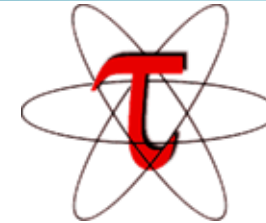


TAU Performance System®

14th Annual MVAPICH Users Group Meeting

2:30 pm – 3:00 pm ET, Monday, August 20th, 2026
The Ohio State University, Columbus, OH

Sameer Shende
Research Professor and Director,
Performance Research Laboratory, OACISS, University of Oregon
President and Director, ParaTools, Inc.
sameer@cs.uoregon.edu
https://tau.uoregon.edu/TAU_MUG26.pdf

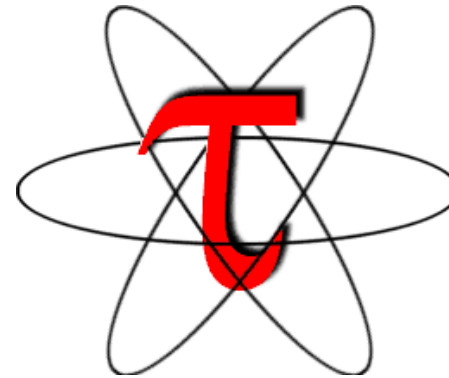


UNIVERSITY
OF OREGON

ParaTools

Acknowledgments

- The MVAPICH team The Ohio State University
 - <https://mvapich.cse.ohio-state.edu>
- TAU team at the University of Oregon
 - <https://tau.uoregon.edu>



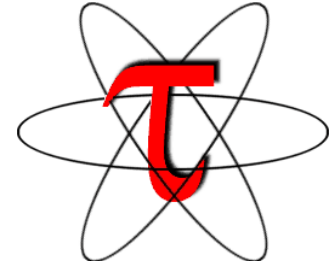
Motivation and Challenges

- With growing hardware complexity, it is getting harder to accurately measure and optimize the performance of our HPC and AI/ML workloads.
- TAU Performance System[®]:
 - Deliver a scalable, portable, performance evaluation toolkit for HPC and AI/ML workloads.
 - <https://tau.uoregon.edu>

Motivation: Improving Productivity

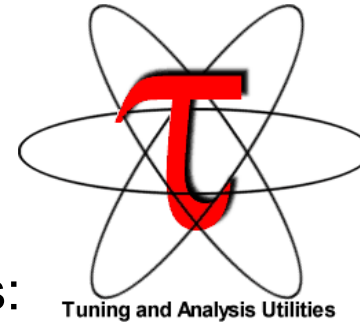
- TAU Performance System®:
 - Deliver a scalable, portable, performance evaluation toolkit for HPC and AI/ML workloads
 - <https://tau.uoregon.edu>

TAU Performance System[®]



- Tuning and Analysis Utilities (25+ year project)
- Comprehensive performance profiling and tracing
 - Integrated, scalable, flexible, portable
 - Targets all parallel programming/execution paradigms
- Integrated performance toolkit
 - Instrumentation, measurement, analysis, visualization
 - Widely-ported performance profiling / tracing system
 - Performance data management and data mining
 - Open source (BSD-style license)
 - Uses performance and control variables to interface with MVAPICH2
- Integrates with application frameworks
- <https://tau.uoregon.edu>

TAU Performance System®



UNIVERSITY
OF OREGON

ParaTools

- Versatile profiling and tracing toolkit that supports:
 - MPI, CUDA, ROCm, DPC++/SYCL (Level Zero), OpenCL, OpenMP (OpenMP Tools Interface for Target Offload), OpenSHMEM
- Scalable, portable, performance evaluation toolkit for HPC and AI/ML workloads that supports:
 - C++/C/DPC++, Fortran, Python
- Supports PAPI, Likwid for hardware performance counter information
- Instrumentation includes support for PETSc (Perfstubs), GPTL, PhiProf, XGC (CAMTIMERS), Kokkos, MPI, pthread, event-based sampling, GPU runtimes
- A single tool (tau_exec) is used to launch un-instrumented, un-modified binaries
- TAU's paraprof, pprof, perfexplorer for profile analysis; Vampir, Jumpshot, Perfetto.dev for traces
- <https://tau.uoregon.edu>

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? In kernels on GPUs and at the level of each statement inside GPU kernels.
- How many instructions are executed in these code regions?
Floating point, Level 1 and 2 *data cache misses*, hits, branches taken? What is the extent of vectorization for loops? Accessing counters on both CPUs and GPUs.
- How much time did my application spend waiting at a barrier in MPI collective operations?
- 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?
- How much energy does the application use in Joules? What is the peak power usage?
- What are the I/O characteristics of the code? What is the peak read and write *bandwidth* of individual calls, total volume?
- 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`

- Pthread: `% mpirun -np 16 tau_exec -T mpi,thread -ebs ./a.out`
- CUDA: `% mpirun -np 16 tau_exec -T cupti,mpi -cupti -ebs ./a.out`
- ROCm: `% mpirun -np 16 tau_exec -T rocm,mpi -rocm -ebs ./a.out`
- Intel Level Zero: `% mpirun -np 16 tau_exec -T l0,mpi -l0 -ebs ./a.out`
- Python: `% tau_python ./foo.py`

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;`

Perfetto.dev: `% tau_trace2json tau.trc tau.edf -chrome -ignoreatomic -o app.json`

Chrome browser: `chrome://tracing` (Load -> app.json) or Perfetto.dev

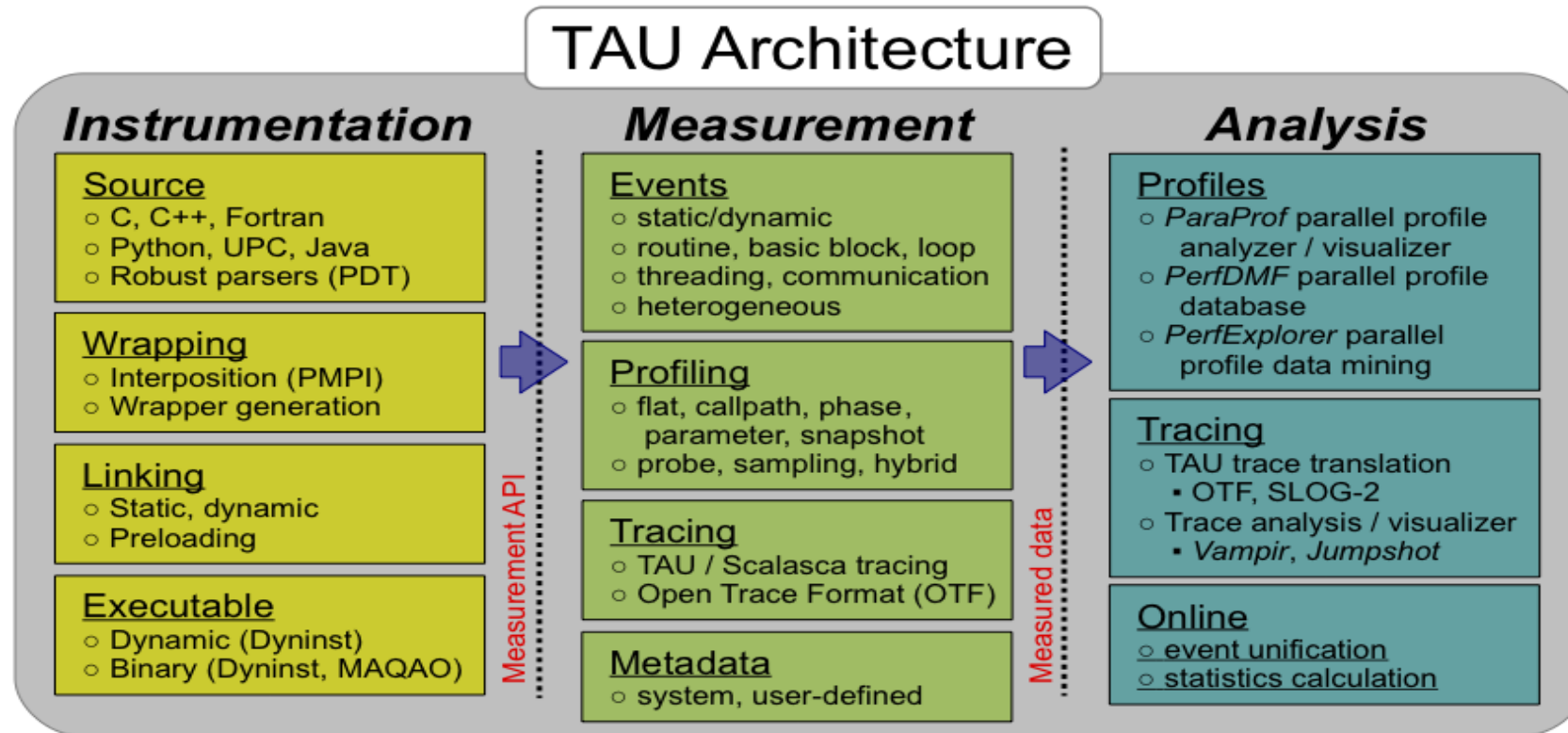
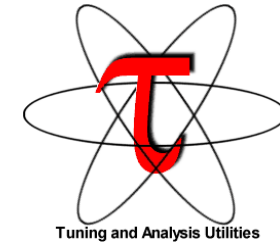
- Jumpshot: `tau2slog2 tau.trc tau.edf -o app.slog2; jumpshot app.slog2`

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, Spark
- Automatic instrumentation

Measurement and analysis support

- MPI (MVAPICH), OpenSHMEM, ARMCI, PGAS, DMAPP
- Supports Intel oneAPI compilers
- pthreads, OpenMP, OMPT interface, hybrid, other thread models
- GPU: OpenCL, oneAPI DPC++/SYCL (Level Zero), OpenACC, Kokkos, RAJA
- Parallel profiling and tracing

Analysis

- Parallel profile analysis (ParaProf), data mining (PerfExplorer)
- Performance database technology (TAUdb)
- 3D profile browser

Instrumentation

Add hooks in the code to perform measurements

- **Source instrumentation using a preprocessor**
 - Add timer start/stop calls in a copy of the source code.
 - Use Program Database Toolkit (PDT) for parsing source code.
 - Requires recompiling the code using TAU shell scripts (tau_cc.sh, tau_f90.sh)
 - Selective instrumentation (filter file) can reduce runtime overhead and narrow instrumentation focus.
- **Compiler-based instrumentation**
 - Use system compiler to add a special flag to insert hooks at routine entry/exit.
 - Requires recompiling using TAU compiler scripts (tau_cc.sh, tau_f90.sh...)
- **Runtime preloading of TAU's Dynamic Shared Object (DSO)**
 - No need to recompile code! Use `mpirun tau_exec ./app` with options.

TAU's Support for Runtime Systems

- *MPI*
 - PMPI profiling interface
 - MPI_T tools interface using performance and control variables
 - MPI Collective Sync time: time in an implicit barrier in MPI collective operations
- *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
- *Intel Level Zero*
 - Captures time spent in kernels on GPUs using oneAPI Level Zero
 - Captures time spent in Intel Level Zero runtime calls
- *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
- *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
- *Julia*
 - Julia instrumentation at the compiler/IR level to rewrite code, manual
 - Tracks time spent in Julia code

Examples of Multi-Level Instrumentation

MPI + OpenMP

MPI_T + PMPI + OMPT may be used to track MPI and OpenMP

MPI + CUDA

PMPI + CUPTI interfaces

OpenCL + ROCm

Rocprofiler + OpenCL instrumentation interfaces

Kokkos + OpenMP

Kokkos profiling API + OMPT to transparently track events

Kokkos + pthread + MPI

Kokkos + pthread wrapper interposition library + PMPI layer

Python + CUDA

Python + CUPTI + pthread profiling interfaces (e.g., Tensorflow, PyTorch)

MPI + OpenCL

PMPI + OpenCL profiling interfaces

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)

TAU Execution Command (tau_exec)

Uninstrumented execution

```
% mpirun -np 256 ./a.out
```

Track GPU operations

```
% mpirun -np 256 tau_exec -rocm ./a.out (-rocm_pc for PC sampling on GPU)
```

```
% mpirun -np 256 tau_exec -cupti ./a.out (-cupti_pc for PC sampling on GPU)
```

```
% mpirun -np 256 tau_exec -cupti -um ./a.out (for Unified Memory) (-cupti_pc for PC sampling on GPU)
```

```
% mpirun -np 256 tau_exec -l0 ./a.out (-l0_pc for PC sampling on GPU)
```

```
% mpirun -np 256 tau_exec -opencl ./a.out
```

```
% mpirun -np 256 tau_exec -openacc ./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 or LLVM compilers that support OMPT)

```
% 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 export TAU_METRICS=TIME,PAPI_L1_DCM... -ebs_resolution=<file | function | line>

export TAU_METRICS=ROCM_|L0|CUPTI_<GPU_metric> for access to GPU counters

Example: TAU Configuration on Aurora

```
% module use /soft/modulefiles; module load tau; ls $TAU/Makefile*  
/soft/perftools/tau/tau-2.35.2/x86_64/lib/Makefile.tau-level_zero-gptl-icpx-papi-ompt-mpi-python-pdt-openmp-l0metrics  
/soft/perftools/tau/tau-2.35.2/x86_64/lib/Makefile.tau-level_zero-icpx-papi-ompt-python-pdt-openmp-l0metrics  
/soft/perftools/tau/tau-2.35.2/x86_64/lib/Makefile.tau-level_zero-itnotify-gptl-icpx-papi-ompt-mpi-python-pdt-openmp-l0metrics
```

For an uninstrumented binary:

```
% mpiexec -n 4 gpu_tile_compact.sh tau_exec -l0 ./a.out
```

Picks the configuration represented by

```
/soft/perftools/tau/tau-2.35.2/x86_64/lib/Makefile.tau-level_zero-gptl-icpx-papi-ompt-mpi-python-pdt-openmp-l0metrics
```

To use OpenMP instrumentation:

```
% export TAU_OMPT_SUPPORT_LEVEL=full
```

```
% export OMP_NUM_THREADS=<N>
```

```
% mpiexec -n 4 gpu_tile_compact.sh tau_exec -T ompt,mpi -ompt -l0 ./a.out
```

```
% pprof -a | more
```

```
% paraprof
```

```
% paraprof --pack foo.ppk
```

```
# Copy it to your local machine and launch: % paraprof foo.ppk
```

Using TAU on Aurora at ALCF

```
% tar xf /soft/perftools/tau/tar/workshop.tgz; cd workshop; cat README
% qsub -I -l walltime=0:29:00 -l filesystems=home:flare -A ATPESC2026 -q ATPESC -l select=1 -X
% module use /soft/modulefiles; module load tau;

• Un-instrumented run with MPI          % mpiexec -np N ./a.out
• Profiling an un-instrumented application (use tau_exec -ebs with any of the following for event-based sampling):
• MPI without GPUs:                    % mpiexec -np N tau_exec -ebs ./a.out
• DPC++/SYCL with MPI:                 % mpiexec -np N gpu_tile_compact.sh tau_exec -T level_zero -l0 ./a.out
• # Use -l0_pc instead of -l0 for sampling on GPU. export TAU_METRICS=L0_ComputeBasic (see tau_l0_avail)
• DPC++/SYCL without MPI:             % tau_exec -T serial -l0 ./a.out
• OpenMP: export TAU_OMPT_SUPPORT_LEVEL=full
• OpenMP with MPI:                    % mpiexec -np N gpu_tile_compact.sh tau_exec tau_exec -ompt ./a.out
• OpenMP without MPI:                 % tau_exec -T serial -ompt ./a.out

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

Tracing:

Perfetto.dev:      % export TAU_TRACE=1; export TAU_FORMAT=perfetto; mpiexec -np N tau_exec ./a.out;

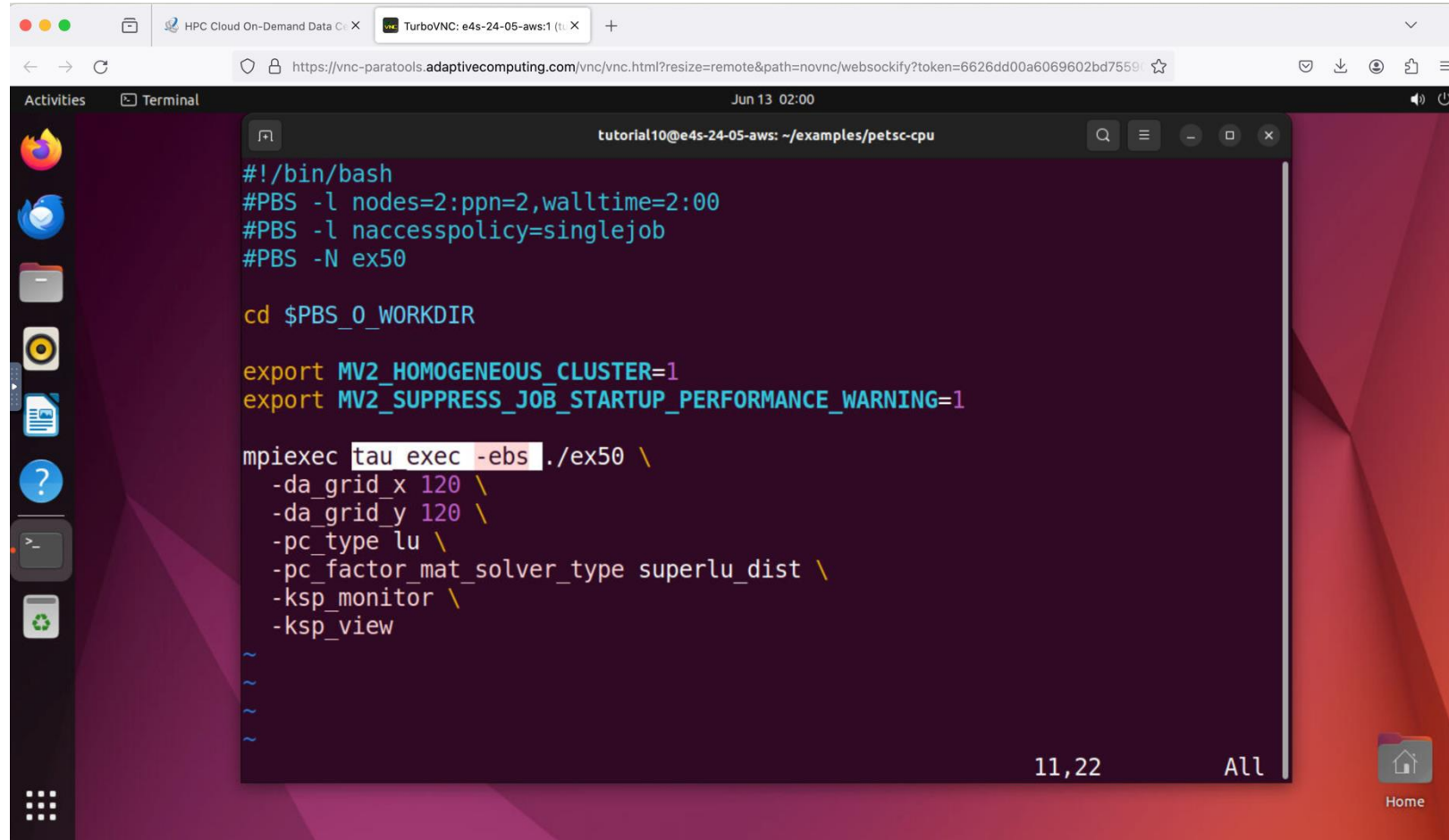
Jumpshot:          % export TAU_TRACE=1; mpiexec -np N gpu_tile_compact.sh tau_exec [Options] ./a.out;
                  % tau_treemerge.pl; tau2slog2 tau.trc tau.edf -o app.slog2; jumpshot app.slog2 &
```

Using TAU on Frontier at OLCF

```
tar xf /sw/frontier/ums/ums002/paratools/tar/workshop.tgz
cd workshop; cat README
salloc -n 4 -A <ACCOUNT> -N 1 -t 0:55:00 -q debug -C nvme --gpus 2
module load ums ums002 tau
srun -n 4 ./a.out
srun -n 4 tau_exec -rocm -ebs ./a.out
srun -n 4 tau_exec -rocm_pc -ebs ./a.out
export TAU_METRICS=ROCM_SQ_WAVES      # see rocprofv3 -L for a complete
list of counters
srun -n 4 tau_exec -rocm -ebs ./a.out

pprof -a | more
paraprof --pack foo.ppk
On your desktop: % paraprof foo.ppk
```

Using TAU on ParaTools Pro for E4S™ cloud images with MVAPICH on AWS, GCP, OCI, Azure, and IBM Cloud



