WORKSHOP

ALCF
Hands-on HPC Workshop

September 23-25 & October 7-9, 2025
Argonne National Laboratory

TAU

Sameer Shende, University of Oregon, ParaTools, Inc.

U.S. DEPARTMENT of ENERGY

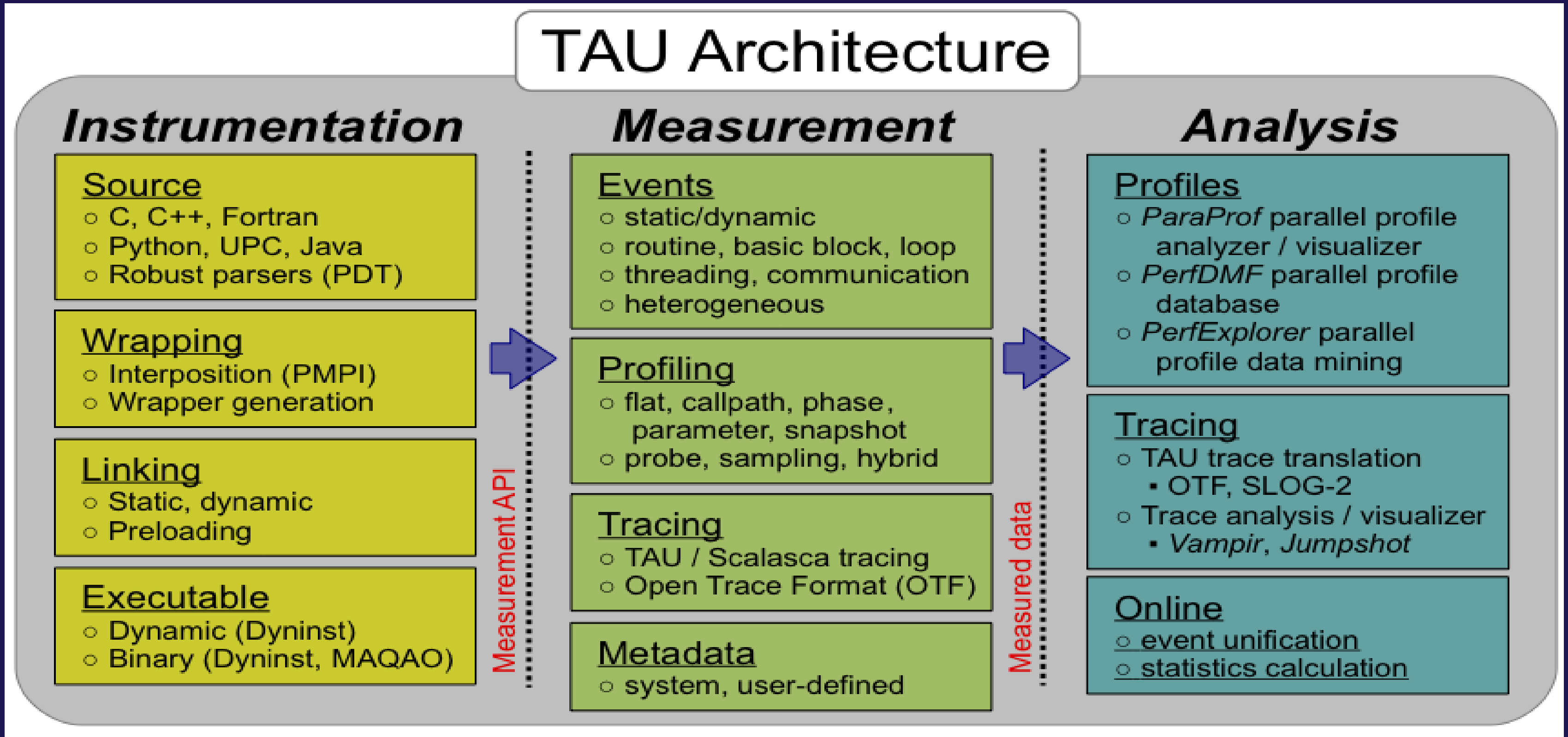
Argonne NATIONAL LABORATORY

Using TAU on Aurora

```
tar xf /soft/perftools/tau/tar/workshop.tgz
cd workshop; cat README
qsub -I  -l walltime=0:29:00 -lfilesystems=home:flare -A alcf_training
-q alcf_training -l select=1 -X
module use /soft/modulefiles
module load tau
module load tau/tau2
module load frameworks

mpiexec -n 4 ./a.out
mpiexec -n 4 tau_exec -l0 -ebs ./a.out

mpiexec -n 4 tau_exec -l0 -ebs python ./foo.py
pprof -a  | more
paraprof --pack foo.ppk
On your desktop: % paraprof foo.ppk
```

TAU Performance System®

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



E4S: Extreme-scale Scientific Software Stack

https://e4s.io

TAU is part of the E4S project

- E4S, an HPSF project, is an HPC-AI software **ecosystem for science** and a community effort to provide open-source software packages for developing, deploying and running scientific applications on HPC platforms.
- E4S has built a comprehensive, coherent software stack that enables application developers to productively develop highly parallel applications that effectively target diverse exascale architectures.
- E4S provides a curated, Spack based software distribution of 100+ HPC (**TAU**, HPCToolkit, OpenFOAM, Gromacs, Nek5000, LAMMPS), EDA (e.g., Xyce), and AI/ML packages (e.g., NVIDIA NeMo™, NVIDIA BioNeMo™, Vllm, HuggingFace CLI, TensorFlow, PyTorch, OpenCV, TorchBraid, Scikit-Learn, Pandas, JAX, LBANN with support for GPUs where available).
- Base images and full featured containers (with GPU support).
- Commercial support for E4S through ParaTools, Inc. for installation, maintaining an issue tracker, and application engagement.
- E4S for commercial clouds: Adaptive Computing's Heidi AI with ParaTools Pro for E4S™ image for **AWS, GCP, Azure, OCI.**
- With E4S Spack binary build caches, E4S supports both bare-metal and containerized deployment for GPU based platforms.
  - x86_64, ppc64le (IBM Power 10), aarch64 (ARM64) with support for CPUs and GPUs from NVIDIA, AMD, and Intel
  - Container images on DockerHub and E4S website of pre-built binaries of ECP ST products.
- e4s-chain-spack.sh to chain two Spack instances allows us to install new packages in home directory and use other tools.
- e4s-cl container launch tool allows binary distribution of applications by swapping MPI in the containerized app w/ system MPI.
- e4s-alc is an à la carte tool to customize container images by adding system and Spack packages to an existing image.
- E4S 25.06 released on June 6, 2025: https://e4s.io/talks/E4S_25.06.pdf

TAU Performance System®

- Instrumentation
  - Fortran, C++, C, UPC, Java, Python, Chapel, Spark
  - Automatic instrumentation
- Measurement and analysis support
  - MPI (MVAPICH2, Intel MPI), 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 **mpiexec tau_exec ./app** with options.
  - —It will preload TAU's DSO and intercept MPI calls and generate profile/trace data.

