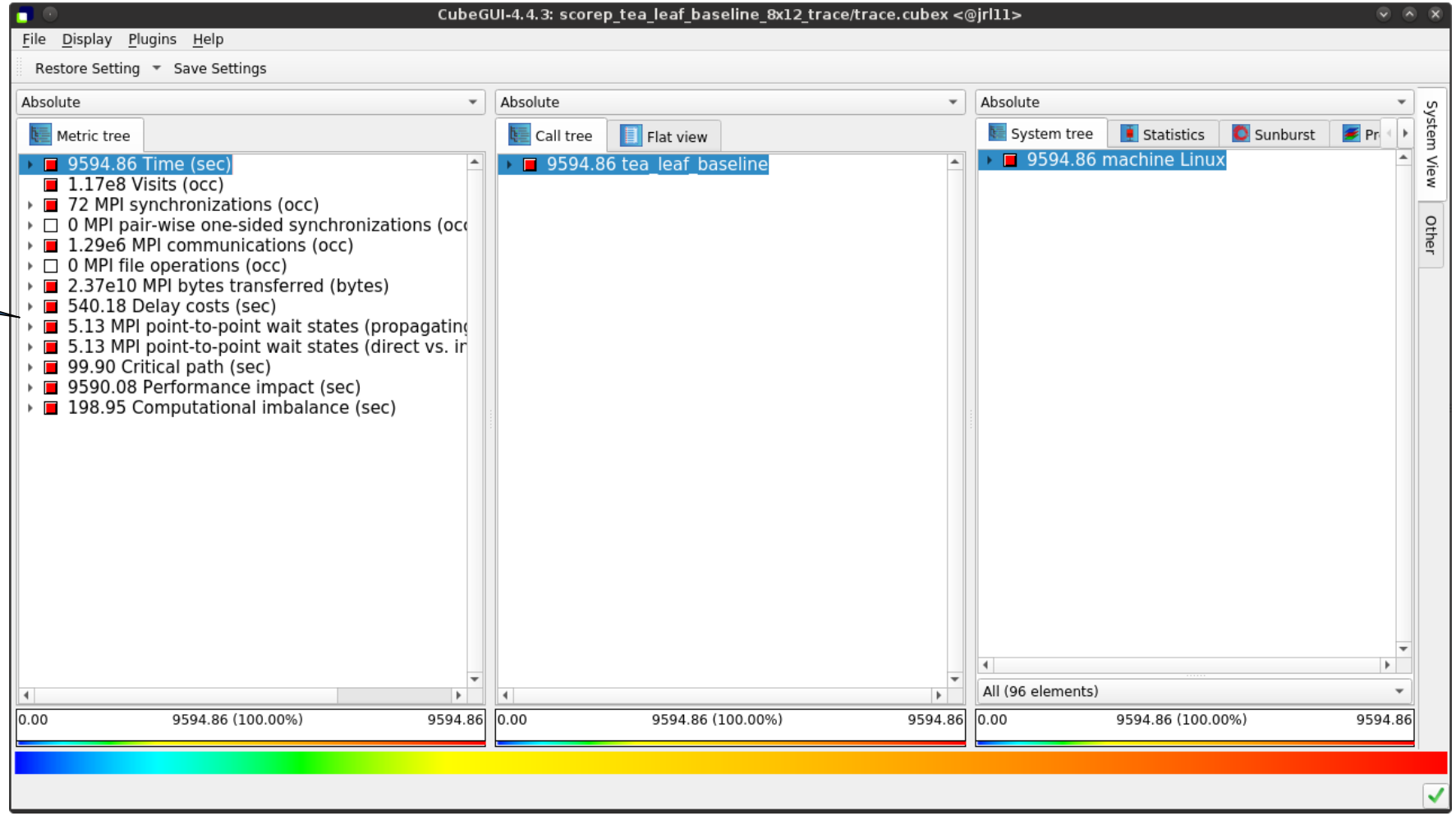
- HPC mini-app developed by the UK Mini-App Consortium
  - Solves the linear 2D heat conduction equation on a spatially decomposed regular grid using a 5 point stencil with implicit solvers
  - Part of the Mantevo 3.0 suite
  - Available on GitHub: https://uk-mac.github.io/TeaLeaf/
- Measurements of TeaLeaf reference v1.0 taken on Jureca cluster @ JSC
  - Using Intel 19.0.3 compilers, Intel MPI 2019.3, Score-P 5.0, and Scalasca 2.5
  - Run configuration
    - 8 MPI ranks with 12 OpenMP threads each
    - Distributed across 4 compute nodes (2 ranks per node)
    - Test problem "5": 4000 × 4000 cells, CG solver

```
% cube scorep_tea_leaf_baseline_8x12_trace/trace.cubex
                              [GUI showing post-processed trace analysis report]
```

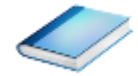
Scalasca analysis report exploration (opening view)



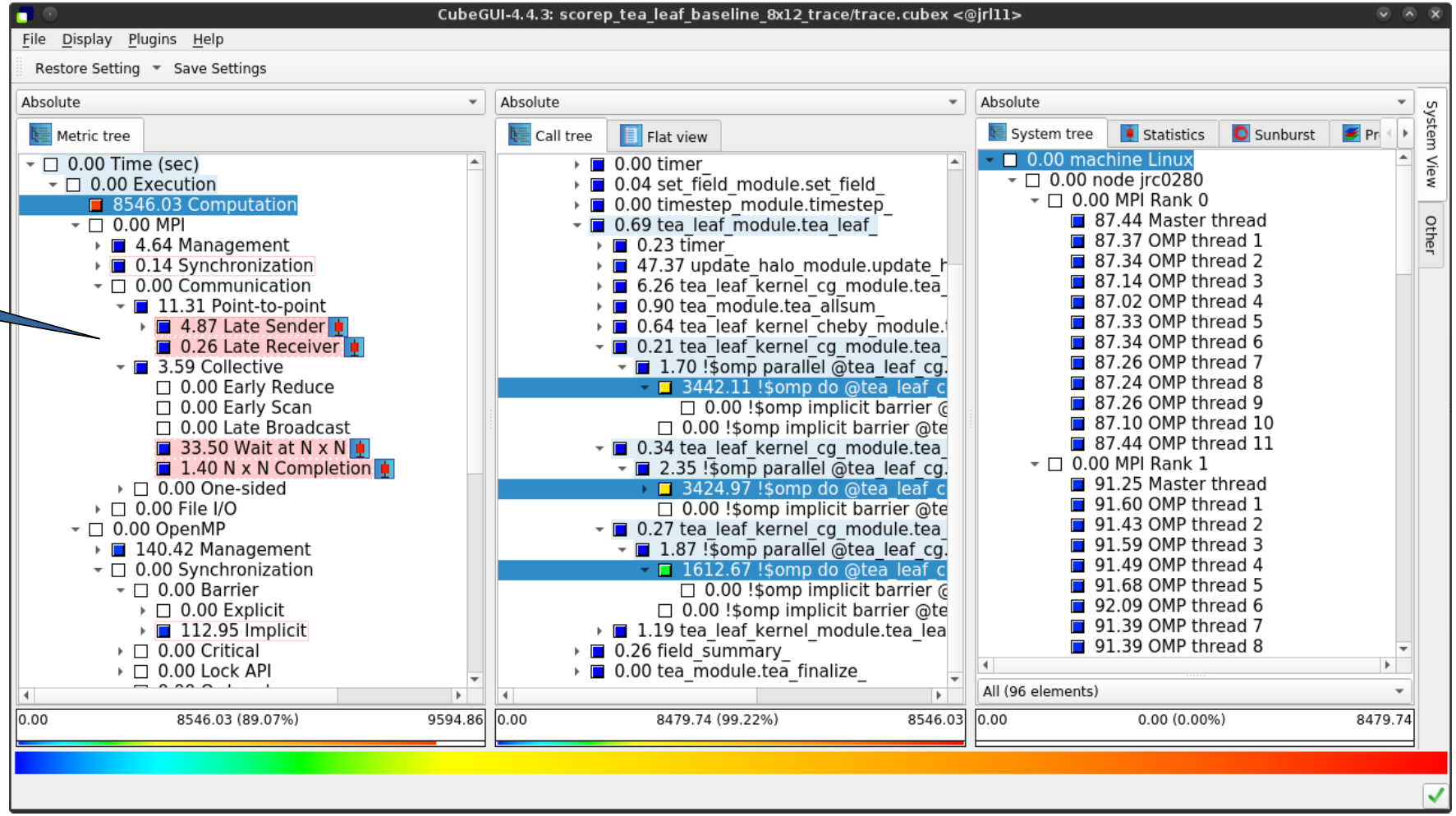
Additional top-level metrics produced by the trace analysis…



Scalasca wait-state metrics

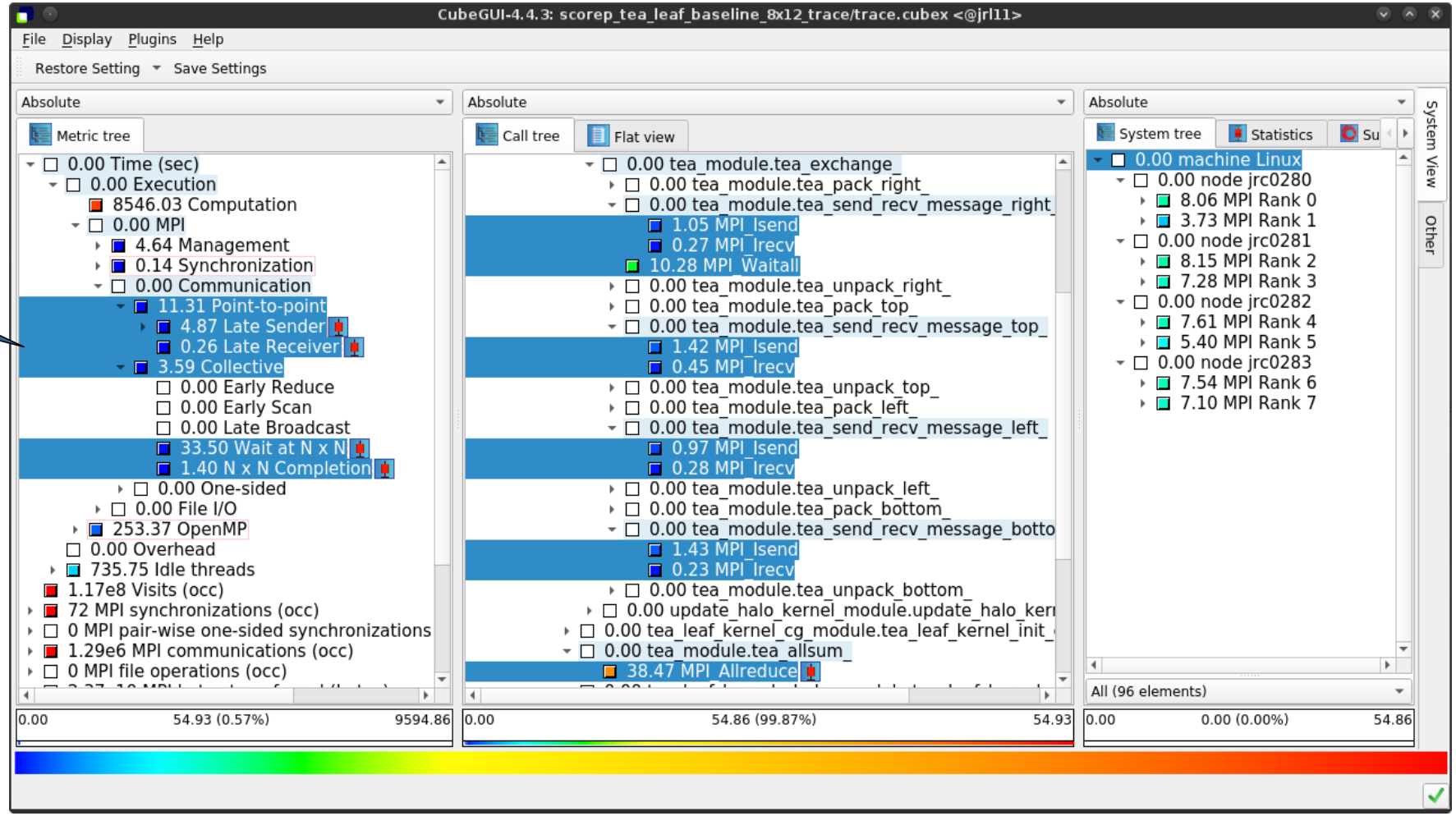


…plus additional wait-state metrics as part of the “Time” hierarchy

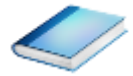


TeaLeaf Scalasca report analysis (I)

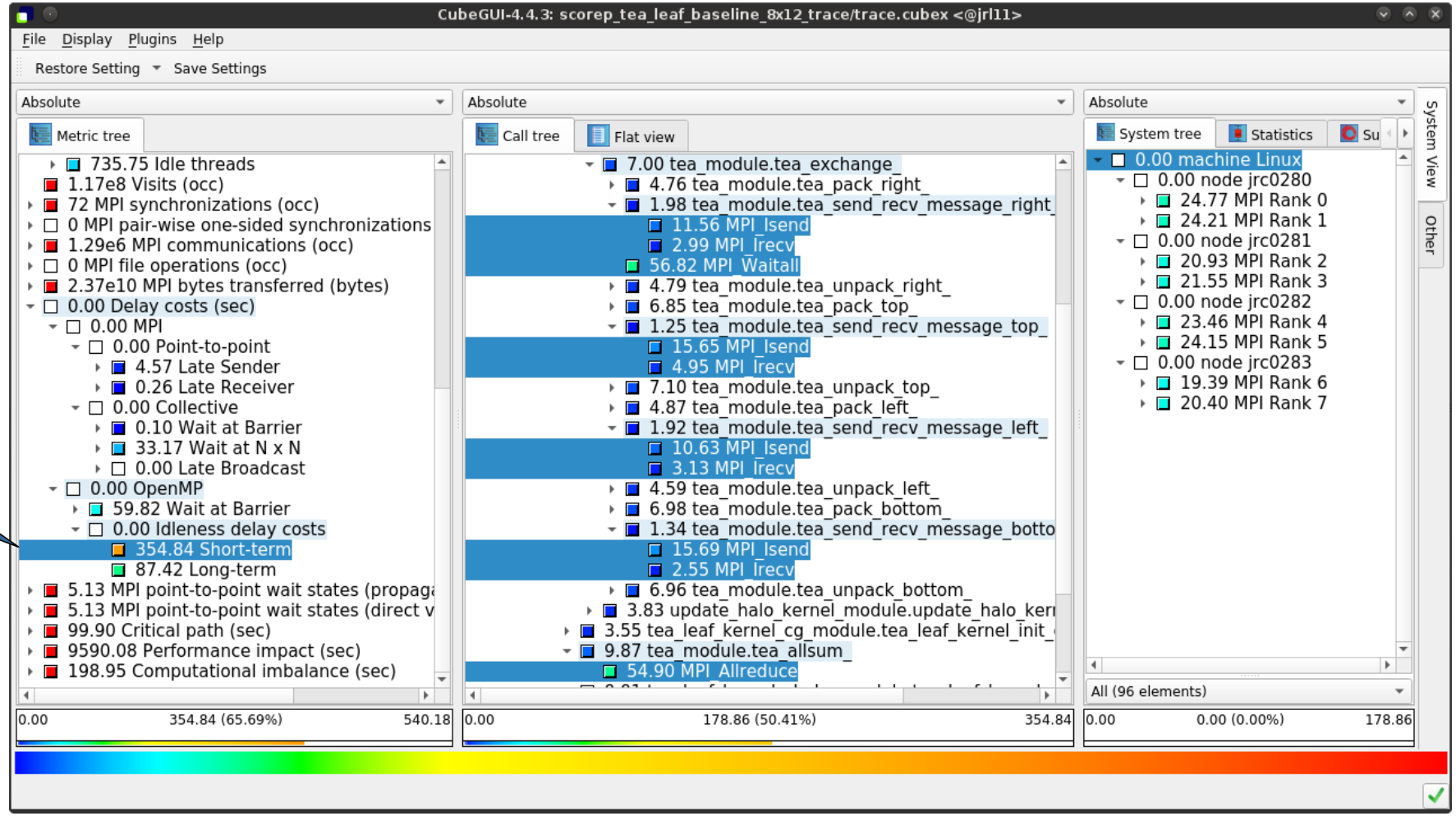
While MPI communication time and wait states are small (~0.6% of the total execution time)…



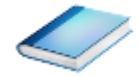
TeaLeaf Scalasca report analysis (II)



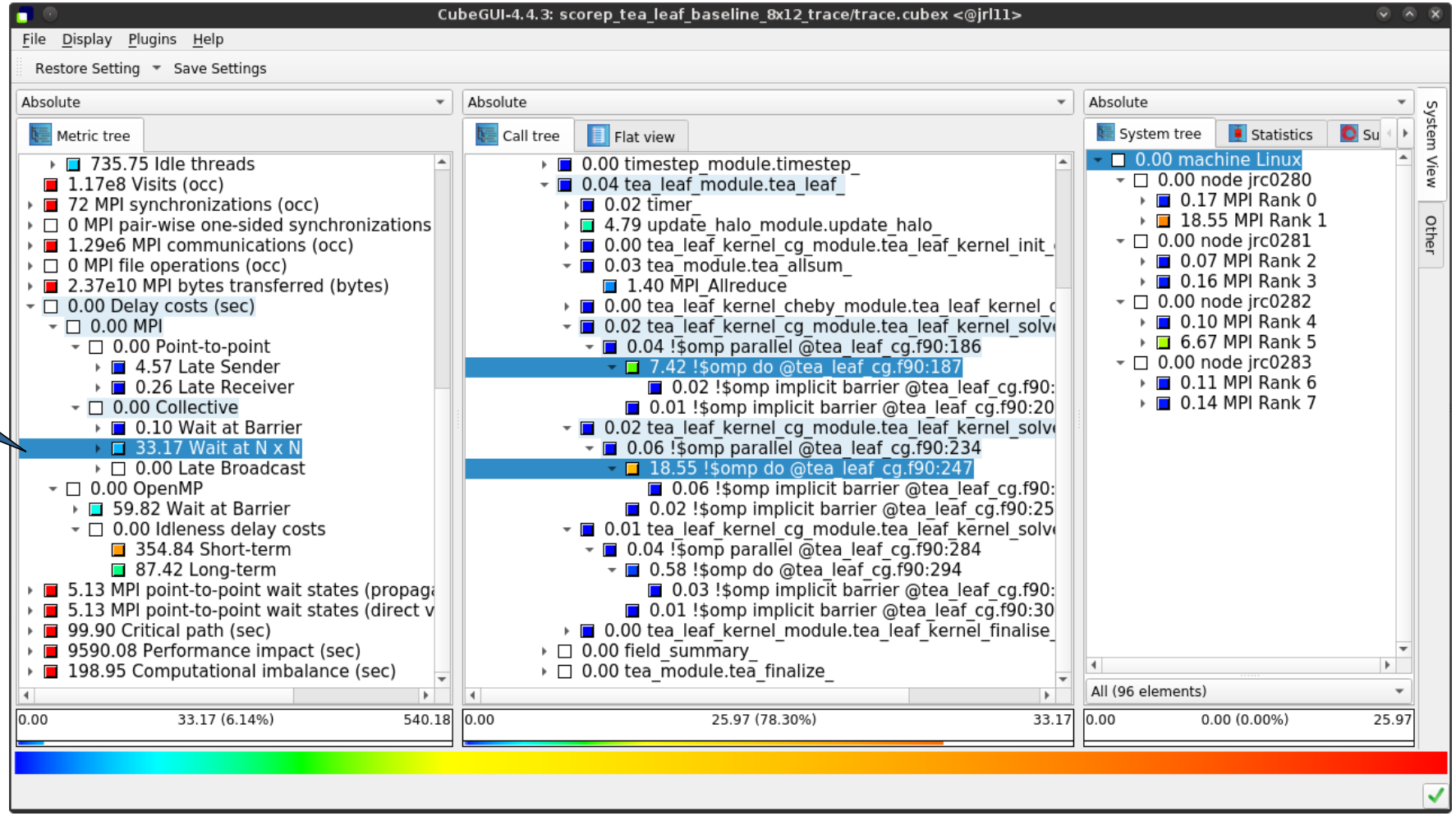
…they directly cause a significant amount of the OpenMP thread idleness



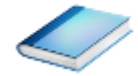
TeaLeaf Scalasca report analysis (III)



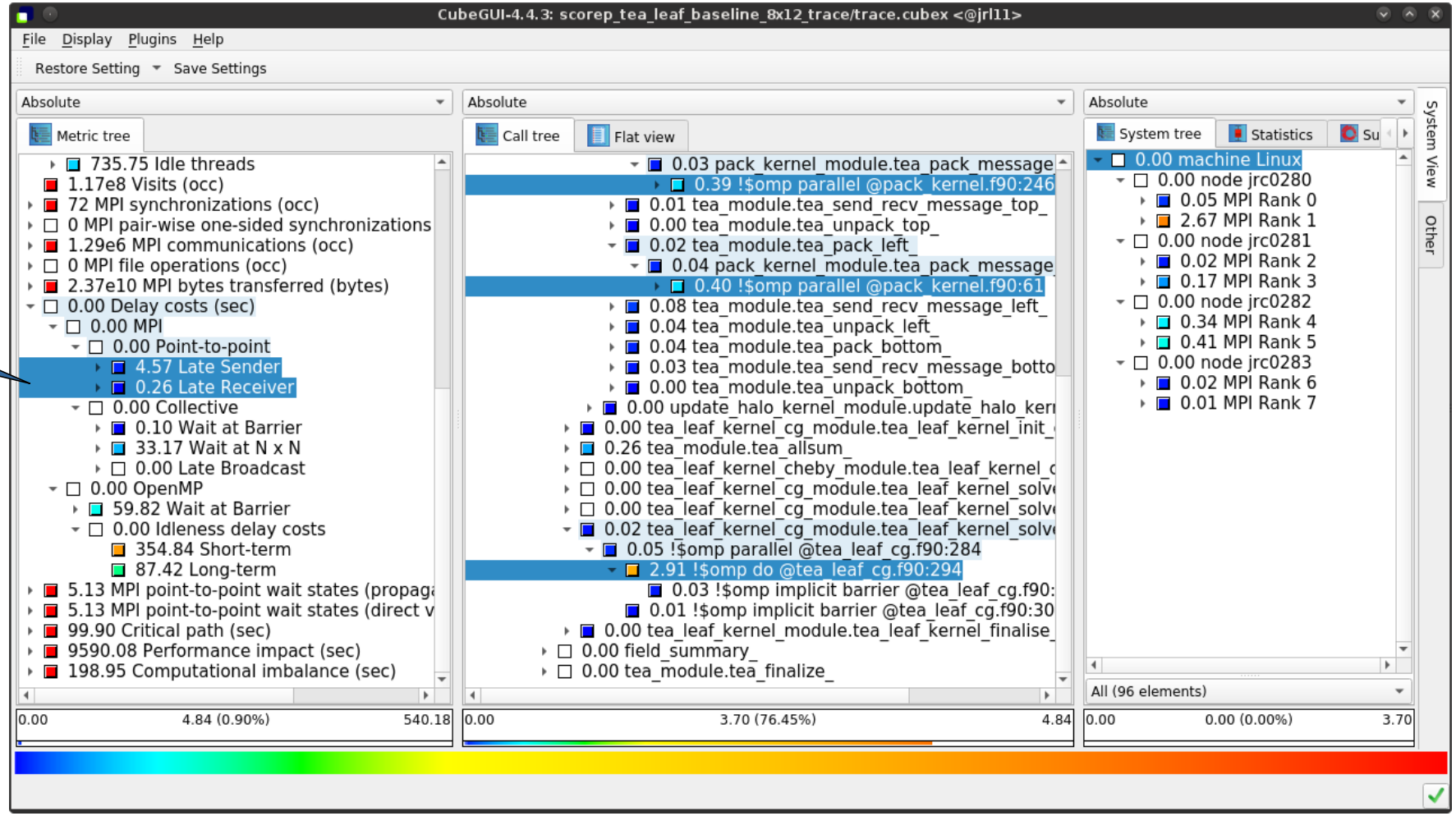
The “Wait at NxN” collective wait states are mostly caused by the first 2 OpenMP `do` loops of the solver (on ranks 5 & 1, resp.)…



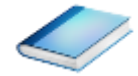
TeaLeaf Scalasca report analysis (IV)



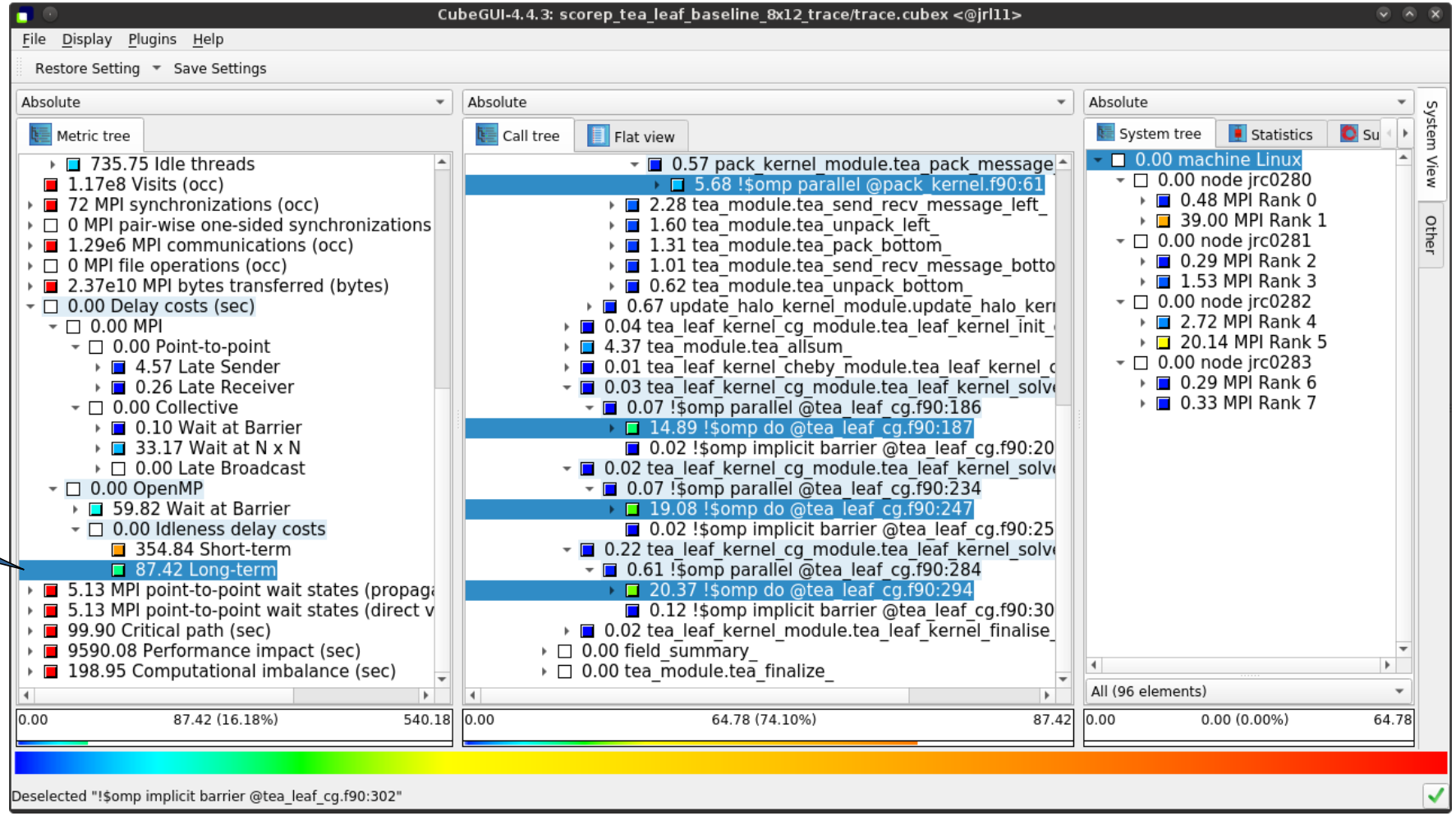
…while the MPI point-to-point wait states are caused by the 3rd solver `do` loop (on rank 1) and two loops in the halo exchange