The screenshot shows a terminal window titled 'tutorial10@e4s-24-05-aws: ~/examples/petsc-cpu'. The session includes the following commands and output:

```
#!/bin/bash
#PBS -l nodes=2:ppn=2,walltime=2:00
#PBS -l naccesspolicy=singlejob
#PBS -N ex50

cd $PBS_O_WORKDIR

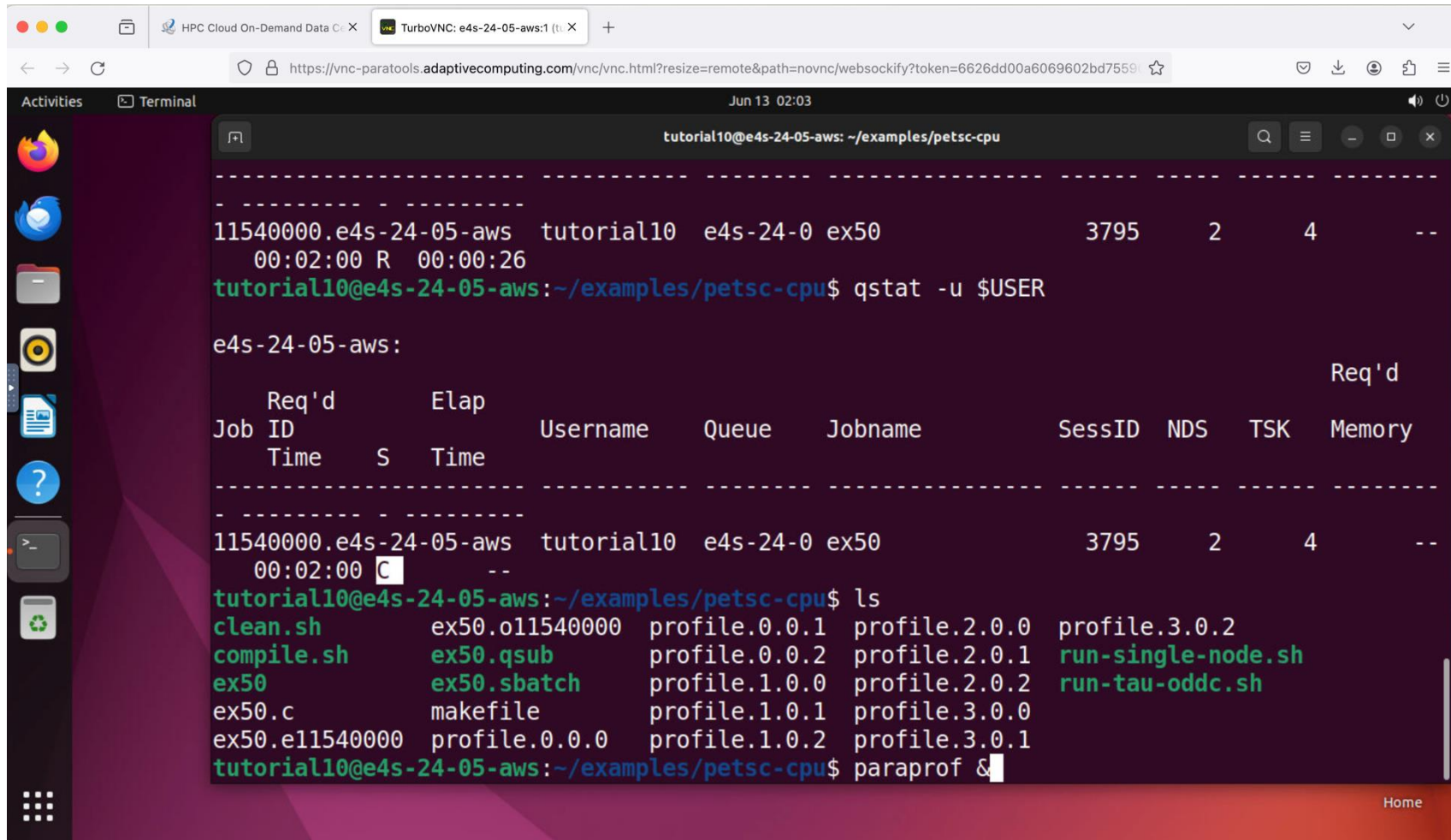
export MV2_HOMOGENEOUS_CLUSTER=1
export MV2_SUPPRESS_JOB_STARTUP_PERFORMANCE_WARNING=1

mpiexec tau exec -ebs ./ex50 \
  -da_grid_x 120 \
  -da_grid_y 120 \
  -pc_type lu \
  -pc_factor_mat_solver_type superlu_dist \
  -ksp_monitor \
  -ksp_view
```

At the bottom of the terminal, the output '11,22 All' is visible.

```
cd ~/examples/petsc-cpu
cat tau.sbatch
./compile.sh
sbatch tau.sbatch
pprof -a
paraprof &
```

TAU's ParaProf Profile Browser: Source Code Browser



```
tutorial10@e4s-24-05-aws: ~/examples/petsc-cpu
-----
11540000.e4s-24-05-aws tutorial10 e4s-24-0 ex50 3795 2 4 --
00:02:00 R 00:00:26
tutorial10@e4s-24-05-aws:~/examples/petsc-cpu$ qstat -u $USER

e4s-24-05-aws:

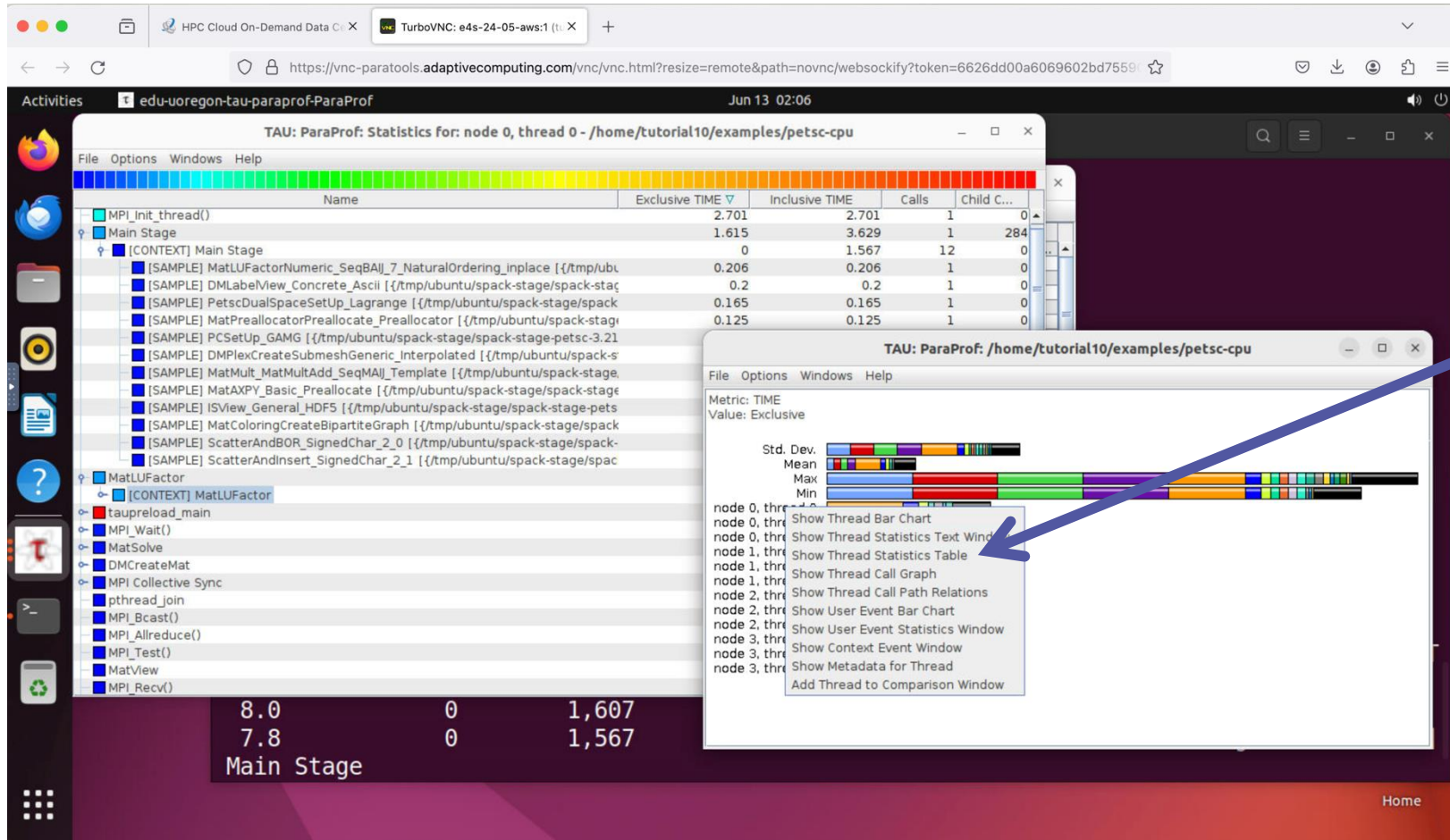
Req'd Elap
Job ID Time S Time Username Queue Jobname SessID NDS TSK Memory
-----
11540000.e4s-24-05-aws tutorial10 e4s-24-0 ex50 3795 2 4 --
00:02:00 C --

tutorial10@e4s-24-05-aws:~/examples/petsc-cpu$ ls
clean.sh ex50.o11540000 profile.0.0.1 profile.2.0.0 profile.3.0.2
compile.sh ex50.qsub profile.0.0.2 profile.2.0.1 run-single-node.sh
ex50 ex50.sbatch profile.1.0.0 profile.2.0.2 run-tau-oddc.sh
ex50.c makefile profile.1.0.1 profile.3.0.0
ex50.e11540000 profile.0.0.0 profile.1.0.2 profile.3.0.1
tutorial10@e4s-24-05-aws:~/examples/petsc-cpu$ paraprof &
```

sbatch tau.sbatch
queue -u \$USER

After it completes
ls
paraprof &

TAU's paraprof browser with PETSc performance profile



paraprof

Choose
Show thread statistics table by
right clicking on
node 0, thread 0.

Using pprof: TAU's text based profile browser

```
tutorial10@e4s-24-05-aws: ~/examples/petsc-cpu$ paraprof &
[1] 152885
tutorial10@e4s-24-05-aws:~/examples/petsc-cpu$ pprof -a | more
Reading Profile files in profile.*

NODE 0;CONTEXT 0;THREAD 0:

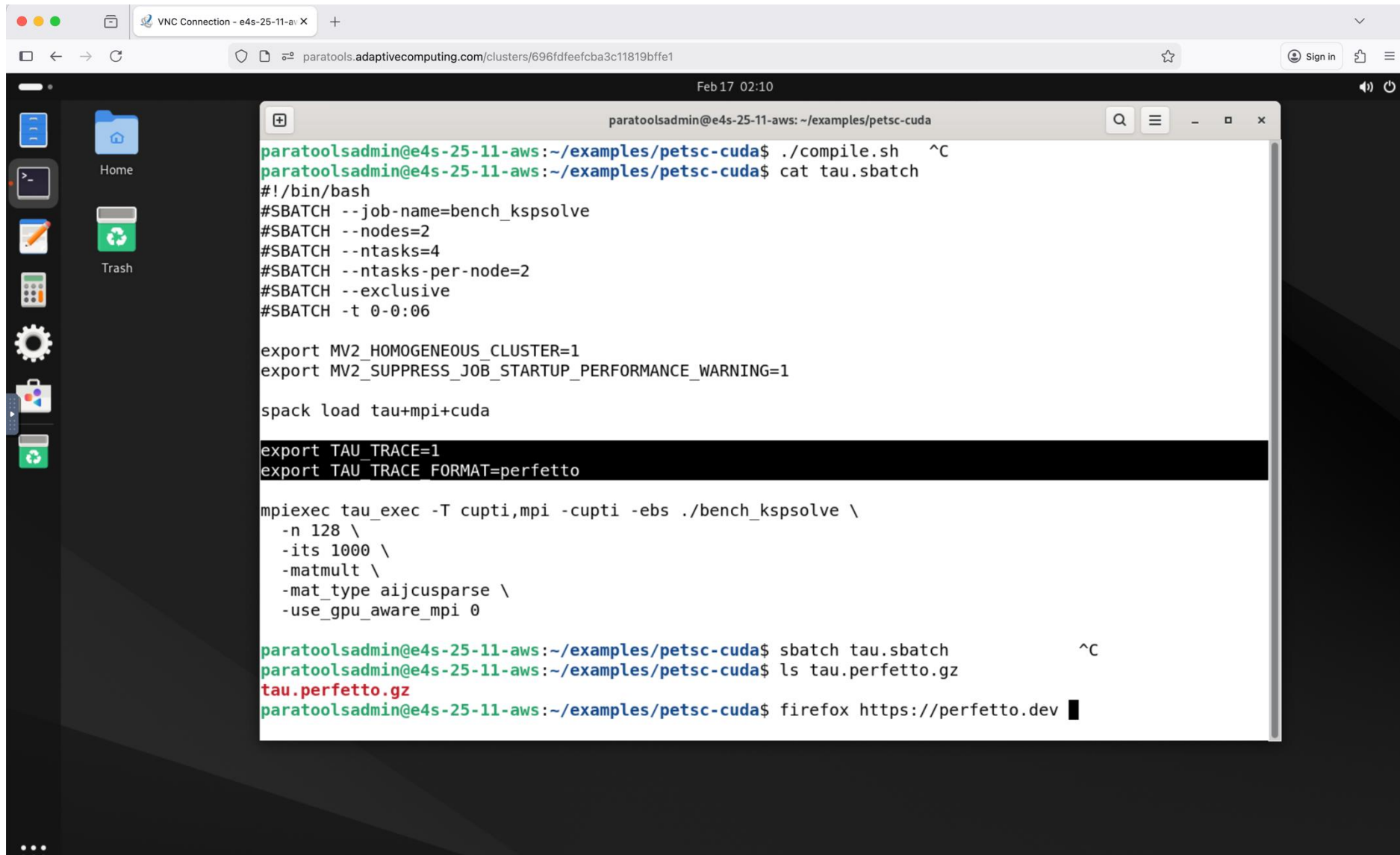
-----
%Time   Exclusive   Inclusive   #Call   #Subrs   Inclusive   Name
      msec     total msec                usec/call
-----
100.0    0.266       20,080      1        1   20080268 .TAU application
100.0      597       20,080      1       159   20080002 taupreload_main
65.2   13,080     13,095      1        2   13095901 MPI_Finalize()
18.1    1,615      3,629      1       284   3629490 Main Stage
13.4    2,700      2,700      1        0   2700776 MPI_Init_thread()
 8.7        1      1,756      1        27   1756523 PCSetUp
 8.7    1,512      1,754      1     10258   1754558 MatLUFactor
 8.0        0      1,607     44        0    36527 MatLUFactor => [CONTEXT
] MatLUFactor
 8.0        0      1,607     44        0    36527 [CONTEXT] MatLUFactor
 7.8        0      1,567     12        0   130617 Main Stage => [CONTEXT]
Main Stage
```

pprof -a | more

Here we see PETSc timers translated into TAU timers using the Perfstubs library.

No modification to the source, build system, or the binary!

PETSc CUDA with Perfetto Traces generated with TAU



The screenshot shows a VNC connection to a terminal window. The terminal prompt is `paratoolsadmin@e4s-25-11-aws: ~/examples/petsc-cuda`. The user runs `./compile.sh` and `cat tau.sbatch`. The `tau.sbatch` file contains the following content:

```
#!/bin/bash
#SBATCH --job-name=bench_kpsolve
#SBATCH --nodes=2
#SBATCH --ntasks=4
#SBATCH --ntasks-per-node=2
#SBATCH --exclusive
#SBATCH -t 0-0:06

export MV2_HOMOGENEOUS_CLUSTER=1
export MV2_SUPPRESS_JOB_STARTUP_PERFORMANCE_WARNING=1

spack load tau+mpi+cuda

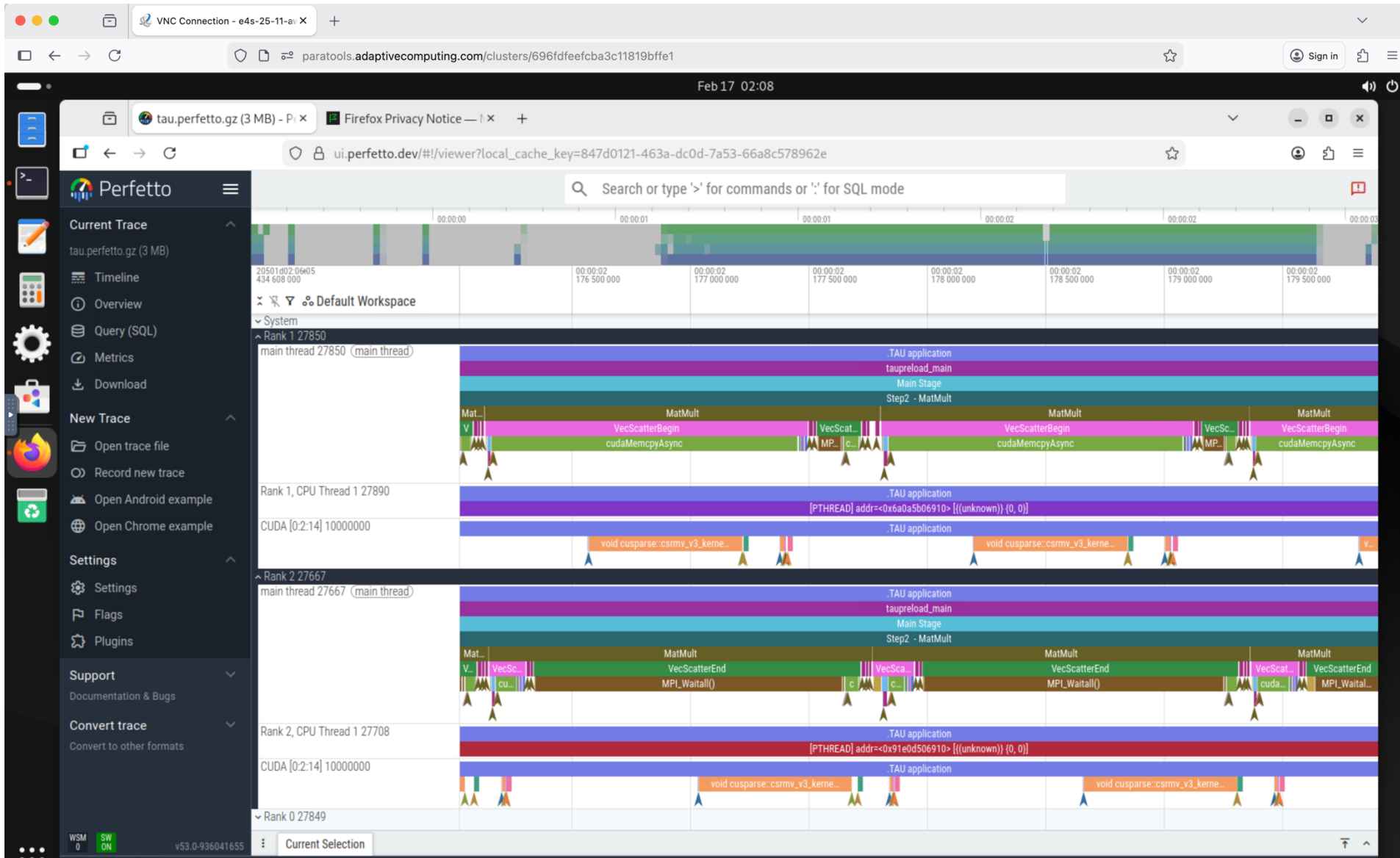
export TAU_TRACE=1
export TAU_TRACE_FORMAT=perfetto

mpiexec tau_exec -T cupti,mpi -cupti -ebs ./bench_kpsolve \
  -n 128 \
  -its 1000 \
  -matmult \
  -mat_type aijcusparse \
  -use_gpu_aware_mpi 0
```

The user then runs `sbatch tau.sbatch`, `ls tau.perfetto.gz`, and `firefox https://perfetto.dev`.