Configurations of TAU installed on Aurora

```
module use /soft/modulefiles;

module load tau;

module load tau/tau2

module load frameworks

ls $TAU/Makefile*

/soft/perftools/tau/tau2/x86_64/lib/
Makefile.tau-level_zero-icpx-papi-ompt-mpi-pthread-python-pdt-openmp

/soft/perftools/tau/tau2/x86_64/lib/
Makefile.tau-level_zero-icpx-papi-ompt-pthread-python-pdt-openmp

mpiexec -n 12 tau_exec -l0 -ebs ./a.out will choose the MPI configuration by
default. For non MPI jobs, please use -T serial. For e.g.,

tau_exec -T serial -ebs -l0 python ./foo.py and will LD_PRELOAD:
/soft/perftools/tau/tau2/x86_64/lib/shared-level_zero-icpx-papi-ompt-pthread-python-
pdt-openmp/libTAU.so
```

Using TAU for source instrumentation

- TAU supports several measurement and thread options
  - Phase profiling, profiling with hardware counters, MPI library, Python…
  - Each measurement configuration of TAU corresponds to a unique stub makefile and library that is generated when you configure it
- To instrument source code automatically using PDT
  - Choose an appropriate TAU stub makefile in <arch>/lib:

```
% module use /soft/modulefiles; module load tau; load tau/tau2
% export TAU_MAKEFILE=/soft/perftools/tau/tau2/x86_64/lib/Makefile.tau-level_zero-icpx-papi-ompt-mpi-pthread-python-pdt-openmp

% export TAU_OPTIONS='-optVerbose …' (see tau_compiler.sh )
% export PATH=$TAUDIR/x86_64/bin:$PATH
Use tau_f90.sh, tau_cxx.sh, tau_upc.sh, or tau_cc.sh as F90, C++, UPC, or C compilers respectively:
% mpif90 foo.f90      changes to
% tau_f90.sh foo.f90
```

- Set runtime environment variables, execute application and analyze performance data:
  - % pprof (for text based profile display)
  - % paraprof (for GUI)

Configurations of TAU installed on Polaris

```
module use /soft/modulefiles; module load tau; ls $TAU/Makefile*

/soft/perftools/tau/tau-2.33.2/craycnl/lib/Makefile.tau-gnu-papi-mpi-cupti-pdt

/soft/perftools/tau/tau-2.33.2/craycnl/lib/Makefile.tau-gnu-papi-mpi-pdt

/soft/perftools/tau/tau-2.33.2/craycnl/lib/Makefile.tau-gnu-papi-mpi-pthread-cupti-pdt

/soft/perftools/tau/tau-2.33.2/craycnl/lib/Makefile.tau-gnu-pdt

/soft/perftools/tau/tau-2.33.2/craycnl/lib/Makefile.tau-gnu-tbb-pdt

/soft/perftools/tau/tau-2.33.2/craycnl/lib/Makefile.tau-nvidia-papi-mpi-cupti-pdt

/soft/perftools/tau/tau-2.33.2/craycnl/lib/Makefile.tau-nvidia-papi-mpi-pdt

/soft/perftools/tau/tau-2.33.2/craycnl/lib/Makefile.tau-nvidia-pdt


aprun -n 16 tau_exec -T gnu-papi-mpi-pthread-cupti-pdt -ebs ./a.out
will choose a configuration represented by:

/soft/perftools/tau/tau-2.33.2/craycnl/lib/Makefile.tau-gnu-papi-mpi-pthread-cupti-pdt
```

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
- *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 + pthread*
  - PMPI + pthread interfaces
- *MPI + Intel ® oneAPI DPC++/SYCL*
- PMPI + Level Zero interfaces
- *OpenCL + Python*
  - OpenCL + Python instrumentation interfaces
- *Kokkos + OpenMP*
  - Kokkos profiling API + OMPT to transparently track events
- *Kokkos + pthread + MPI*
  - Kokkos + pthread wrapper interposition library + PMPI layer
- *MPI + OpenCL*
  - PMPI + OpenCL profiling interfaces

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

Binary instrumentation of libraries: HDF5

STEP
SOFTWARE TOOLS
ECOSYSTEM PROJECT

```
$ pprof
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        0.272           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
```

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

TAU Configurations

% ls /usr/projects/packages/tau/tau-2.34.1/craycnl/lib/Makefile*

/usr/projects/packages/tau/tau-2.34.1/craycnl/lib/Makefile.tau-cray-papi-ompt-mpi-pdt-openmp

/usr/projects/packages/tau/tau-2.34.1/craycnl/lib/Makefile.tau-cray-pdt

For an uninstrumented binary:

% salloc –N 1

% srun –n 16 –T cray,mpi,papi,pdt ./a.out

Picks the configuration represented by

/usr/projects/packages/tau/tau-2.34.1/craycnl/lib/Makefile.tau-cray-papi-ompt-mpi-pdt-openmp

To use OpenMP instrumentation:

% export TAU_OMPT_SUPPORT_LEVEL=full

% export OMP_NUM_THREADS=<N>

% srun –n 16 tau_exec –T ompt,mpi –ompt –ebs ./a.out

To use TAU's source code instrumentation:

% export TAU_MAKEFILE=/usr/projects/packages/tau/tau-2.34.1/craycnl/lib/Makefile.tau-cray-papi-ompt-mpi-pdt-openmp

% make CC=tau_cc.sh F90=tau_f90.sh CXX=tau_cxx.sh ; mpirun –np 16 ./a.out

% pprof –a | more

% paraprof

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

TAU Configurations available

% module load tau

% ls $TAU/Makefile*

/packages/tau-2.34/x86_64/lib/Makefile.tau-intel-papi-mpi-pthread-pdt

/packages/tau-2.34/x86_64/lib/Makefile.tau-intel-papi-ompt-mpi-pdt-openmp

/packages/tau-2.34/x86_64/lib/Makefile.tau-papi-mpi-pdt

/packages/tau-2.34/x86_64/lib/Makefile.tau-papi-pdt

/packages/tau-2.34/x86_64/lib/Makefile.tau-papi-pthread-pdt

For an uninstrumented binary:

% mpirun –np 16 tau_exec –T mpi,papi,pdt ./a.out

Picks the configuration represented by

/packages/tau-2.34/x86_64/lib/Makefile.tau-intel-papi-mpi-pdt

To use OpenMP instrumentation:

% export TAU_OMPT_SUPPORT_LEVEL=full

% export OMP_NUM_THREADS=<N>

% mpirun –np 16 tau_exec –T ompt,mpi –ompt –ebs ./a.out
To use TAU's source code instrumentation:

% export TAU_MAKEFILE=/packages/tau-2.34/x86_64/lib/Makefile.tau-intel-papi-mpi-pdt

% make CC=tau_cc.sh F90=tau_f90.sh CXX=tau_cxx.sh ; mpirun –np 16 ./a.out

% pprof –a | more

% paraprof

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

Tags in tau_exec and other tools

```
% cd $TAU; ls Makefile.*
Makefile.tau-icpx-papi-mpi-pdt
% srun -n 4 ./matrix
```

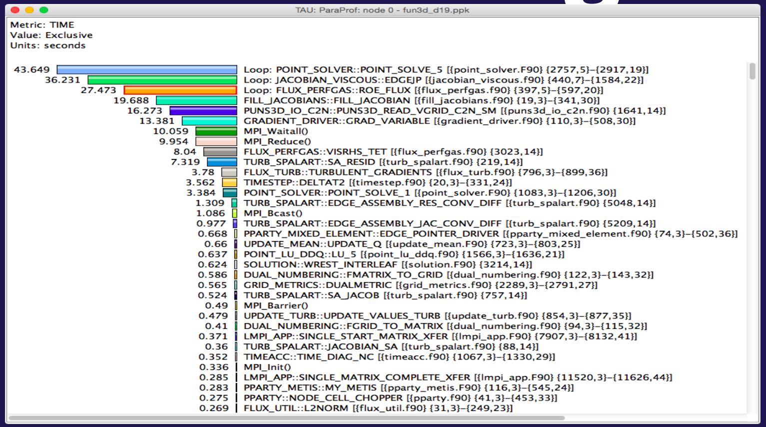
- % tau_exec -T icpx,mpi,pdt ./a.out
- Chooses Makefile.tau-icpx-mpi,pdt and associated libraries.

- % tau_exec -T serial,pdt ./a.out
- Chooses Makefile.tau-pdt or the shortest Makefile name without –mpi.