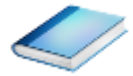
TeaLeaf Scalasca report analysis (V)



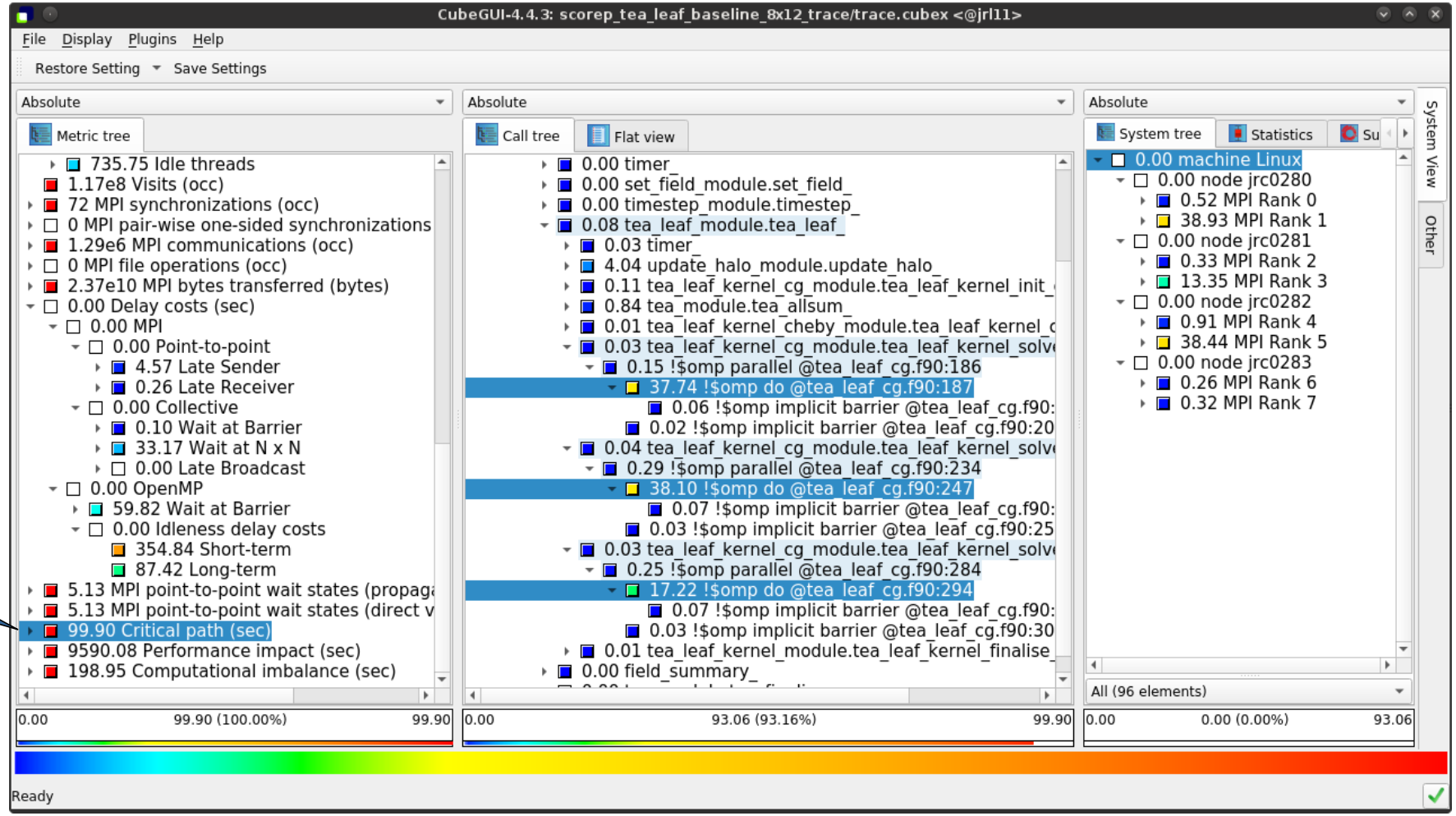
Various OpenMP `do` loops (incl. the solver loops) also cause OpenMP thread idleness on other ranks via propagation



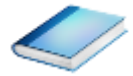
TeaLeaf Scalasca report analysis (VI)



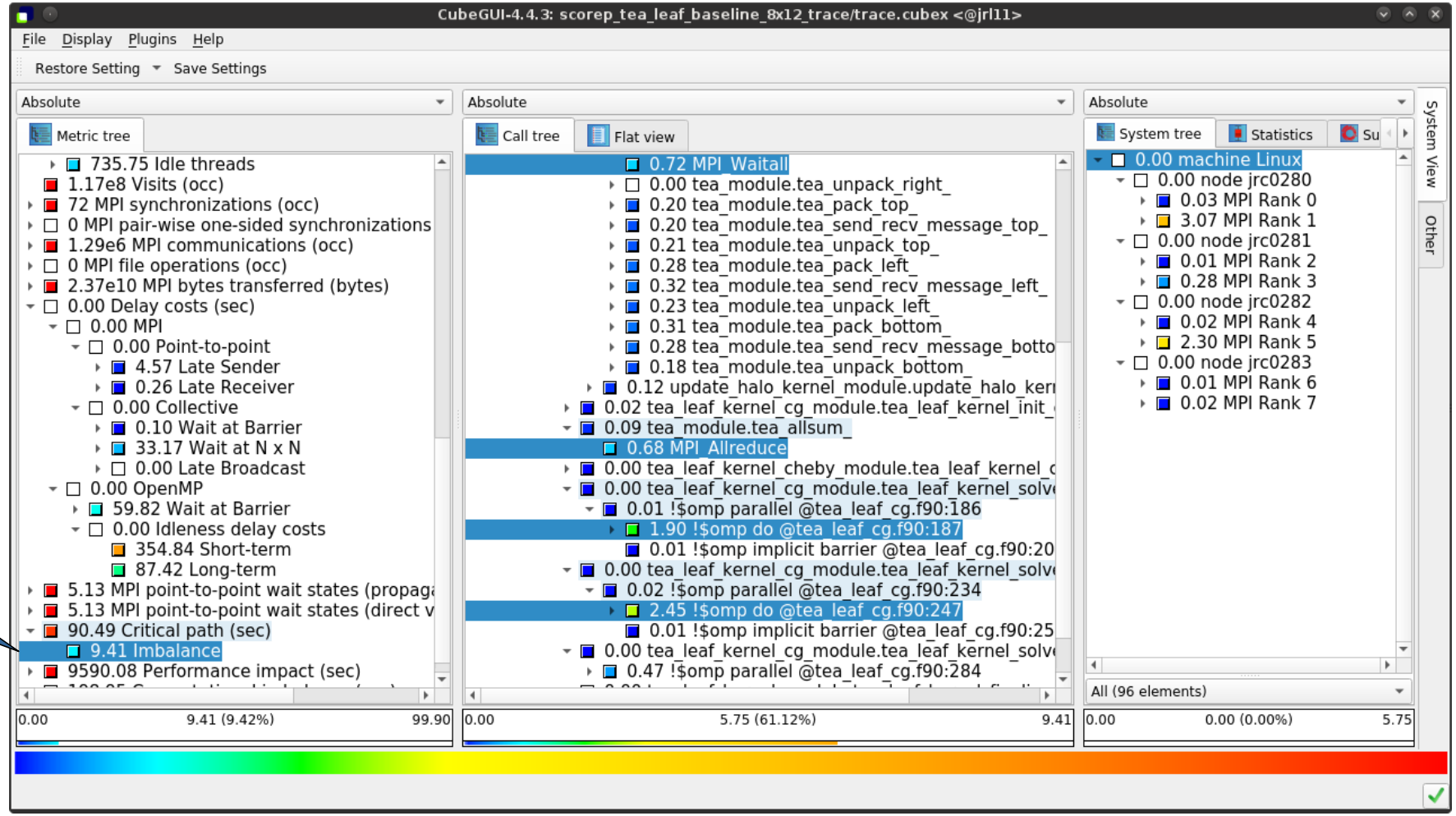
The Critical Path also highlights the three solver loops…



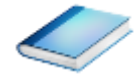
TeaLeaf Scalasca report analysis (VII)



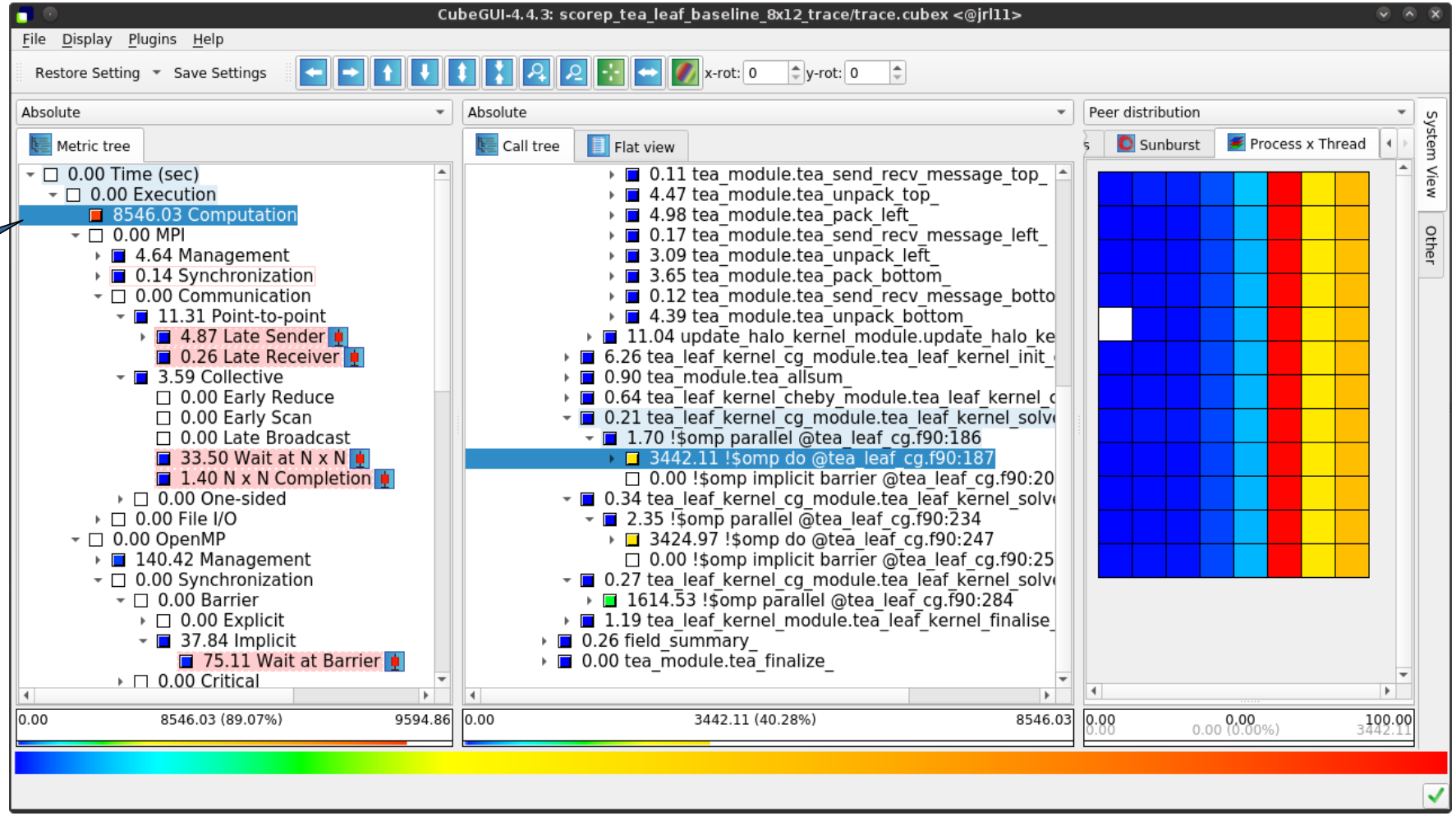
…with imbalance (time on critical path above average) mostly in the first two loops and MPI communication



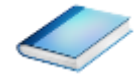
TeaLeaf Scalasca report analysis (VIII)



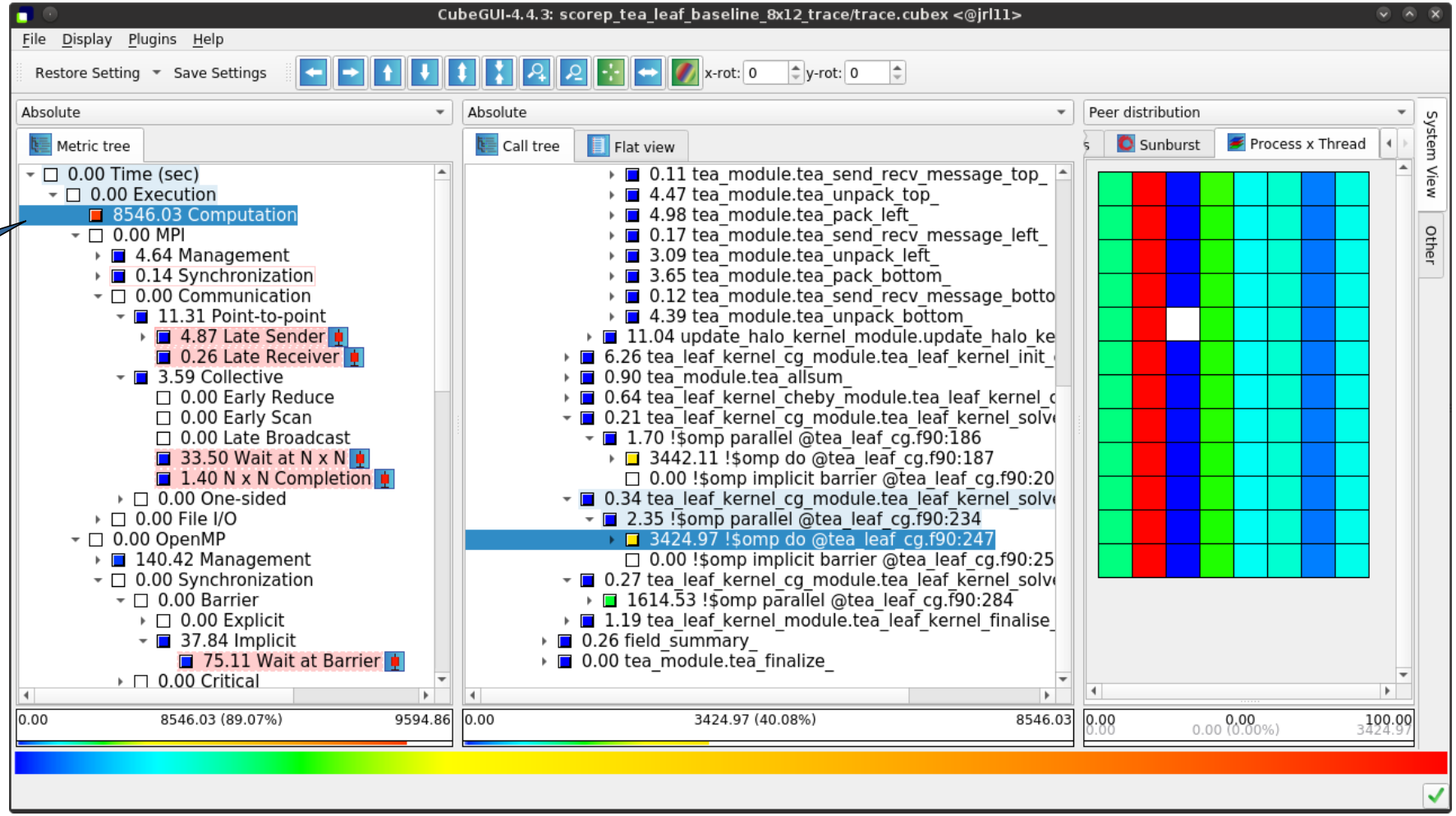
Computation time of 1st…



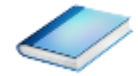
TeaLeaf Scalasca report analysis (IX)



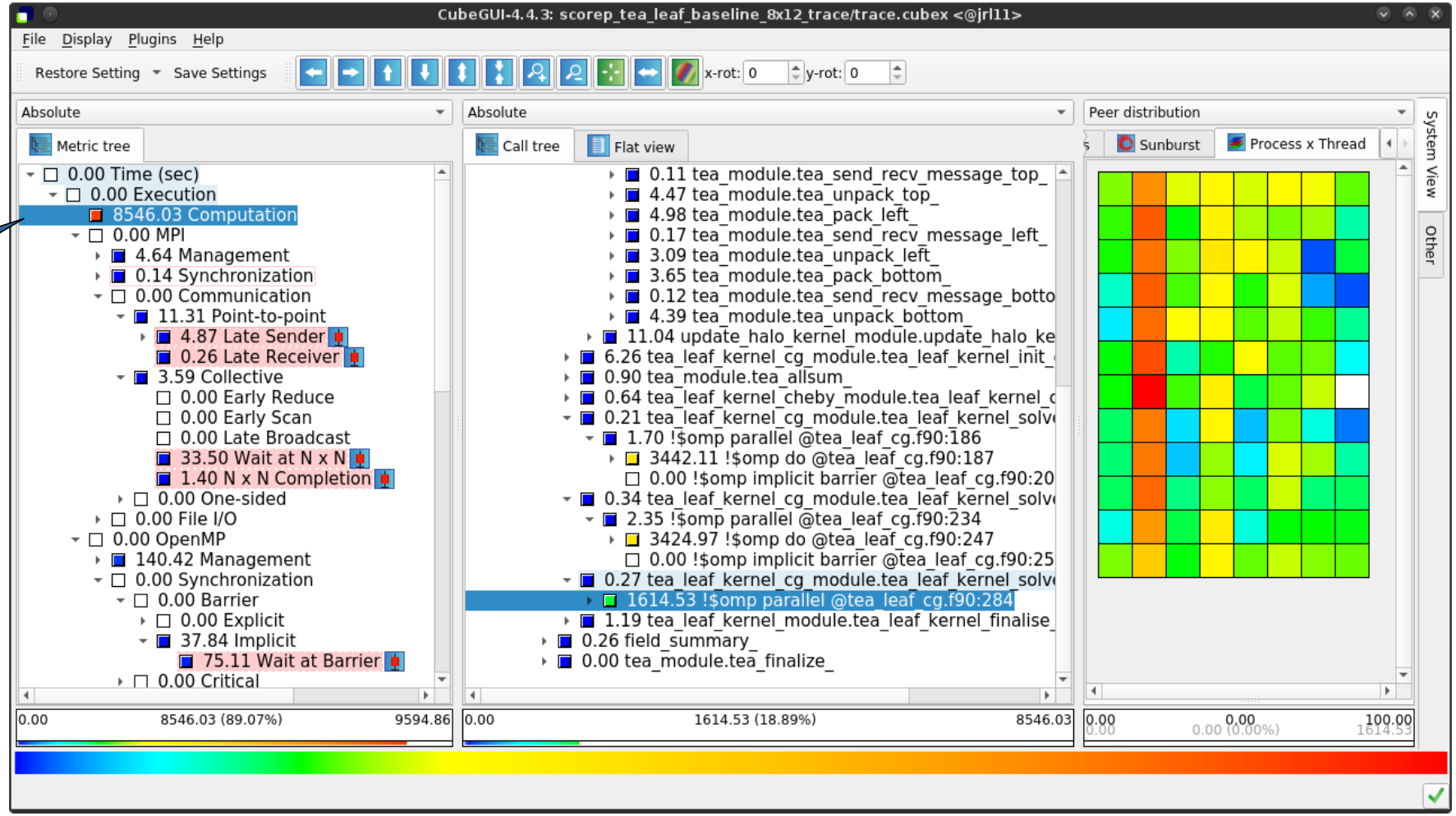
…and 2nd `do` loop mostly balanced within each rank, but vary considerably across ranks…



TeaLeaf Scalasca report analysis (X)



…while the 3rd `do` loop also shows imbalance within each rank



TeaLeaf analysis summary

- The first two OpenMP do loops of the solver are well balanced within a rank, but are imbalanced across ranks
  - ➔ Requires a global load balancing strategy
- The third OpenMP do loop, however, is imbalanced within ranks,
  - causing direct "Wait at OpenMP Barrier" wait states,
  - which cause indirect MPI point-to-point wait states,
  - which in turn cause OpenMP thread idleness
  - ➔ Low-hanging fruit

- Adding a `SCHEDULE(guided)` clause reduced
  - the MPI point-to-point wait states by ~66%
  - the MPI collective wait states by ~50%
  - the OpenMP "Wait at Barrier" wait states by ~55%
  - the OpenMP thread idleness by ~11%
  - ➔ **Overall runtime (wall-clock) reduction by ~5%**

Scalasca Trace Tools: Further information

- Collection of trace-based performance tools
  - Specifically designed for large-scale systems
  - Features an automatic trace analyzer providing wait-state, critical-path, and delay analysis
  - Supports MPI, OpenMP, POSIX threads, and hybrid MPI+OpenMP/Pthreads
- Available under 3-clause BSD open-source license

- Documentation & sources:
  - https://www.scalasca.org
- Contact:
  - mailto: scalasca@fz-juelich.de

trace tools scalasca

Acknowledgement

This tutorial is sponsored by the DEEP-SEA project.

www.deep-projects.eu
@DEEPprojects
@deep-projects

The DEEP Projects have received funding from the European Commission's FP7, H2020, and EuroHPC Programmes, under Grant Agreements n° 287530, 610476, 754304, and 955606.

The EuroHPC Joint Undertaking (JU) receives support from the European Union's Horizon 2020 research and innovation programme and Germany, France, Spain, Greece, Belgium, Sweden, United Kingdom, Switzerland.

Reference material

Scalasca command – One command for (almost) everything

```
% scalasca
Scalasca 2.6.1
Toolset for scalable performance analysis of large-scale parallel applications
usage: scalasca [OPTION]... ACTION <argument>...
    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

Options:
   -c, --show-config     show configuration summary and exit
   -h, --help            show this help and exit
   -n, --dry-run         show actions without taking them
       --quickref        show quick reference guide and exit
       --remap-specfile  show path to remapper specification file and exit
   -v, --verbose         enable verbose commentary
   -V, --version         show version information and exit
```

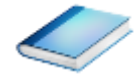
- The ‘`scalasca -instrument`’ command is deprecated and will be remove in the next major release
  ⇨ use Score-P instrumenter directly

Scalasca convenience command: scan / scalasca -analyze

```
% scan
Scalasca 2.6.1: 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.
  -R #runs         : Specify the number of measurement runs per config.
  -M cfgfile       : Specify a config file for a multi-run measurement.
  -P preset        : Specify a preset for a multi-run measurement, e.g., 'pop'.
  -L               : List available multi-run presets.
  -D cfgfile       : Check a multi-run config file for validity and dump
                   : the processed configuration for comparison.
```

- Scalasca measurement collection & analysis nexus

Automatic measurement configuration



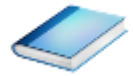
- scan configures Score-P measurement by automatically setting some environment variables and exporting them
  - E.g., experiment title, profiling/tracing mode, filter file, …
  - Precedence order:
    - Command-line arguments
    - Environment variables already set
    - Automatically determined values
- 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)

Scalasca convenience command: square / scalasca -examine

```
% square
Scalasca 2.6.1: analysis report explorer
usage: square [OPTIONS] <experiment archive | cube file>
  -C <none | quick | full> : Level of sanity checks for newly created reports
  -c <number>              : Consider number of counters when doing scoring (-s)
  -F                       : Force remapping of already existing reports
  -f filtfile              : Use specified filter file when doing scoring (-s)
  -s                       : Skip display and output textual score report
  -v                       : Enable verbose mode
  -n                       : Do not include idle thread metric
  -S <mean | merge>        : Aggregation method for summarization results of
                             each configuration (default: merge)
  -T <mean | merge>        : Aggregation method for trace analysis results of
                             each configuration (default: merge)
  -A                       : Post-process every step of a multi-run experiment
  -I                       : Ignore structural sanity checks and force aggregation
                             of measurements in a multi-run experiment
  -x <scorep-score opt>    : Pass option(s) to scorep-score
```

- Scalasca analysis report explorer (Cube)

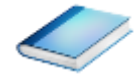
Scalasca advanced command: scout - Scalasca automatic trace analyzer



```
% scout.hyb --help
SCOUT   (Scalasca 2.6.1)
Copyright (c) 1998-2022 Forschungszentrum Juelich GmbH
Copyright (c) 2014-2021 RWTH Aachen University
Copyright (c) 2009-2014 German Research School for Simulation Sciences GmbH

Usage: <launchcmd> scout.hyb [OPTION]... <ANCHORFILE | EPIK DIRECTORY>
Options:
  --statistics       Enables instance tracking and statistics [default]
  --no-statistics    Disables instance tracking and statistics
  --critical-path    Enables critical-path analysis [default]
  --no-critical-path Disables critical-path analysis
  --rootcause        Enables root-cause analysis [default]
  --no-rootcause     Disables root-cause analysis
  --single-pass      Single-pass forward analysis only
  --time-correct     Enables enhanced timestamp correction
  --no-time-correct  Disables enhanced timestamp correction [default]
  --verbose, -v      Increase verbosity
  --help             Display this information and exit
```

- Provided in serial (.ser), OpenMP (.omp), MPI (.mpi) and MPI+OpenMP (.hyb) variants