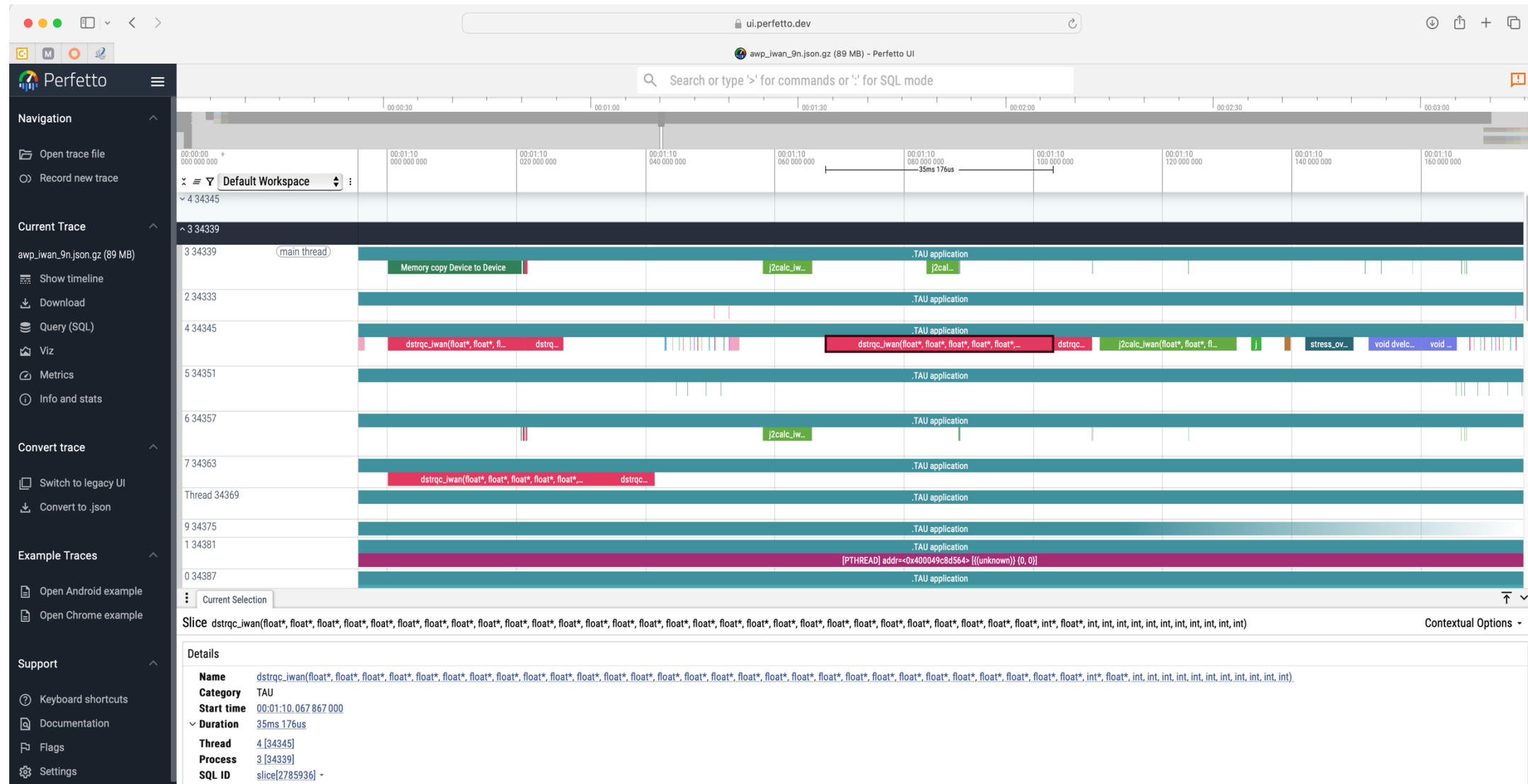
Add
`export TAU_TRACE=1`
`export TAU_TRACE_FORMAT=`
`perfetto`

Visualizing PETSC CUDA Traces with <https://Perfetto.dev>

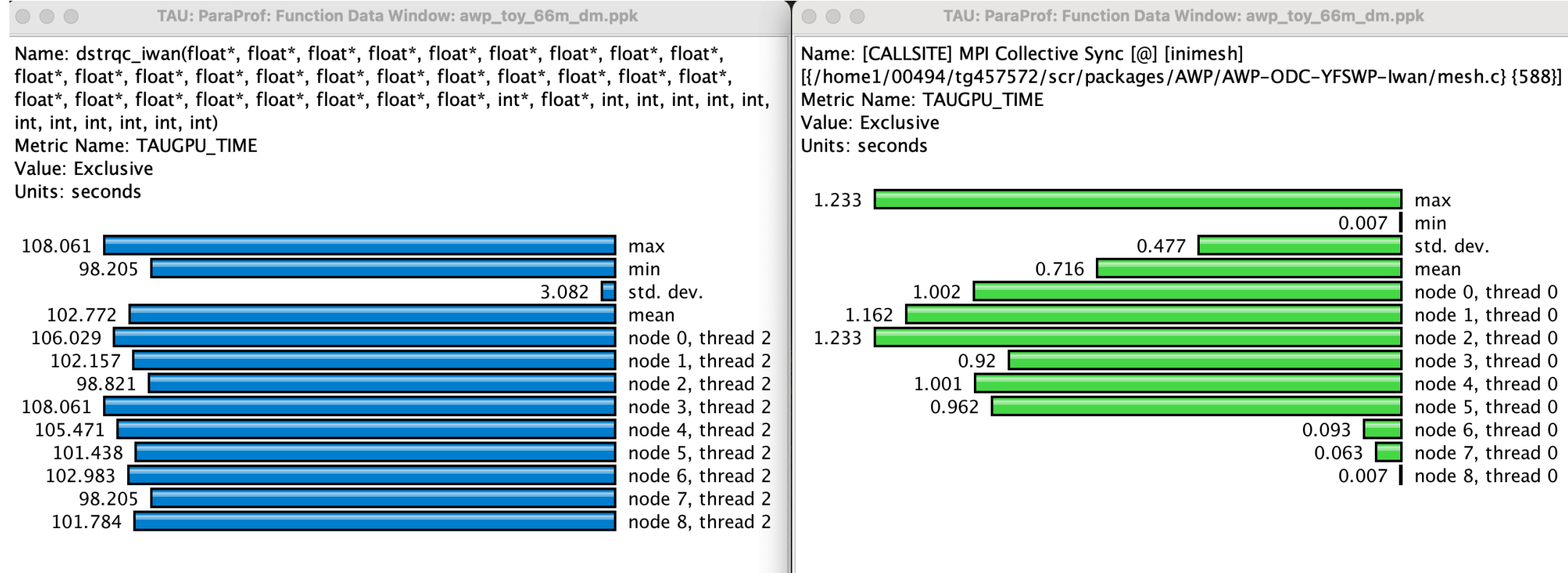


wasd
W = widen
S = Shrink
A = Left
D = Right

Visualizing AWP-ODC traces on Vista (TACC)

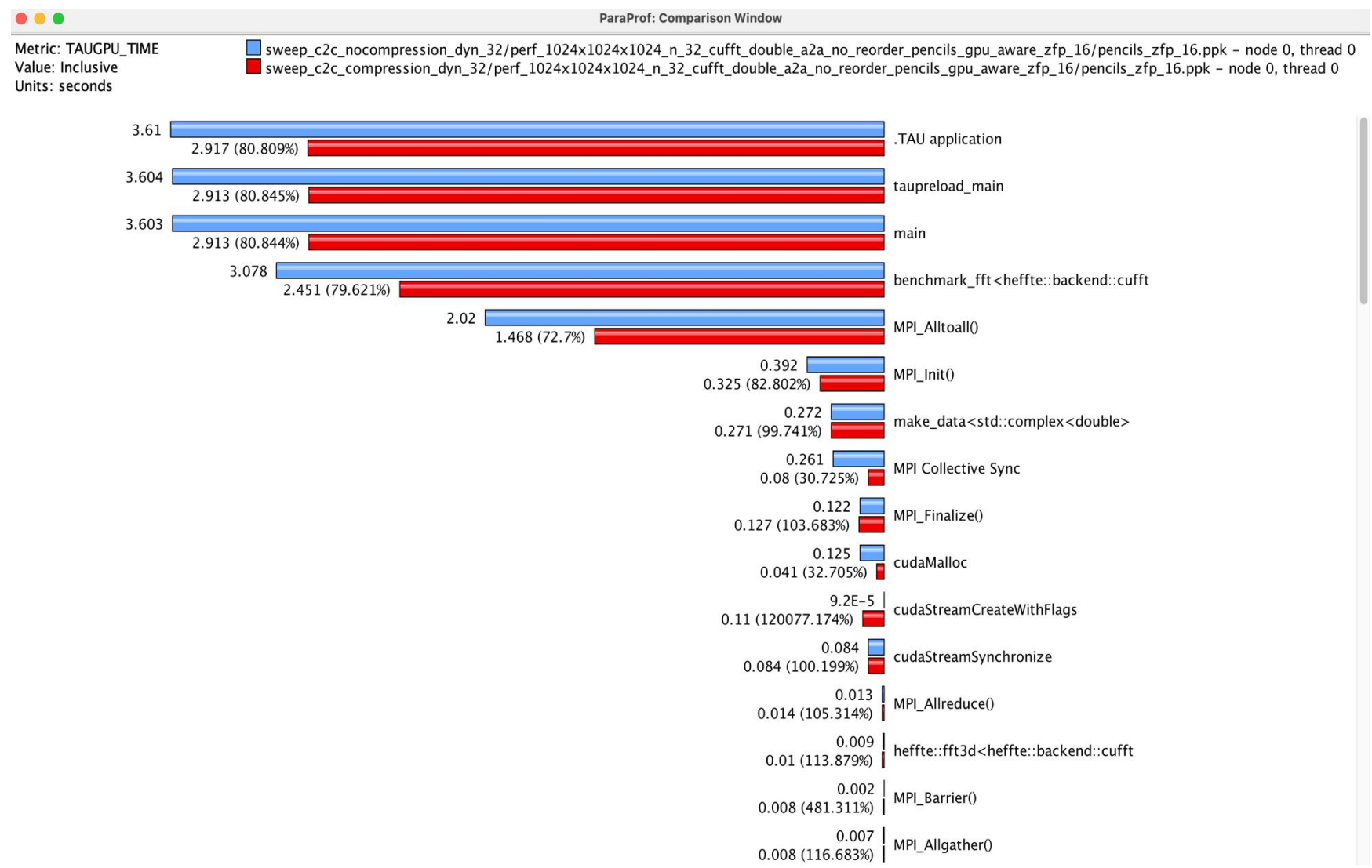


TAU Profiling Results for AWP-ODC on Vista@TACC



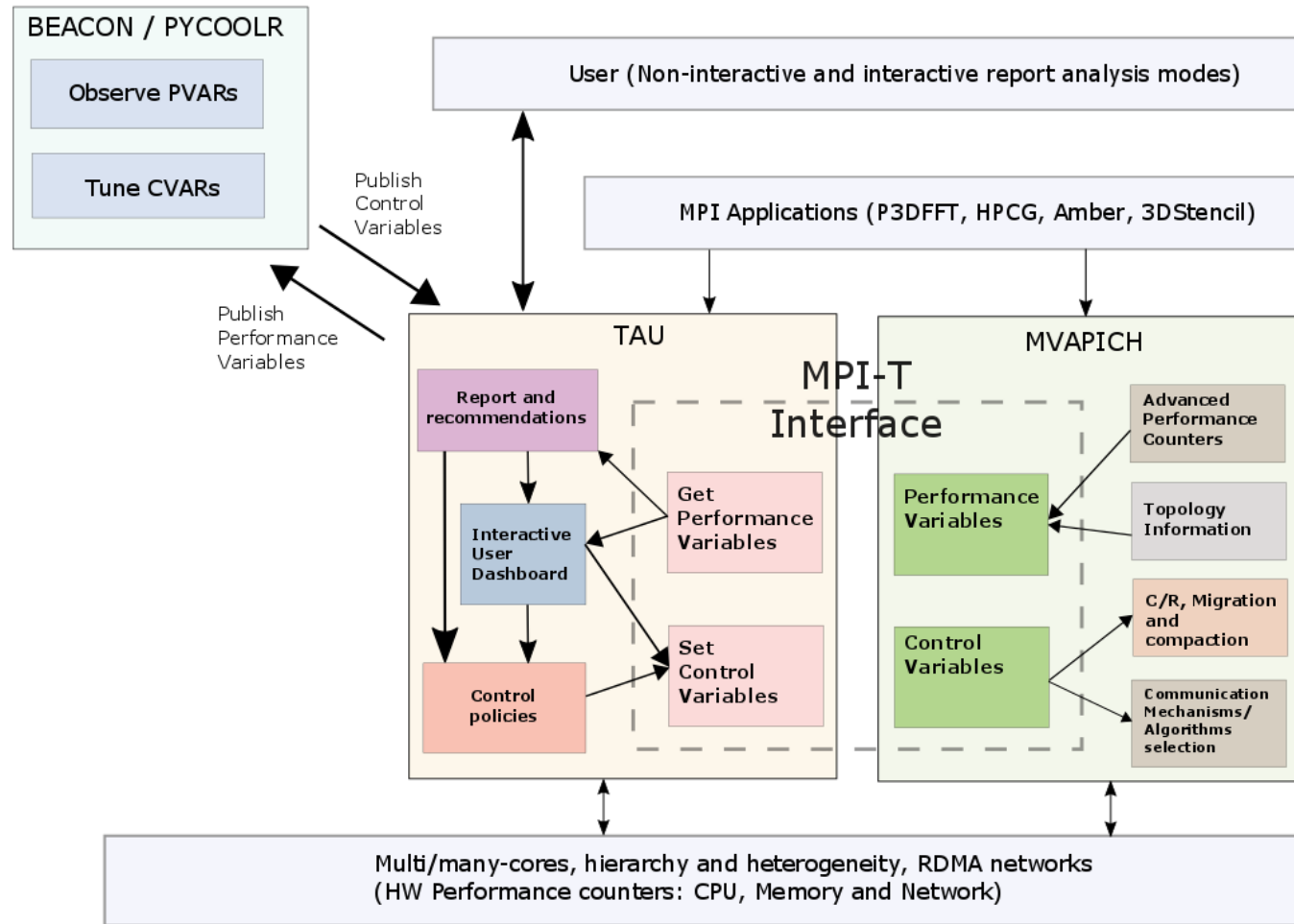
Profiling AWP-ODC Iwan testcase on Vista at TACC with TAU showing the distribution of performance of a key GPU kernel across multiple GH200 GPUs (left) and the time spent in a collective operation stalled at an MPI_Barrier call (right) across multiple MPI ranks.

TAU Results for HeFFTe using MVAPICH's Data Compression



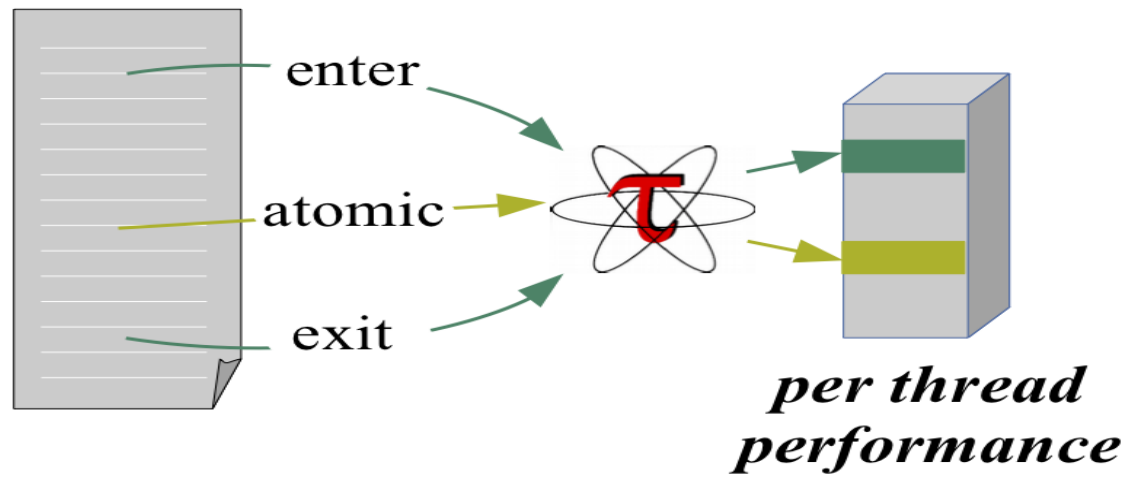
heFFTe performance data viewed in TAU's paraprof for sweep c2c testcase on 32 MPI ranks on Vista at TACC shows performance improvement with compression enabled in MVAPICH.

Integrating TAU with MVAPICH through MPI_T Interface

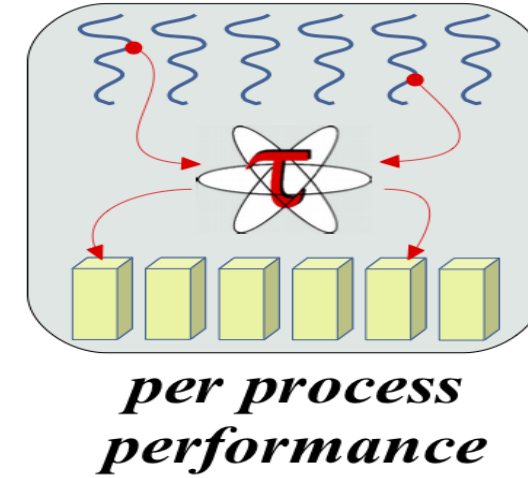


- Enhance existing support for MPI_T in MVAPICH2 to expose a richer set of performance and control variables
- Get and display MPI Performance Variables (PVARs) made available by the runtime in TAU
- Control the runtime's behavior via MPI Control Variables (CVARs)
- Add support to MVAPICH2 and TAU for interactive performance engineering sessions

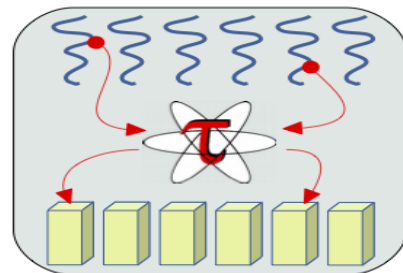
TAU Performance Measurement Model



enter/exit events
are "interval" events

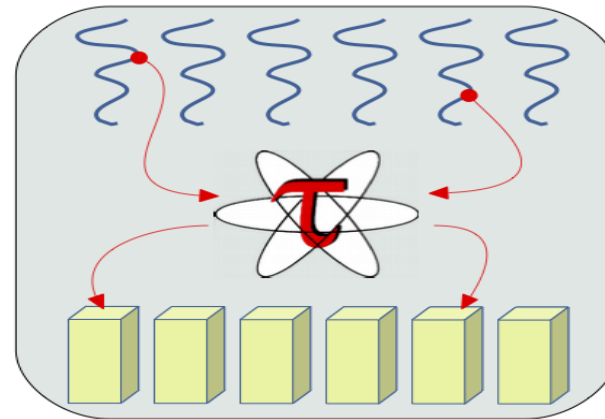


(in shared memory)



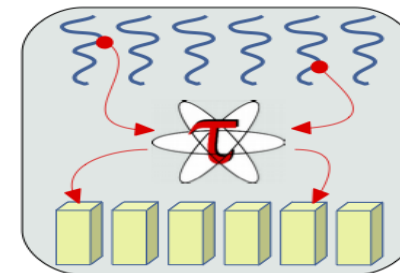
Process 0

...



Process i

...