- -T <list_of_tags> is used in several TAU tools:
  - tau_run
  - tau_python
  - tau_rewrite
  - tau_exec
  - tau_gen_wrapper

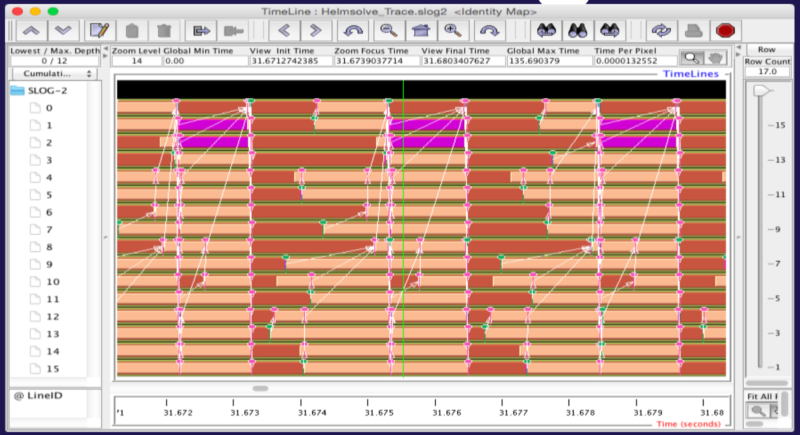
Profiling and Tracing

Profiling



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

Tracing



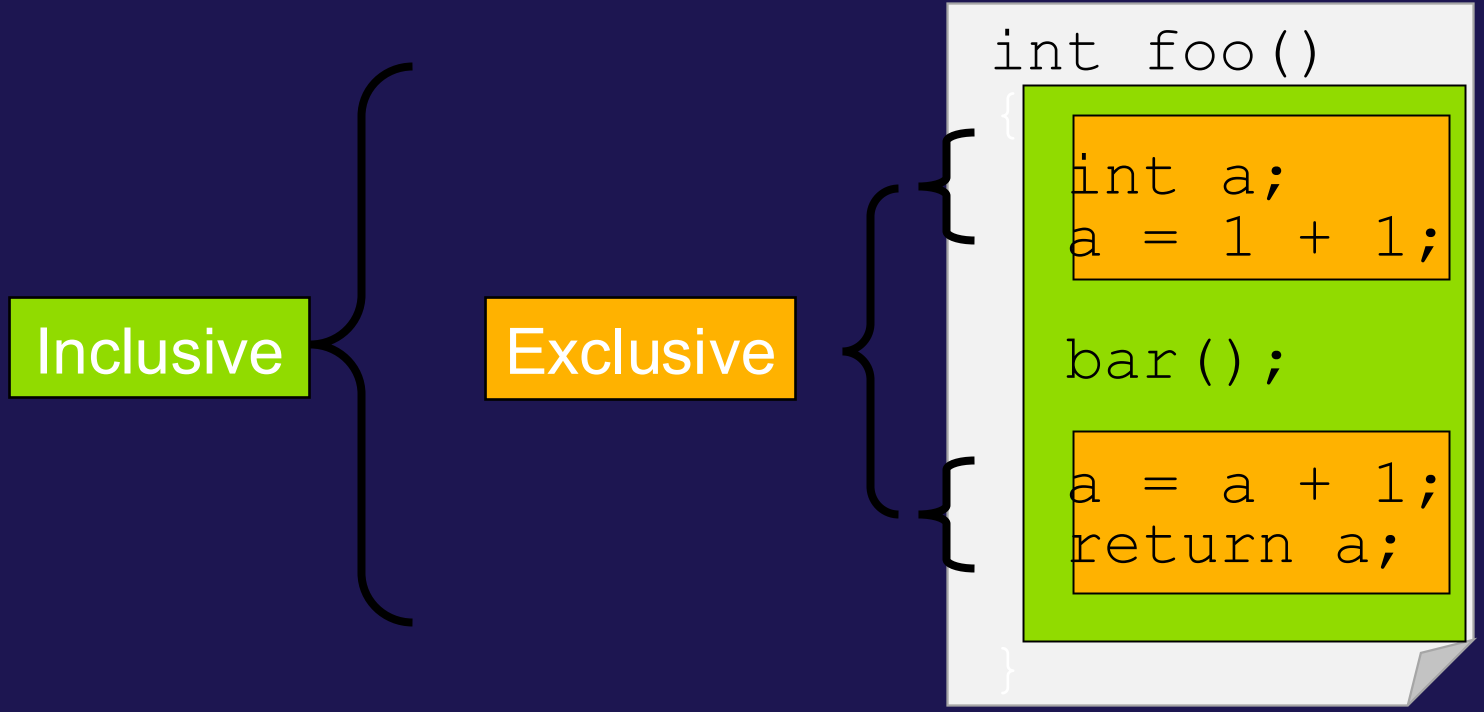
- Tracing shows you when the events take place on a timeline

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

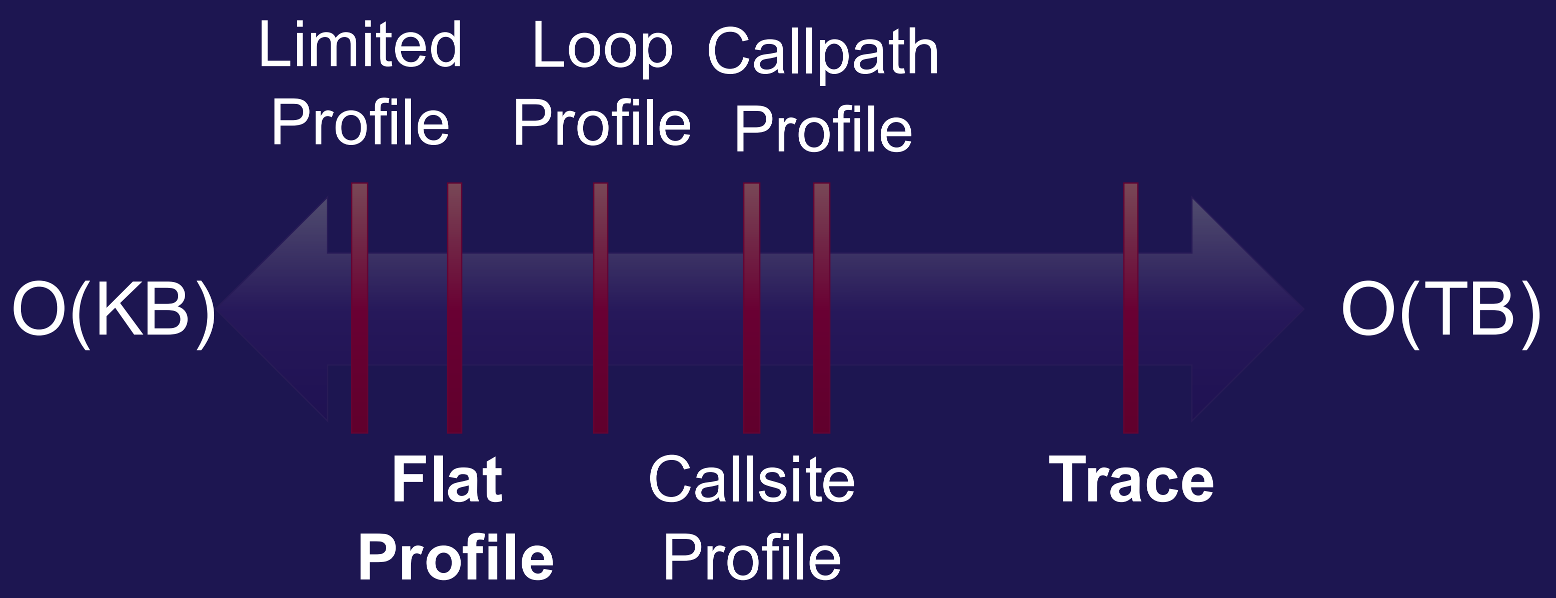
Tracing shows you **when** the events take place on a timeline

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?