Scalasca advanced command: clc_synchronize

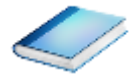


- Scalasca trace event timestamp consistency correction

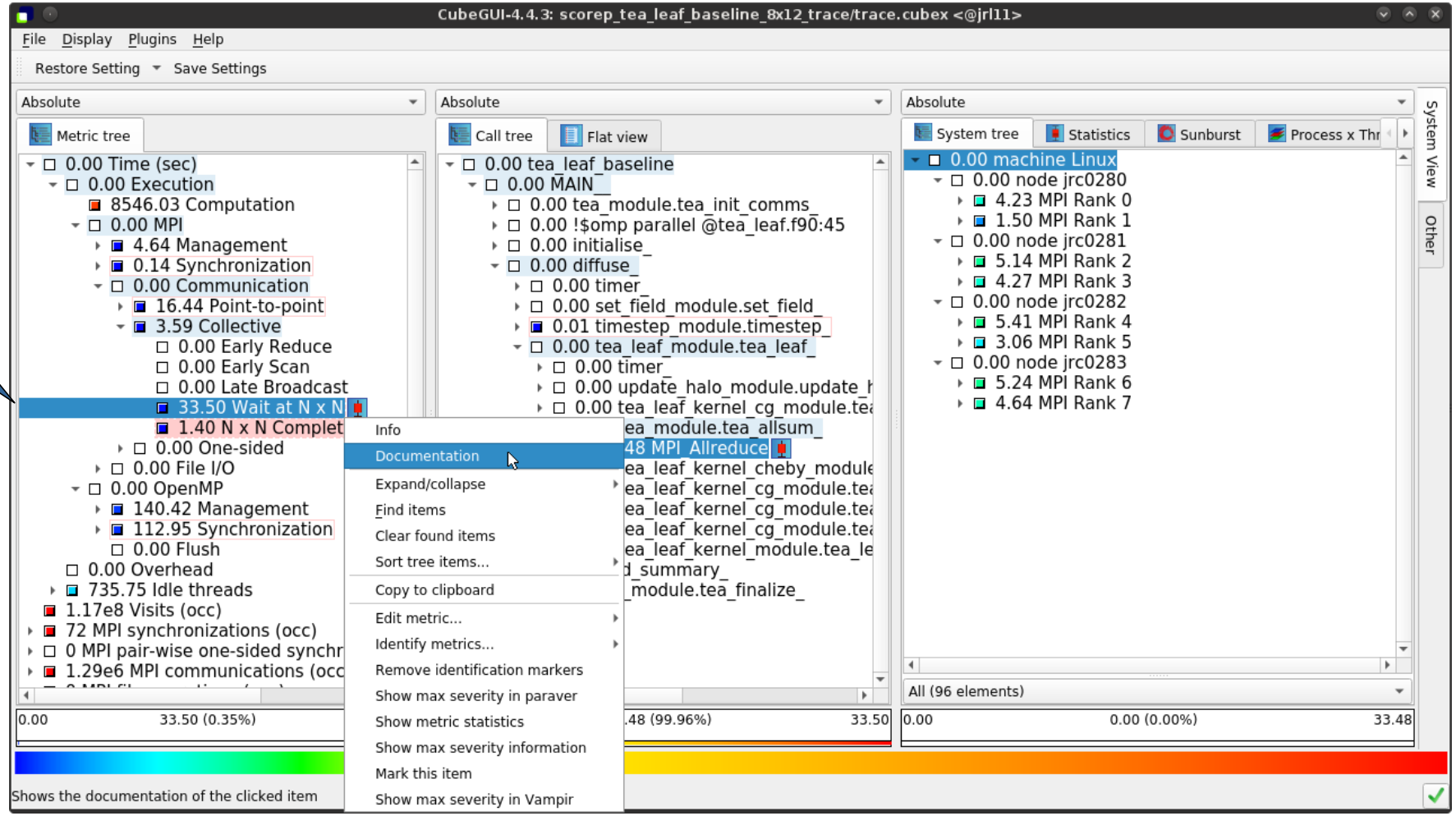
```
Usage: <launchcmd> clc_synchronize.hyb <ANCHORFILE | EPIK_DIRECTORY>
```

  - Provided in MPI (.mpi) and MPI+OpenMP (.hyb) variants
  - Takes as input a trace experiment archive where the events may have timestamp inconsistencies
    - E.g., multi-node measurements on systems without adequately synchronized clocks on each compute node
  - Generates a new experiment archive (always called ./clc_sync) containing a trace with event timestamp inconsistencies resolved
    - E.g., suitable for detailed examination with a time-line visualizer

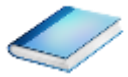
Online metric description



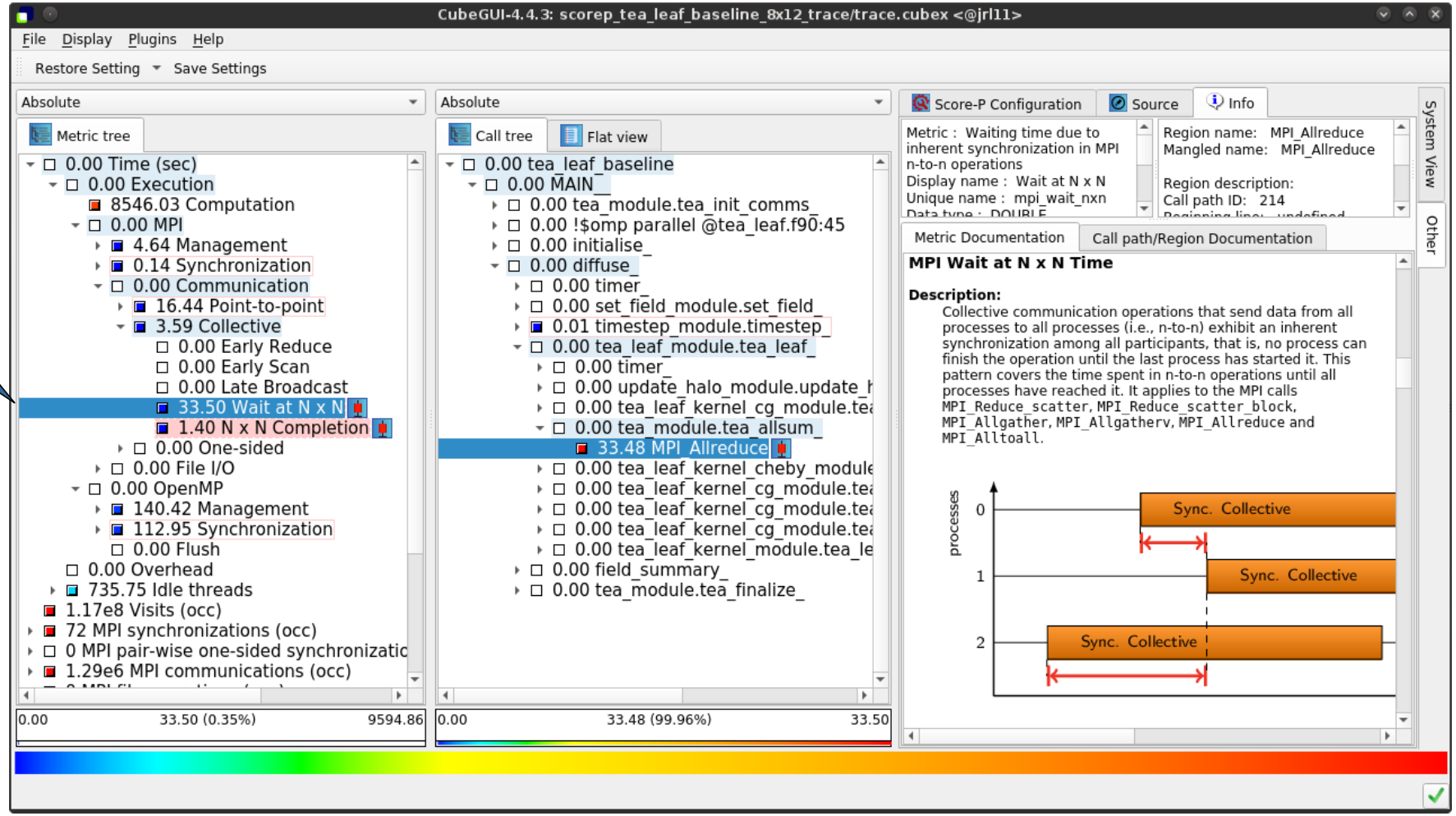
Access online metric description via context menu (right-click)



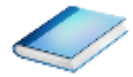
Online metric description (cont.)



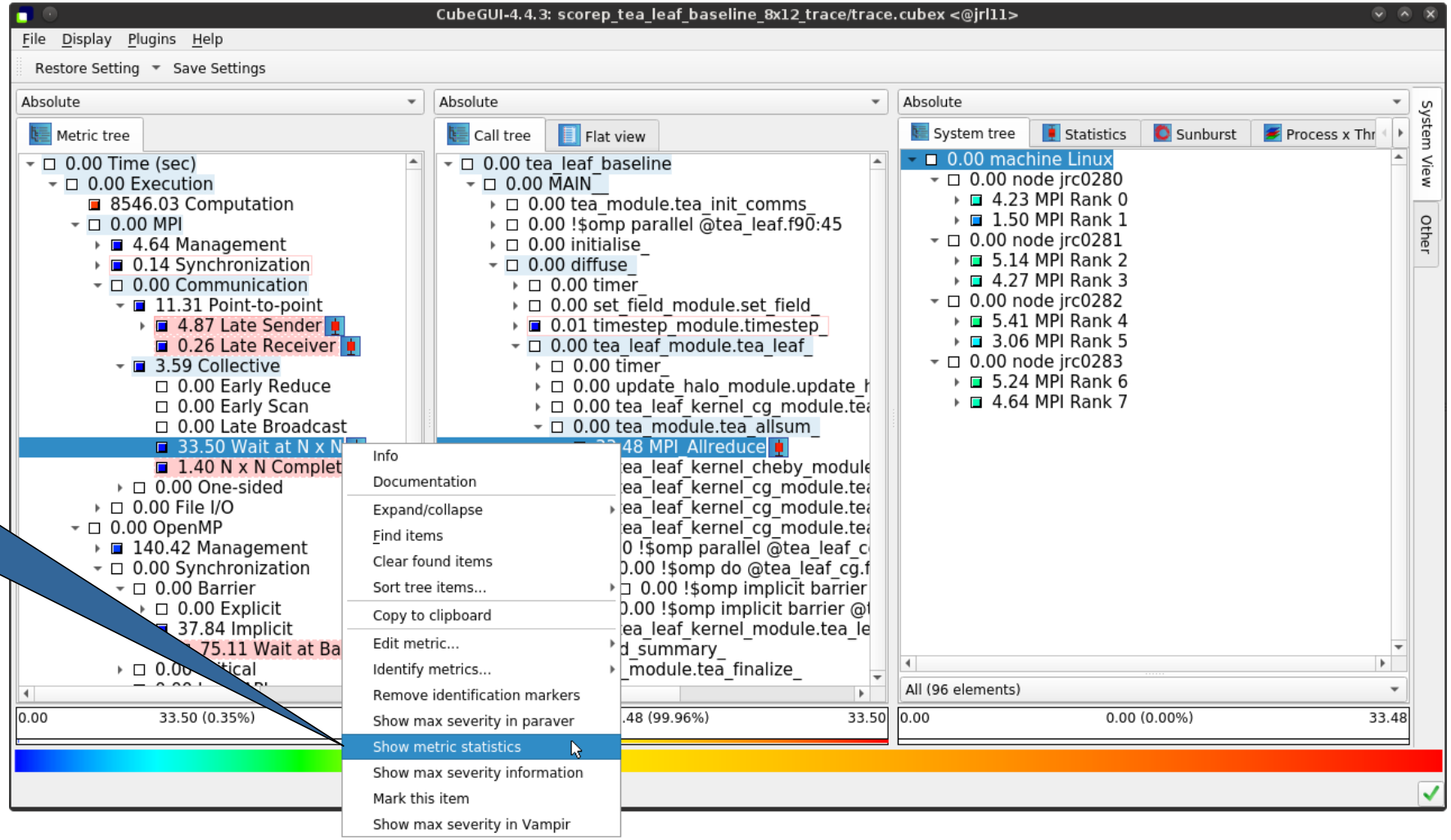
Selection of different metric automatically updates description



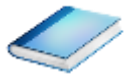
Metric statistics



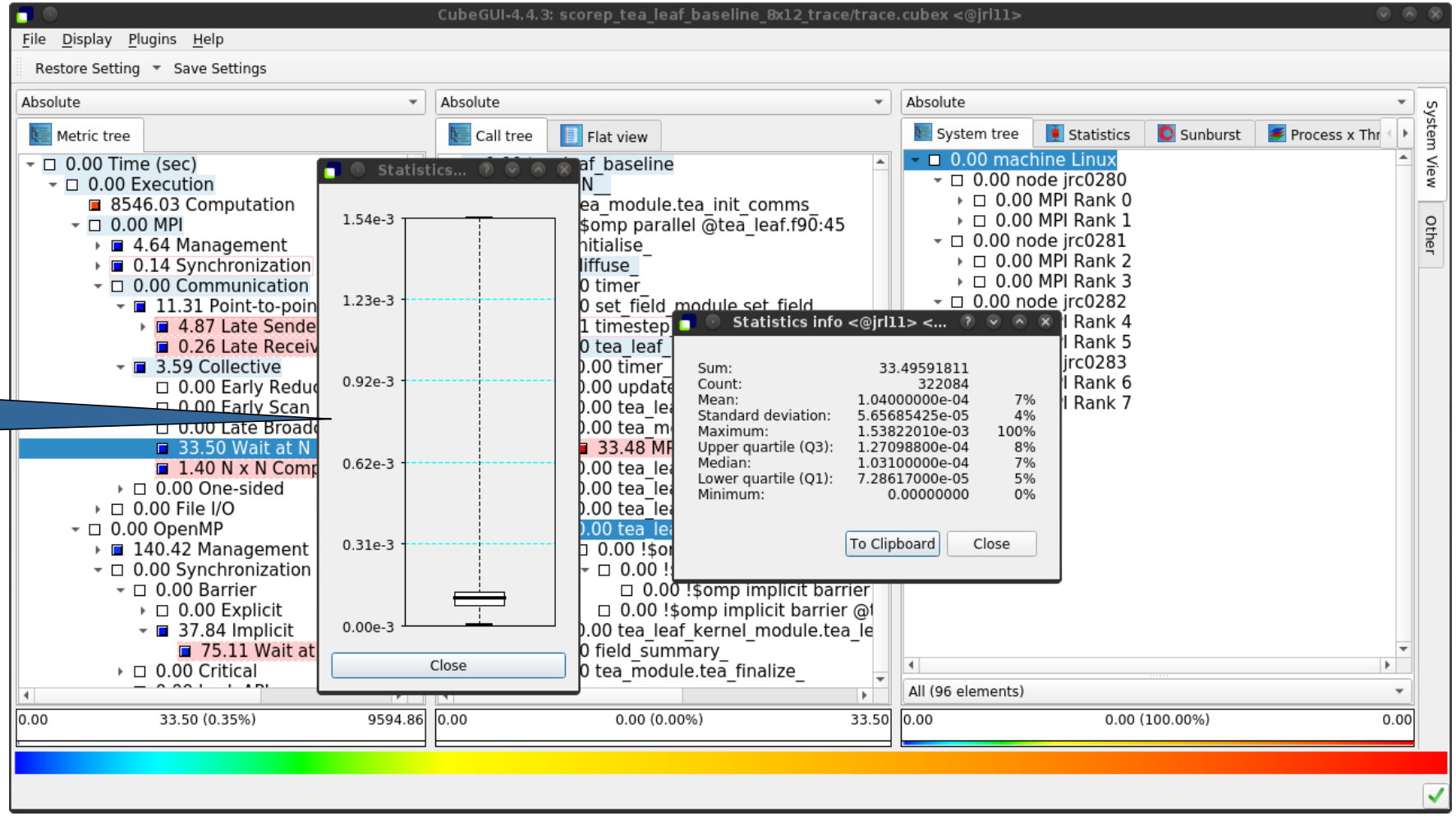
Access metric statistics for metrics marked with box plot icon from context menu



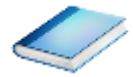
Metric statistics (cont.)



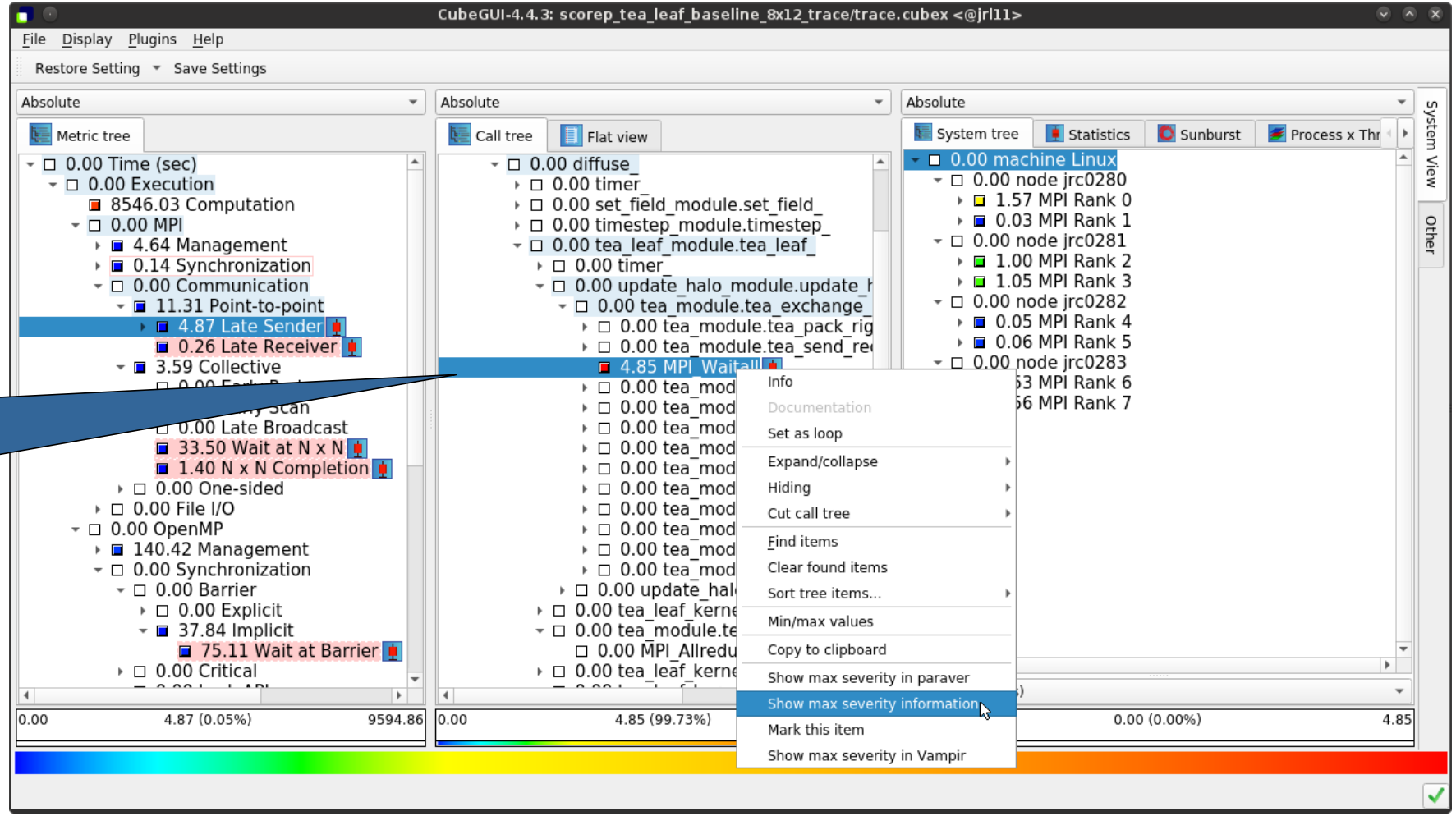
Shows instance statistics box plot, click to get details



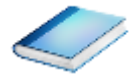
Metric instance statistics



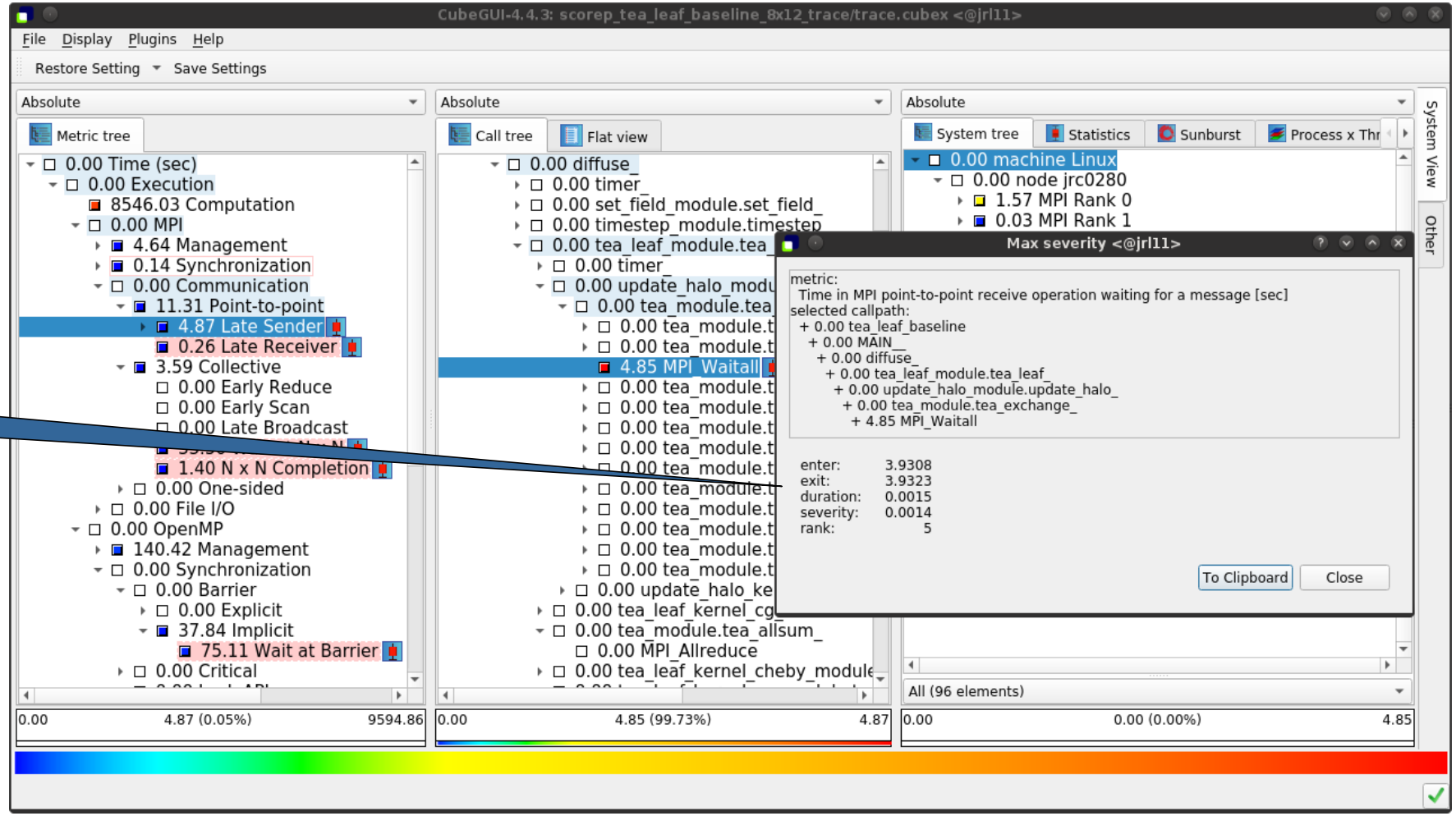
Access most-severe instance information for call paths marked with box plot icon via context menu



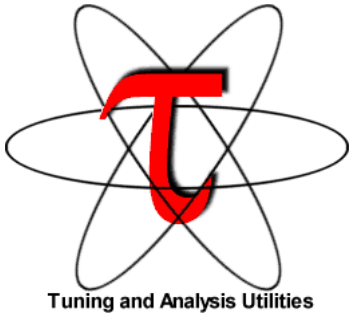
Metric instance statistics (cont.)