Process N-1

application-wide
performance data

TAU Plugin Architecture

Extend TAU *event* interface for plugins

Events: *interval*, *atomic*

Specialized on event ID

Synchronous operation

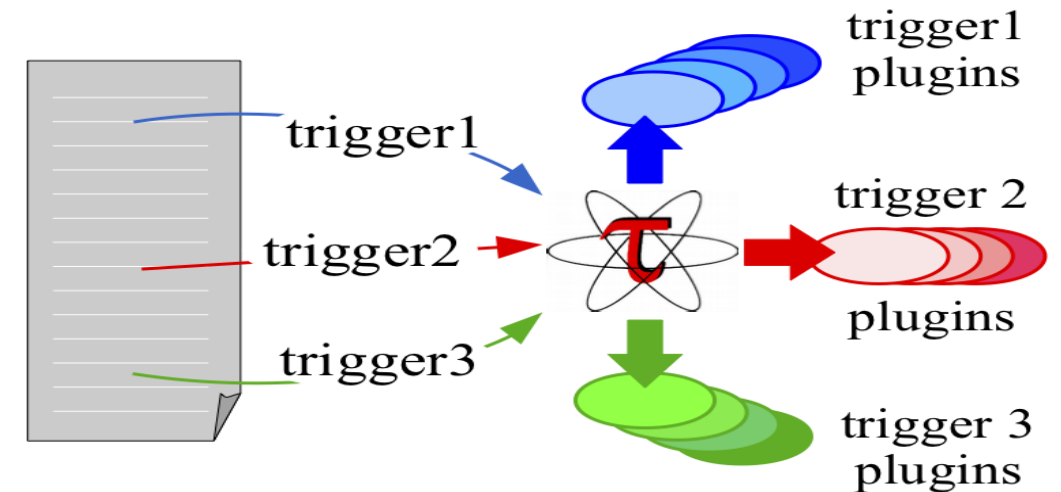
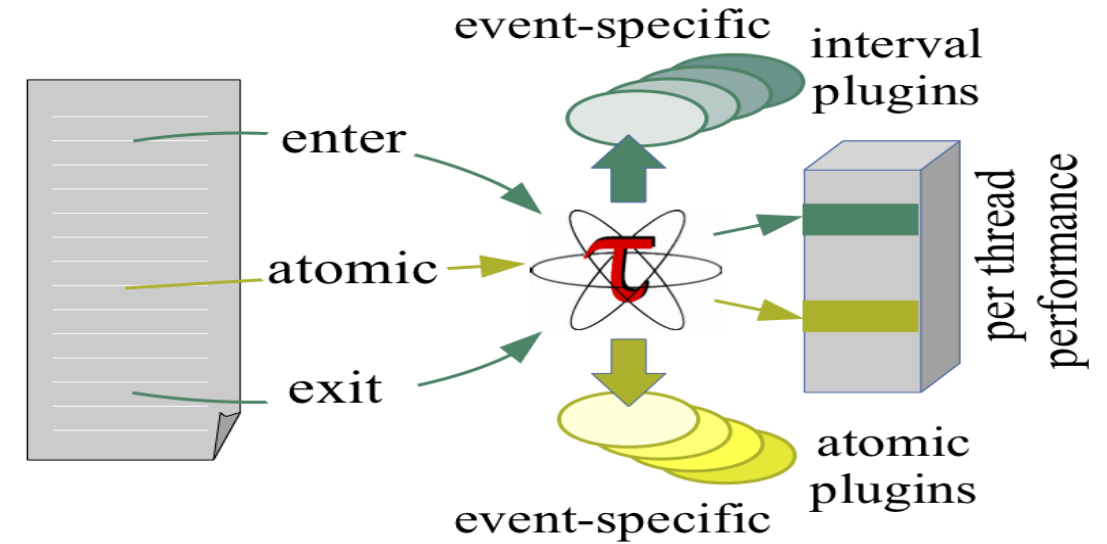
Create TAU interface for *trigger* plugins

Named trigger

Pass application data

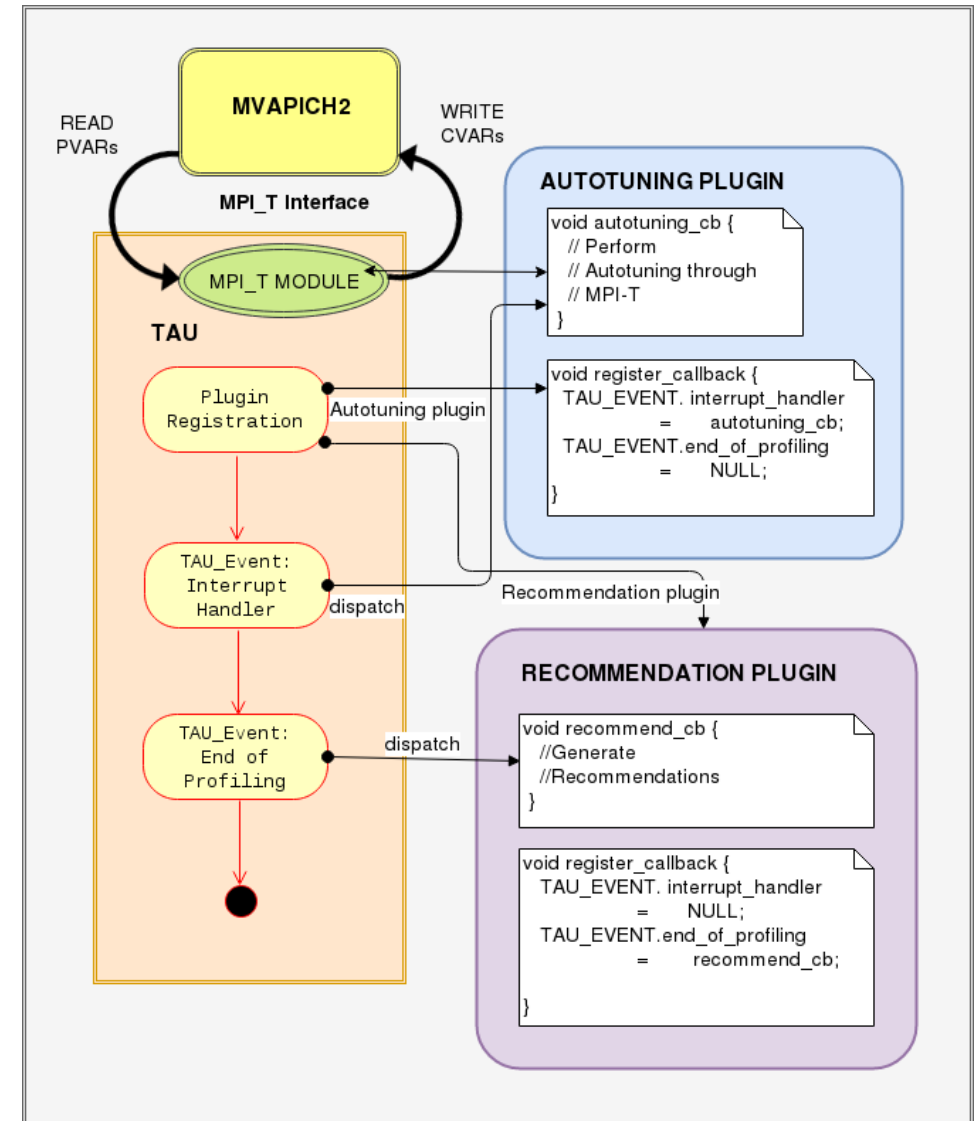
Synchronous

Asynchronous using agent plugin



Plugin-based Infrastructure for Non-Interactive Tuning

- TAU supports a *fully-customizable* plugin infrastructure based on callback event handler registration for salient states inside TAU:
 - Function Registration / Entry / Exit
 - Phase Entry / Exit
 - Atomic Event Registration / Trigger
 - Init / Finalize Profiling
 - Interrupt Handler
 - MPI_T*
- Application can define its own “trigger” states and associated plugins
 - Pass arbitrary data to trigger state plugins



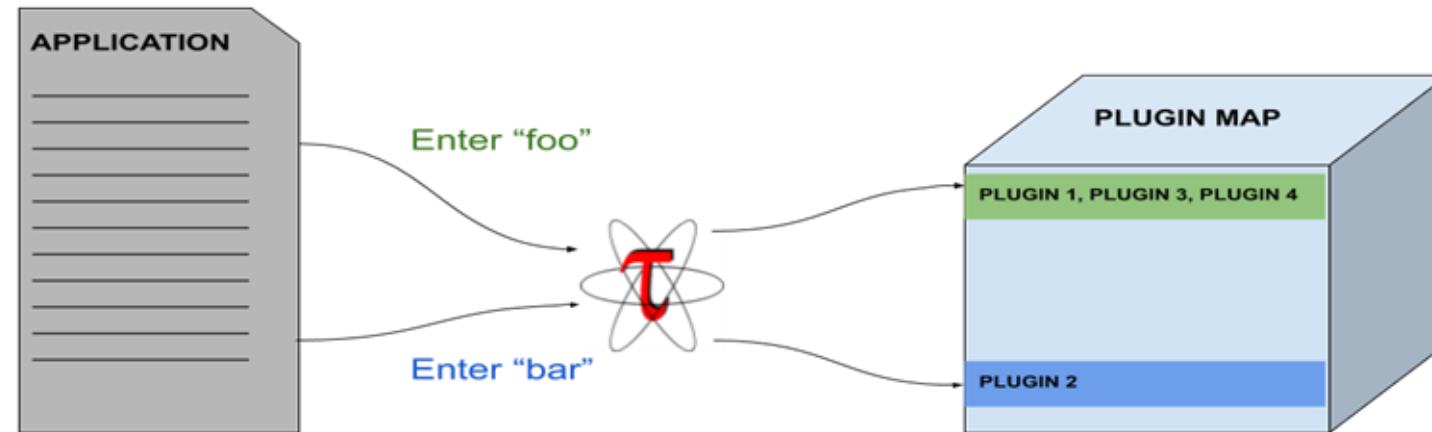
TAU Runtime Control of Plugin

- TAU defines a plugin API to deliver access control to the internal plugin map
- User can specify a regular expression to control plugins executed for a class of named states at runtime

Access to map on a process is serialized: application is expected to access map through main thread

TAU Customization

- TAU states can be *named* or *generic*
- TAU distinguishes named states in a way that allows for separation of occurrence of a state from the action associated with it
Function entry for “foo” and “bar” represent distinguishable states in TAU
- TAU maintains an internal map of a list of plugins associated with each state



PVARs Exposed by MVAPICH2

TAU: ParaProf Manager		
File Options Help		
Applications	TrialField	Value
Standard Applications	MPI_T PVAR[0]: mem_allocated	Current level of allocated memory within the MPI library
Default App	MPI_T PVAR[10]: mv2_num_2level_comm_success	Number of successful 2-level comm creations
Default Exp	MPI_T PVAR[11]: mv2_num_shmem_coll_calls	Number of times MV2 shared-memory collective calls were invoked
lulesh.ppk	MPI_T PVAR[12]: mpit_progress_poll	CH3 RDMA progress engine polling count
TIME	MPI_T PVAR[13]: mv2_smp_read_progress_poll	CH3 SMP read progress engine polling count
Default (jdbc:h2:/home)	MPI_T PVAR[14]: mv2_smp_write_progress_poll	CH3 SMP write progress engine polling count
	MPI_T PVAR[15]: mv2_smp_read_progress_poll_success	Unsuccessful CH3 SMP read progress engine polling count
	MPI_T PVAR[16]: mv2_smp_write_progress_poll_succ...	Unsuccessful CH3 SMP write progress engine polling count
	MPI_T PVAR[17]: rdma_ud_retransmissions	CH3 RDMA UD retransmission count
	MPI_T PVAR[18]: mv2_coll_bcast_binomial	Number of times MV2 binomial bcast algorithm was invoked
	MPI_T PVAR[19]: mv2_coll_bcast_scatter_doubling_all...	Number of times MV2 scatter+double allgather bcast algorithm was invoked
	MPI_T PVAR[1]: mem_allocated	Maximum level of memory ever allocated within the MPI library
	MPI_T PVAR[20]: mv2_coll_bcast_scatter_ring_allgather	Number of times MV2 scatter+ring allgather bcast algorithm was invoked
	MPI_T PVAR[21]: mv2_coll_bcast_scatter_ring_allgath...	Number of times MV2 scatter+ring allgather shm bcast algorithm was invoked
	MPI_T PVAR[22]: mv2_coll_bcast_shmem	Number of times MV2 shmem bcast algorithm was invoked
	MPI_T PVAR[23]: mv2_coll_bcast_knomial_internode	Number of times MV2 knomial internode bcast algorithm was invoked
	MPI_T PVAR[24]: mv2_coll_bcast_knomial_intranode	Number of times MV2 knomial intranode bcast algorithm was invoked
	MPI_T PVAR[25]: mv2_coll_bcast_mcast_internode	Number of times MV2 mcast internode bcast algorithm was invoked
	MPI_T PVAR[26]: mv2_coll_bcast_pipelined	Number of times MV2 pipelined bcast algorithm was invoked
	MPI_T PVAR[27]: mv2_coll_alltoall_inplace	Number of times MV2 in-place alltoall algorithm was invoked
	MPI_T PVAR[28]: mv2_coll_alltoall_bruck	Number of times MV2 brucks alltoall algorithm was invoked
	MPI_T PVAR[29]: mv2_coll_alltoall_rd	Number of times MV2 recursive-doubling alltoall algorithm was invoked
	MPI_T PVAR[2]: num_malloc_calls	Number of MPIT_malloc calls
	MPI_T PVAR[30]: mv2_coll_alltoall_sd	Number of times MV2 scatter-destination alltoall algorithm was invoked
	MPI_T PVAR[31]: mv2_coll_alltoall_pw	Number of times MV2 pairwise alltoall algorithm was invoked
	MPI_T PVAR[32]: mpit_alltoall_mv2_pw	Number of times MV2 pairwise alltoallv algorithm was invoked
	MPI_T PVAR[33]: mv2_coll_allreduce_shm_rd	Number of times MV2 shm rd allreduce algorithm was invoked
	MPI_T PVAR[34]: mv2_coll_allreduce_shm_rs	Number of times MV2 shm rs allreduce algorithm was invoked
	MPI_T PVAR[35]: mv2_coll_allreduce_shm_intra	Number of times MV2 shm intra allreduce algorithm was invoked
	MPI_T PVAR[36]: mv2_coll_allreduce_intra_p2p	Number of times MV2 intra p2p allreduce algorithm was invoked
	MPI_T PVAR[37]: mv2_coll_allreduce_2lvl	Number of times MV2 two-level allreduce algorithm was invoked
	MPI_T PVAR[38]: mv2_coll_allreduce_shmem	Number of times MV2 shmem allreduce algorithm was invoked
	MPI_T PVAR[39]: mv2_coll_allreduce_mcast	Number of times MV2 multicast-based allreduce algorithm was invoked
	MPI_T PVAR[3]: num_malloc_calls	Number of MPIT_malloc calls
	MPI_T PVAR[40]: mv2_reg_cache_hits	Number of registration cache hits
	MPI_T PVAR[41]: mv2_reg_cache_misses	Number of registration cache misses
	MPI_T PVAR[42]: mv2_vbuf_allocated	Number of VBUFs allocated
	MPI_T PVAR[43]: mv2_vbuf_allocated_array	Number of VBUFs allocated
	MPI_T PVAR[44]: mv2_vbuf_freed	Number of VBUFs freed
	MPI_T PVAR[45]: mv2_ud_vbuf_allocated	Number of UD VBUFs allocated
	MPI_T PVAR[46]: mv2_ud_vbuf_freed	Number of UD VBUFs freed
	MPI_T PVAR[47]: mv2_vbuf_free_attempts	Number of time we attempted to free VBUFs
	MPI_T PVAR[48]: mv2_vbuf_free_attempt_success_time	Average time for number of times we successfully freed VBUFs
	MPI_T PVAR[49]: mv2_vbuf_free_attempt_success_time	Average time for number of times we successfully freed VBUFs
	MPI_T PVAR[4]: num_memalign_calls	Number of MPIT_memalign calls
	MPI_T PVAR[50]: mv2_vbuf_allocate_time	Average time for number of times we allocated VBUFs
	MPI_T PVAR[51]: mv2_vbuf_allocate_time	Average time for number of times we allocated VBUFs