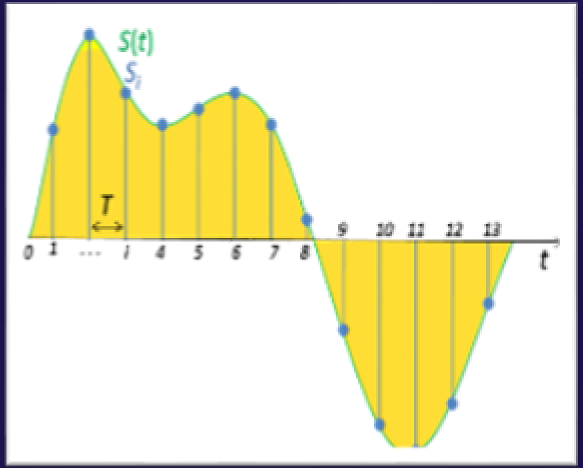
Performance Data Measurement

Direct via Probes

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

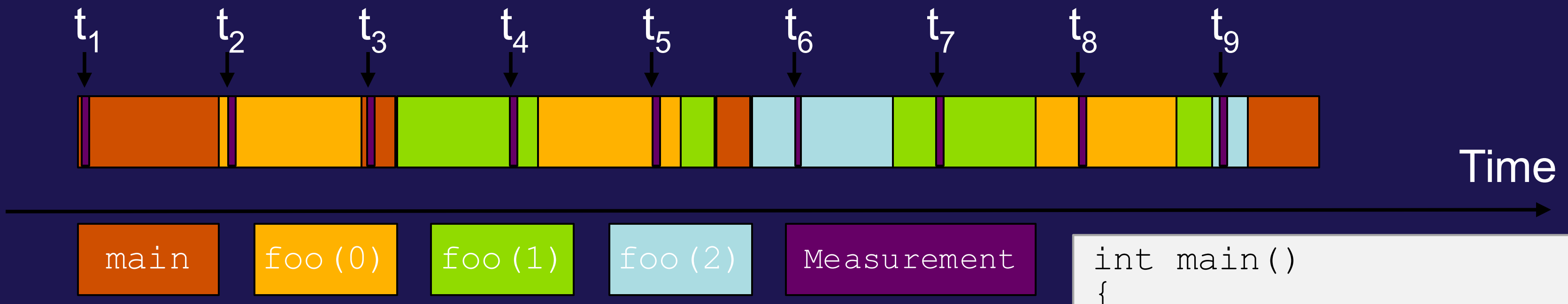
- Exact measurement
- Fine-grain control
- Calls inserted into code

Indirect via Sampling



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

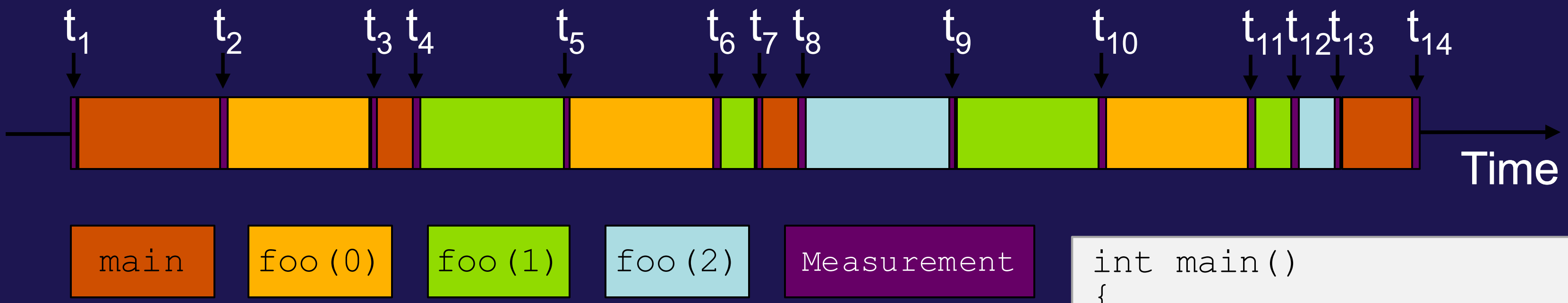
Sampling



- 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);