Shows instance details



Performance Data Management with TAU PerfExplorer

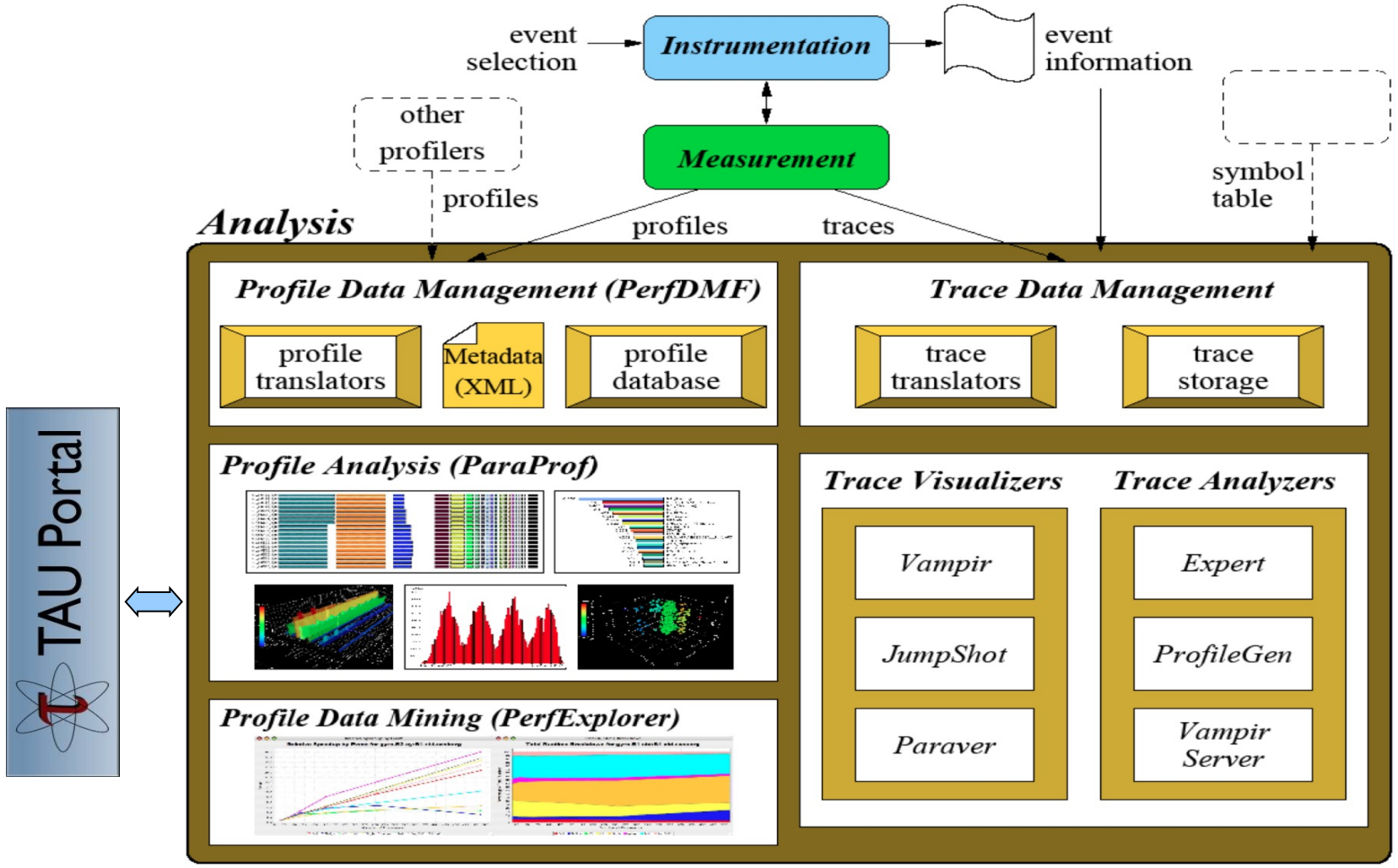


Sameer Shende
sameer@cs.uoregon.edu
University of Oregon

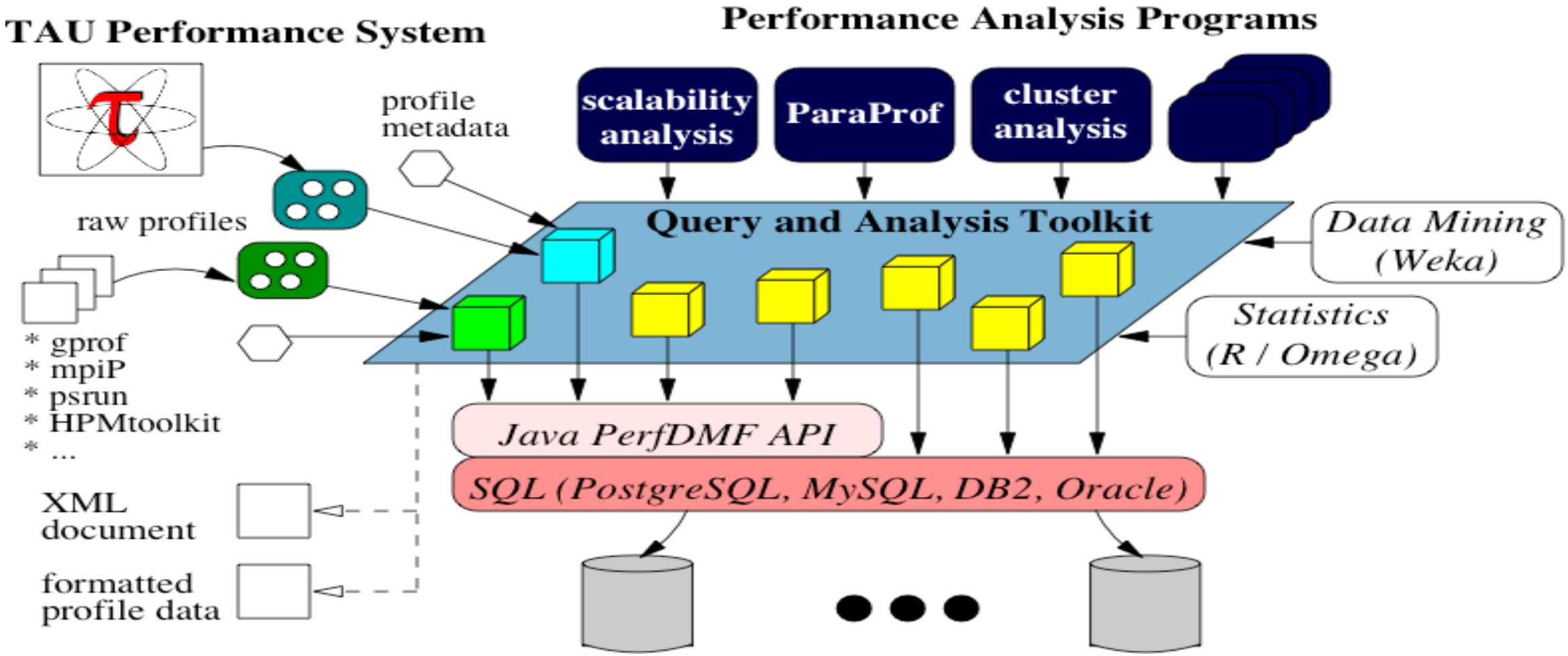
http://tau.uoregon.edu/TAU_PerfExplorer_SC23.pdf

TAU's Analysis Tools: PerfExplorer

TAU Analysis



TAUdb: Performance Data Management Framework



Using TAUdb

- Configure TAUdb (Done by each user)

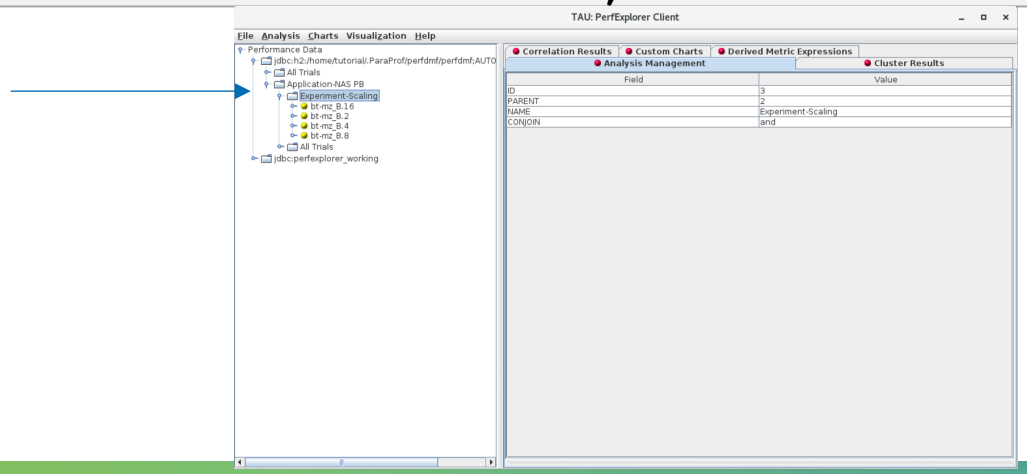
  % taudb_configure --create-default
  - Choose derby, PostgreSQL, MySQL, Oracle or DB2
  - Hostname
  - Username
  - Password
  - Say yes to downloading required drivers (we are not allowed to distribute these)
  - Stores parameters in your ~/.ParaProf/taudb.cfg file
- Configure PerfExplorer (Done by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

  % perfexplorer_configure
- Execute PerfExplorer

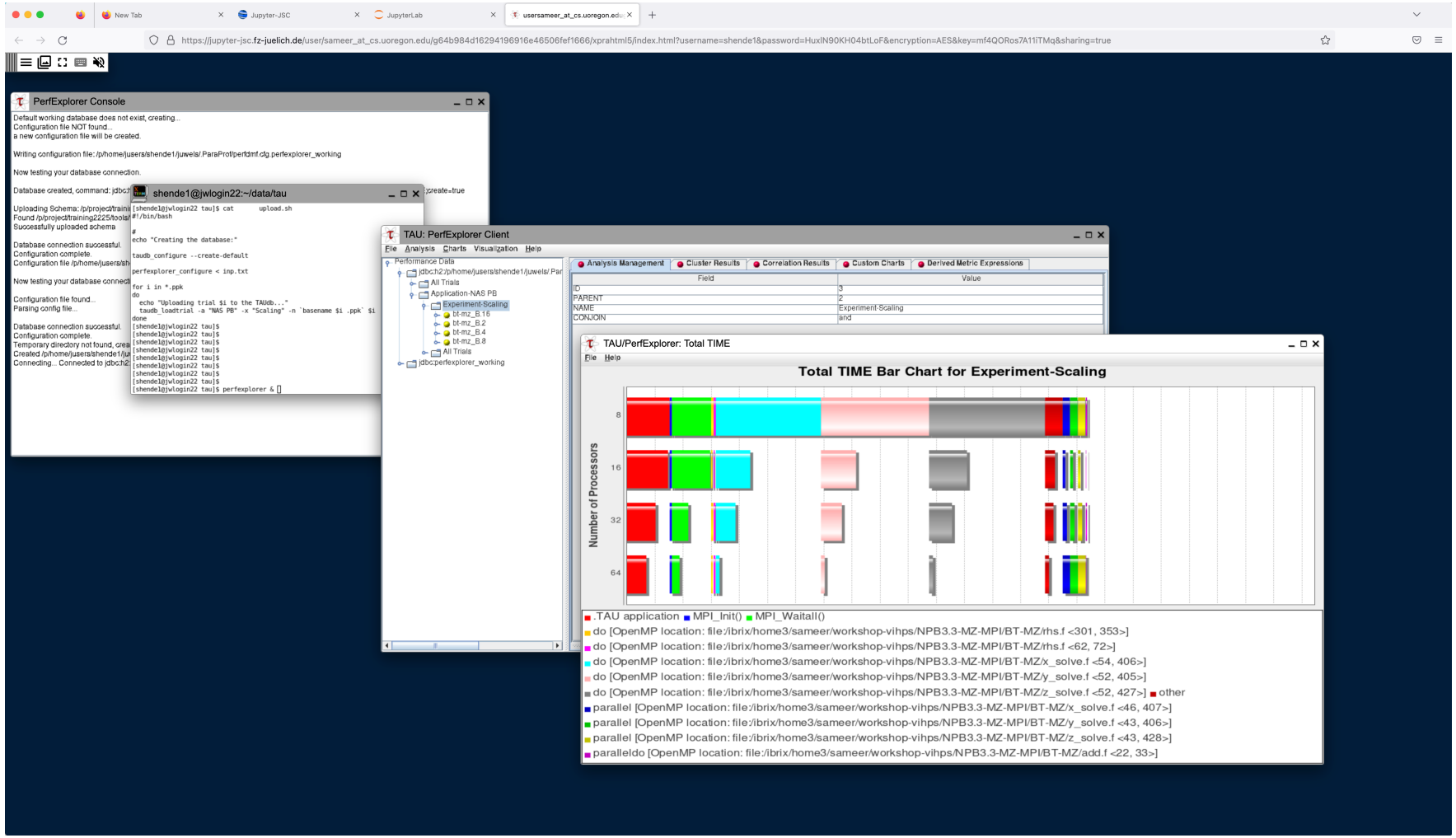
  % perfexplorer

PerfExplorer Hands-on

```
% source /p/project/training2341/setup.sh
% wget http://tau.uoregon.edu/data.tgz
% tar xf data.tgz; cd data/tau
% cat ./upload.sh
% ./upload.sh
% perfexplorer

# Click and expand first database jdbc:/home/tutorial/.ParaProf/perfdmf.cfg
# Click and expand Application-NAS PB and Experiment-Scaling
# Select by clicking Experiment-Scaling
# Go to Charts -> Total Execution Time, then select other charts
```



PerfExplorer Hands-on



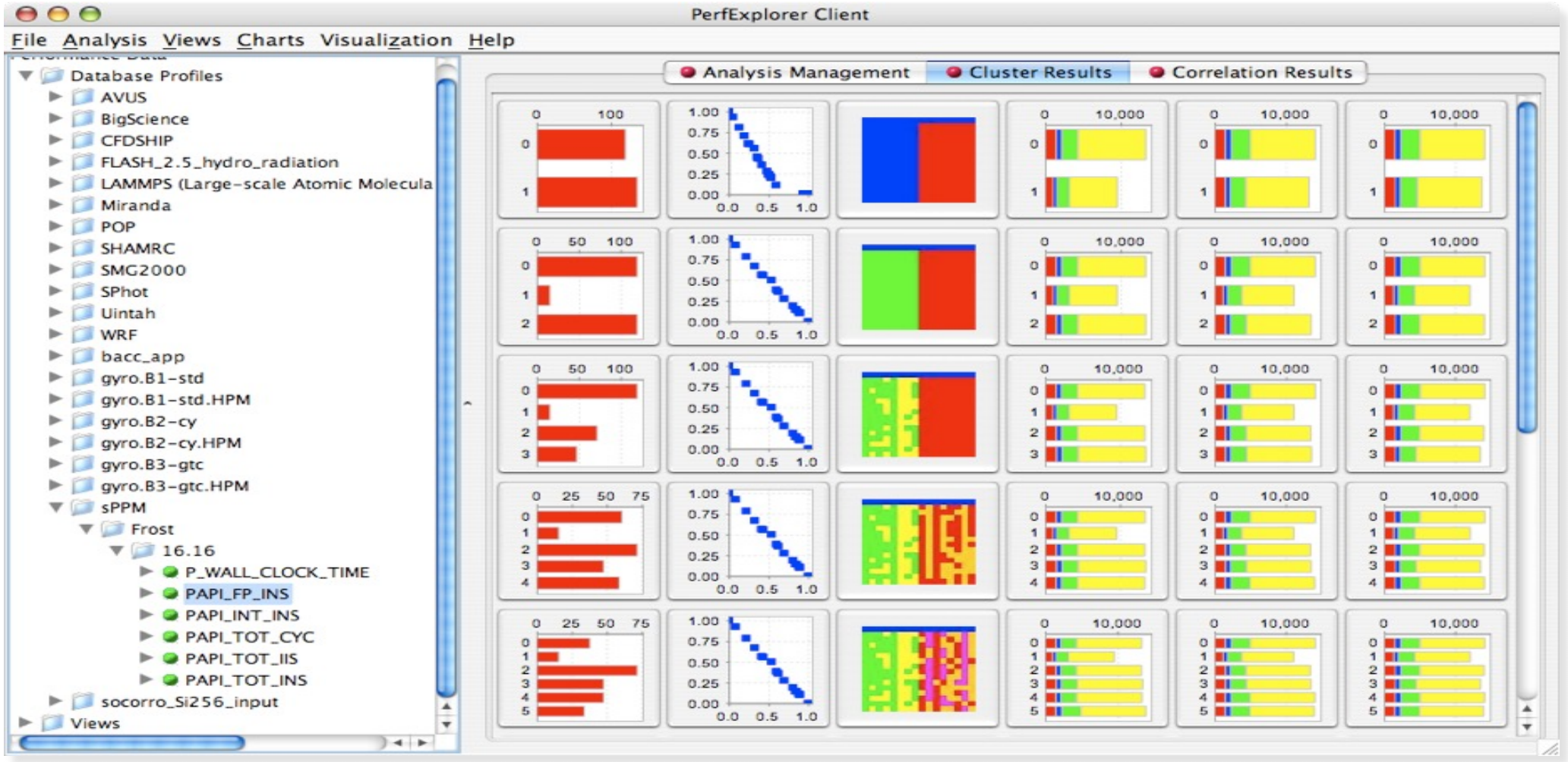
Performance Data Mining (PerfExplorer)

- 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: Using Cluster Analysis

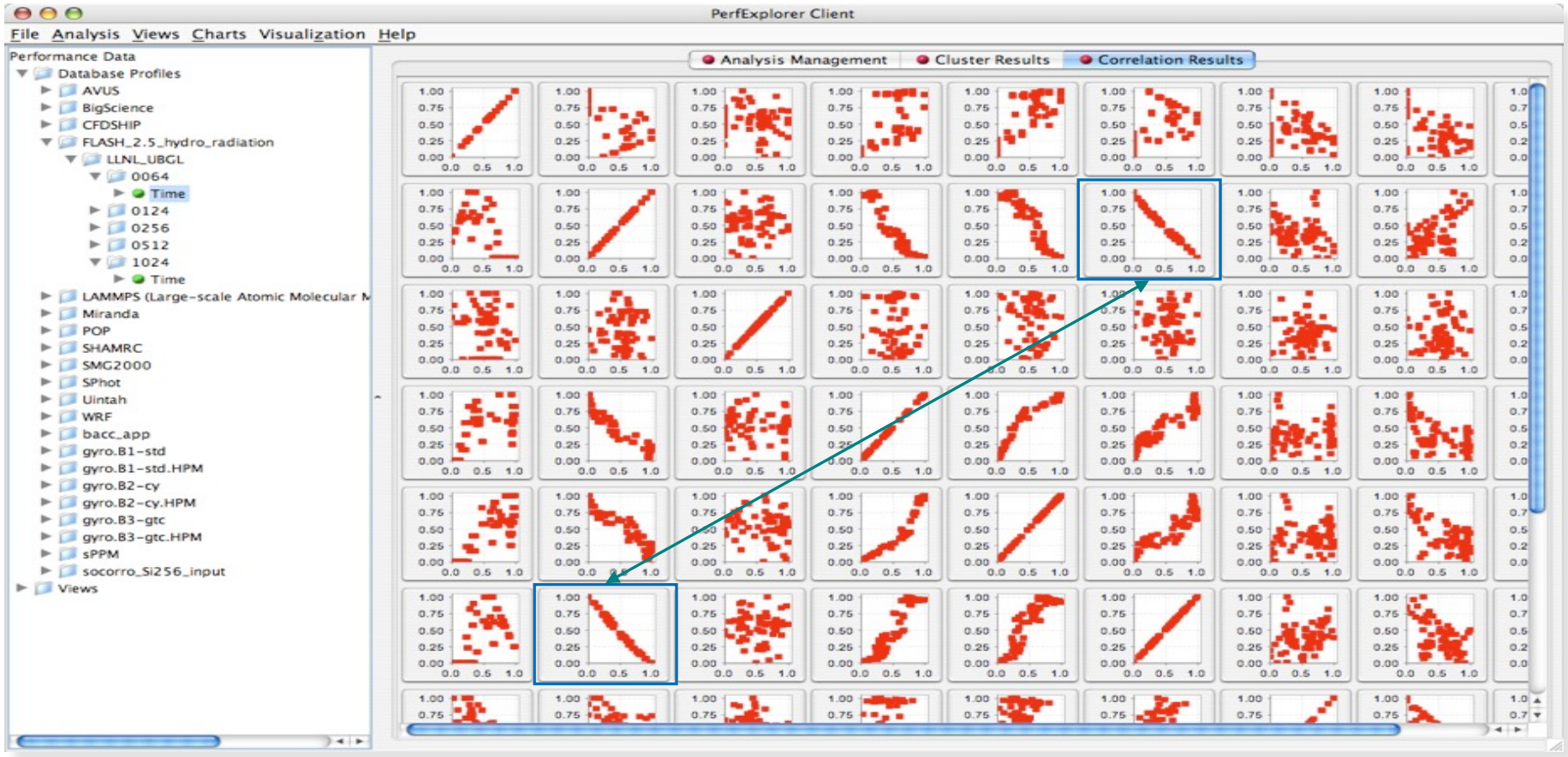
- Performance data represented as vectors - each dimension is the cumulative time for an event
- *k*-means: *k* random centers are selected and instances are grouped with the "closest" (Euclidean) center
- New centers are calculated and the process repeated until stabilization or max iterations
- Dimension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

PerfExplorer - Cluster Analysis (sPPM)



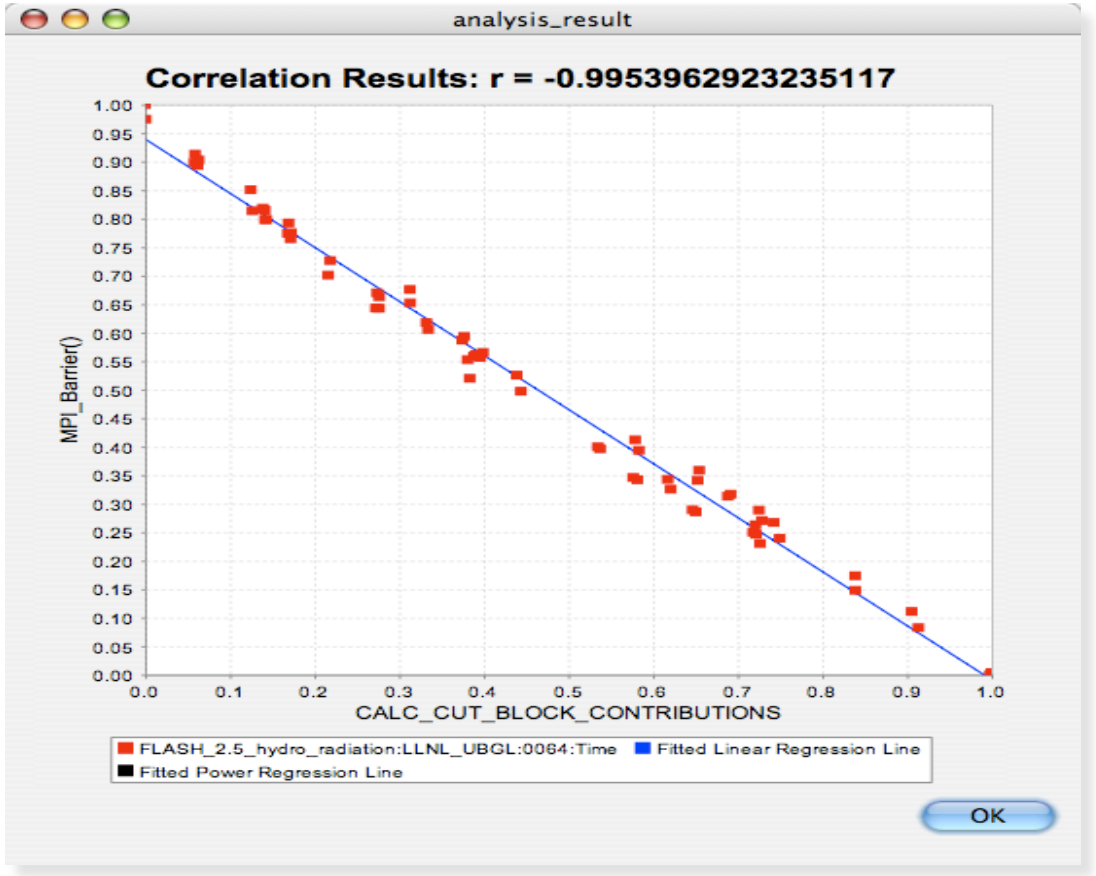
PerfExplorer - Correlation Analysis (Flash)

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



PerfExplorer - Correlation Analysis (Flash)

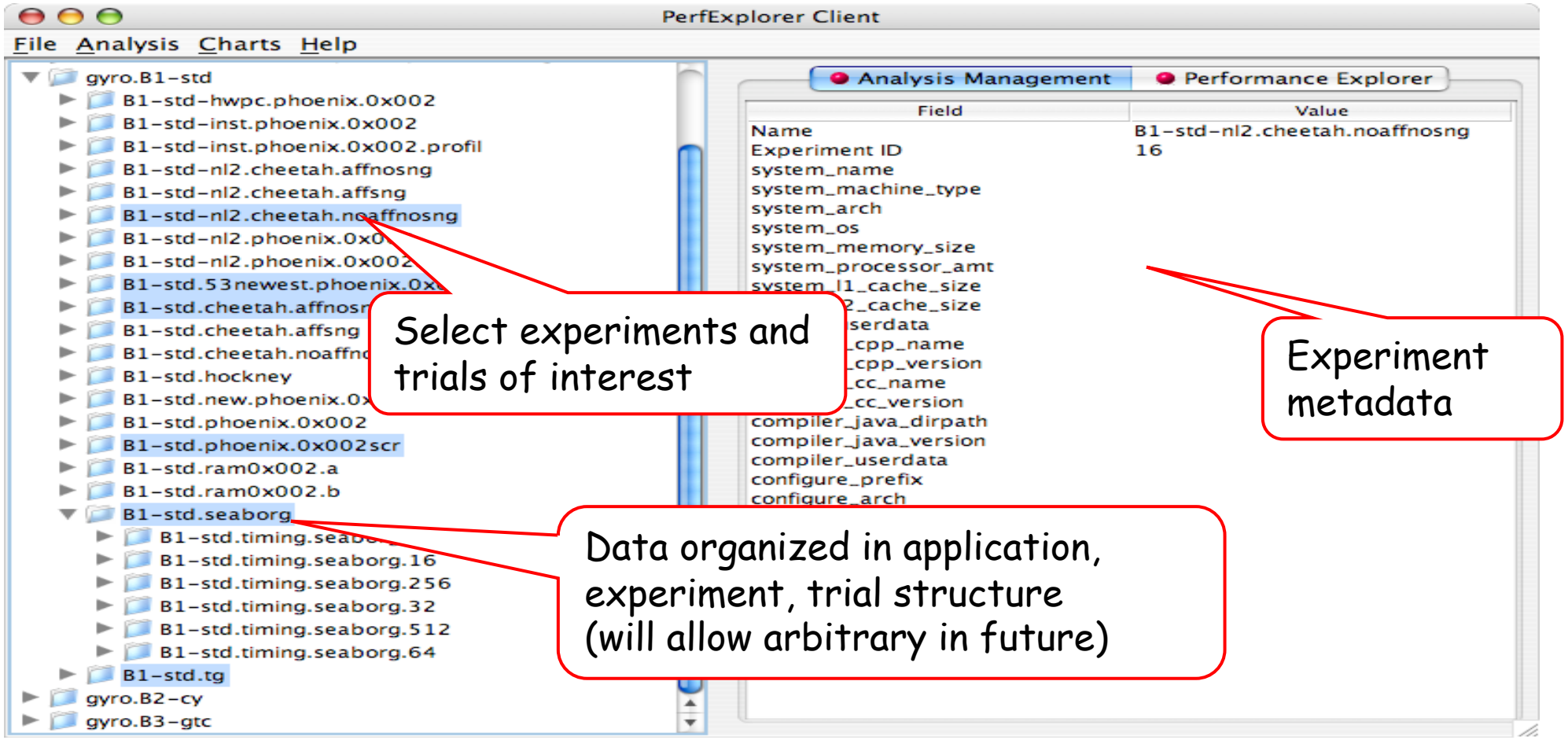
- -0.995 indicates strong, negative relationship
- As CALC_CUT_BLOCK_CONTRIBUTIONS() increases in execution time, MPI_Barrier() decreases