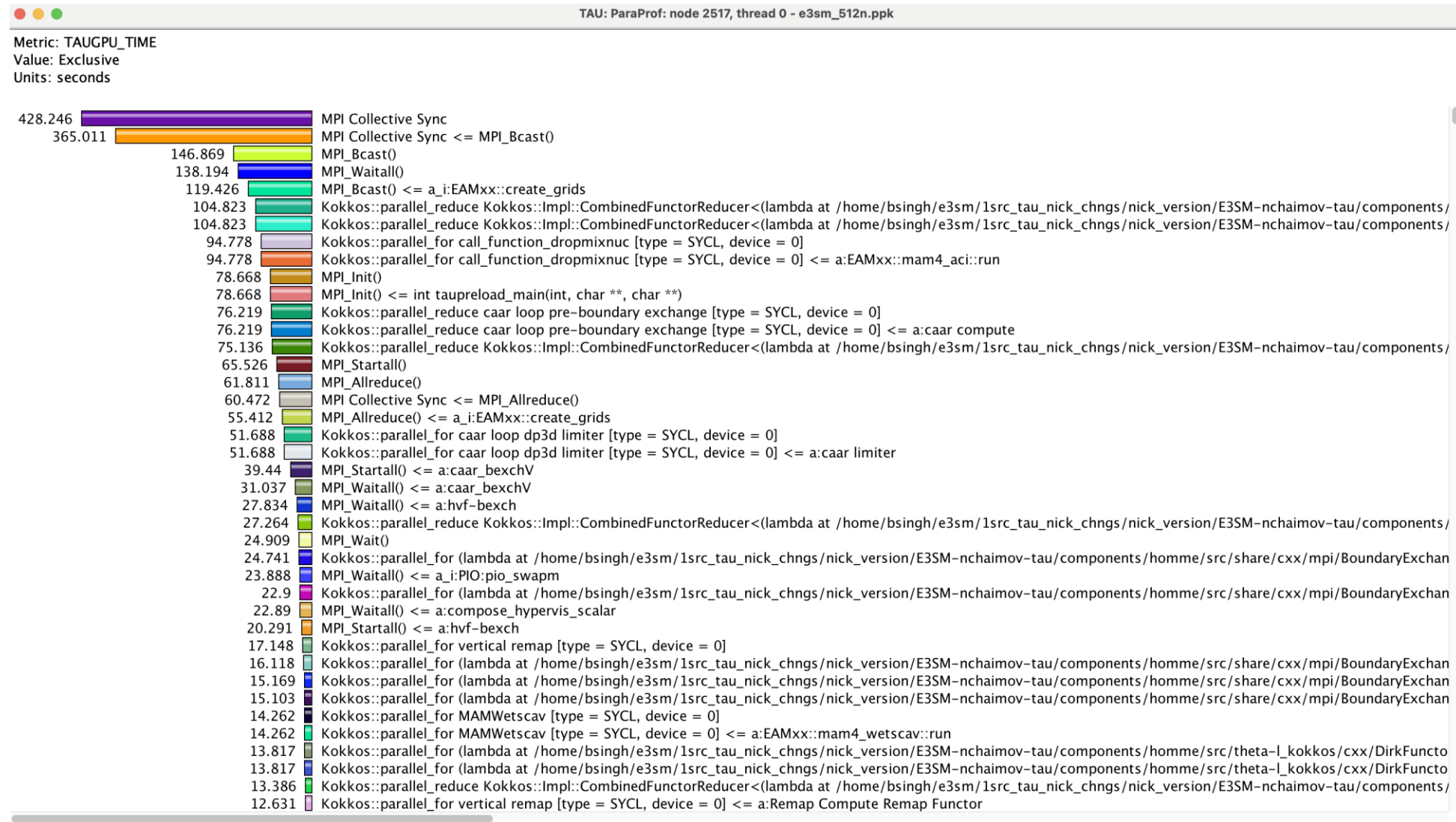
CVARs Exposed by MVAPICH2

TAU: ParaProf Manager		
File	Options	Help
Applications	TrialField	Value
Standard Applications	Local Time	2016-08-16T10:11:04-07:00
Default App	MPI Processor Name	cerberus.nic.uoregon.edu
Default Exp	MPIR_CVAR_ABORT_ON_LEAKED_HANDLES	If true, MPI will call MPI_Abort at MPI_Finalize if any MPI object handles have been leaked. For example,...
lulesh.ppk	MPIR_CVAR_ALLGATHERV_PIPELINE_MSG_SIZE	The smallest message size that will be used for the pipelined, large-message, ring algorithm in the MPI_...
TIME	MPIR_CVAR_ALLGATHER_LONG_MSG_SIZE	For MPI_Allgather and MPI_Allgatherv, the long message algorithm will be used if the send buffer size is ...
Default (jdb:h2:/home)	MPIR_CVAR_ALLGATHER_SHORT_MSG_SIZE	For MPI_Allgather and MPI_Allgatherv, the short message algorithm will be used if the send buffer size is...
	MPIR_CVAR_ALLREDUCE_SHORT_MSG_SIZE	the short message algorithm will be used if the send buffer size is <= this value (in bytes)
	MPIR_CVAR_ALLTOALL_MEDIUM_MSG_SIZE	the medium message algorithm will be used if the per-destination message size (sendcount*size(sendtyp...
	MPIR_CVAR_ALLTOALL_SHORT_MSG_SIZE	the short message algorithm will be used if the per-destination message size (sendcount*size(sendtype)) ...
	MPIR_CVAR_ALLTOALL_THROTTLE	max no. of irecv/isends posted at a time in some alltoall algorithms. Setting it to 0 causes all irecvs/isen...
	MPIR_CVAR_ASYNC_PROGRESS	If set to true, MPICH will initiate an additional thread to make asynchronous progress on all communicati...
	MPIR_CVAR_BCAST_LONG_MSG_SIZE	Let's define short messages as messages with size < MPIR_CVAR_BCAST_SHORT_MSG_SIZE, and mediu...
	MPIR_CVAR_BCAST_MIN_PROCS	Let's define short messages as messages with size < MPIR_CVAR_BCAST_SHORT_MSG_SIZE, and mediu...
	MPIR_CVAR_BCAST_SHORT_MSG_SIZE	Let's define short messages as messages with size < MPIR_CVAR_BCAST_SHORT_MSG_SIZE, and mediu...
	MPIR_CVAR_CH3_EAGER_MAX_MSG_SIZE	This cvar controls the message size at which CH3 switches from eager to rendezvous mode.
	MPIR_CVAR_CH3_ENABLE_HCOLL	If true, enable HCOLL collectives.
	MPIR_CVAR_CH3_INTERFACE_HOSTNAME	If non-NULL, this cvar specifies the IP address that other processes should use when connecting to this pr...
	MPIR_CVAR_CH3_NOLOCAL	If true, force all processes to operate as though all processes are located on another node. For example,...
	MPIR_CVAR_CH3_ODD_EVEN_CLIQUES	If true, odd procs on a node are seen as local to each other, and even procs on a node are seen as local t...
	MPIR_CVAR_CH3_PORT_RANGE	The MPIR_CVAR_CH3_PORT_RANGE environment variable allows you to specify the range of TCP ports ...
	MPIR_CVAR_CH3_RMA_ACC_IMMED	Use the immediate accumulate optimization
	MPIR_CVAR_CH3_RMA_GC_NUM_COMPLETED	Threshold for the number of completed requests the runtime finds before it stops trying to find more co...
	MPIR_CVAR_CH3_RMA_GC_NUM_TESTED	Threshold for the number of RMA requests the runtime tests before it stops trying to check more reques...
	MPIR_CVAR_CH3_RMA_LOCK_IMMED	Issue a request for the passive target RMA lock immediately. Default behavior is to defer the lock reque...
	MPIR_CVAR_CH3_RMA_MERGE_LOCK_OP_UNLOCK	Enable/disable an optimization that merges lock, op, and unlock messages, for single-operation passive ta...
	MPIR_CVAR_CH3_RMA_NREQUEST_NEW_THRESHOLD	Threshold for the number of new requests since the last attempt to complete pending requests. Higher ...
	MPIR_CVAR_CH3_RMA_NREQUEST_THRESHOLD	Threshold at which the RMA implementation attempts to complete requests while completing RMA oper...
	MPIR_CVAR_CHOP_ERROR_STACK	If >0, truncate error stack output lines this many characters wide. If 0, do not truncate, and if <0 use a ...
	MPIR_CVAR_COLL_ALIAS_CHECK	Enable checking of aliasing in collective operations
	MPIR_CVAR_COMM_SPLIT_USE_QSORT	Use qsort(3) in the implementation of MPI_Comm_split instead of bubble sort.
	MPIR_CVAR_CTXID_EAGER_SIZE	The MPIR_CVAR_CTXID_EAGER_SIZE environment variable allows you to specify how many words in th...
	MPIR_CVAR_DEBUG_HOLD	If true, causes processes to wait in MPI_Init and MPI_Initthread for a debugger to be attached. Once the ...
	MPIR_CVAR_DEFAULT_THREAD_LEVEL	Sets the default thread level to use when using MPI_INIT.
	MPIR_CVAR_DUMP_PROVIDERS	If true, dump provider information at init
	MPIR_CVAR_ENABLE_COLL_FT_RET	DEPRECATED! Will be removed in MPICH-3.2 Collectives called on a communicator with a failed process...
	MPIR_CVAR_ENABLE_SMP_ALLREDUCE	Enable SMP aware allreduce.
	MPIR_CVAR_ENABLE_SMP_BARRIER	Enable SMP aware barrier.
	MPIR_CVAR_ENABLE_SMP_BCAST	Enable SMP aware broadcast (See also: MPIR_CVAR_MAX_SMP_BCAST_MSG_SIZE)
	MPIR_CVAR_ENABLE_SMP_COLLECTIVES	Enable SMP aware collective communication.
	MPIR_CVAR_ENABLE_SMP_REDUCE	Enable SMP aware reduce.
	MPIR_CVAR_ERROR_CHECKING	If true, perform checks for errors, typically to verify valid inputs to MPI routines. Only effective when M...
	MPIR_CVAR_GATHERV_INTER_SSEND_MIN_PROCS	Use Ssend (synchronous send) for intercommunicator MPI_Gatherv if the "group B" size is >= this value....
	MPIR_CVAR_GATHER_INTER_SHORT_MSG_SIZE	use the short message algorithm for intercommunicator MPI_Gather if the send buffer size is < this value...
	MPIR_CVAR_GATHER_VSMALL_MSG_SIZE	use a temporary buffer for intracommunicator MPI_Gather if the send buffer size is < this value (in bytes...
	MPIR_CVAR_IBA_EAGER_THRESHOLD	0 (old) -> 204800 (new), This set the switch point between eager and rendezvous protocol
	MPIR_CVAR_MAX_INLINE_SIZE	This set the maximum inline size for data transfer
	MPIR_CVAR_MAX_SMP_ALLREDUCE_MSG_SIZE	Maximum message size for which SMP-aware allreduce is used. A value of '0' uses SMP-aware allreduce ...

Using MVAPICH2 and TAU with Multiple CVARs

- To set CVARs or read PVARs using TAU for an uninstrumented binary:
% export TAU_TRACK_MPI_T_PVARS=1
% export TAU_MPI_T_CVAR_METRICS=
 MPIR_CVAR_VBUF_POOL_REDUCED_VALUE[1],
 MPIR_CVAR_IBA_EAGER_THRESHOLD
% export TAU_MPI_T_CVAR_VALUES=32,64000
% export PATH=/path/to/tau/x86_64/bin:\$PATH
% mpirun -np 1024 *tau_exec -T mvapich,mpit* ./a.out
% paraprof

MPI Collective Sync in TAU: Time wasted in barrier



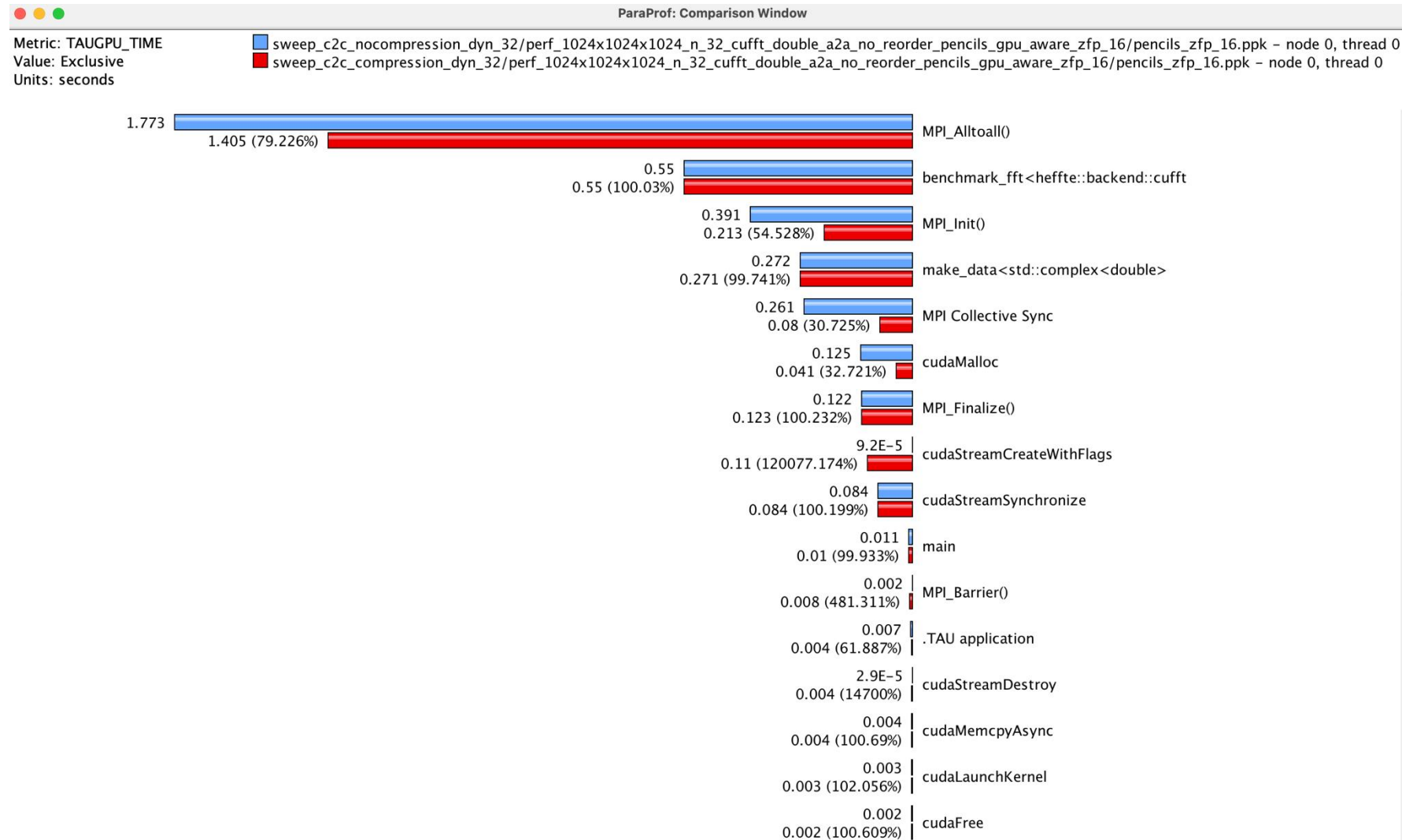
MPI Collective Sync in TAU: Time wasted in an implicit barrier

TAU: ParaProf: Call Path Data n,c,t, 2517,0,0 - e3sm_512n.ppk

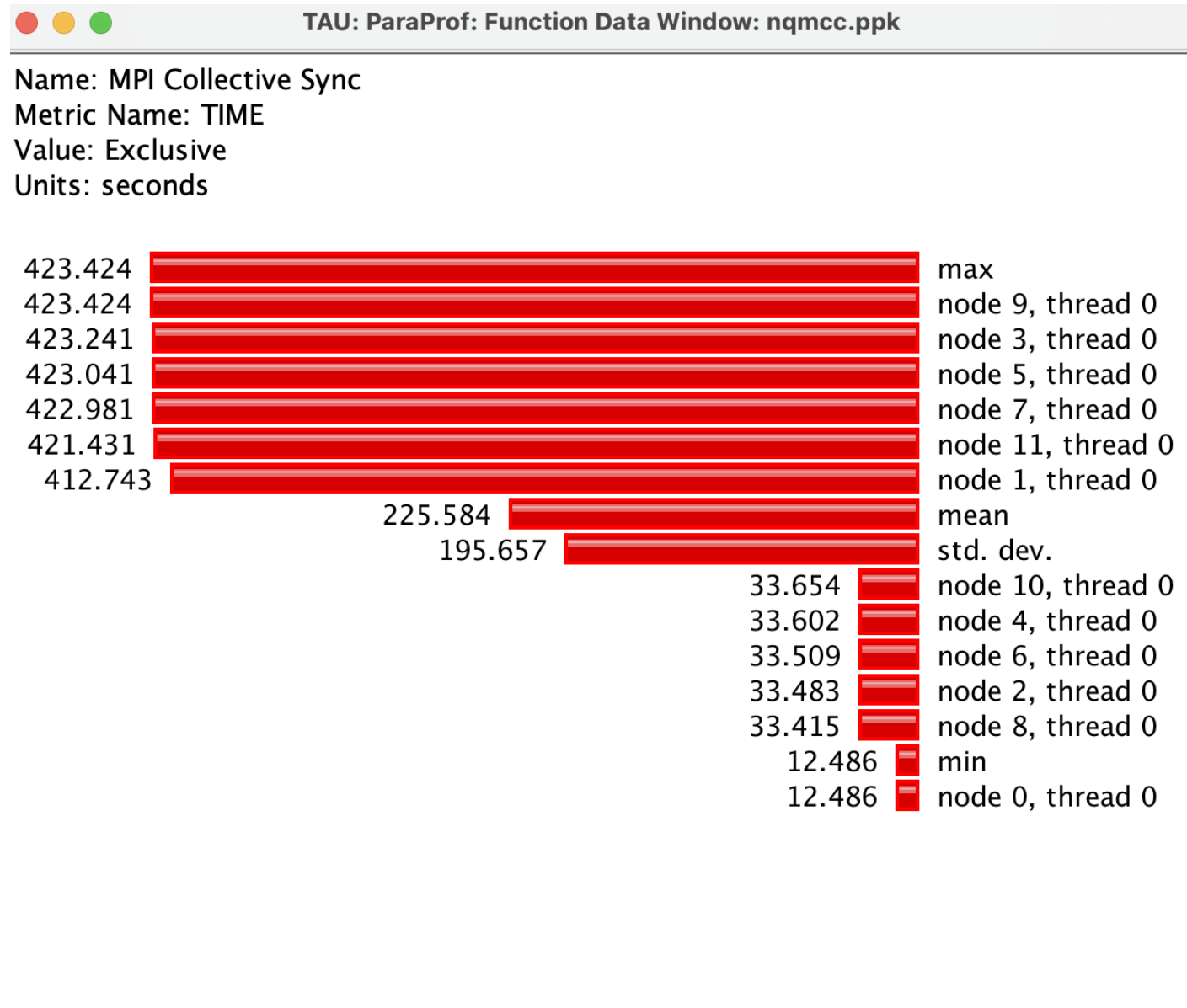
Metric Name: TAUGPU_TIME
Sorted By: Exclusive
Units: seconds

Exclusive	Inclusive	Calls/Tot.Calls	Name[id]
365.011	365.011	1292216/1610353	MPI_Bcast()
3.0E-4	3.0E-4	1/1610353	MPI_Scan()
60.472	60.472	317955/1610353	MPI_Allreduce()
1.073	1.073	71/1610353	MPI_Reduce()
0.565	0.565	5/1610353	MPI_Allgather()
0.333	0.333	25/1610353	MPI_Comm_dup()
0.573	0.573	51/1610353	MPI_Gather()
0.173	0.173	20/1610353	MPI_Scatterv()
0.046	0.046	9/1610353	MPI_Allgatherv()
--> 428.246	428.246	1610353	MPI Collective Sync
0.046	0.974	519/1292216	l_i:PIO:pio_inq_varid_file_vid
0.01	0.028	122/1292216	l_i:PIO:get_var_0d_double
0.839	1.705	9864/1292216	PIO:PIOc_write_darray
0.203	0.22	6/1292216	l_i:comp_init
4.4E-4	0.035	5/1292216	PIO:put_var1_double
0.053	0.148	630/1292216	l_i:PIO:pio_inq_vardimid_file_vid
0.006	0.036	67/1292216	CPL:seq_timemgr_clockInit
2.865	6.205	25889/1292216	a_i:EAMxx::initialize_fields
1.1E-4	0.036	1/1292216	CPL:seq_hist_write
0.013	0.08	154/1292216	l_i:PIO:get_var_1d_int
0.007	0.031	86/1292216	CPL:comp_init_cc_atm2
7.6E-4	0.001	9/1292216	i:PIO:pio_inq_varndims_file_vid
0.003	0.007	38/1292216	CPL:comp_init_cc_esp
3.4E-4	7.3E-4	3/1292216	i:cice_pice_readLBUB_UB_readpio
1.82	4.426	108/1292216	i_i:cice_mct_init
0.156	6.196	1438/1292216	CPL:o2c_infoexch

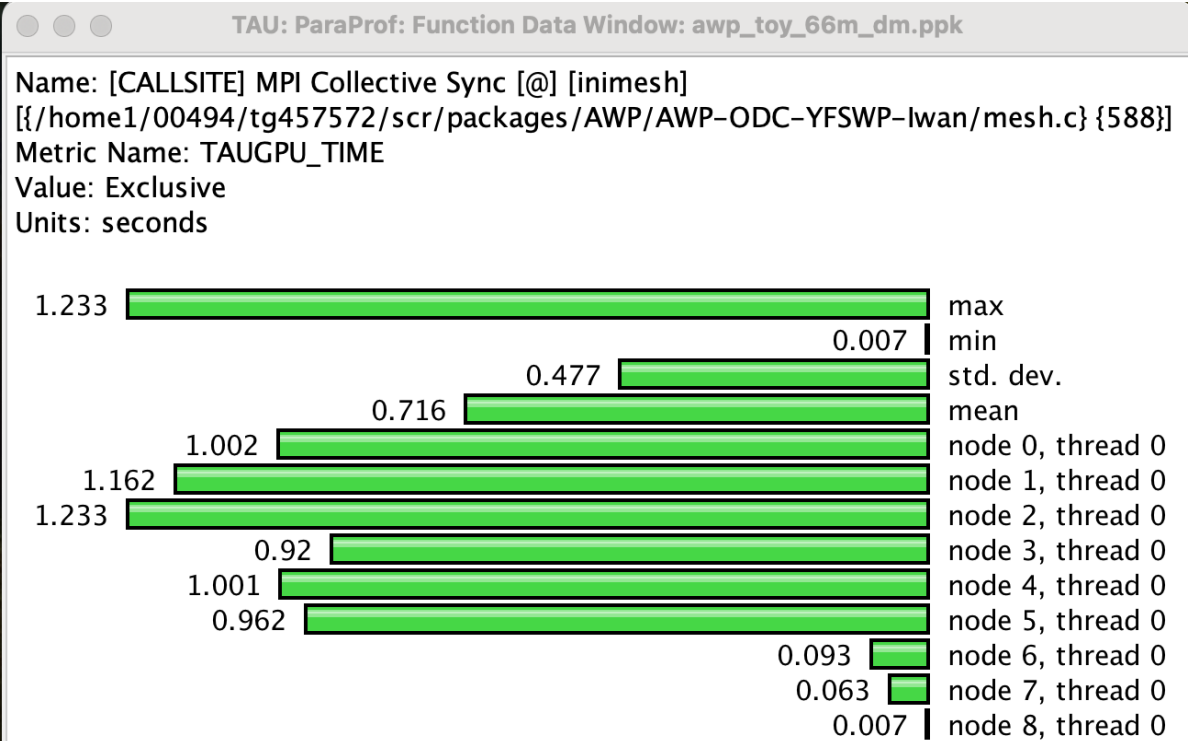
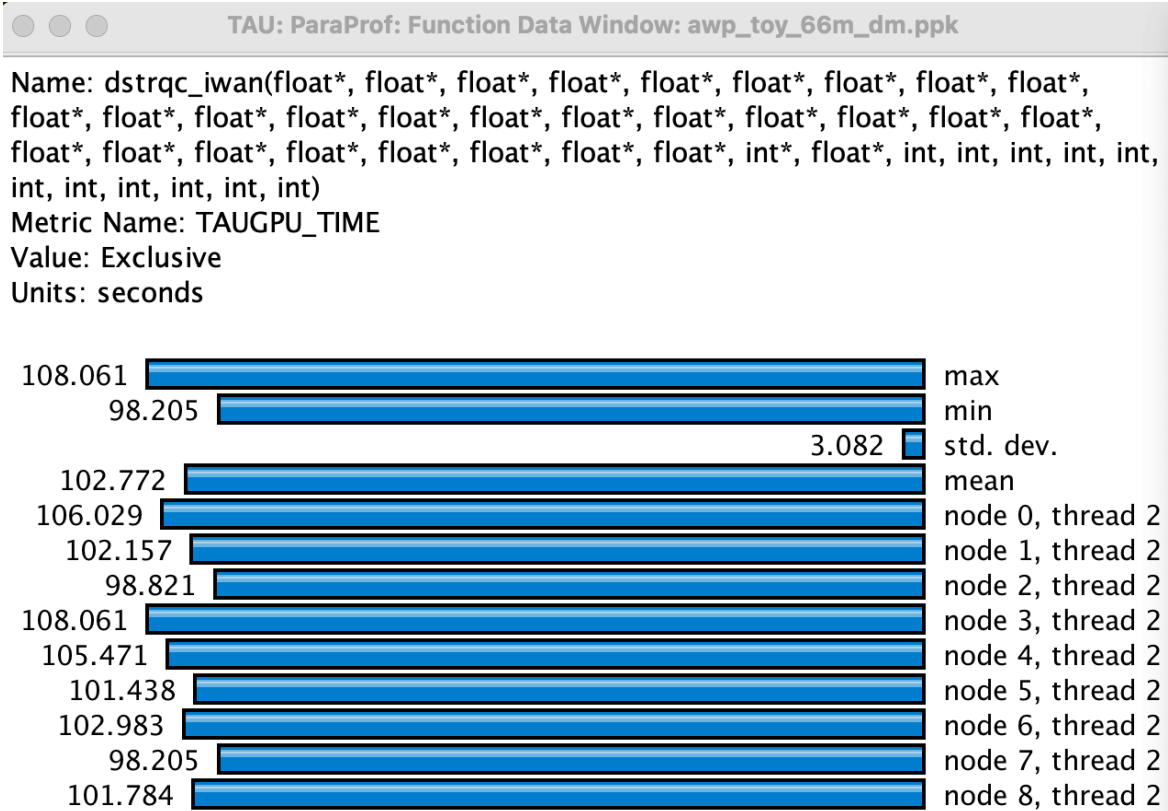
HeFFTe: Comparing collective sync time with TAU's comparison window