}
```

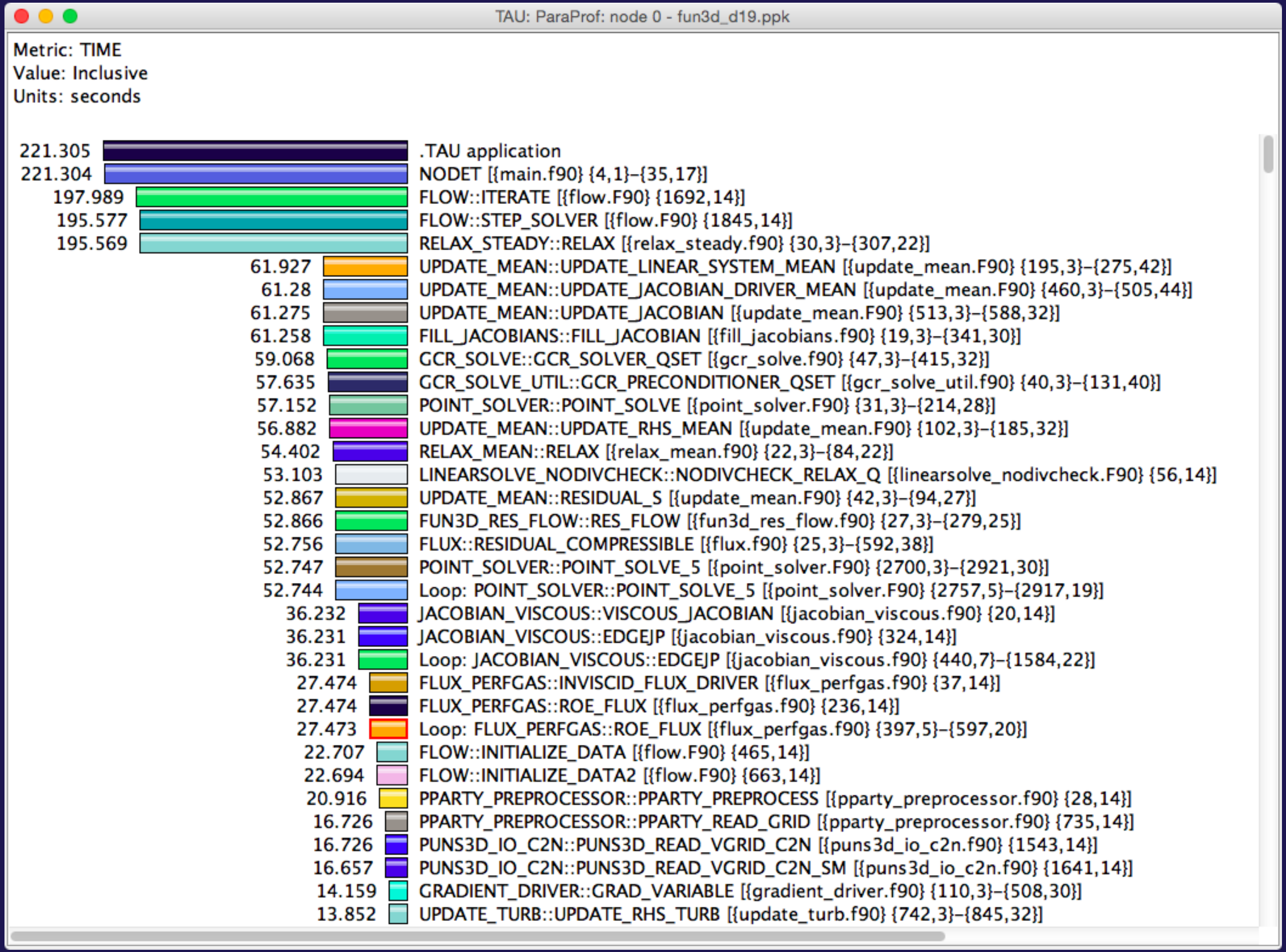
Instrumentation



- 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

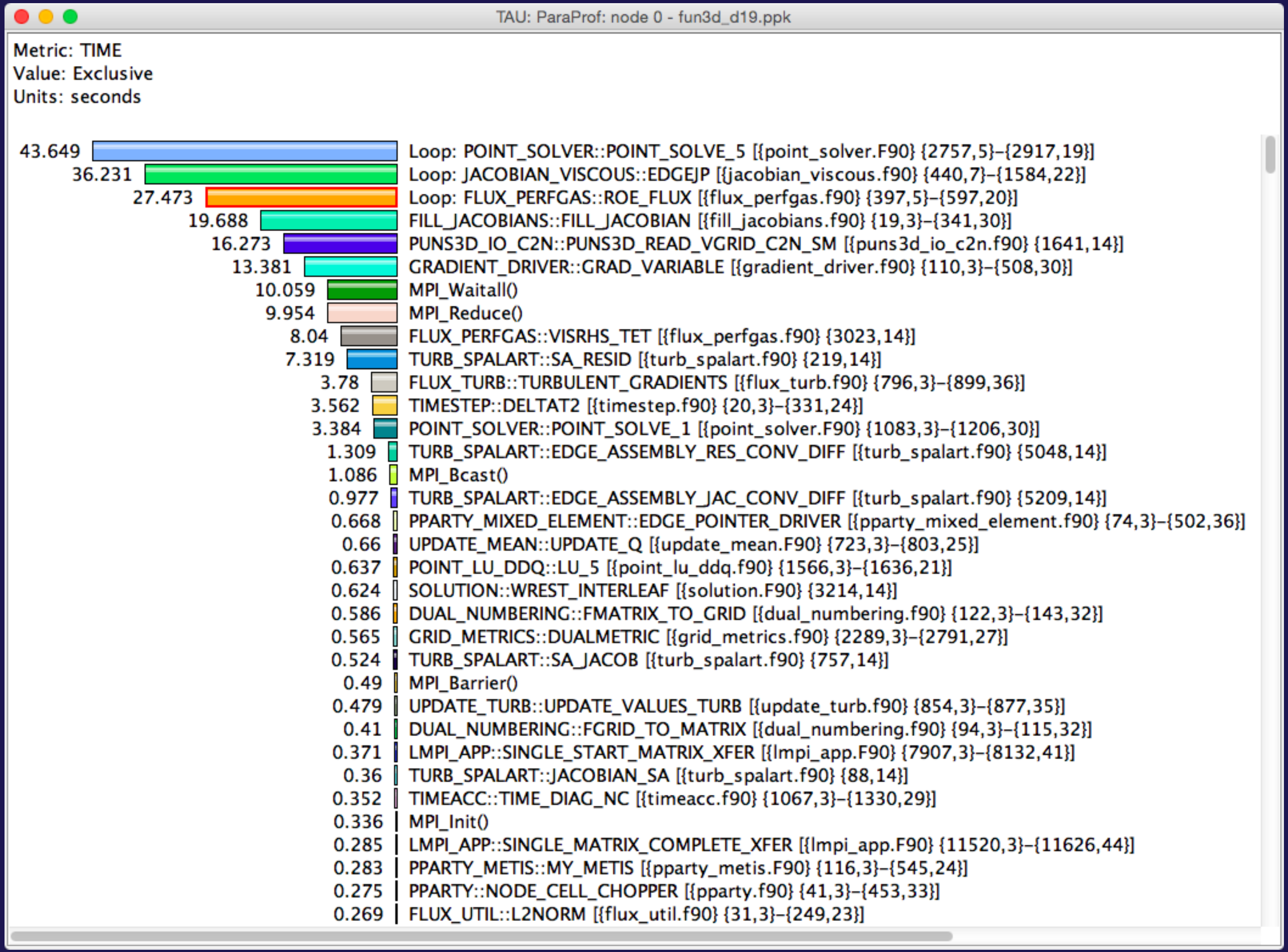
```
int main()
{
  int i;
  TAU_START("main");
  for (i=0; i < 3; i++)
    foo(i);
  TAU_STOP("main");
  return 0;
}