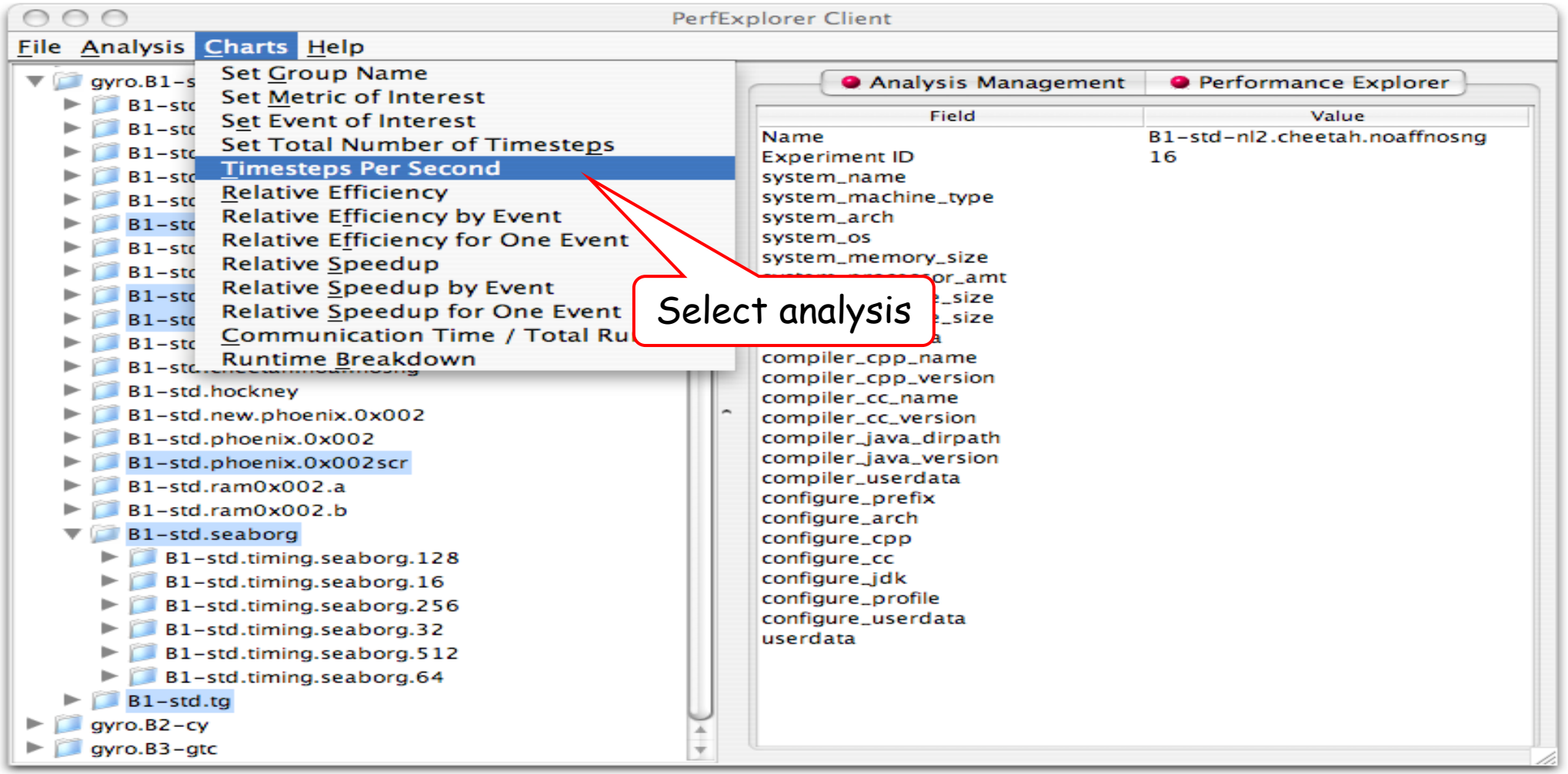
PerfExplorer - Comparative Analysis

- 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

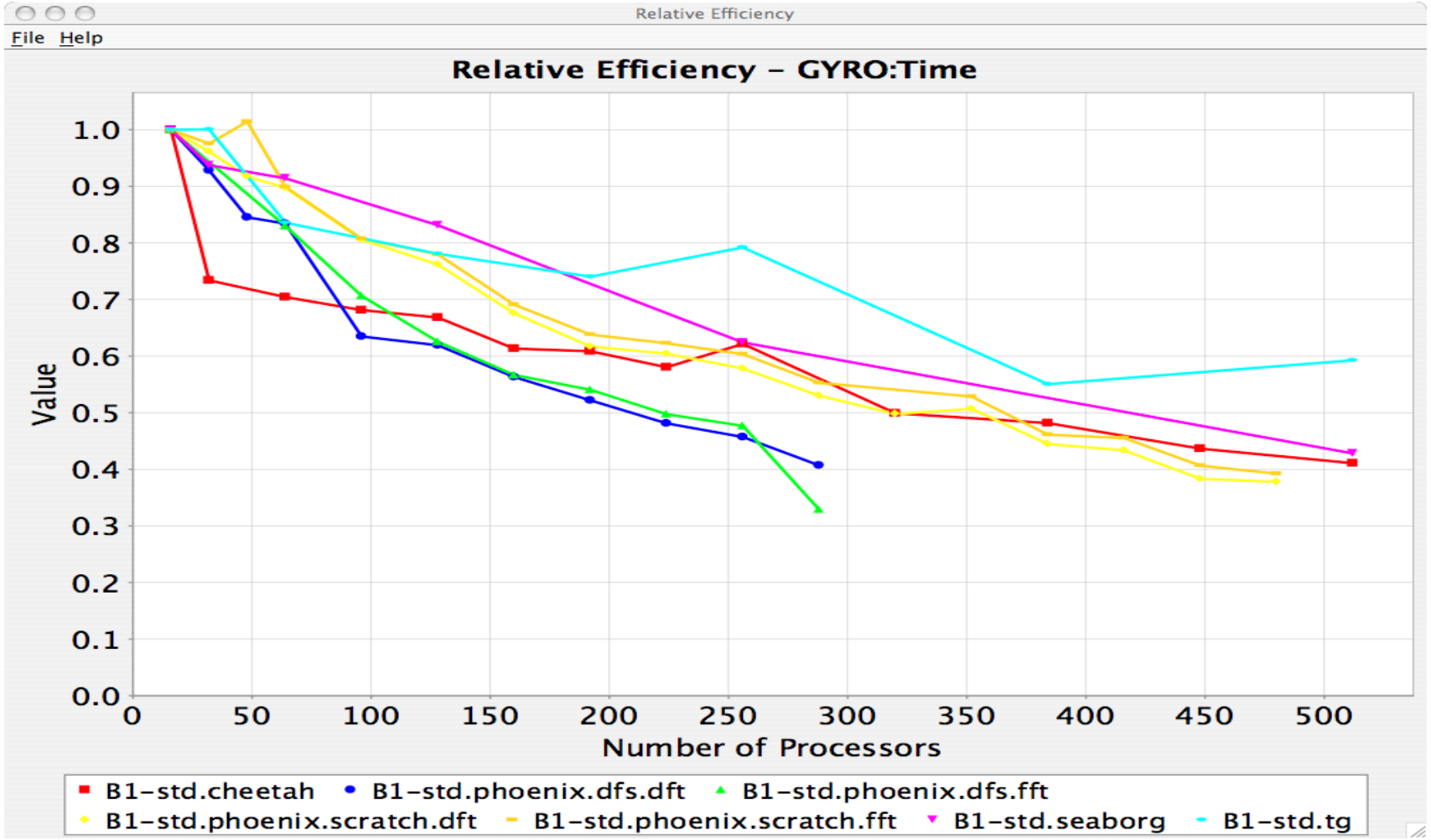
PerfExplorer - Interface



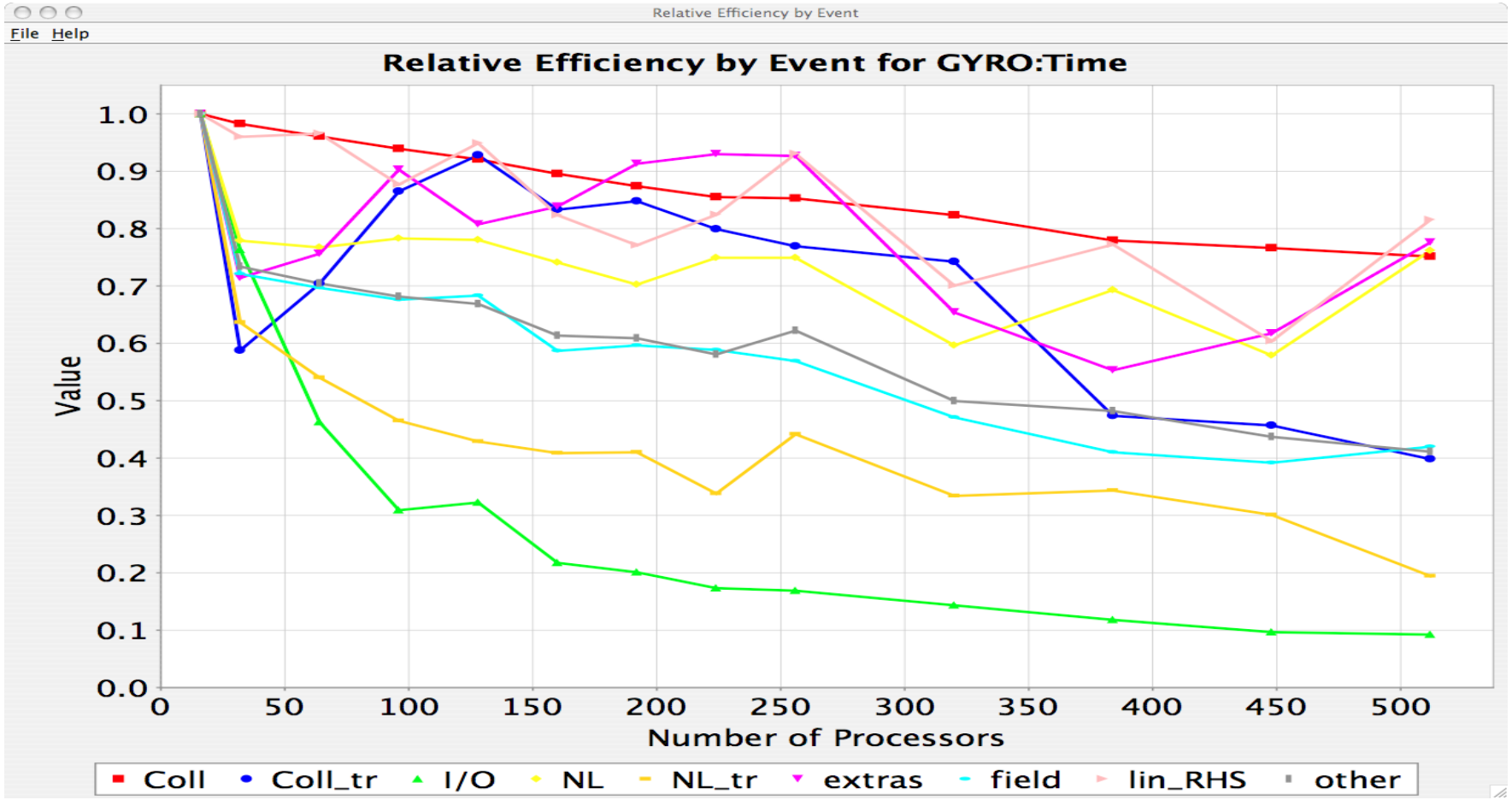
PerfExplorer - Interface



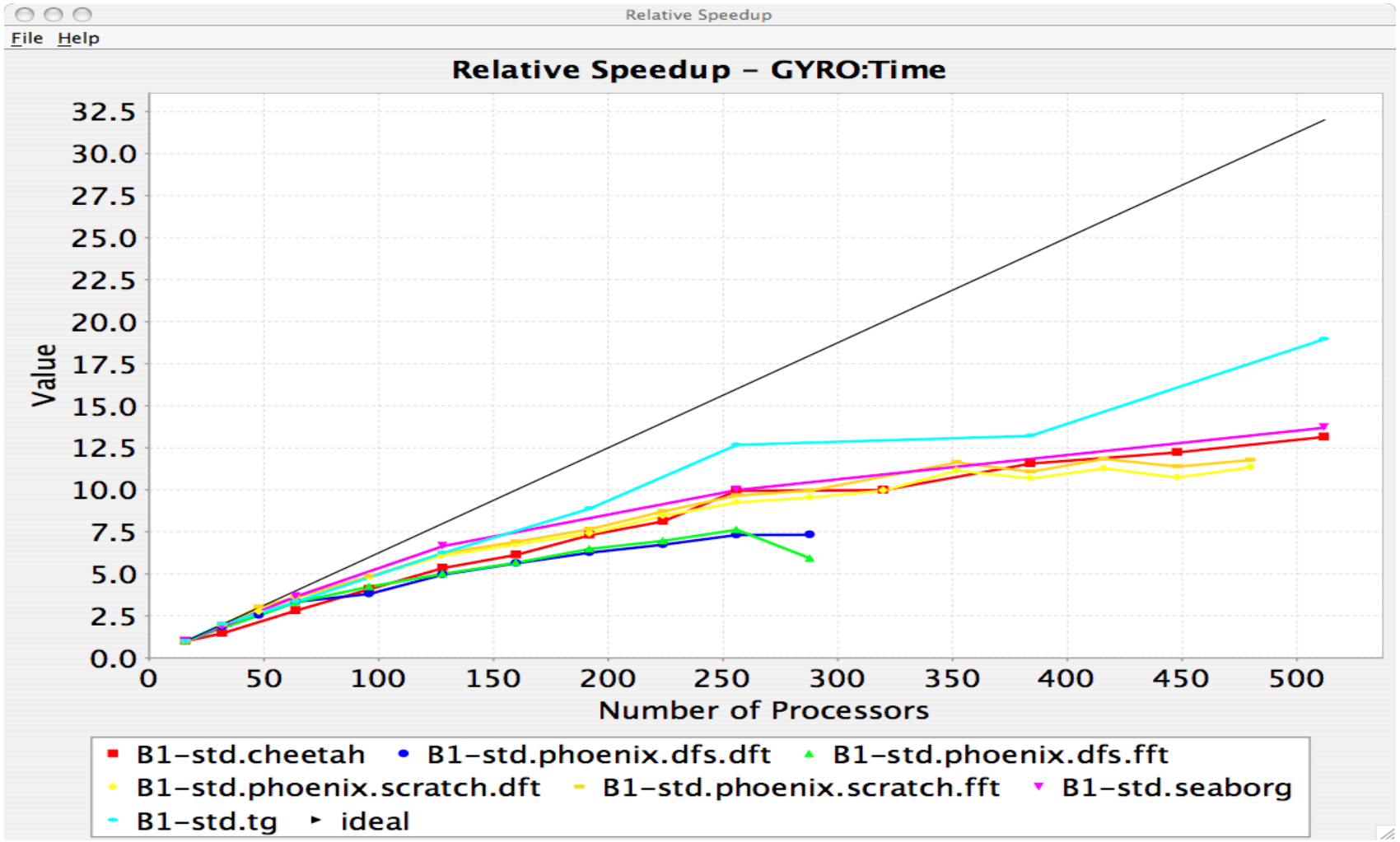
PerfExplorer - Relative Efficiency Plots



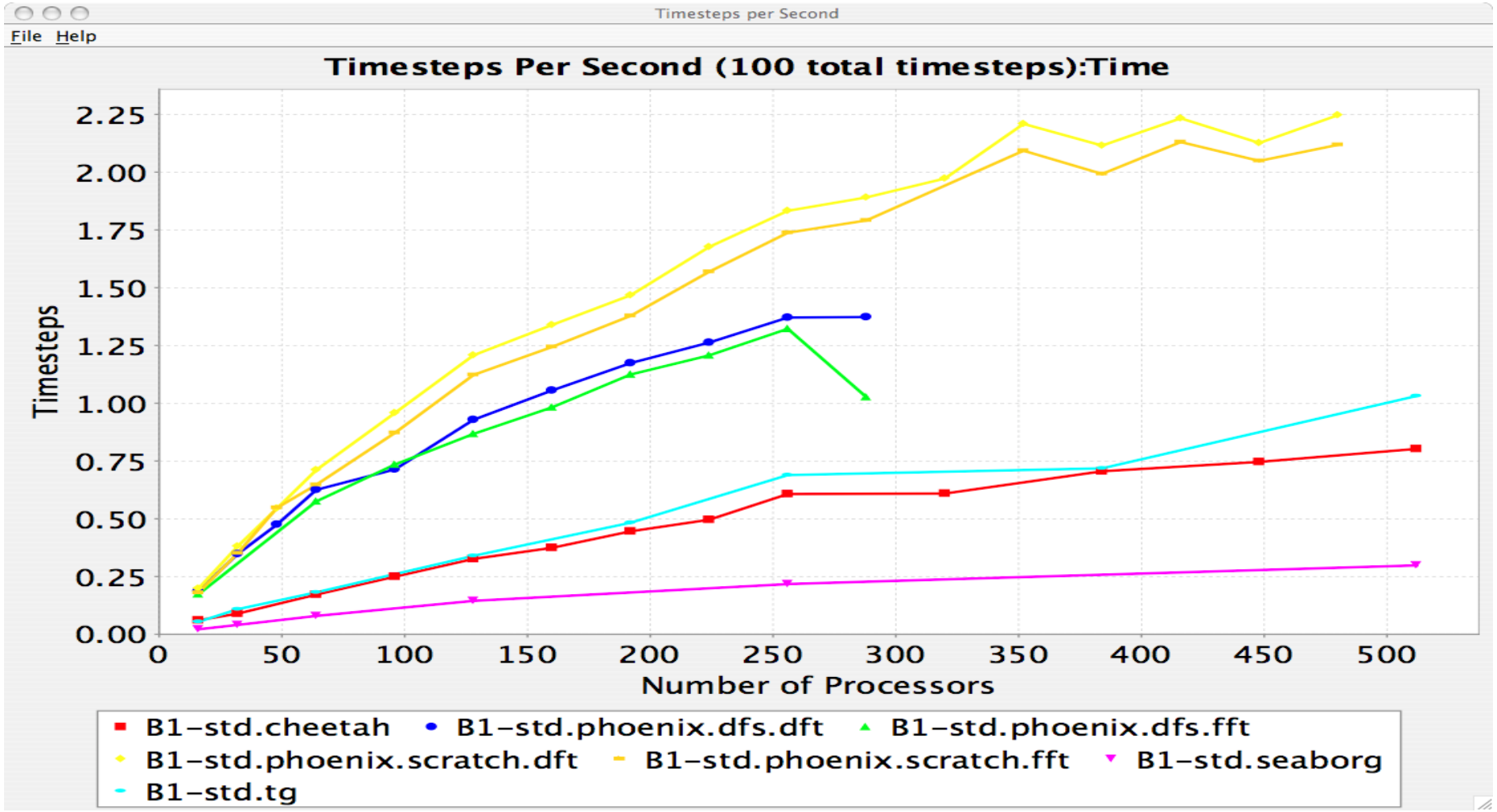
PerfExplorer - Relative Efficiency by Routine