MPI Collective Sync in TAU: Time wasted in an implicit barrier

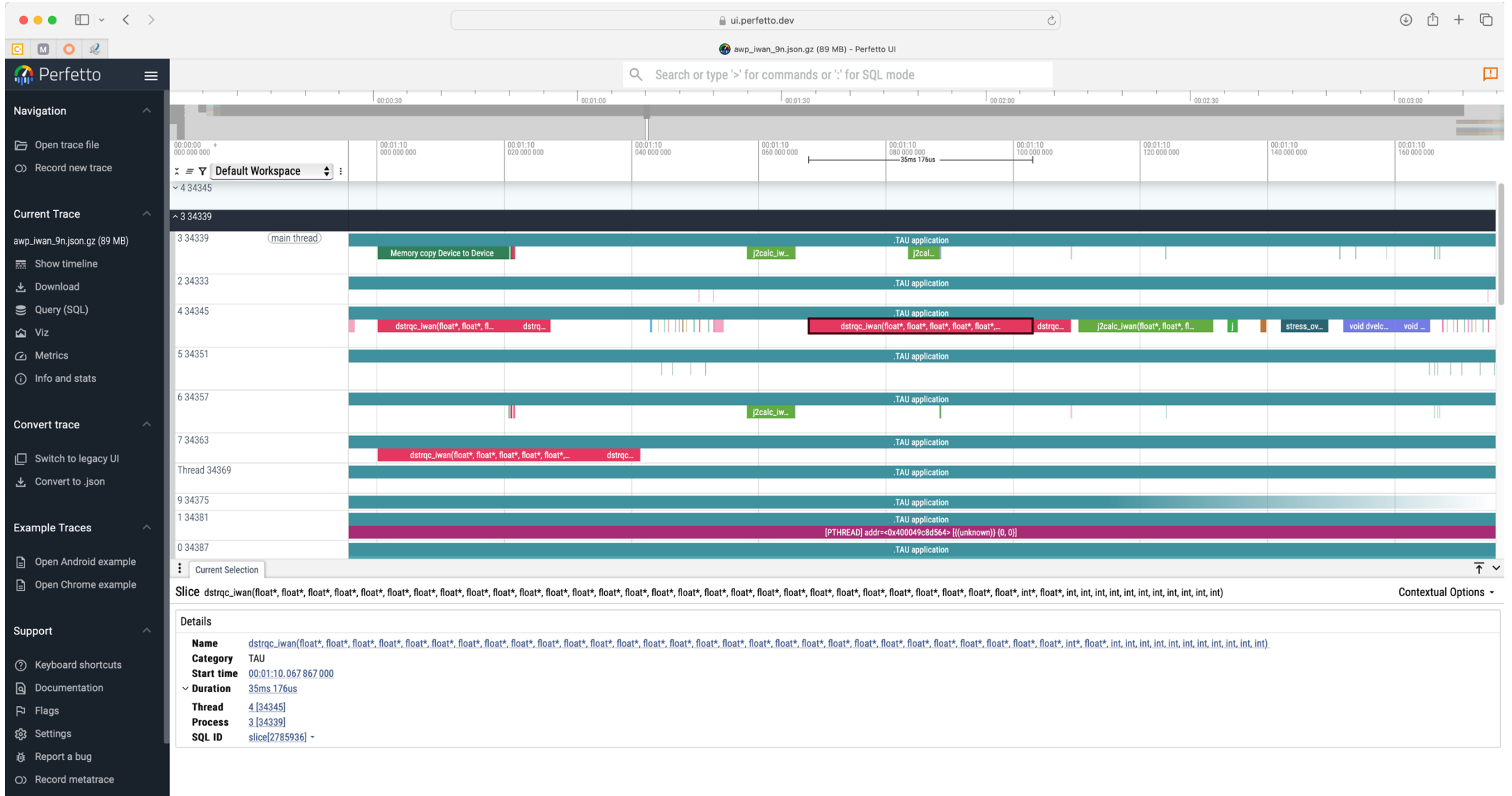


AWP-ODC: Callsite location of MPI Collective Sync



```
export TAU_CALLSITE=1; aprun -n 64 tau_exec ./a.out
```

AWP-ODC: Tracing with TAU and Perfetto.dev



Binary instrumentation of libraries: Work in progress

```
% tau_run a.out -o a.inst
```

instruments a binary. Other flags `-T <tags>`, `-f <selective instrumentation file>`

```
% tau_run -l /path/to/libhdf5.so.310 -o libhdf5.so.310
```

instruments a DSO

```
% tau_exec ./a.out
```

executes the uninstrumented application with the instrumented shared object.

To use with DyninstAPI 13 on x86_64:

1. Load spack: `source spack/share/spack/setup-env.sh`

2. Install dyninst: `spack install dyninst@13 %gcc@11`

3. Configure tau with dyninst:

3.1 `spack find -p dyninst boost tbb elfutils`

3.2 Copy the paths for each package into the configure line

3.3 `./configure -bfd=download -dyninst=<dir> -tbb=<dir> -boost=<dir> -elf=<dir>; <set paths>; make install`

With AMD GPUs:

`./configure -bfd=download -mpi -rocm=/opt/rocm-6.0.0 -rocprofiler=/opt/rocm-6.0.0 -dyninst=download; make install`

Binary instrumentation of libraries: HDF5

```
$ pprof
Reading Profile files in profile.*
```

```
NODE 0;CONTEXT 0;THREAD 0:
```

%Time	Exclusive msec	Inclusive total msec	#Call	#Subrs	Inclusive Name usec/call
100.0	0.272	68	1	1	68245 .TAU application
99.6	1	67	1	26	67973 taupreload_main
65.8	0.008	44	6	1	7484 H5open
65.8	6	44	2	14	22448 H5_init_library
36.0	4	24	1	12	24563 H5VL_init_phase2
27.8	1	18	1	319	18943 H5T_init
19.8	0.193	13	179	179	76 H5T__register_int
19.5	0.302	13	179	310	74 H5T__register
19.0	4	12	155	2555	84 H5T__path_find_real
13.0	2	8	1	79	8857 H5P_init_phase1
12.7	0.663	8	2	51	4349 H5F_open
11.2	0.348	7	1	6	7610 H5Fcreate
10.5	0.386	7	1	6	7138 H5F__create_api_common
9.8	0.406	6	1	2	6707 H5VL_file_create
9.2	0.005	6	1	1	6299 H5VL__native_file_create
7.1	1	4	488	976	10 H5T_copy
6.5	1	4	1	363	4452 H5E_init
5.6	0.013	3	4	12	956 H5I_dec_app_ref
5.6	0.013	3	2	10	1896 H5Fclose
5.5	0.009	3	2	4	1878 H5F__close_cb
5.5	0.01	3	2	6	1868 H5VL_file_close
5.4	0.013	3	2	4	1852 H5VL__native_file_close
5.4	0.019	3	4	8	924 H5F_try_close.localalias

AWP-ODC [UCSD]: TAU+CUPTI+DyninstAPI

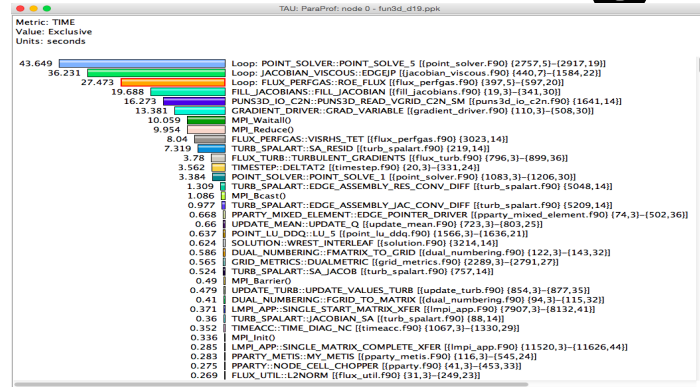
Reading Profile files in profile.*

NODE 0;CONTEXT 0;THREAD 0:

%Time	Exclusive msec	Inclusive total msec	#Call	#Subrs	Inclusive usec/call	Name
100.0	25	6:27.801	1	1	387801618	.TAU application
100.0	3	6:27.776	1	1	387776447	taupreload_main
100.0	205	6:27.772	1	125785	387772922	int main()
60.2	3:53.279	3:53.279	10501	0	22215	cudaDeviceSynchronize
37.4	2:25.156	2:25.156	18300	0	7932	cudaStreamSynchronize
1.8	7,086	7,086	12006	0	590	MPI_Waitall()
0.2	662	940	1	1033	940006	MPI_Init()
0.1	45	284	62151	62151	5	cudaLaunchKernel<char>
0.1	274	274	1024	0	268	cudaStreamCreateWithFlags
0.1	239	239	62151	0	4	cudaLaunchKernel
0.0	173	173	108	0	1607	Grid3D Alloc3D()
0.0	108	108	20750	0	5	cudaMemcpyAsync
0.0	67	72	1	1026	72277	MPI_Finalize()
0.0	9	64	7500	30000	9	dstrqc_iwan_H
0.0	7	58	4000	24000	15	update_swapzone_data_y_H
0.0	7	57	4000	24000	14	update_swapzone_data_x_H
0.0	7	55	4000	24000	14	update_swapzone_buffer_y_H
0.0	28	53	40000	40000	1	cudaFuncSetCacheConfig<void (float*
0.0	7	52	4000	24000	13	update_swapzone_buffer_x_H
0.0	5	45	1500	10500	30	void Cpy2Device_VY()
0.0	2	42	7500	7500	6	dstrqc_iwan
0.0	2	41	8000	8000	5	update_swapzone_data_y
0.0	2	41	8000	8000	5	update_swapzone_data_x
0.0	40	40	1	0	40672	void inisigma2()

Profiling and Tracing

Profiling

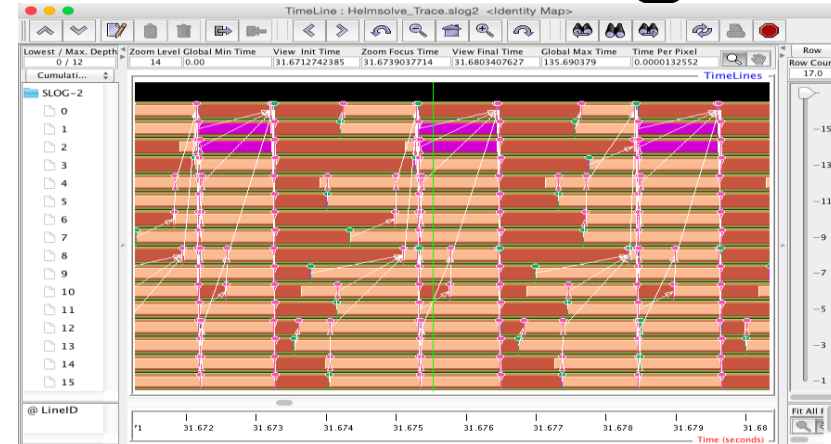


- **Profiling** shows you **how much** (total) time was spent in each routine
- Profiling and tracing

Profiling shows you **how much** (total) time was spent in each routine

Tracing shows you **when** the events take place on a timeline

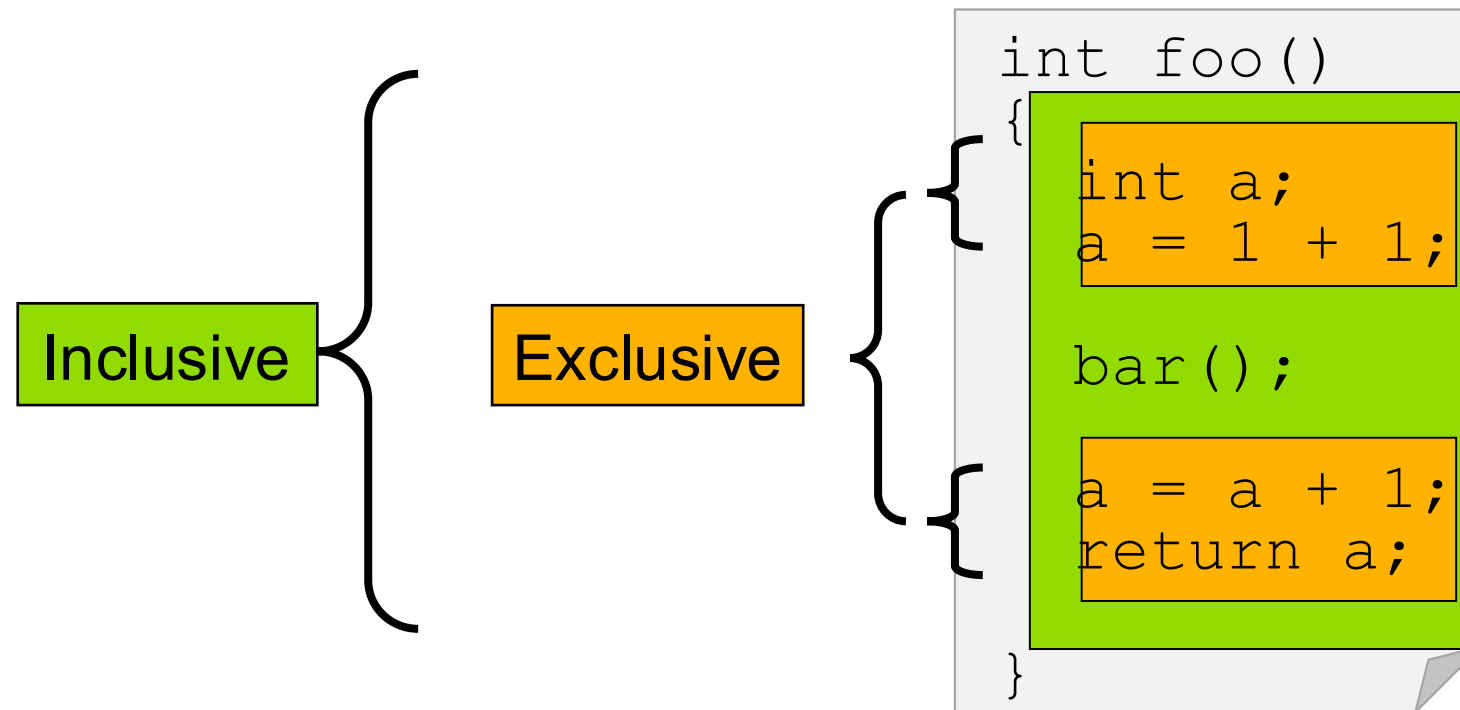
Tracing



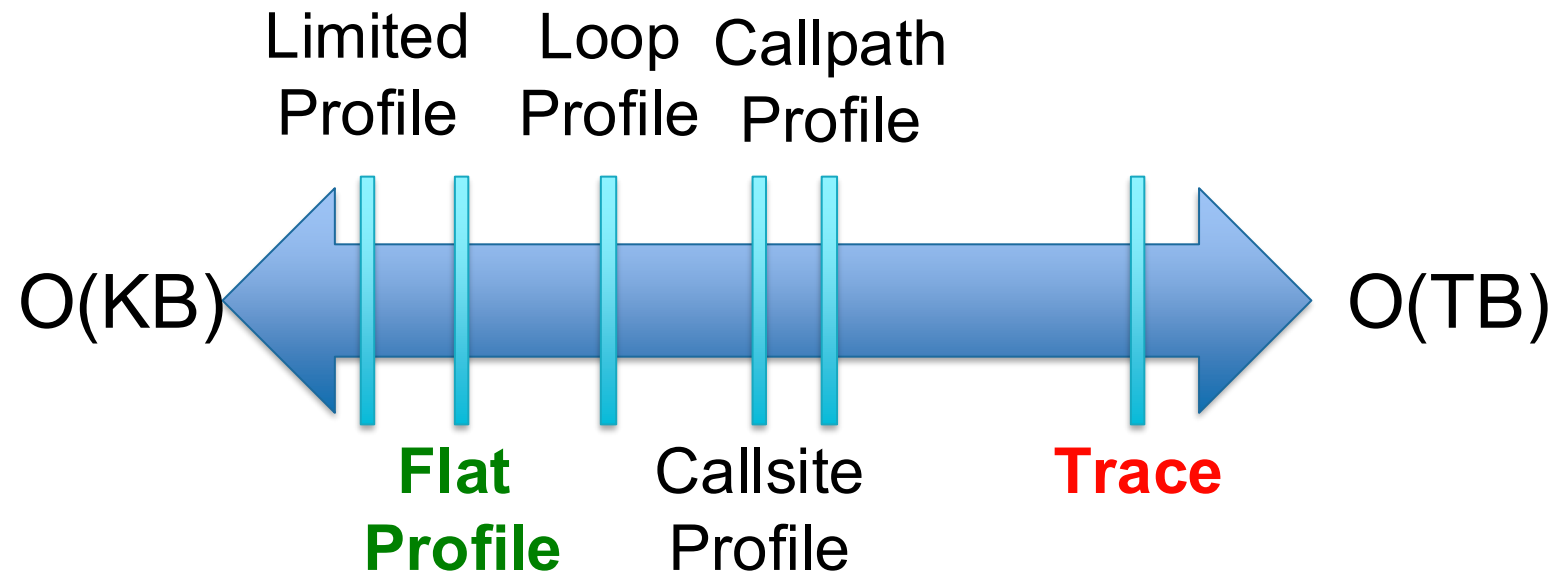
- Tracing shows you when the events take place on a timeline

Inclusive vs. Exclusive values

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



How much data do you want?



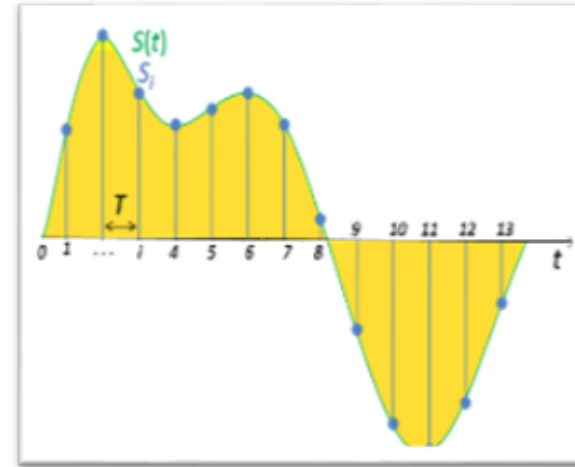
Types of Performance Profiles

- *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

Performance Data Measurement

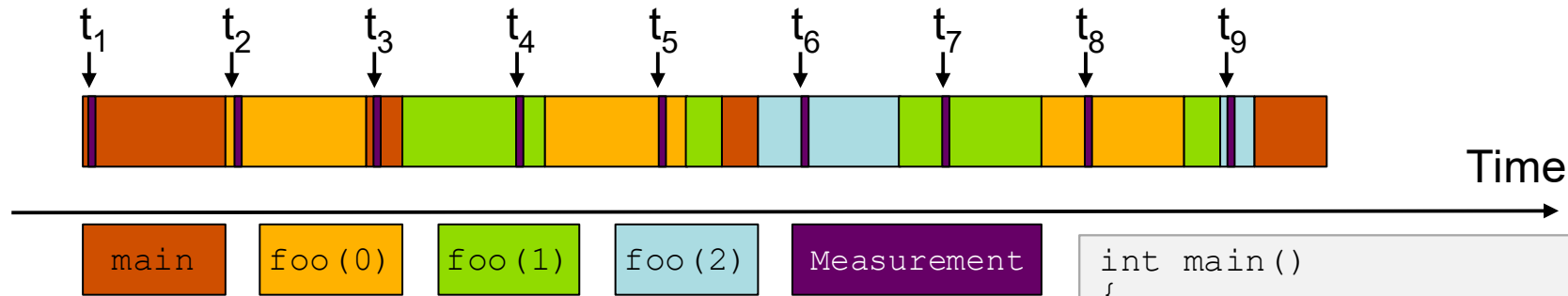
```
Call START('potential')  
// code  
Call STOP('potential')
```