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

Inclusive Measurements



Argonne NATIONAL LABORATORY

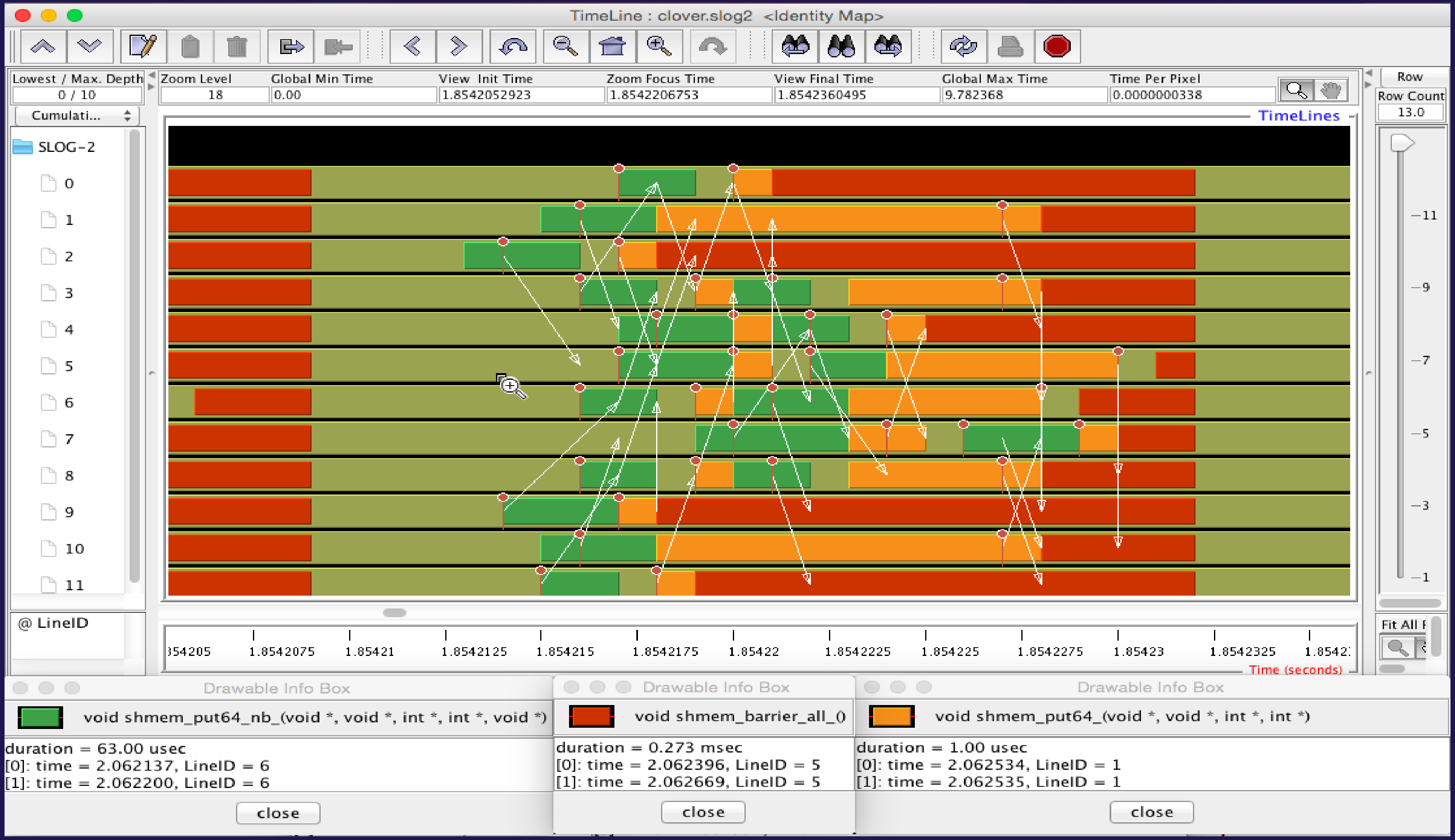
Exclusive Time



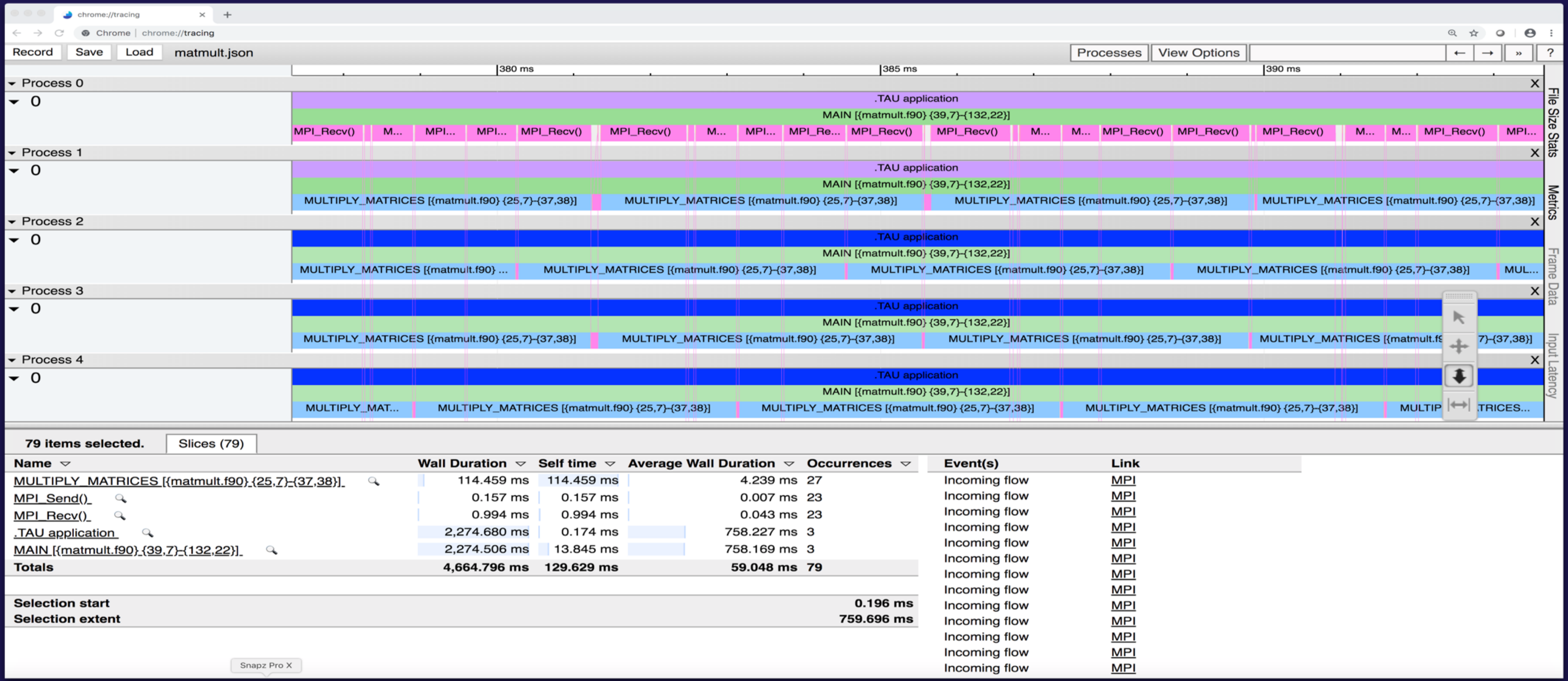
TAU’s Runtime Environment Variables

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>)

Tracing: Jumpshot [ANL] (ships with TAU)



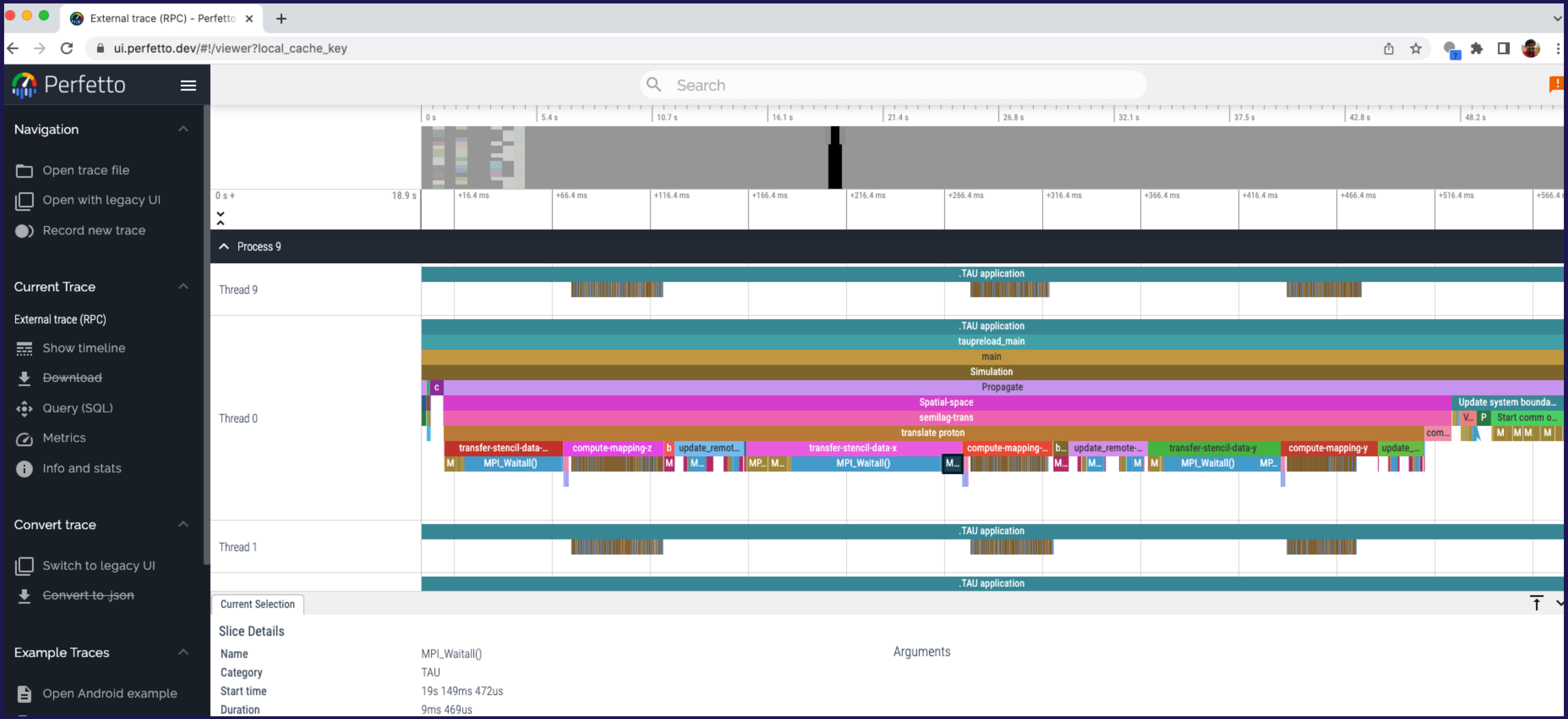
Tracing: Chrome Browser



% export TAU_TRACE=1
% mpirun –np 256 tau_exec ./a.out
% tau_treemerge.pl; tau_trace2json tau.trc tau.edf –chrome –ignoreatomic –o app.json