PerfExplorer - Relative Speedup

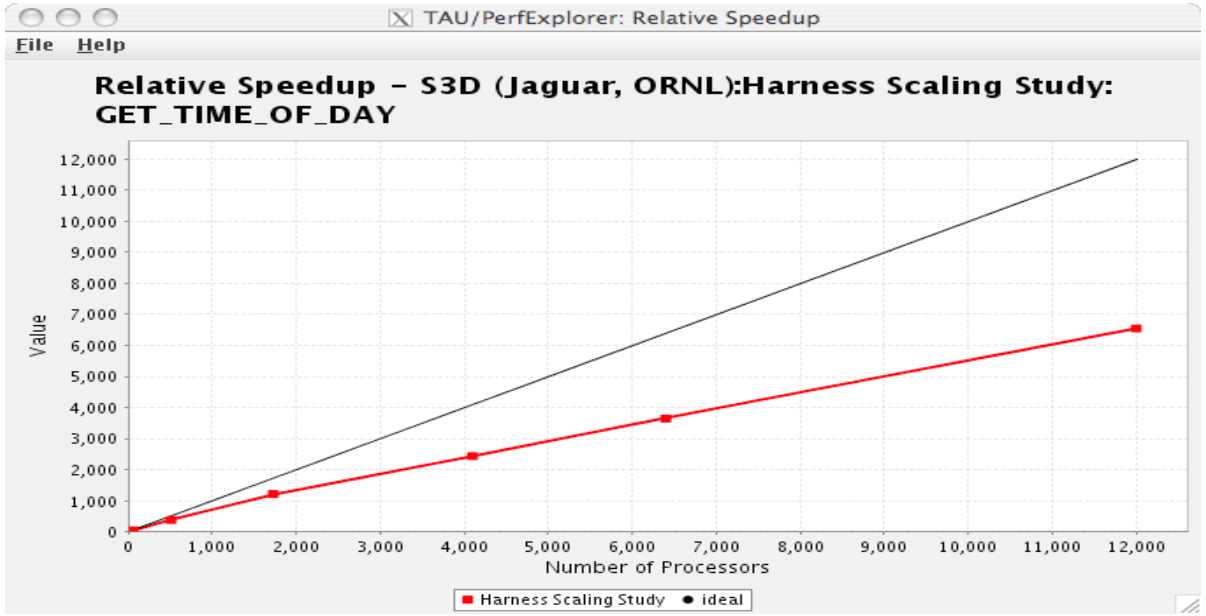


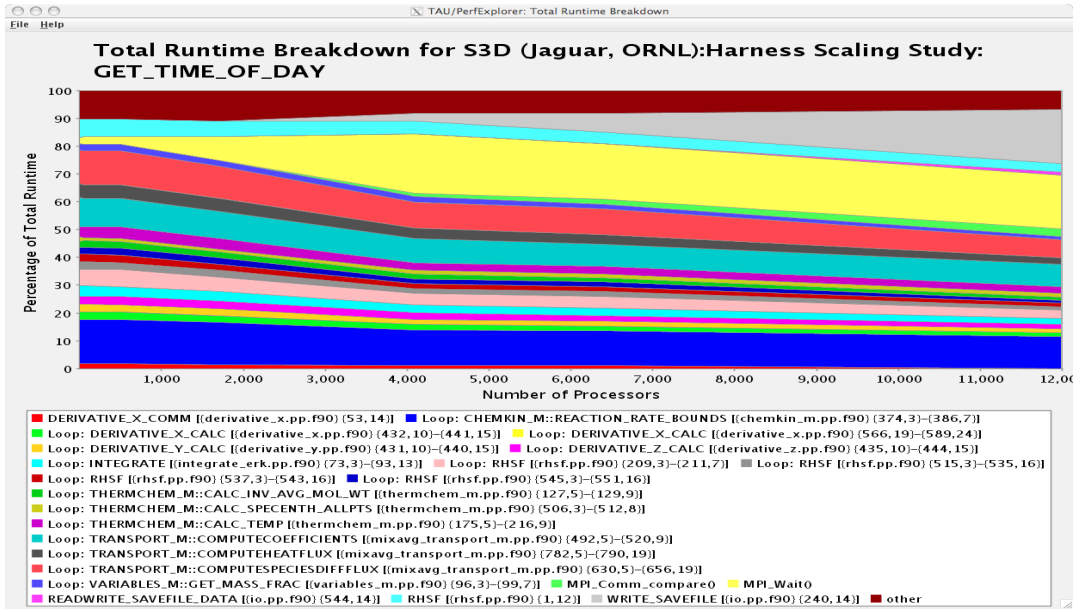
PerfExplorer - Timesteps Per Second



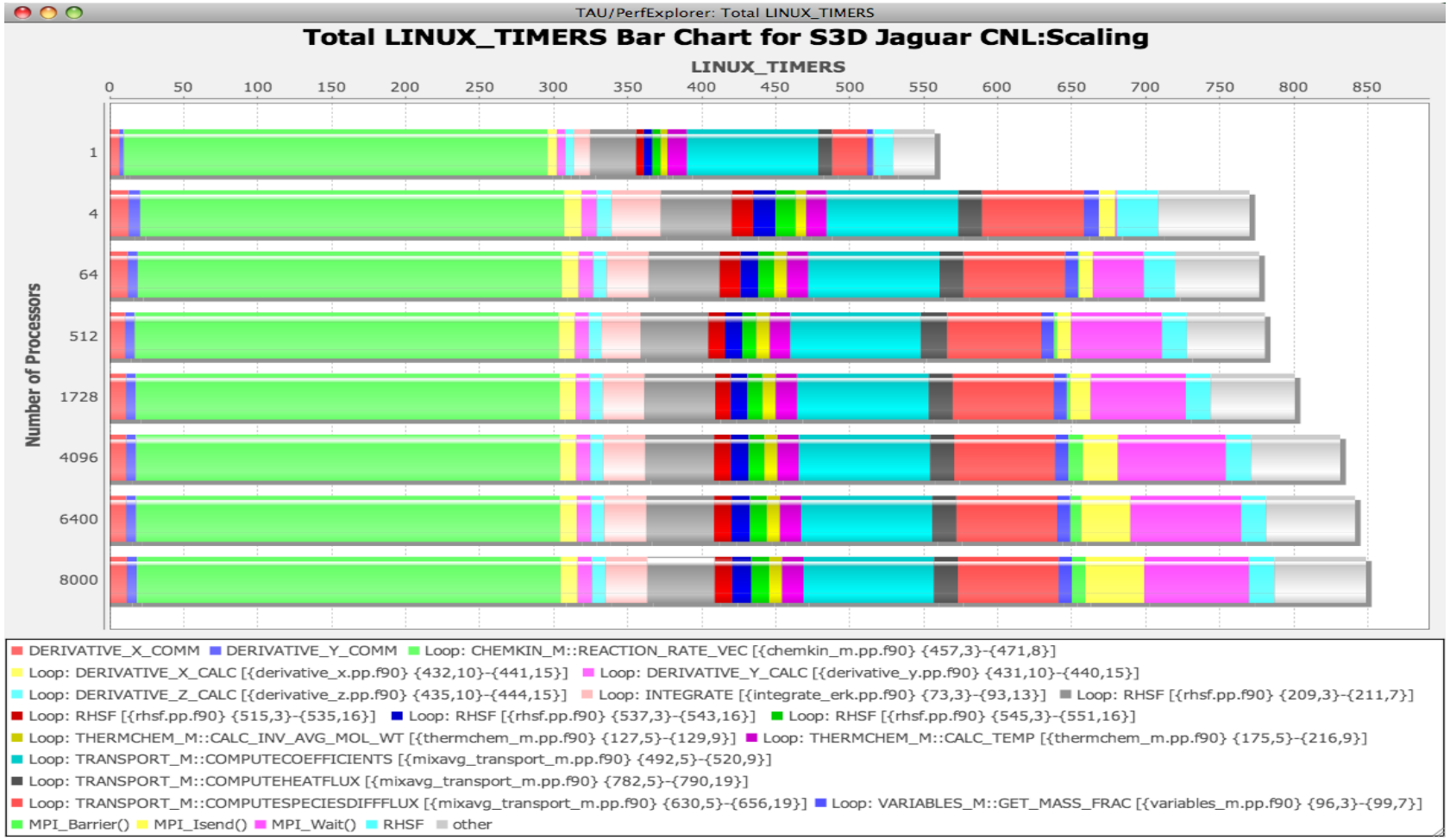
Evaluate Scalability

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

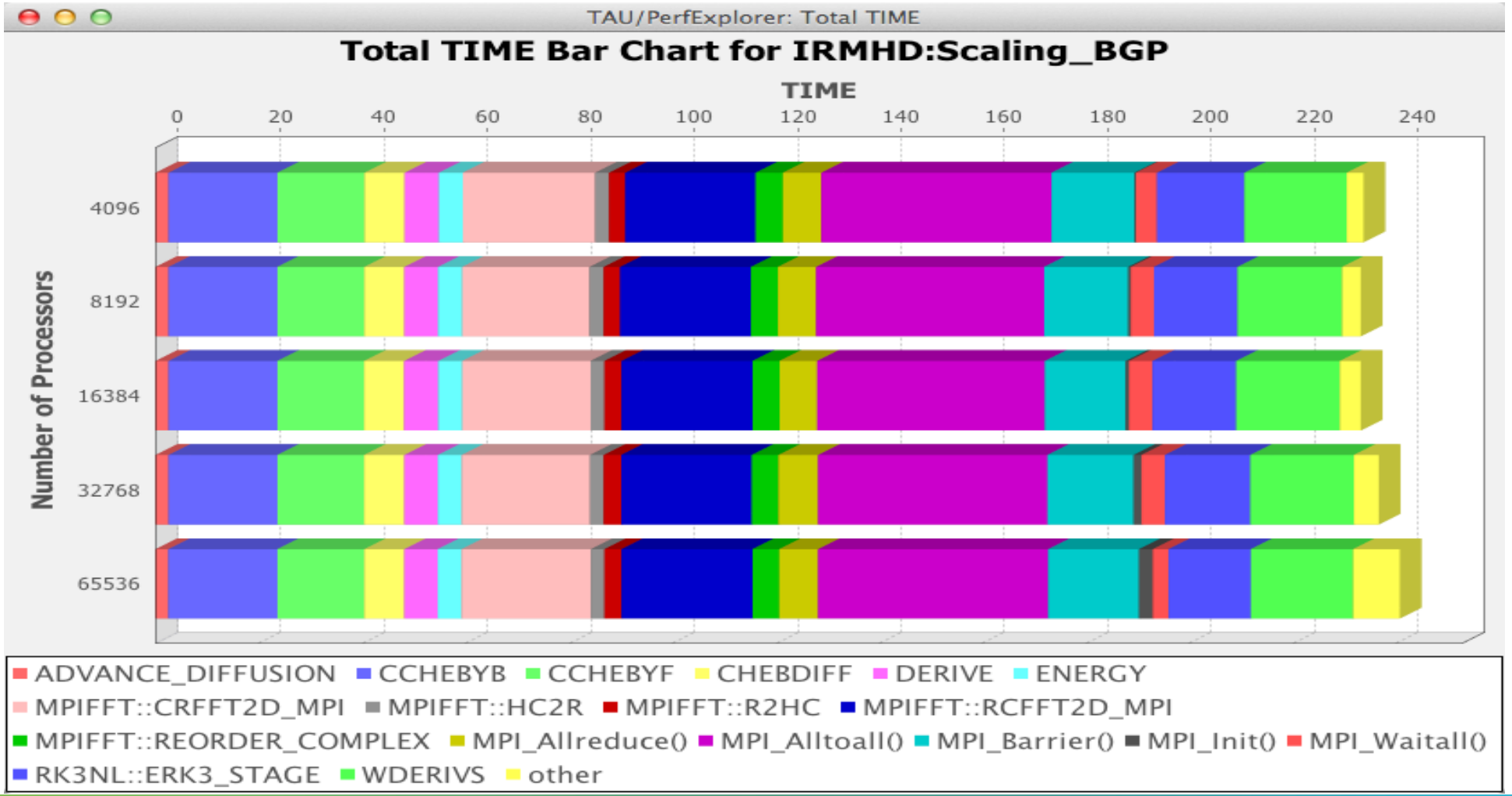




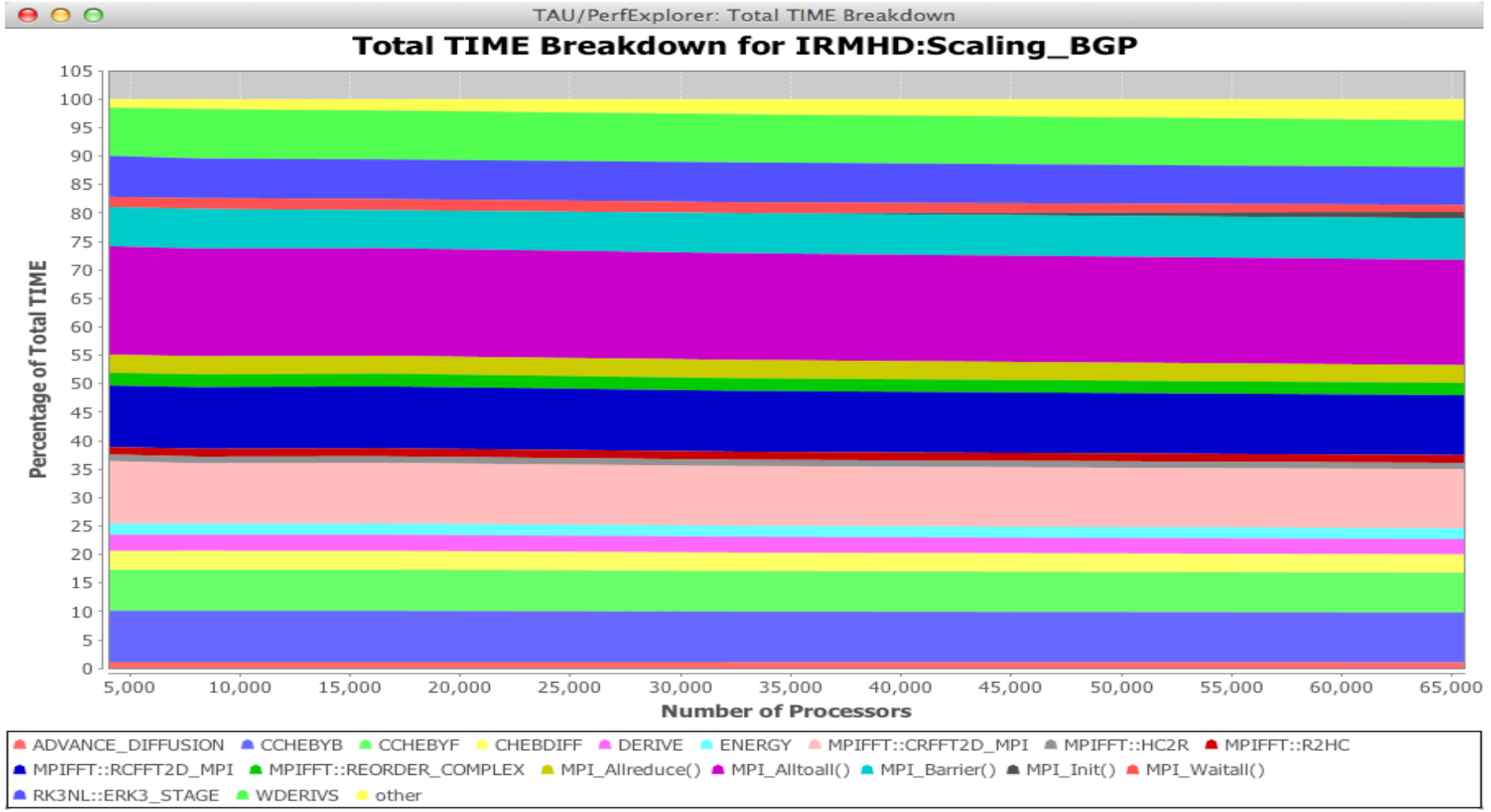
Evaluate Scalability



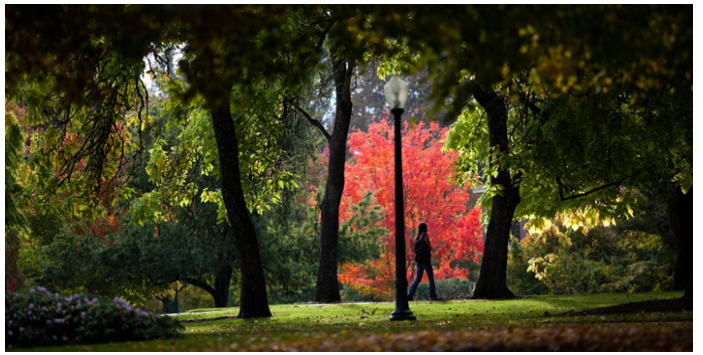
PerfExplorer



PerfExplorer



Performance Research Lab, University of Oregon, Eugene, USA



Support Acknowledgments

- US Department of Energy (DOE)
  - Office of Science contracts, ECP
  - SciDAC, LBL contracts
  - LLNL-LANL-SNL ASC/NNSA contract
  - Battelle, PNNL contract
  - ANL, ORNL contract
- Department of Defense (DoD)
  - PETTT, HPCMP
- National Science Foundation (NSF)
  - Glassbox, SI-2
- NASA
- CEA, France
- Partners:
  - University of Oregon
  - ParaTools, Inc., ParaTools, SAS
  - The Ohio State University
  - University of Tennessee, Knoxville
  - T.U. Dresden, GWT
  - Juelich Supercomputing Center















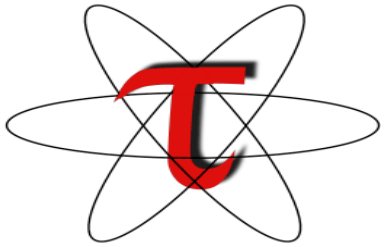




Acknowledgement

This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of two U.S. Department of Energy organizations (Office of Science and the National Nuclear Security Administration) responsible for the planning and preparation of a capable exascale ecosystem, including software, applications, hardware, advanced system engineering, and early testbed platforms, in support of the nation's exascale computing imperative.

Download TAU from U. Oregon



http://tau.uoregon.edu

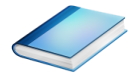
http://www.hpclinux.com [LiveDVD, OVA]

https://e4s.io [Containers for Extreme-Scale Scientific Software Stack]

Free download, open source, BSD license

Score-P:
Specialized Measurements and Analyses

Mastering build systems



- Hooking up the Score-P instrumenter `scorep` into complex build environments like *Autotools* or *CMake* was always challenging
- Score-P provides convenience wrapper scripts to simplify this
- *Autotools* and *CMake* need the used compiler already in the *configure step,* but instrumentation should not happen in this step, only in the *build step*

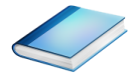
```
% SCOREP_WRAPPER=off \
> cmake .. \
> -DCMAKE_C_COMPILER=scorep-icc \
> -DCMAKE_CXX_COMPILER=scorep-icpc
```

Disable instrumentation in the *configure step*

Specify the wrapper scripts as the compiler to use

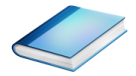
- Allows to pass addition options to the Score-P instrumenter and the compiler via environment variables without modifying the *Makefiles*
- Run `scorep-wrapper --help` for a detailed description and the available wrapper scripts of the Score-P installation

Score-P user instrumentation API



- No replacement for automatic compiler instrumentation
- Can be used to further subdivide functions
  - E.g., multiple loops inside a function
- Can be used to partition application into coarse grain phases
  - E.g., initialization, solver, & finalization
- Enabled with `--user` flag to Score-P instrumenter
- Available for Fortran / C / C++

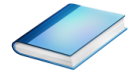
Score-P user instrumentation API (Fortran)



```
#include "scorep/SCOREP_User.inc"

subroutine foo(…)
  ! Declarations
  SCOREP_USER_REGION_DEFINE( solve )

  ! Some code…
  SCOREP_USER_REGION_BEGIN( solve, "<solver>", \
                            SCOREP_USER_REGION_TYPE_LOOP )
  do i=1,100
    [...]
  end do
  SCOREP_USER_REGION_END( solve )
  ! Some more code…
end subroutine
```

- Requires processing by the C preprocessor
  - For most compilers, this can be automatically achieved by having an uppercase file extension, e.g., `main.F` or `main.F90`

Score-P user instrumentation API (C/C++)

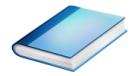


```
#include "scorep/SCOREP_User.h"

void foo()
{
  /* Declarations */
  SCOREP_USER_REGION_DEFINE( solve )

  /* Some code… */
  SCOREP_USER_REGION_BEGIN( solve, "<solver>",
                            SCOREP_USER_REGION_TYPE_LOOP )
  for (i = 0; i < 100; i++)
  {
    [...]
  }
  SCOREP_USER_REGION_END( solve )
  /* Some more code… */
}
```

Score-P user instrumentation API (C++)

```
#include "scorep/SCOREP_User.h"

void foo()
{
  // Declarations

  // Some code…
  {
    SCOREP_USER_REGION( "<solver>",
                        SCOREP_USER_REGION_TYPE_LOOP )
    for (i = 0; i < 100; i++)
    {
      [...]
    }
  }
  // Some more code…
}
```

Score-P measurement control API



- Can be used to temporarily disable measurement for certain intervals
  - Annotation macros ignored by default
  - Enabled with `--user` flag

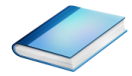
```
#include "scorep/SCOREP_User.inc"

subroutine foo(…)
  ! Some code…
  SCOREP_RECORDING_OFF()
  ! Loop will not be measured
  do i=1,100
    [...]
  end do
  SCOREP_RECORDING_ON()
  ! Some more code…
end subroutine
```

Fortran (requires C preprocessor)

```
#include "scorep/SCOREP_User.h"

void foo(…) {
  /* Some code… */
  SCOREP_RECORDING_OFF()
  /* Loop will not be measured */
  for (i = 0; i < 100; i++) {
    [...]
  }
  SCOREP_RECORDING_ON()
  /* Some more code… */
}
```

C / C++

Enriching measurements with performance counters



- Record metrics from PAPI:

```
% export SCOREP_METRIC_PAPI=PAPI_TOT_CYC
% export SCOREP_METRIC_PAPI_PER_PROCESS=PAPI_L3_TCM
```

  - Use PAPI tools to get available metrics and valid combinations:

```
% papi_avail
% papi_native_avail
```

- Record metrics from Linux perf:

```
% export SCOREP_METRIC_PERF=cpu-cycles
% export SCOREP_METRIC_PERF_PER_PROCESS=LLC-load-misses
```

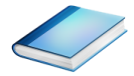
  - Use the `perf` tool to get available metrics and valid combinations:

```
% perf list
```

- Write your own metric plugin
  - Repository of available plugins: https://github.com/score-p

Only the master thread records the metric (assuming all threads of the process access the same L3 cache)

Mastering application memory usage

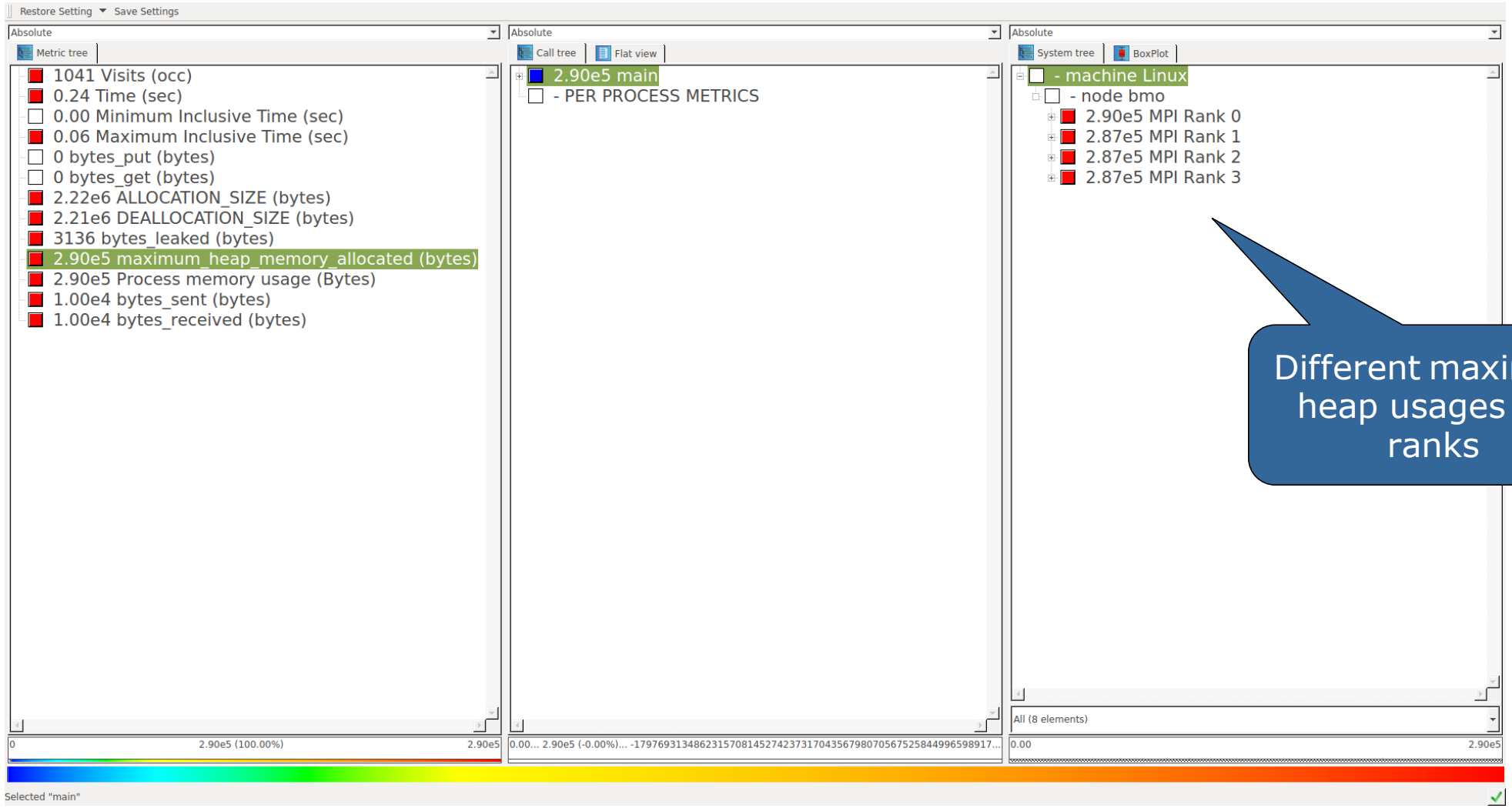


- Determine the maximum heap usage per process
- Find high frequent small allocation patterns
- Find memory leaks
- Support for:
  - C, C++, MPI, and SHMEM (Fortran only for GNU Compilers)
  - Profile and trace generation (profile recommended)
    - Memory leaks are recorded only in the profile
    - Resulting traces are not supported by Scalasca yet

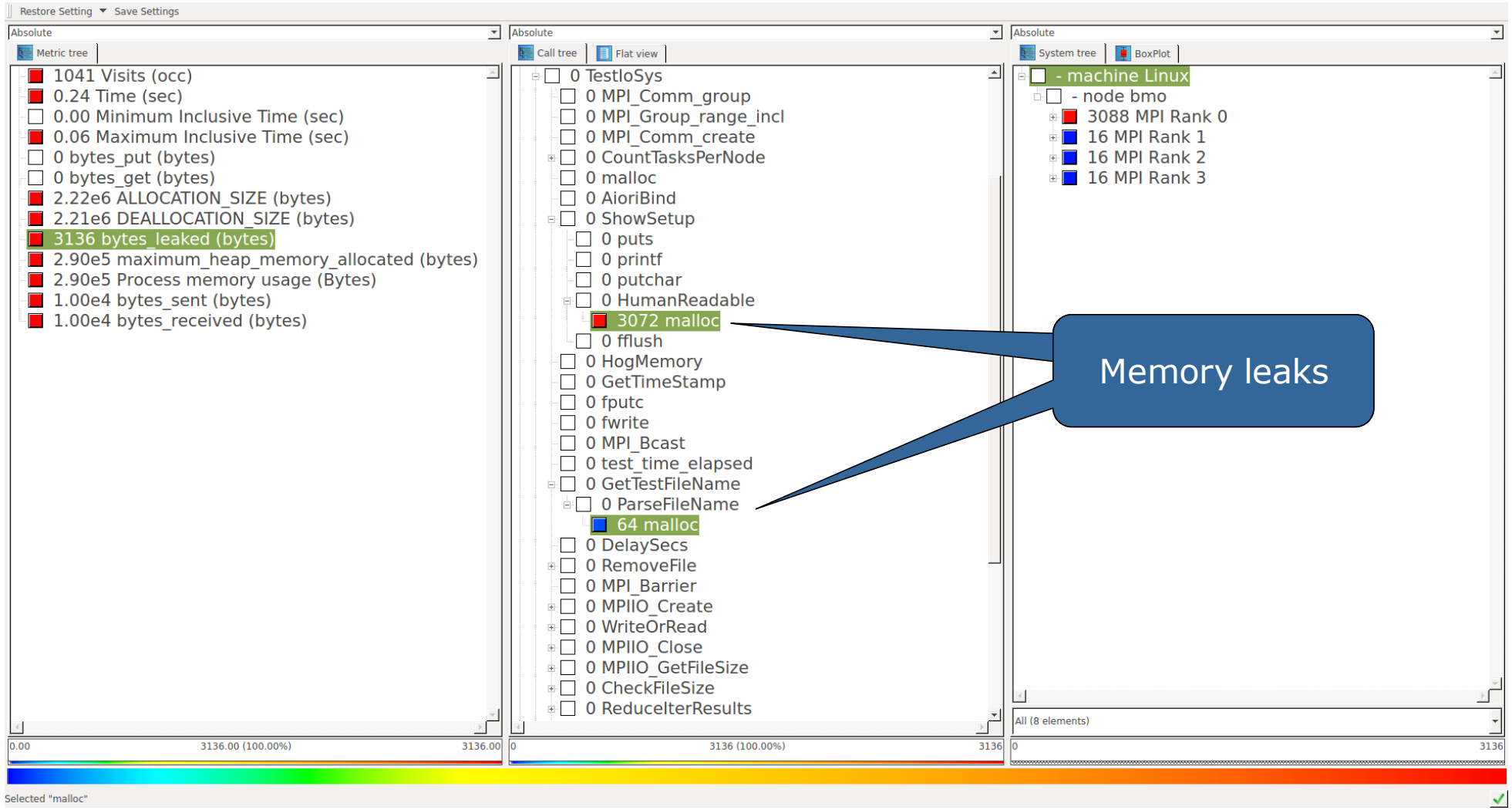
```
% export SCOREP_MEMORY_RECORDING=true
% export SCOREP_MPI_MEMORY_RECORDING=true
```

- Set new configuration variable to enable memory recording

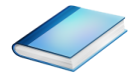
Mastering application memory usage



Mastering application memory usage



Hybrid measurement with sampling

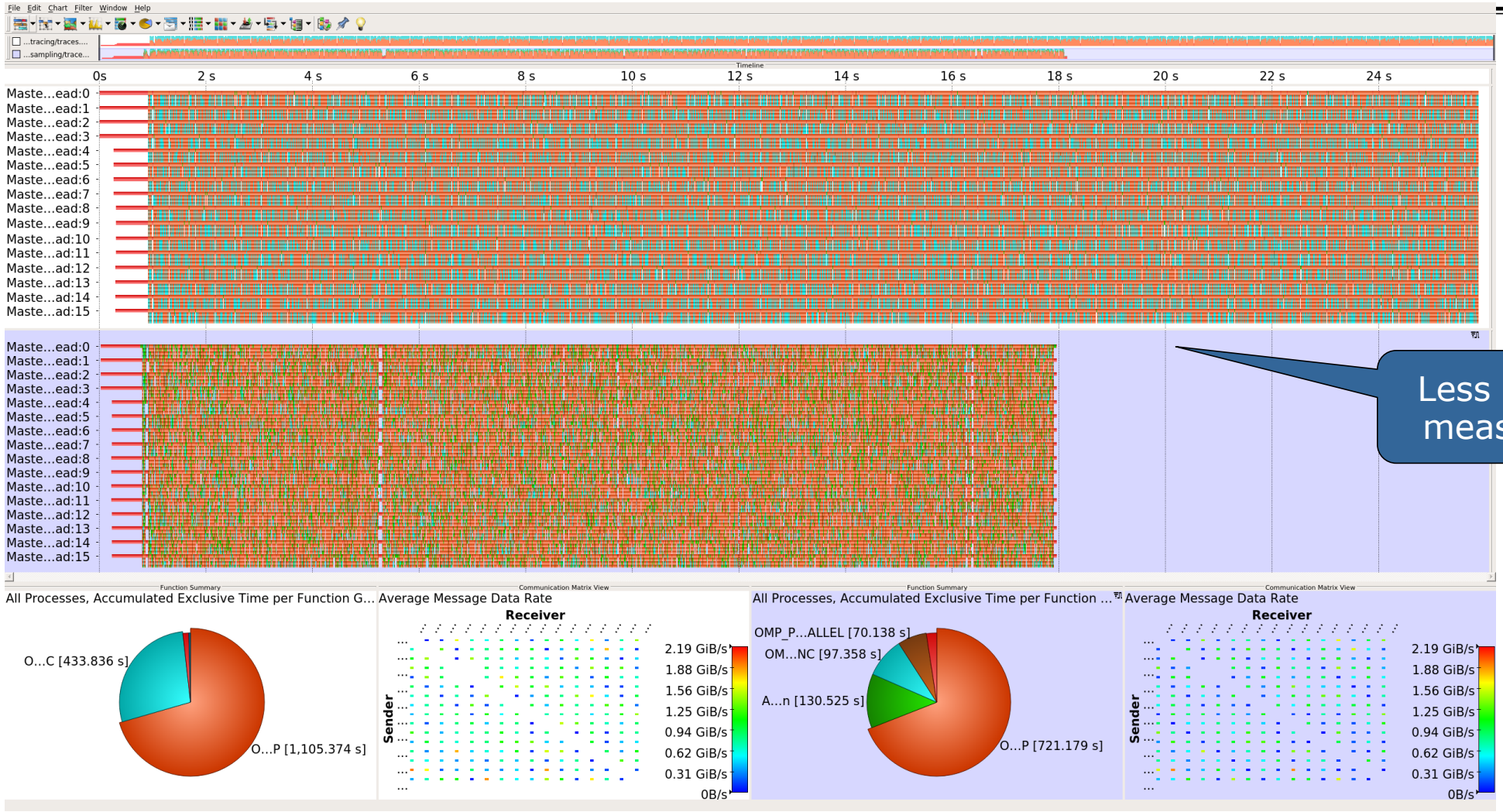


- Automatic compiler instrumentation greatly disturbs C++ applications because of frequent/short function calls => Use sampling instead
- Novel combination of sampling events and instrumentation of MPI, OpenMP, …
  - Sampling replaces only compiler instrumentation (use `--nocompiler`)
  - Instrumentation is still used for parallel activities (MPI, OpenMP, CUDA, I/O)
- Supports profile and trace generation

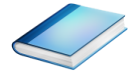
```
% export SCOREP_ENABLE_UNWINDING=true
% # use the default sampling frequency
% #export SCOREP_SAMPLING_EVENTS=perf_cycles@2000000
```