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



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

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

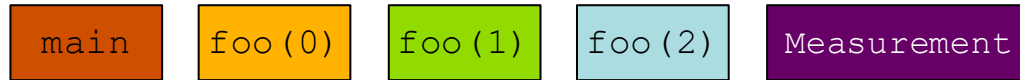
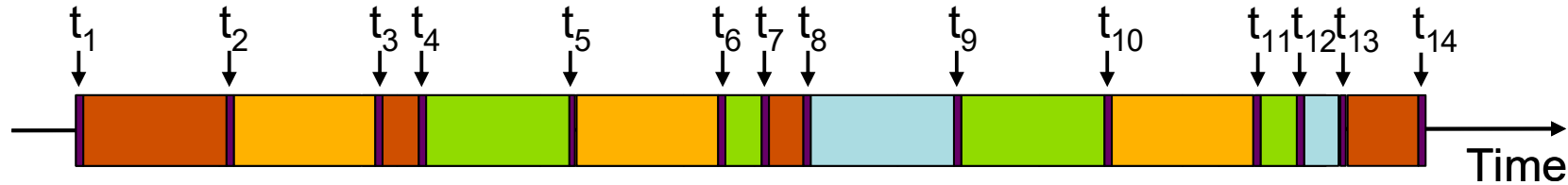
```
int main()
{
    int i;

    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:

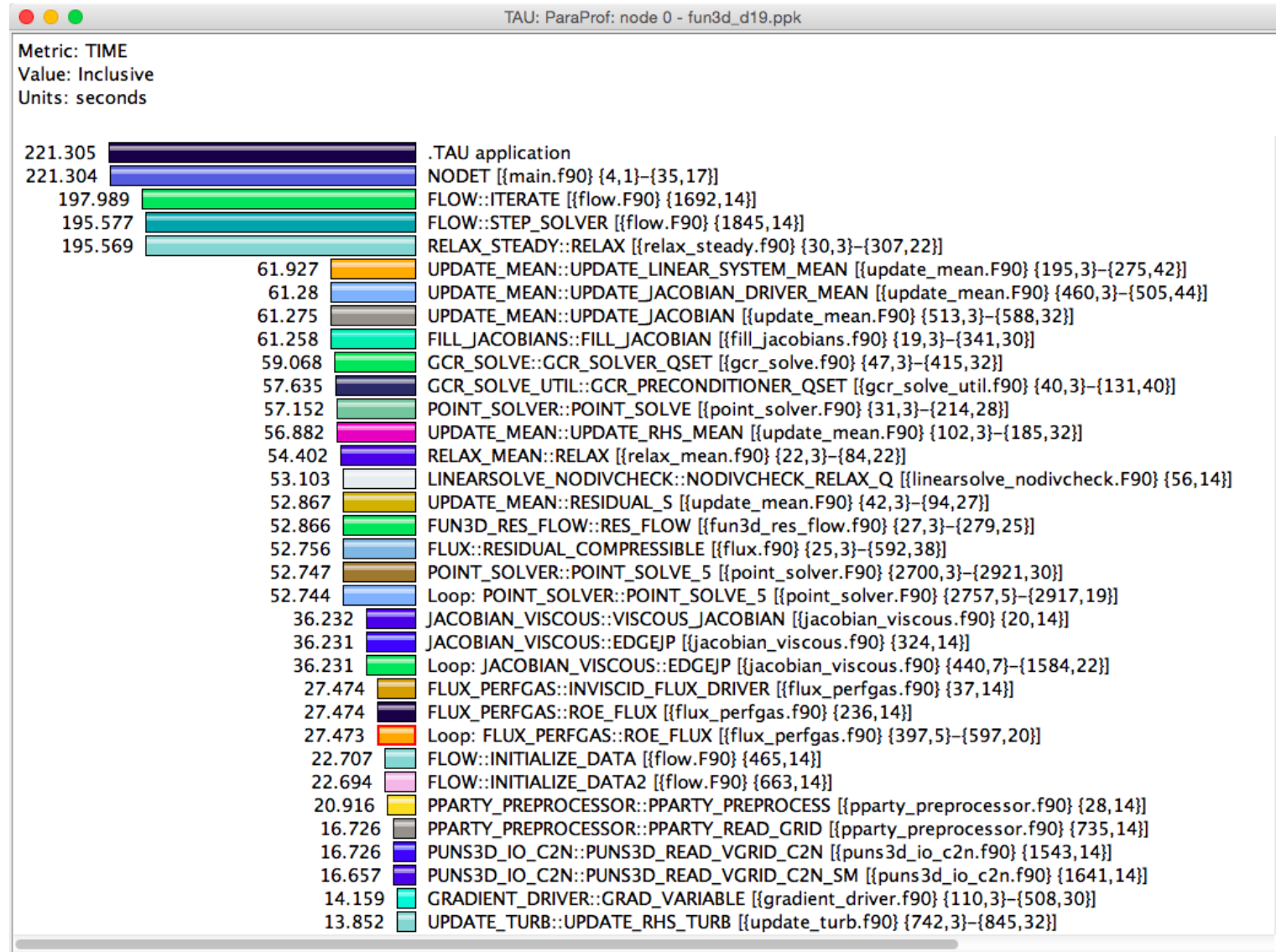
Processing of source-code / executable necessary

Large relative overheads for small functions

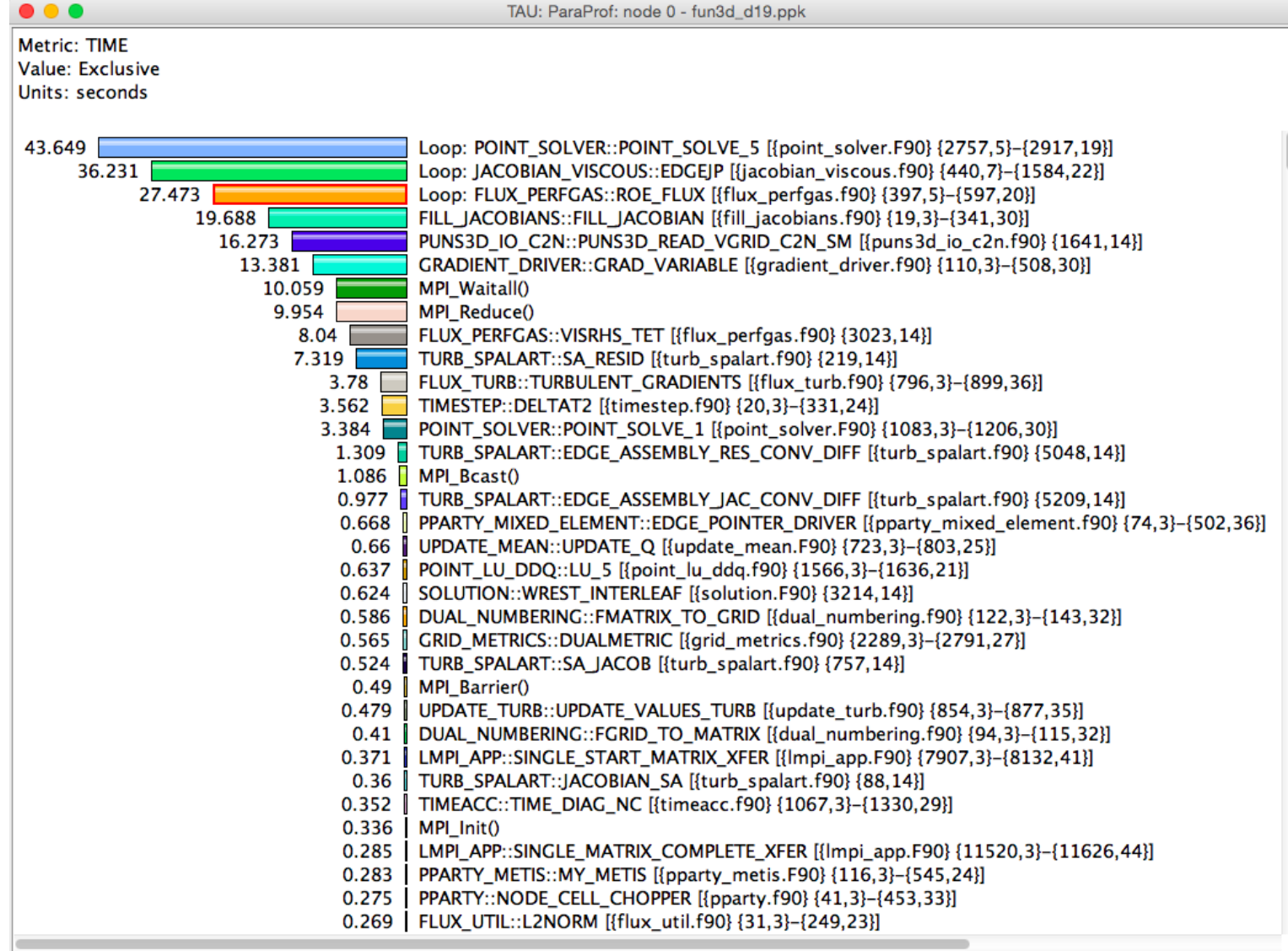
```
int main()
{
    int i;
    TAU_START("main");
    for (i=0; i < 3; i++)
        foo(i);
    TAU_STOP("main");
    return 0;
}