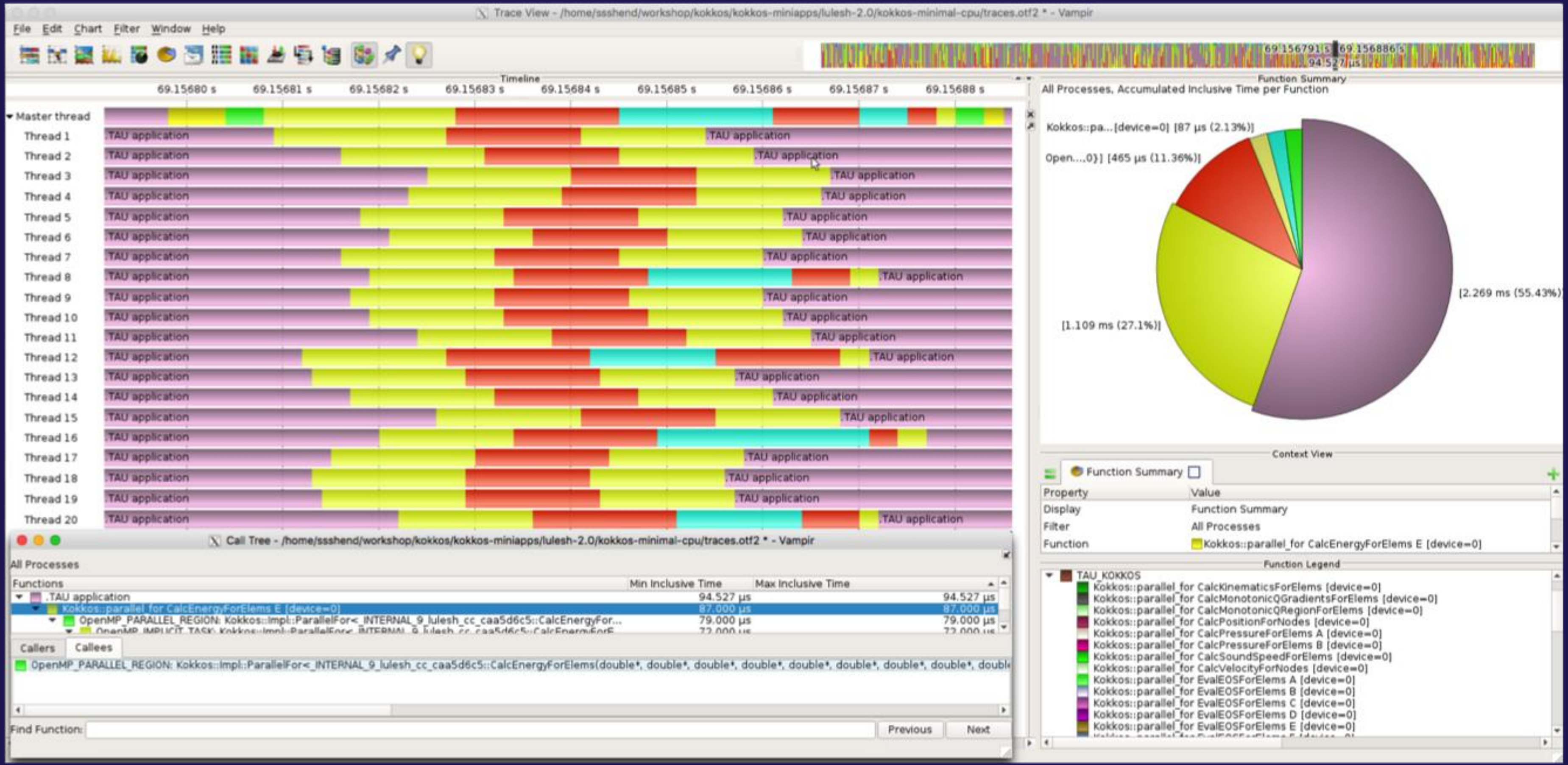
Chrome browser: chrome://tracing (Load -> app.json) OR use https://perfetto.dev