- Set new configuration variable to enable sampling

Mastering C++ applications

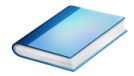


Wrapping calls to 3rd party libraries

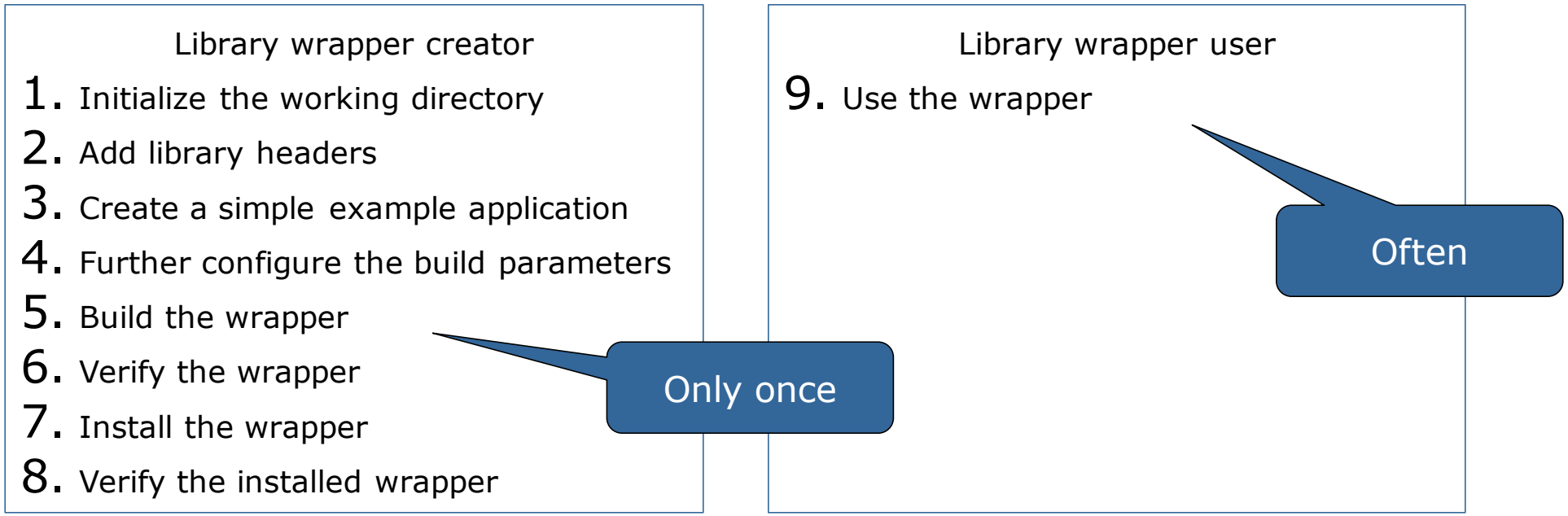


- Score-P does not record function calls to non-instrumented external libraries
- Increase insight into the behavior of the application
  - How does the application use the external library?
  - How does this compares to the usage of other libraries?
- Manual user instrumentation of the application using the library should be avoided
- Vendor provided libraries cannot be instrumented, but API provided in headers

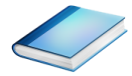
Wrapping calls to 3rd party libraries: Library wrapper generator



- Workflow to generate library wrappers for most C/C++ libraries
- Tailored towards user of the external library, not users of Score-P
- Results can be shared by multiple users
- Workflow driver `scorep-libwrap-init --help` provides instructions



Wrapping calls to 3rd party libraries: Workflow



- Start workflow by telling `scorep-libwrap-init` how you would compile and link an application, e.g., using FFTW

```
% scorep-libwrap-init          \
>   --name=fftw                 \
>   --prefix=$PREFIX            \
>   -x c                        \
>   --cppflags="-O3 -DNDEBUG -openmp -I$FFTW_INC" \
>   --ldflags="-L$FFTW_LIB"     \
>   --libs="-lfftw3f -lfftw3" \
>   working_directory
```

Omit to install into Score-P

Flags used to compile/link

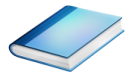
Working directory can be archived for later rebuild

- Generate and build wrapper

```
% cd working_directory
% ls                    # (Check README.md for instructions)
% make                  # Generate and build wrapper
% make check            # See if header analysis matches symbols
% make install          #
% make installcheck     # More checks: Linking etc.
```

Tells you how to use the wrapper with Score-P

Wrapping calls to 3rd party libraries: Usage and result



- List of available wrappers:

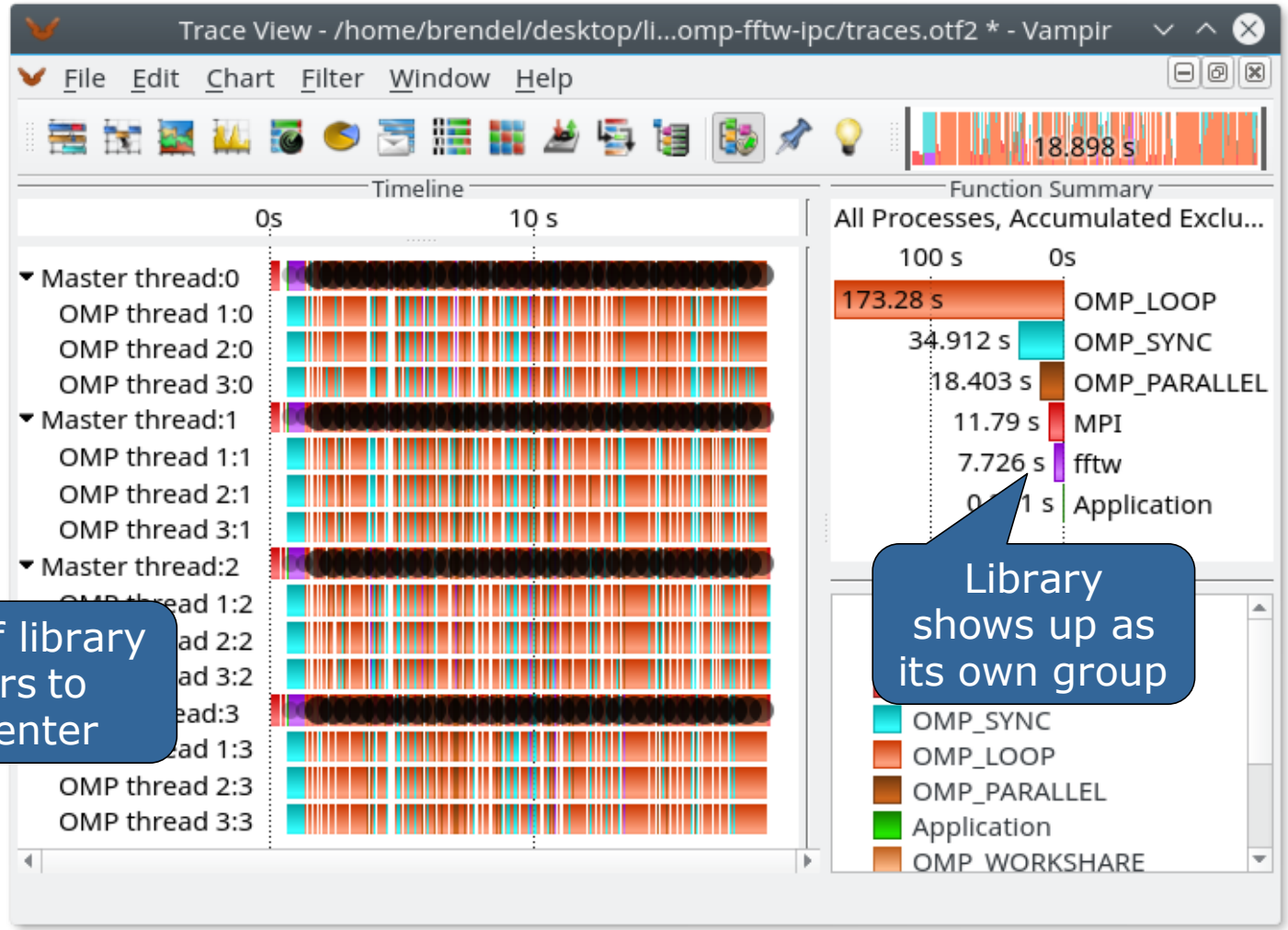
```
% scorep-info libwrap-summary
```

- Instrumentation:

```
% cd <application>
% export \
SCOREP_WRAPPER_INSTRUMENTER_FLAGS=\
    --libwrap=fftw
% make clean
% make
# run application as usual
```

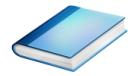
- MPI + OpenMP
- Calls to FFTW library

Pass list of library wrappers to instrumenter



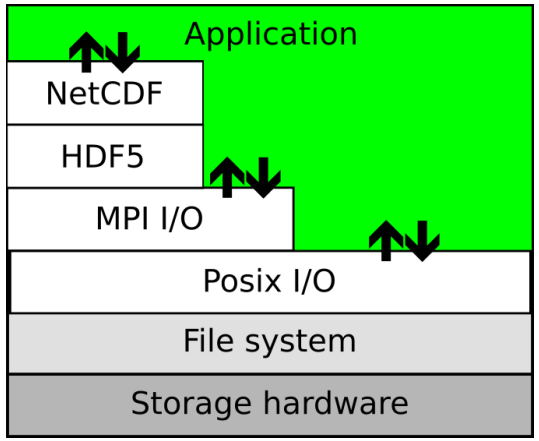
Library shows up as its own group

File I/O recording



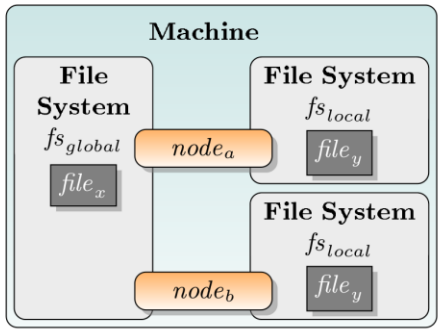
- Omnipresent in todays HPC applications

- Record interaction between multiple layers
  - MPI I/O (`MPI_File_open`)
  - ISO C I/O (`fopen`)
  - POSIX I/O (`open`, interface to OS)

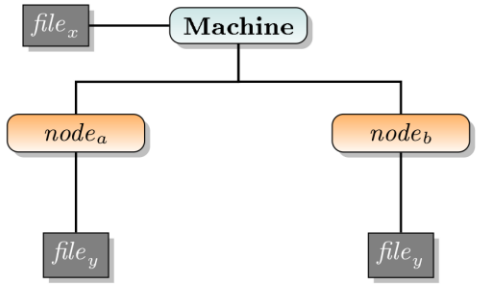


- System tree information determine whether file resides in a shared filesystem

- High level of detail
  => Trace data might increase dramatically



(a) Hardware topology



(b) System tree representation

NetCDF & HDF5 will be supported later

B_EFF I/O

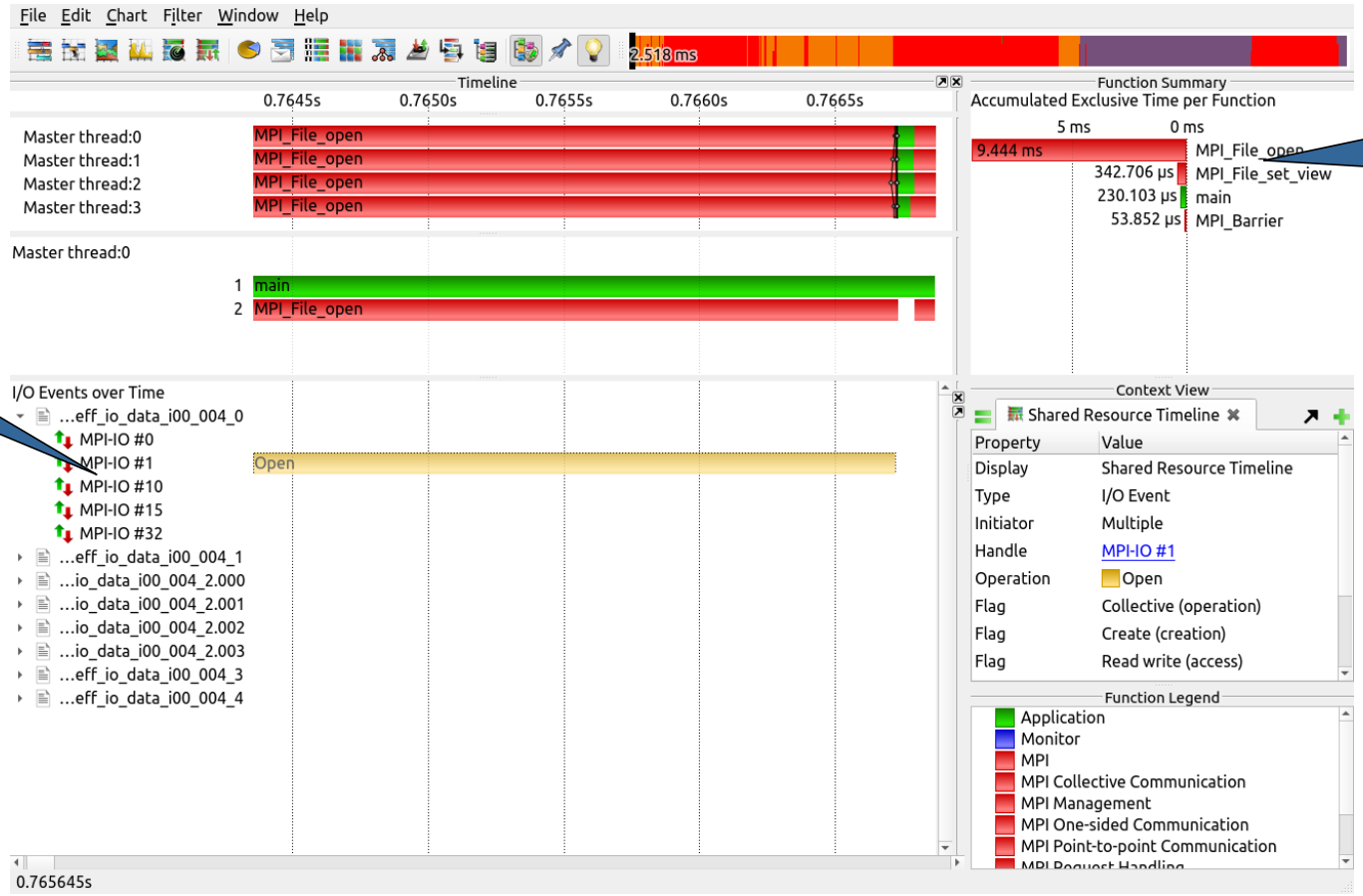
- MPI I/O benchmark
- fs.hlrs.de/projects/par/mpi//b_eff_io/

- MPI I/O is enabled by default

```
% scorep-mpicc -o b_eff_io b_eff_io.c
% export SCOREP_EXPERIMENT_DIRECTORY=scorep-b_eff_io-4-profile
% mpirun -n 4 -c 6 ./b_eff_io -MB 2048 -MT 98304 -rewrite -N 4 -T 60
% scorep-scorep –g scorep-b_eff_io-4-profile/profile.cubex
% export SCOREP_EXPERIMENT_DIRECTORY=scorep-b_eff_io-4-tracing
% export SCOREP_FILTERING_FILE=initial_scorep.filter
% export SCOREP_ENABLE_TRACING=true
% export SCOREP_TOTAL_MEMORY=31MB
% mpirun -n 4 -c 6 ./b_eff_io -MB 2048 -MT 98304 -rewrite -N 4 -T 60
```

Result visualization

Hierarchy of files and corresponding handles



MPI I/O functions recorded

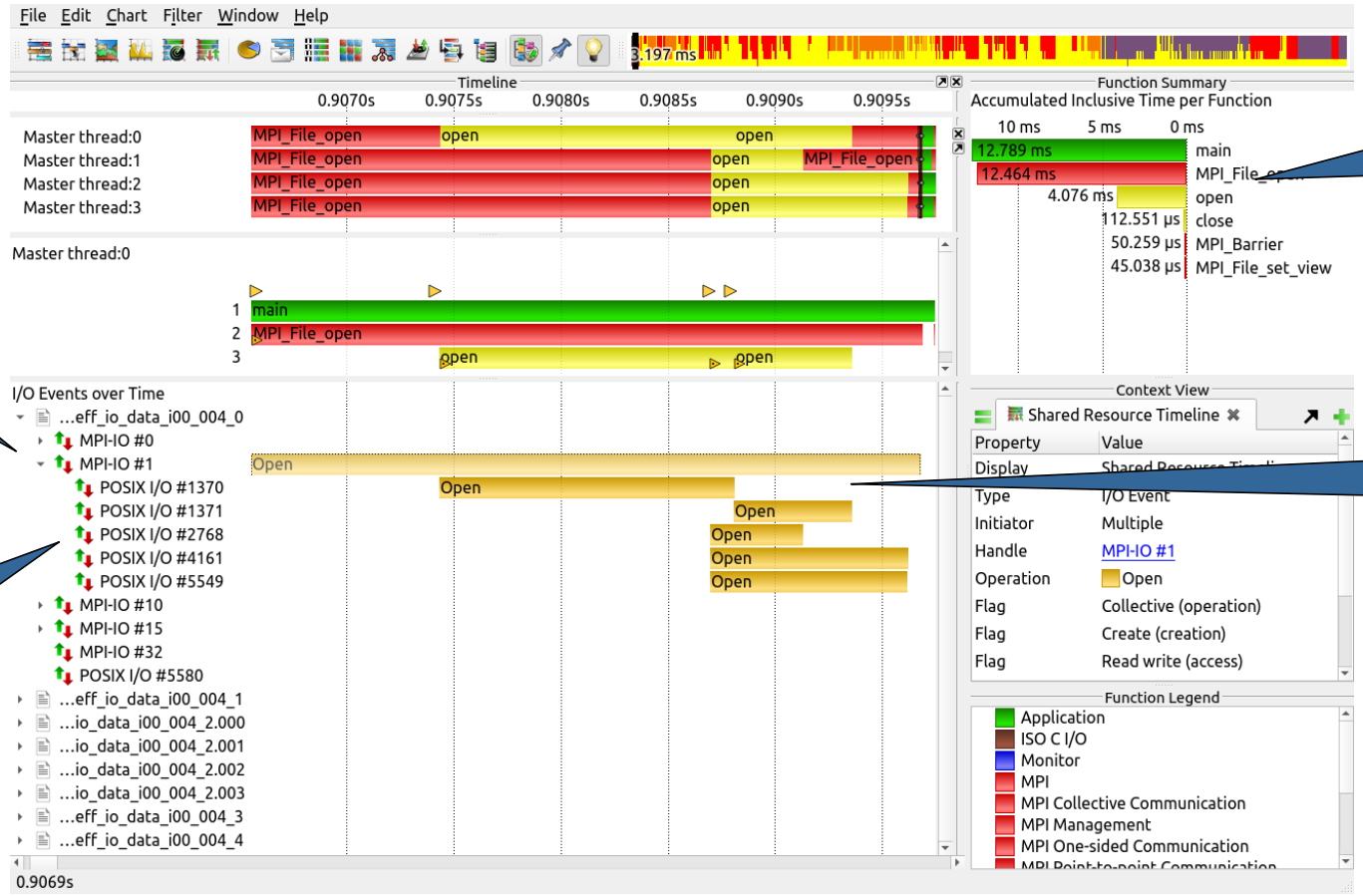
B_EFF I/O

- Enabling ISO C and POSIX I/O instrumentation

- Instrumentation might require threading support

```
% export SCOREP_WRAPPER_INSTRUMENTER_FLAGS=\
    '--io=runtime:posix --thread=pthread'
% scorep-mpicc -o b_eff_io b_eff_io.c
% export SCOREP_EXPERIMENT_DIRECTORY=scorep-b_eff_io-4-profile+posix
% export SCOREP_ENABLE_TRACING=false
% scorep-scorep scorep-b_eff_io-4-profile+posix/profile.cubex
% export SCOREP_EXPERIMENT_DIRECTORY=scorep-b_eff_io-4-tracing+posix
% export SCOREP_ENABLE_TRACING=true
% export SCOREP_TOTAL_MEMORY=61MB
% mpirun -n 4 -c 6 ./b_eff_io -MB 2048 -MT 98304 -rewrite -N 4 -T 60
```

Result visualization



Increased hierarchy

Each rank operates on its own POSIX handle

MPI and POSIX I/O functions recorded

Rank 0 ensures file exists for all ranks

Further information

- Community instrumentation & measurement infrastructure
  - Instrumentation (various methods) and sampling
  - Basic and advanced profile generation
  - Event trace recording
  - Online access to profiling data
- Available under 3-clause BSD open-source license
- Documentation & Sources:
  - https://www.score-p.org
- User guide also part of installation:
  - `<prefix>/share/doc/scorep/{pdf,html}/`
- Support and feedback: support@score-p.org
- Subscribe to news@score-p.org, to be up to date

Exercise: Typical performance bottlenecks and how they can be identified

The VI-HPS Team

Sparse matrix vector multiplication

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

- A sparse matrix is a matrix populated primarily with zeros
- 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]
```

Sparse matrix vector multiplication

- Naive 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

Local installation (JUWELS Booster)

- Set account and default environment (NVHPC + ParaStationMPI) via helper script:

```
% source /p/project/training2341/setup.sh
```

- Load the modules for the tool environment:

```
% module load Score-P CubeGUI
```

- Copy tutorial sources to your WORK directory (or your personal workspace)
  - Only required if not done already (for opening exercise)

```
% cd $WORK
% tar xf $PROJECT/examples/smxv.tar.gz
% cd SMXV
```

SMxV: Makefile

```
% cat Makefile
CC=pgcc

OMP_CFLAGS=-fopenmp
LIBS=-lm

TARGETS=\
     smxv-omp

all: $(TARGETS)

clean:   $(RM) *.o $(TARGETS) smxv-omp.scorep

CPPFLAGS=-DLITTLE_ENDIAN -DITERLIMIT=500

smxv-omp: smxv.c y_Ax.h
        $(CC) $(OMP_CFLAGS) $(CPPFLAGS) $(CFLAGS) -o $@ $< $(LIBS)

scorep: smxv-omp.scorep

smxv-omp.scorep: smxv.c y_Ax.h
        scorep $(CC) $(OMP_CFLAGS) $(CPPFLAGS) $(CFLAGS) -o $@ $< $(LIBS)
```

Load imbalances in OpenMP codes

- Exercise I: Determine scaling behavior for number of threads between 1 and 12

```
% make
% sbatch -c 1 run.sbatch
% sbatch -c 2 run.sbatch
…
% sbatch -c 12 run.sbatch
```

Load imbalances in OpenMP codes

- Exercise II: Collection and analyze measurement

```
% make scorep
% sbatch -c 12 scorep.sbatch
```

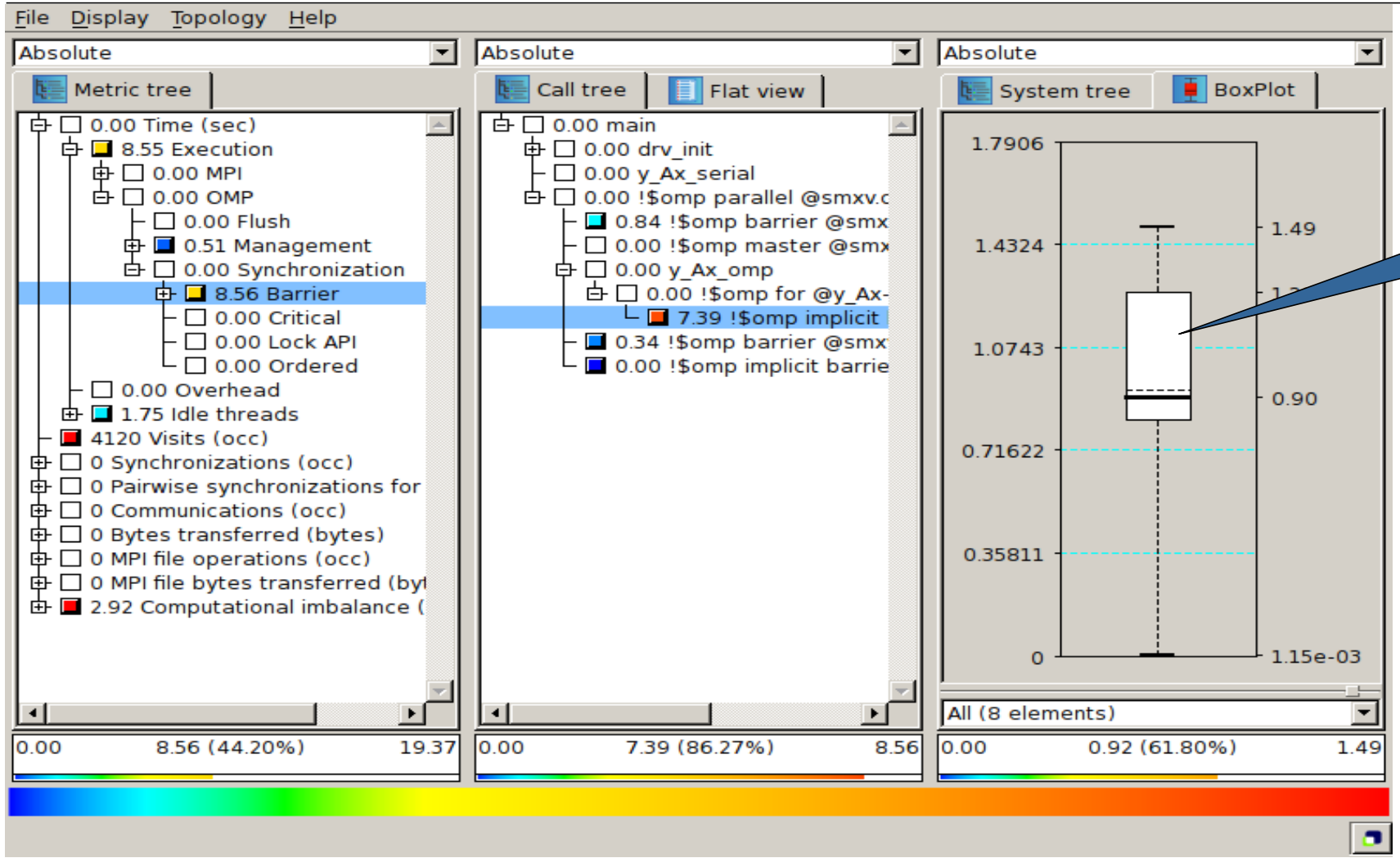
Load imbalances in OpenMP codes: Profile examination

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

- Open measurement with Cube

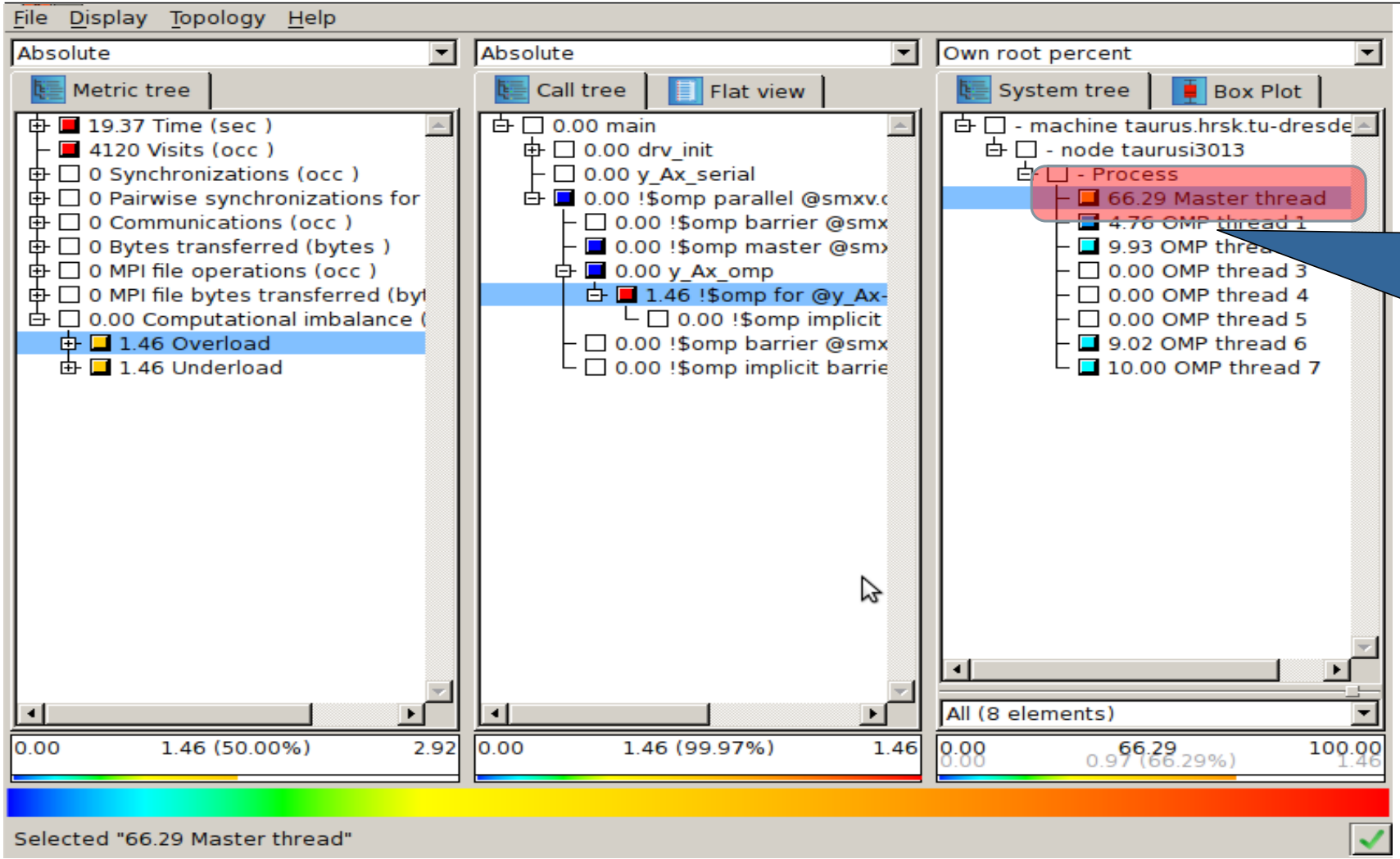
```
% cube scorep-smxv-12/trace.cubex