void foo(int i)
{
    TAU_START("foo");
    if (i > 0)
        foo(i - 1);
    TAU_STOP("foo");
}
```

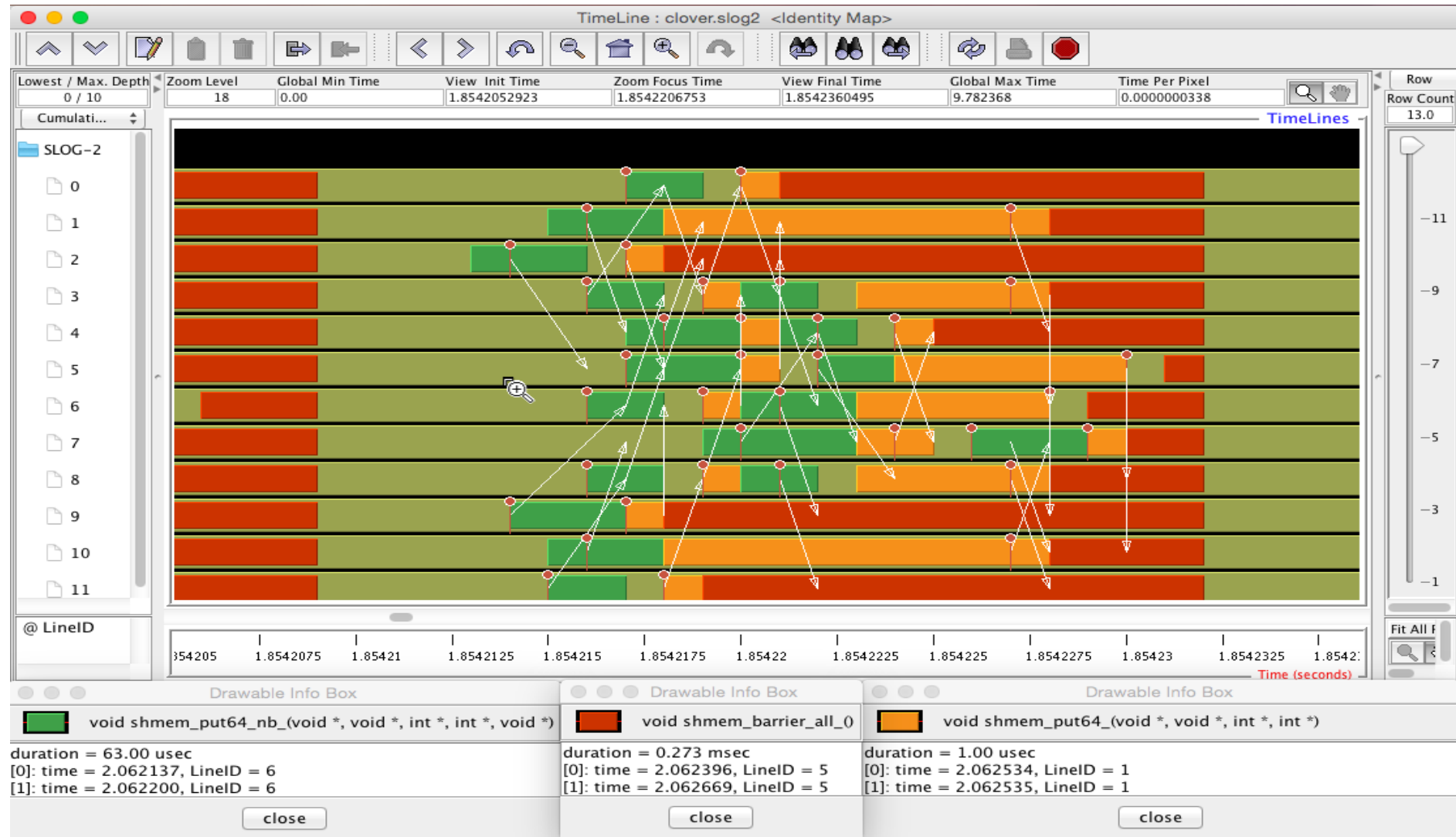
Inclusive Measurements



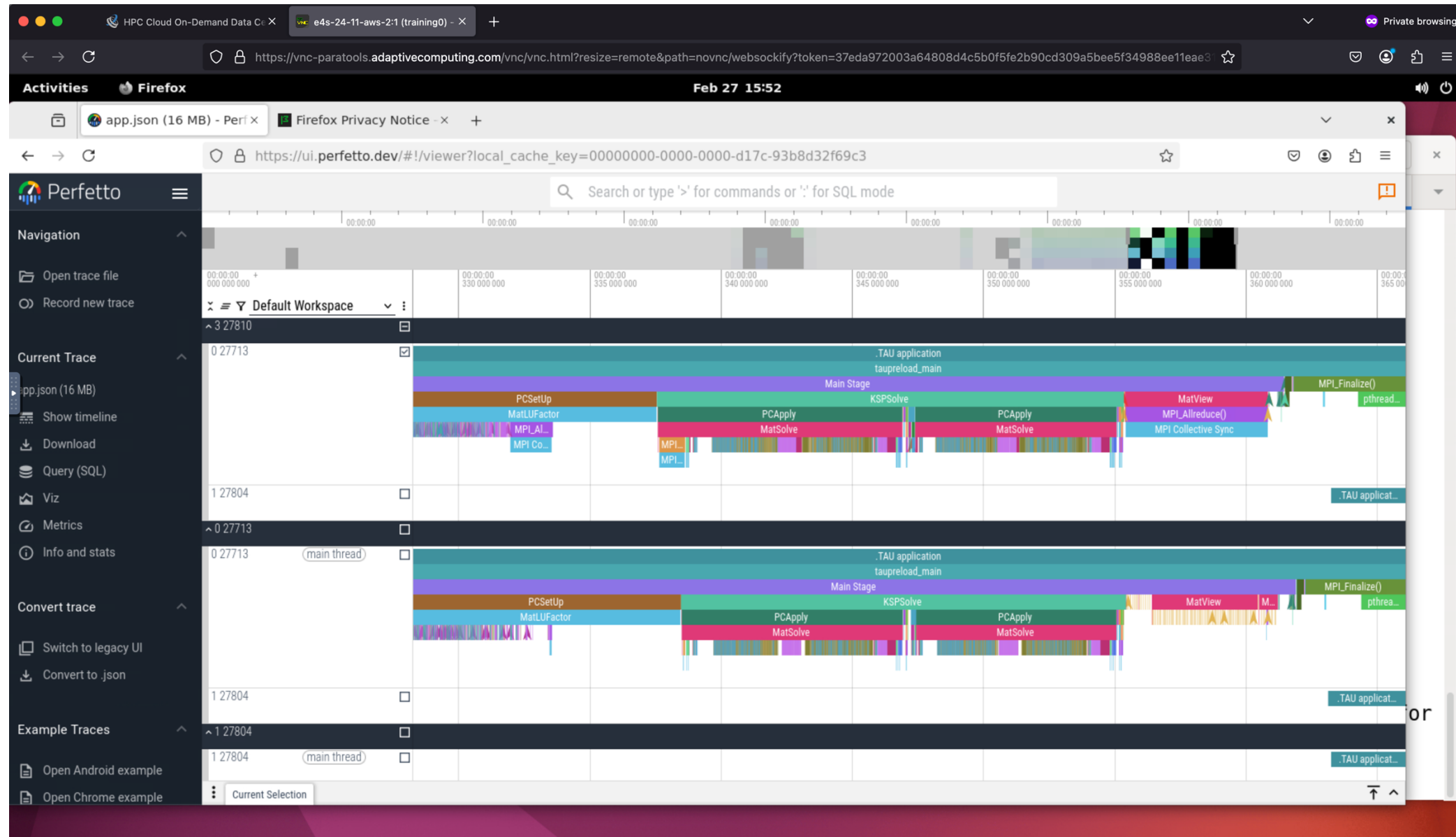
Exclusive Time



Tracing: Jumpshot (ships with TAU)

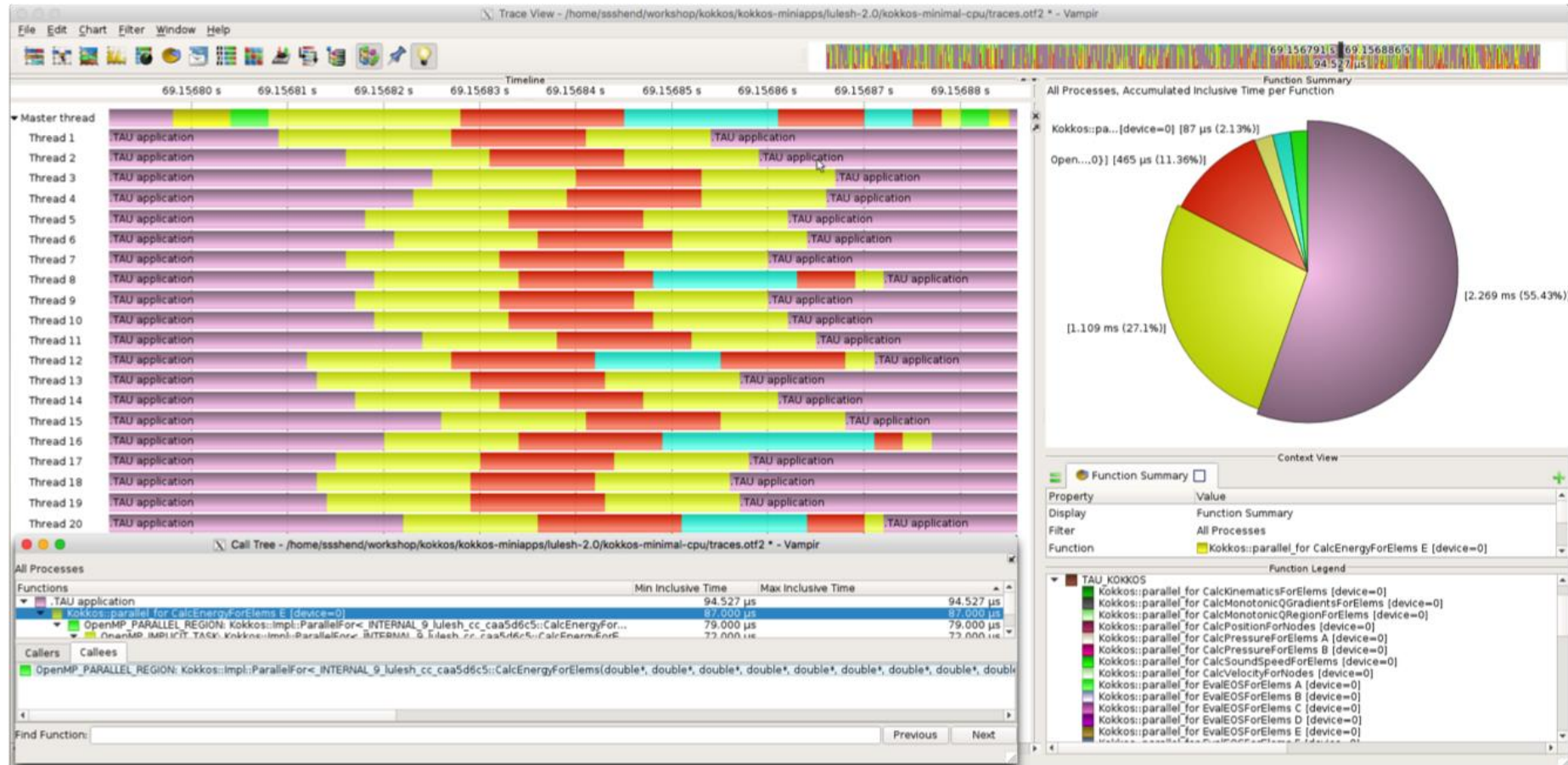


Visualizing Traces with https://Perfetto.dev



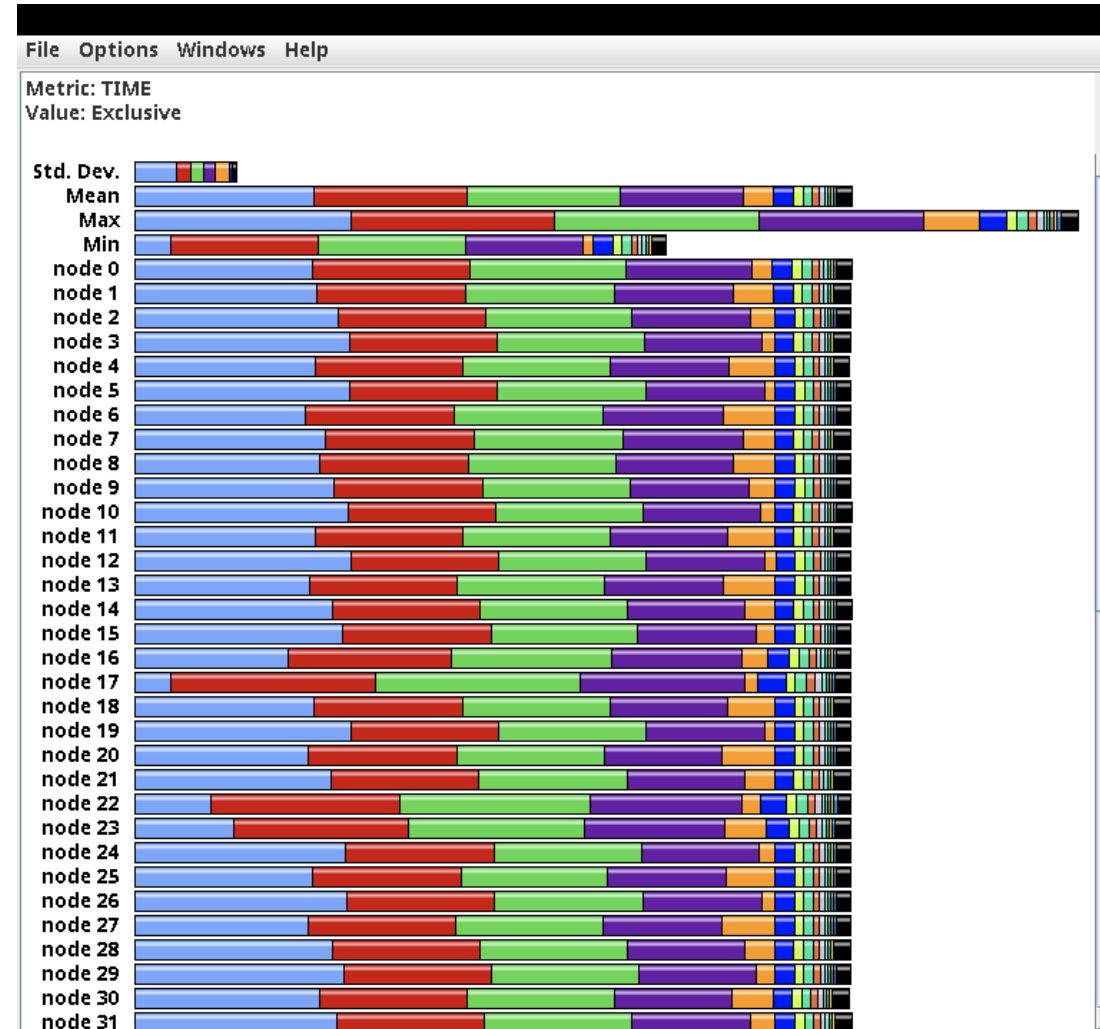
wasd
W = widen
S = Shrink
A = Left
D = Right

Vampir [TU Dresden] Timeline: Kokkos



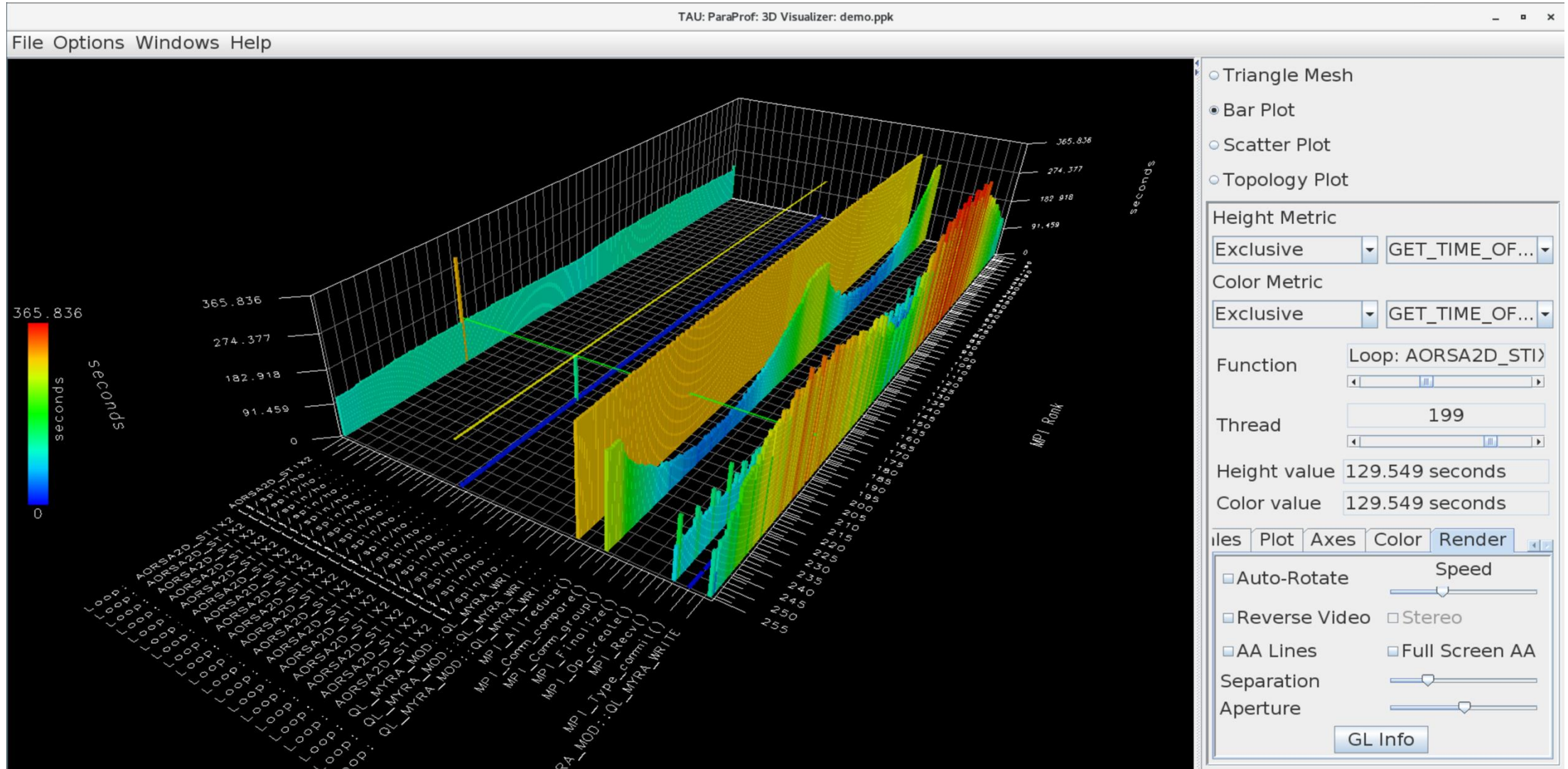
```
% export TAU_TRACE=1; export TAU_TRACE_FORMAT=otf2
% tau_exec -T ompt --ompt ./a.out
% vampir traces.otf2 &
```

ParaProf Profile Browser

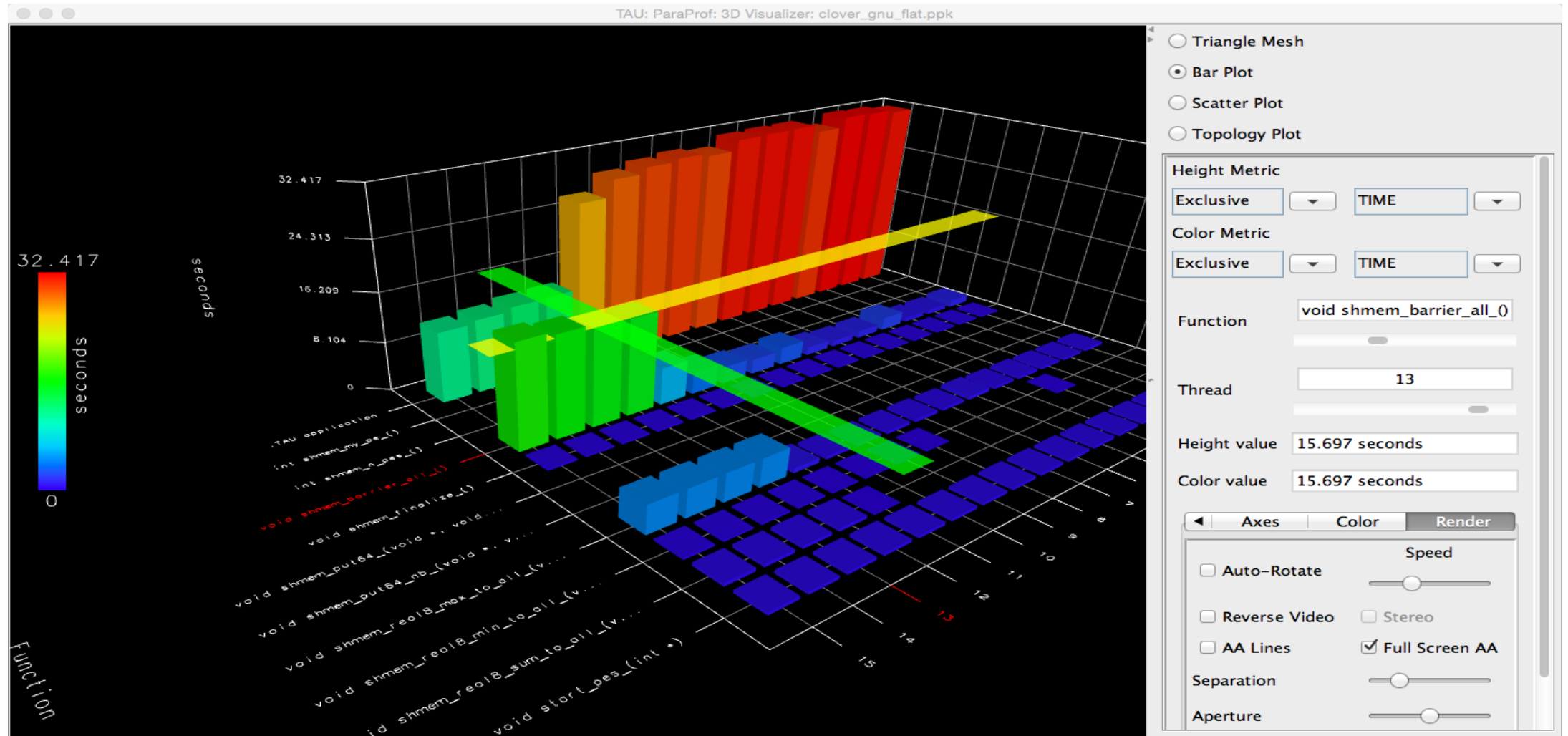


% paraprof

ParaProf 3D Profile Browser



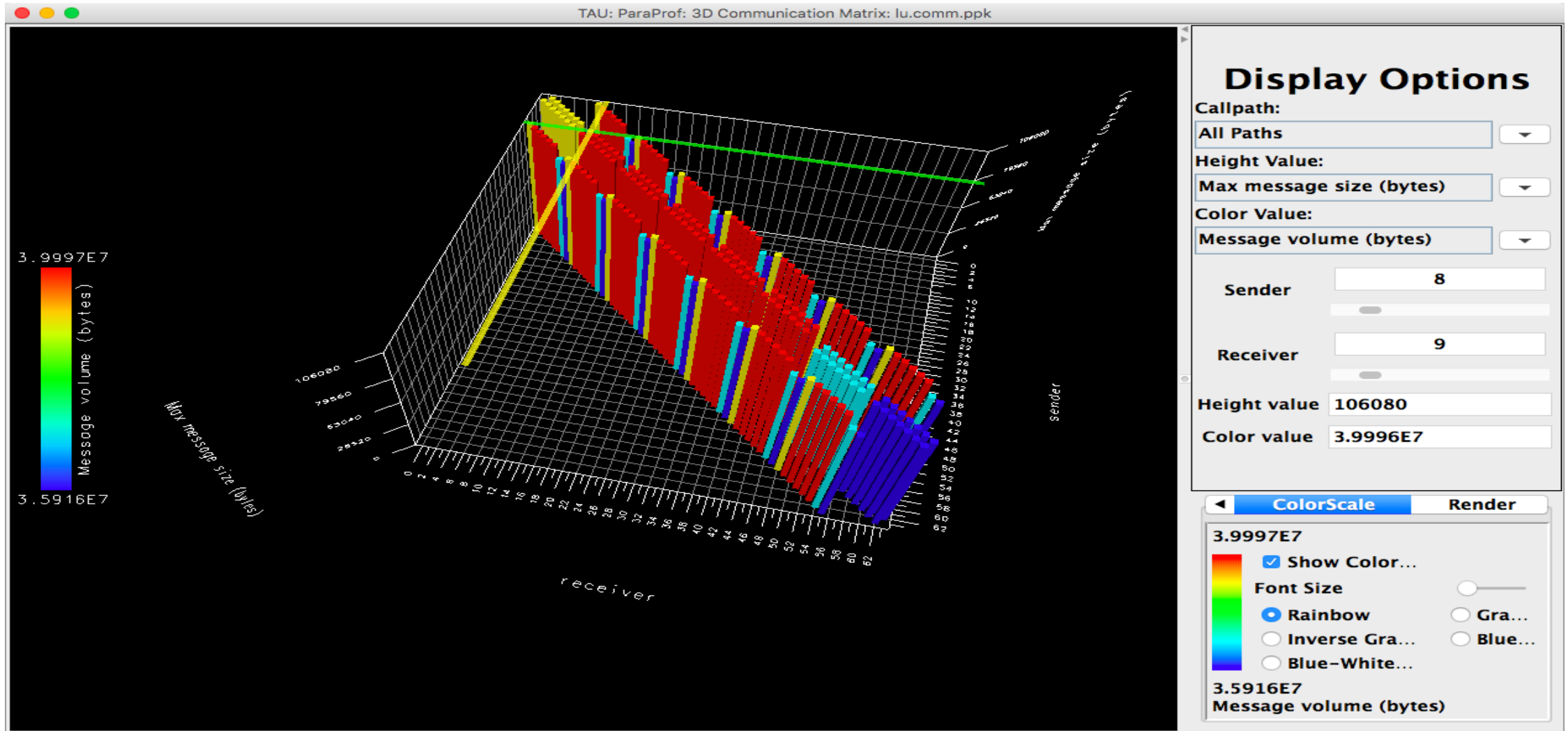
TAU – ParaProf 3D Visualization



% paraprof app.ppk

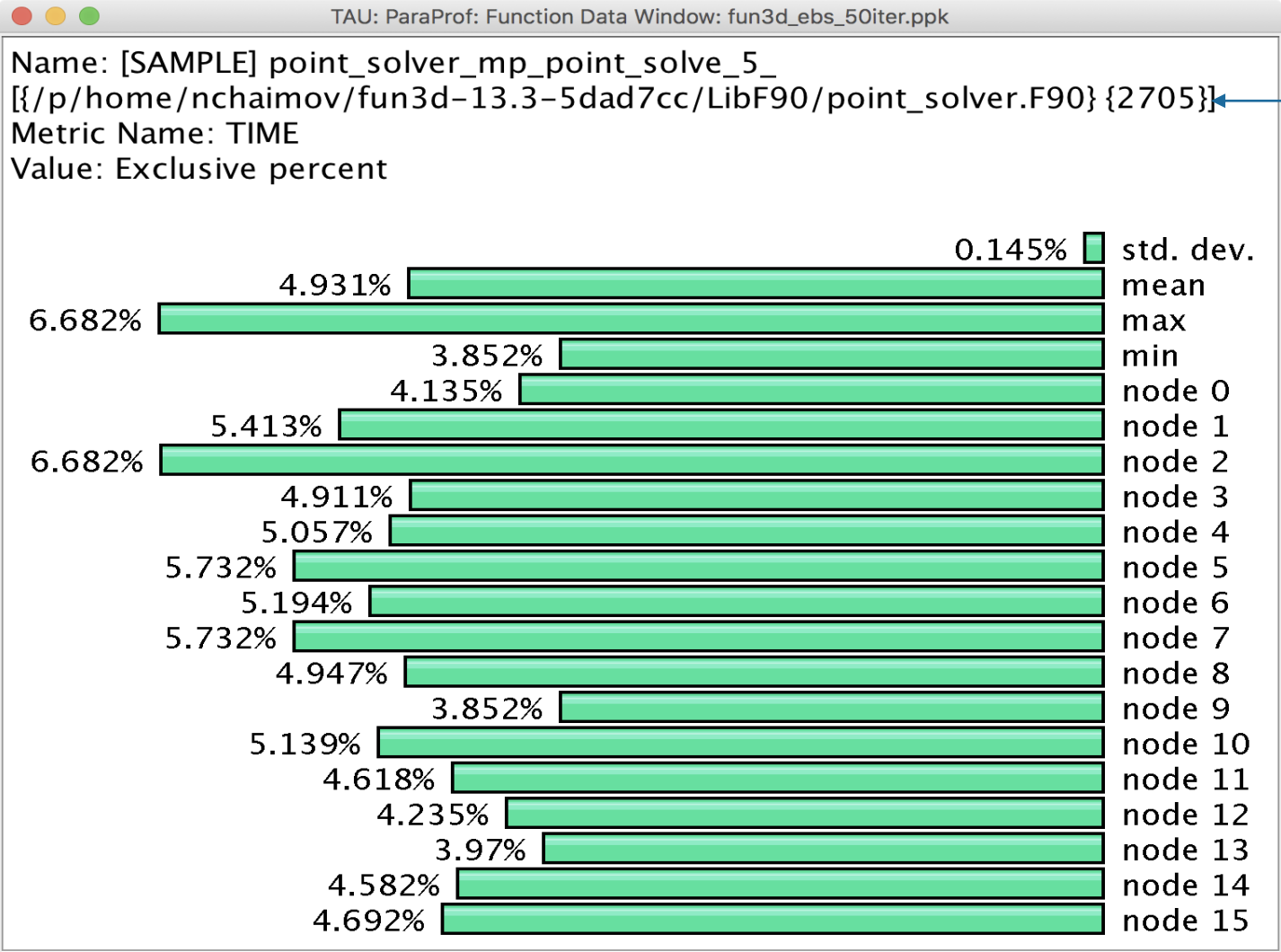
Windows -> 3D Visualization -> Bar Plot (right pane)

TAU – 3D Communication Window



```
% export TAU_COMM_MATRIX=1; mpirun ... tau_exec ./a.out  
% paraprof ; Windows -> 3D Communication Matrix
```

Event Based Sampling (EBS)



File: point_solver.F90
Line: 2705

Uninstrumented!

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

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
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>)

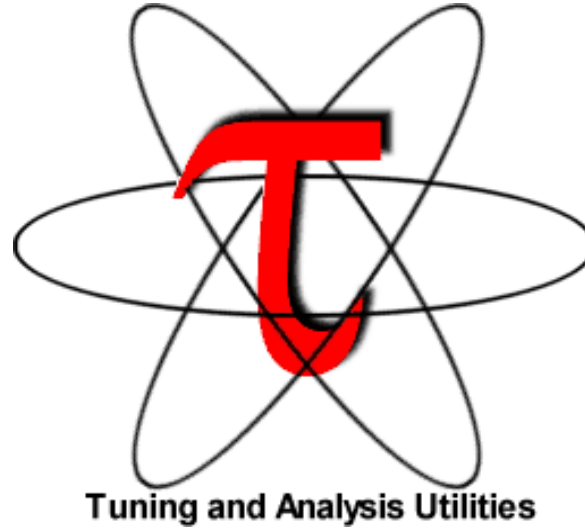
Runtime Environment Variables

Environment Variable	Default	Description
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.

Runtime Environment Variables

Environment Variable	Default	Description
TAU_TRACK_MEMORY_LEAKS	0	Tracks allocates that were not de-allocated (needs <code>–optMemDbg</code> or <code>tau_exec –memory</code>)
TAU_EBS_SOURCE	TIME	Allows using PAPI hardware counters for periodic interrupts for EBS (e.g., <code>TAU_EBS_SOURCE=PAPI_TOT_INS</code> when <code>TAU_SAMPLING=1</code>)
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 <code>TAU_MEMDBG_PROTECT_*</code>)
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 <code>–optMemDbg</code> while building or <code>tau_exec –memory</code>)
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 <code>–optMemDbg</code> or <code>tau_exec –memory</code>)
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
TAU_EVENT_THRESHOLD	0.5	Define a threshold value (e.g., .25 is 25%) to trigger marker events for min/max

Download TAU from U. Oregon

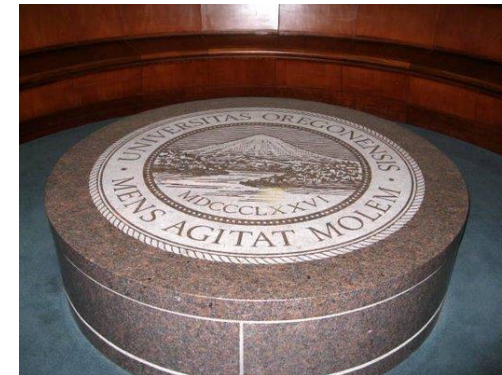


<http://tau.uoregon.edu>

**<https://e4s.io> [TAU in Docker/Singularity containers]
for more information**

Free download, open source, BSD license

Performance Research Laboratory, University of Oregon Eugene



www.uoregon.edu

Support Acknowledgements

- US Department of Energy (DOE)
 - ANL
 - Office of Science contracts, ECP
 - SciDAC, LBL contracts
 - LLNL-LANL-SNL ASC/NNSA contract
 - Battelle, PNNL and ORNL contract
- Department of Defense (DoD)
 - PETTT, HPCMP
- National Science Foundation (NSF)
 - SI2-SSI, Glassbox, E4S Workshop
- NASA
- Intel, NVIDIA, AMD, IBM
- CEA, France
- Partners:
 - University of Oregon
 - The Ohio State University
 - ParaTools, Inc.
 - University of Tennessee, Knoxville
 - T.U. Dresden, GWT
 - Jülich Supercomputing Center



UNIVERSITY
OF OREGON

THE OHIO STATE
UNIVERSITY



ParaTools



Acknowledgment

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Next-Generation Scientific Software Technologies program, under contract numbers DE-AC02-06CH11357, DE-AC05-00OR22725, DOE SBIR DE-SC0022502, Sandia National Laboratories Contract PO 2542488, and NSF OAC award 2311831.



U.S. DEPARTMENT OF
ENERGY

Office of
Science



- <https://science.osti.gov/ascr>
- <https://pesoproject.org>
- <https://ascr-step.org>
- <https://hpsf.io>
- <https://www.energy.gov/technologytransitions/sbirsttr>