                    [CUBE GUI showing trace analysis report]
```

Time spent in OpenMP barriers



Great variation in the distribution of the time spent in the barrier

Computational imbalance



Master thread does 24% of the work (2.00/8.30) and has 66% of the computational overload

Sparse matrix vector multiplication

- Scheduling policy determined via environmental variable `OMP_SCHEDULE`:
  1. `static[,n]`
     - Schedule is fix at start of work share
  2. `dynamic[,n]`
     - Work is split into packages of n-items. Each thread requests new package

- Exercise III: Measure improved algorithm

```
% cat scorep.sbatch
…
export SCOREP_EXPERIMENT_DIRECTORY=scorep-smxv-$SLURM_CPUS_PER_TASK-dynamic
export OMP_SCHEDULE=…

scan -t srun ./smxv-omp.scorep yax_large.bin
% sbatch -c 12 scorep.sbatch
```

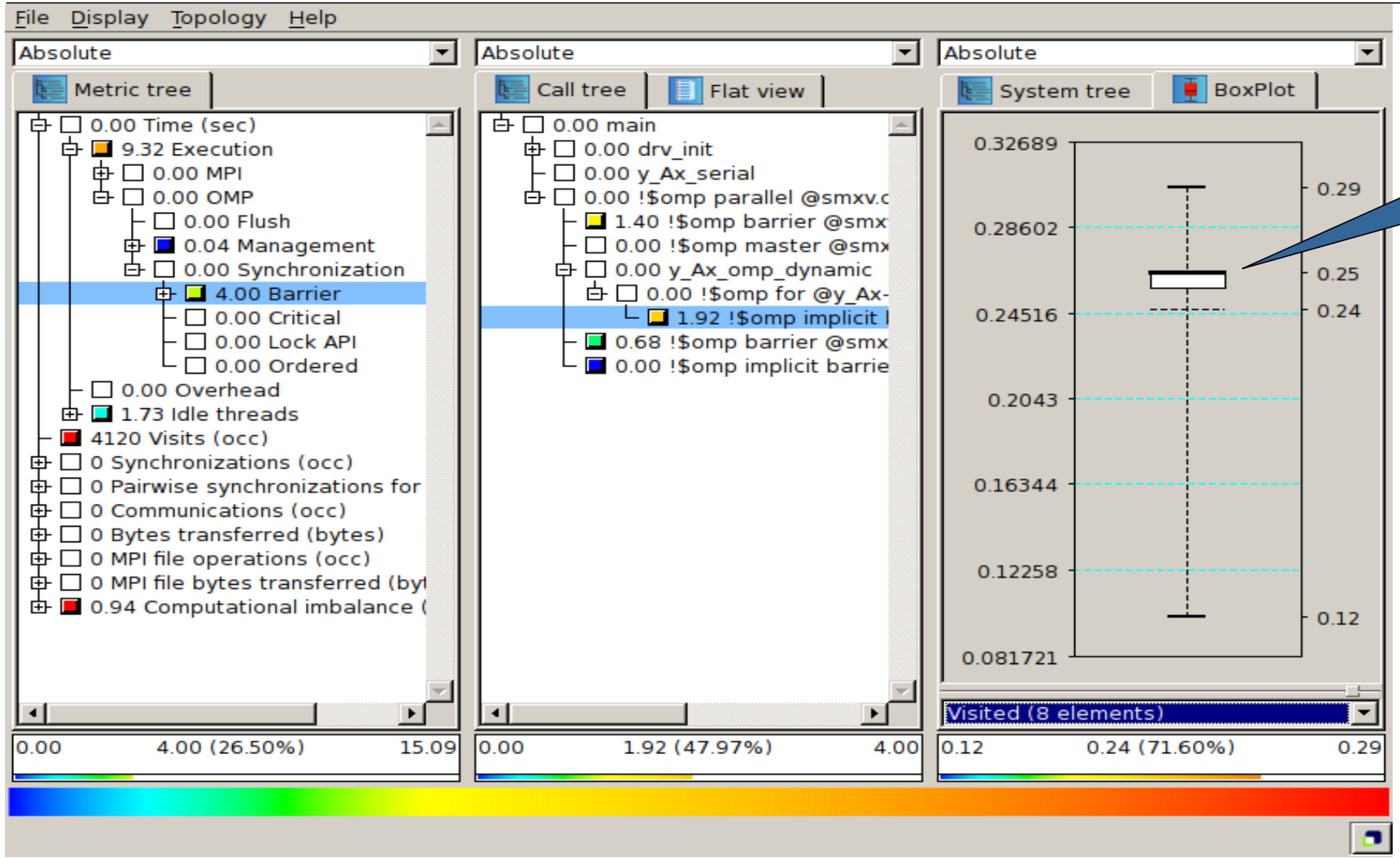
Profile Analysis

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

- Open measurement with Cube

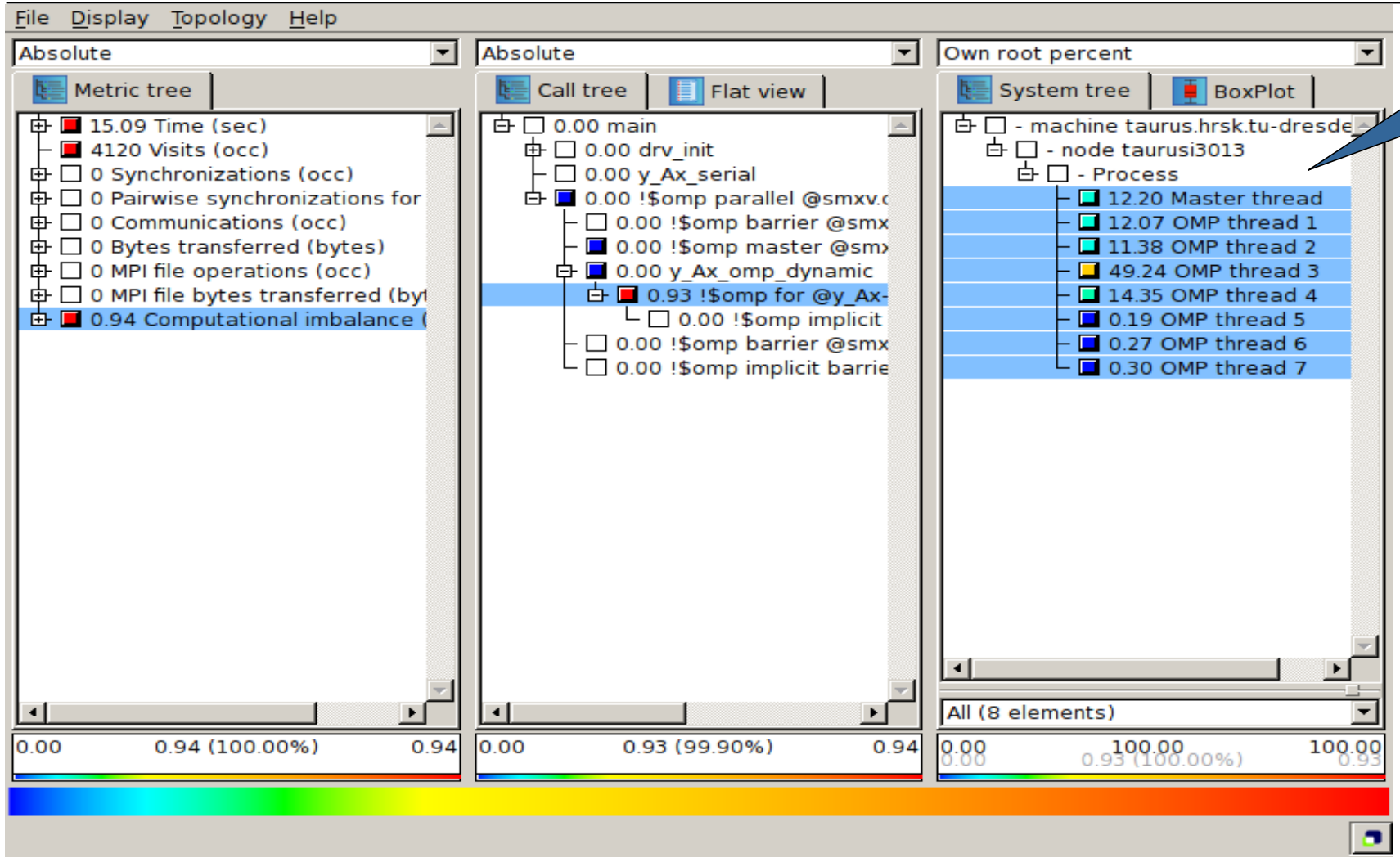
```
% cube scorep-smxv-12-dynamic/trace.cubex

                    [CUBE GUI showing trace analysis report]
```

Time spent in OpenMP barriers



Variation is now much smaller

Computational imbalance

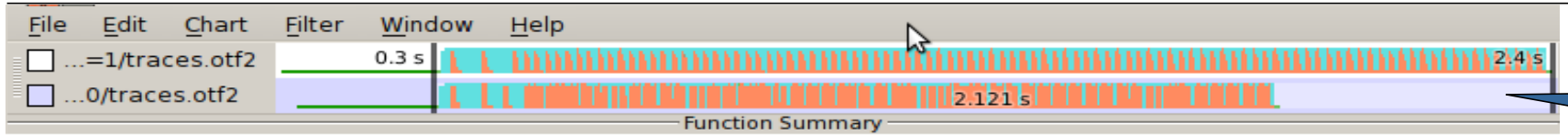


Computational imbalance can still be improved

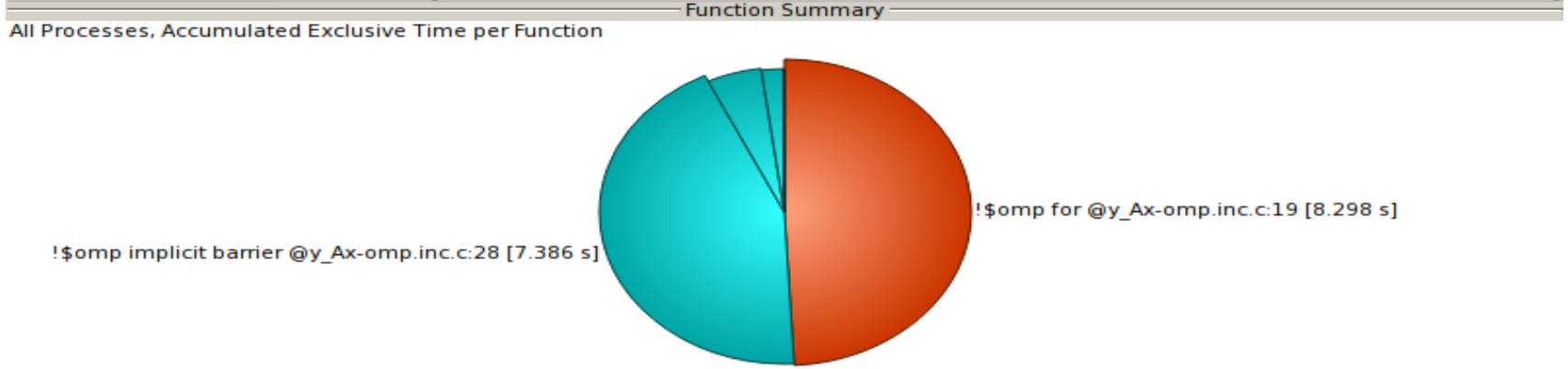
Trace comparison

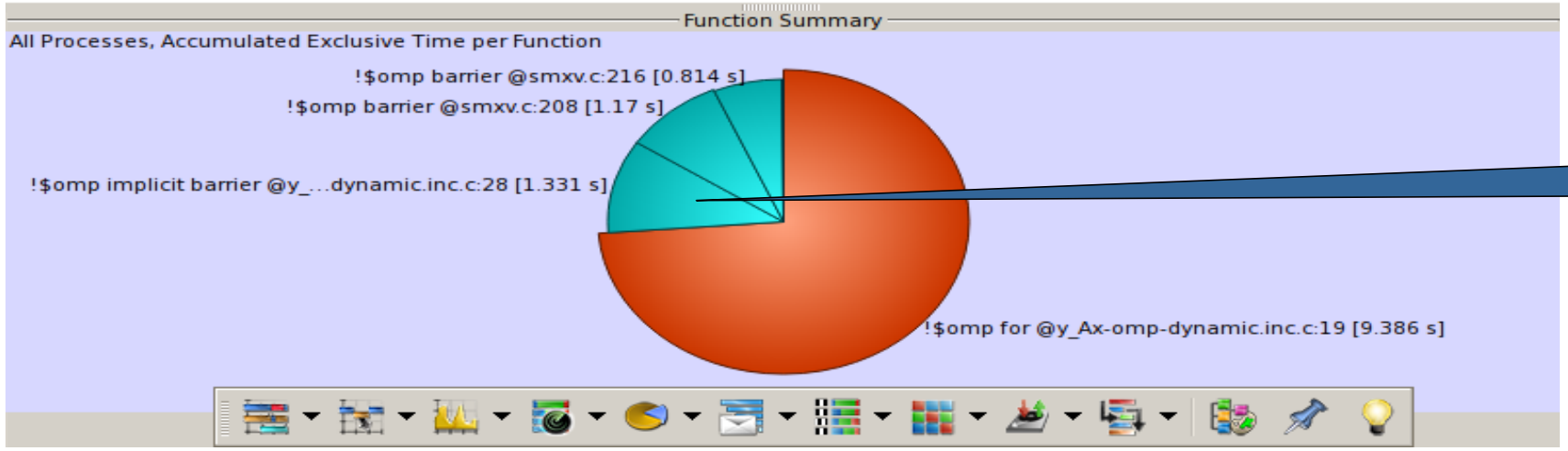
- Open both traces in Vampir

Time spent in OpenMP barriers



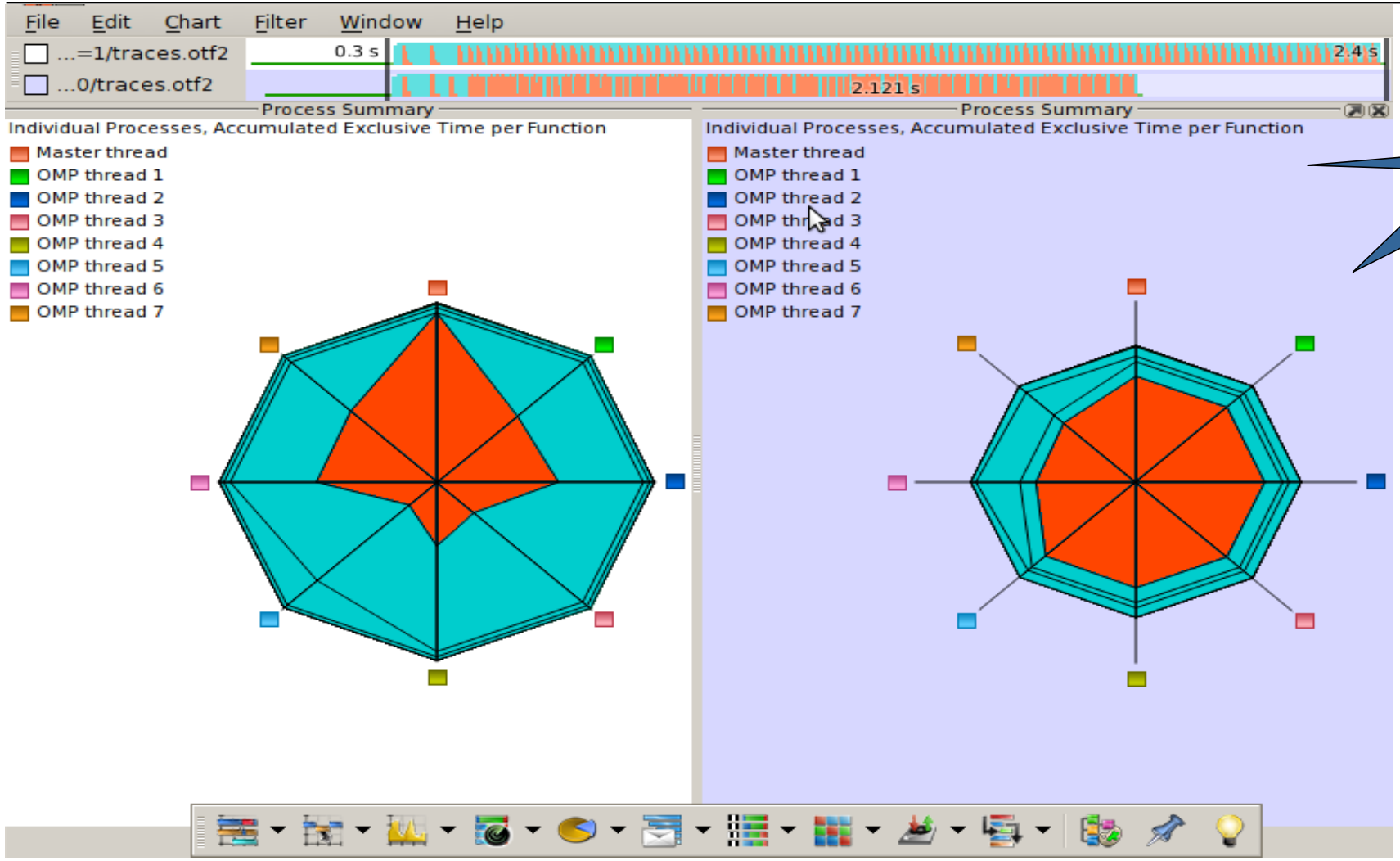
Improved runtime





Less time in OpenMP barrier

Computational imbalance



Great imbalance for time spent in computational code

Validate optimization

- Exercise IV

```
% cat run.sbatch
…
export OMP_SCHEDULE=…

srun ./smxv-omp yax_large.bin
% sbatch -c 12 run.sbatch
```

Review

Markus Geimer
Jülich Supercomputing Centre

Summary

You've been introduced to a variety of tools

- with hints to apply and use the tools 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

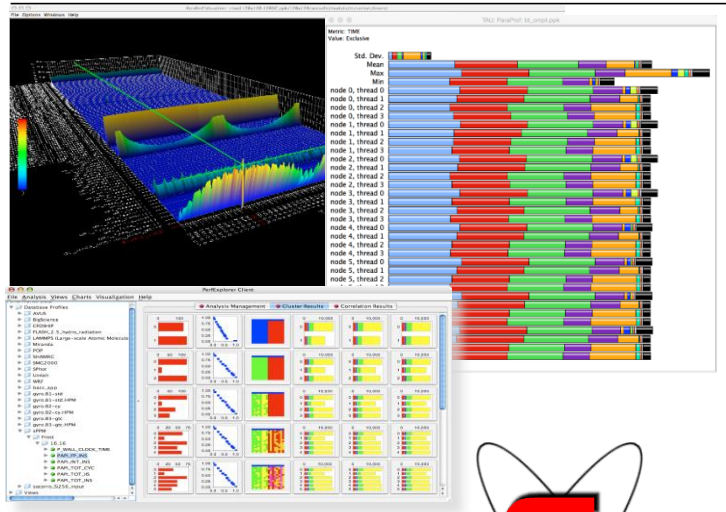
Which tools you use and when you use them likely to depend on the situation

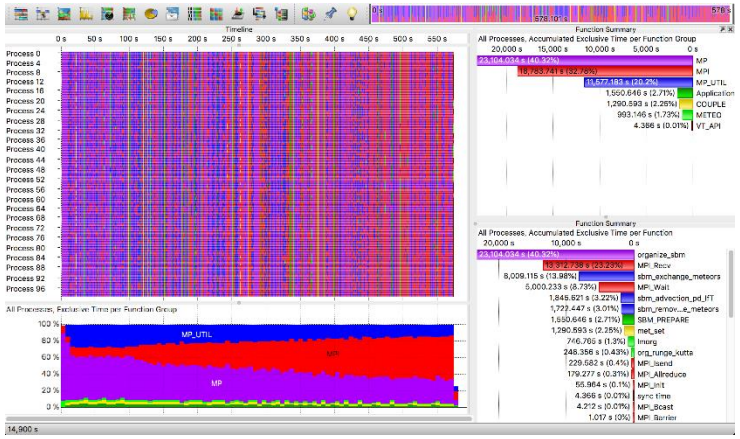
- which are available on (or for) your computer system
- which support your programming paradigms and languages
- which you are familiar (comfortable) with using
- which type of issue you suspect
- which question you want to have answered

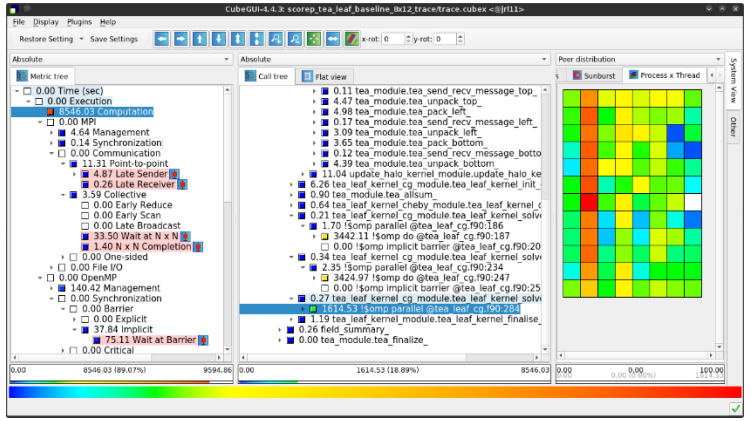
Being aware of (potentially) available tools and their capabilities can help finding the most appropriate tools

Portable multi-platforms tools complement platform-specific/vendor tools

Tools featured in this tutorial







Tuning and Analysis Utilities

VAMPIR

scalasca

Score-P
Scalable performance measurement infrastructure for parallel codes

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 set of corefiles
- parallel debuggers help isolate known problems
- correctness checking tools can identify other issues
  (that might not cause problems right now, but will eventually)
  - e.g., race conditions, invalid/non-compliant usage

Best to start with an overview of execution performance

- fraction of time spent in computation (CPU & GPU) vs comm/synch vs I/O
- which sections of the application/library code are most costly
- Example profilers: **Score-P + Cube/ParaProf, TAU**

and how it changes with scale or different configurations

- processes vs threads, mappings, bindings

Workflow (communication/synchronization)

Communication 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 more expensive
  - generally manifests as waiting time at following collective ops

Workflow (wasted time waiting)

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
  - 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

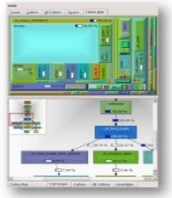
Workflow (core computation)

Effective computation within processors/cores (and GPUs) is also vital

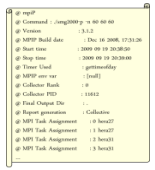
- Optimized libraries may already be available
- Optimizing compilers can also do a lot
  - provided the code is clearly written and not too complex
  - appropriate directives and other hints can also help
- Processor hardware counters can also provide insight
  - although hardware-specific interpretation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

Technologies and their integration

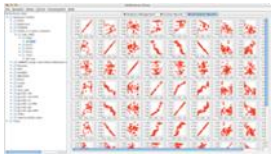
KCACHEGRIND

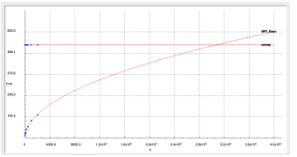


CALIPER / LIKWID / MAP/PR / MAQAO / MPIP / O|SS

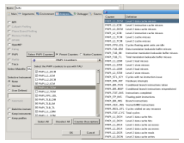


TAU EXTRA-P





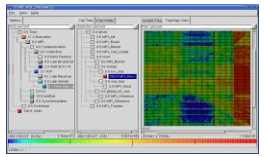
PAPI



Hardware monitoring

Automatic profile & trace analysis

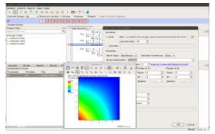
SCALASCA / CUBE



SCORE-P / EXTRAE

MUST / ARCHER

DDT

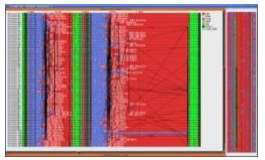


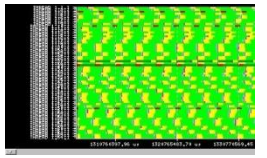
Debugging, error & anomaly detection

JUBE

Visual trace analysis

VAMPIR PARAVER





STAT

Execution

Optimization

LIKWID / MEMCHECKER / SPINDLE / SIONLIB

MAQAO

Further information

Website

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

https://www.vi-hps.org