Perfetto.dev



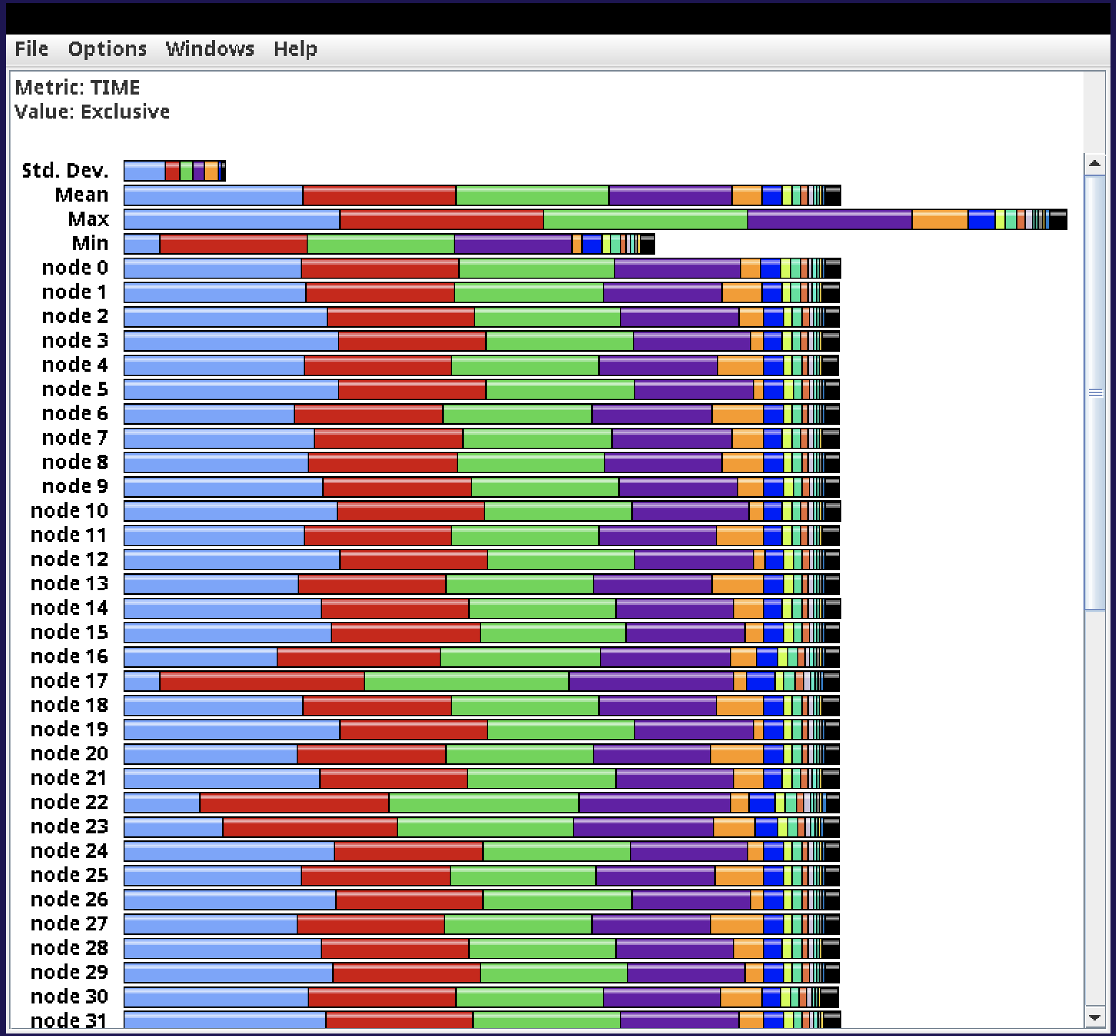
Argonne NATIONAL LABORATORY

Vampir [TU Dresden] Timeline: Kokkos



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

ParaProf Profile Browser

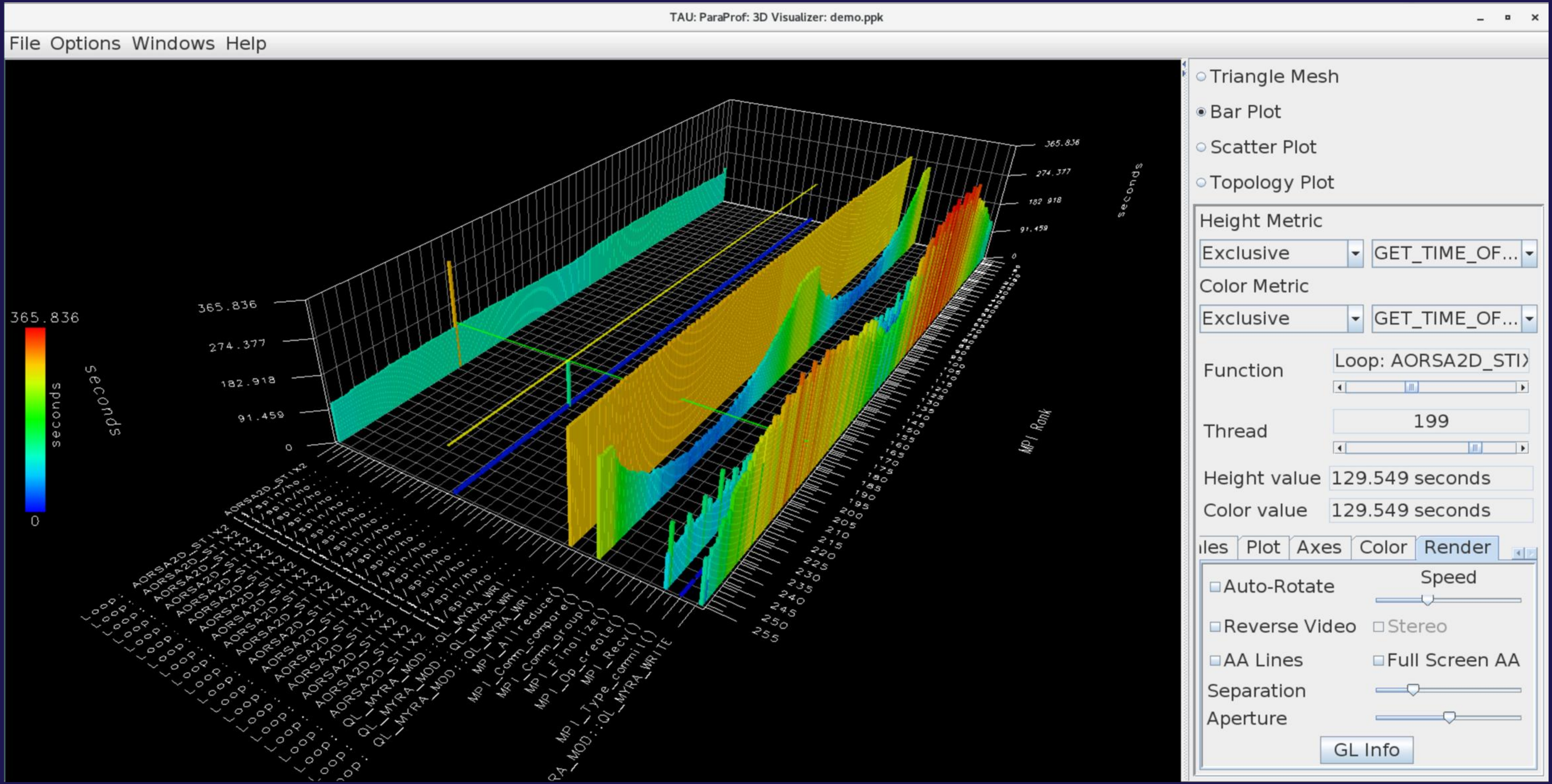



```
% paraprof
```

On a Mac if there are black windows:

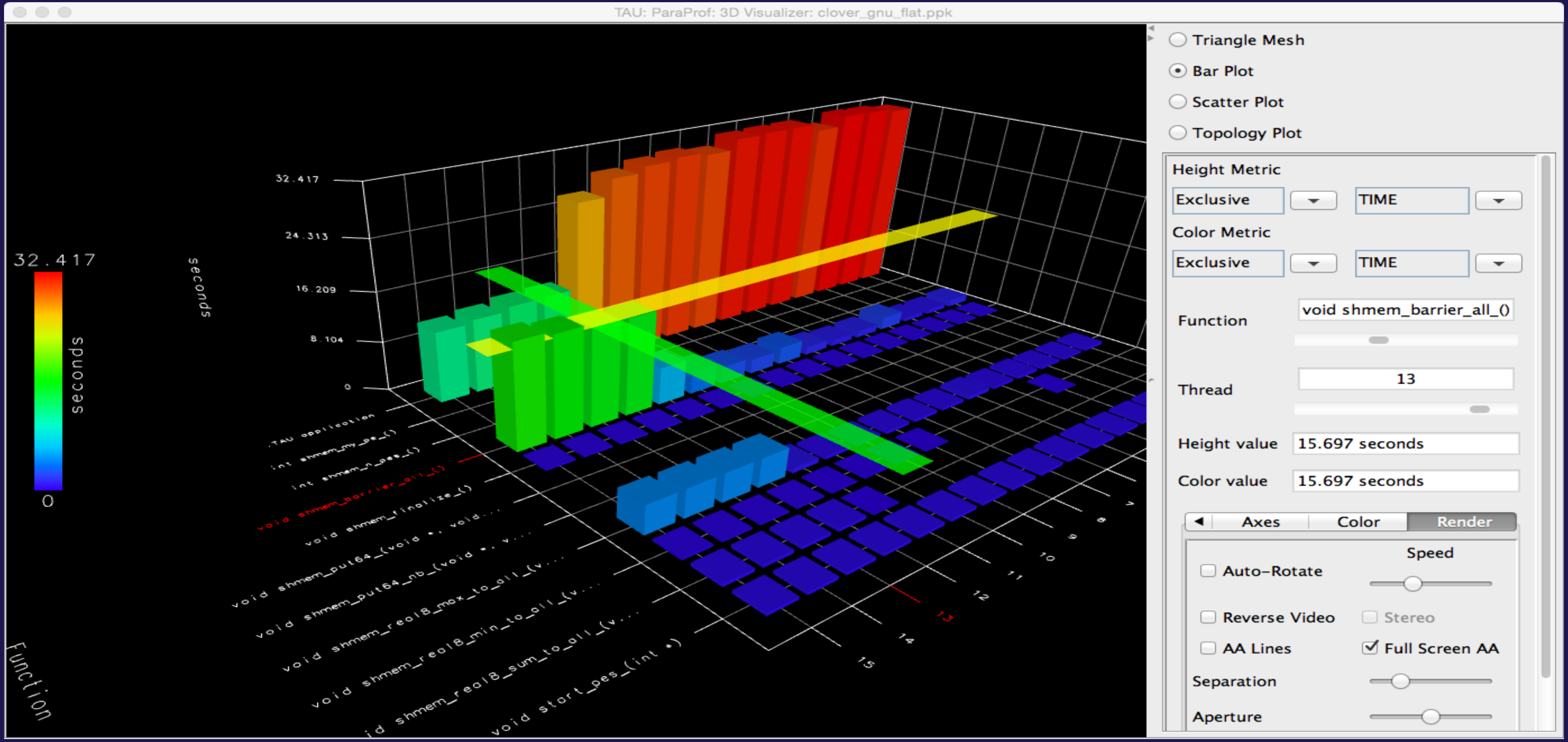
```
% paraprof -fix-xquartz
```

ParaProf 3D Profile Browser



Argonne NATIONAL LABORATORY

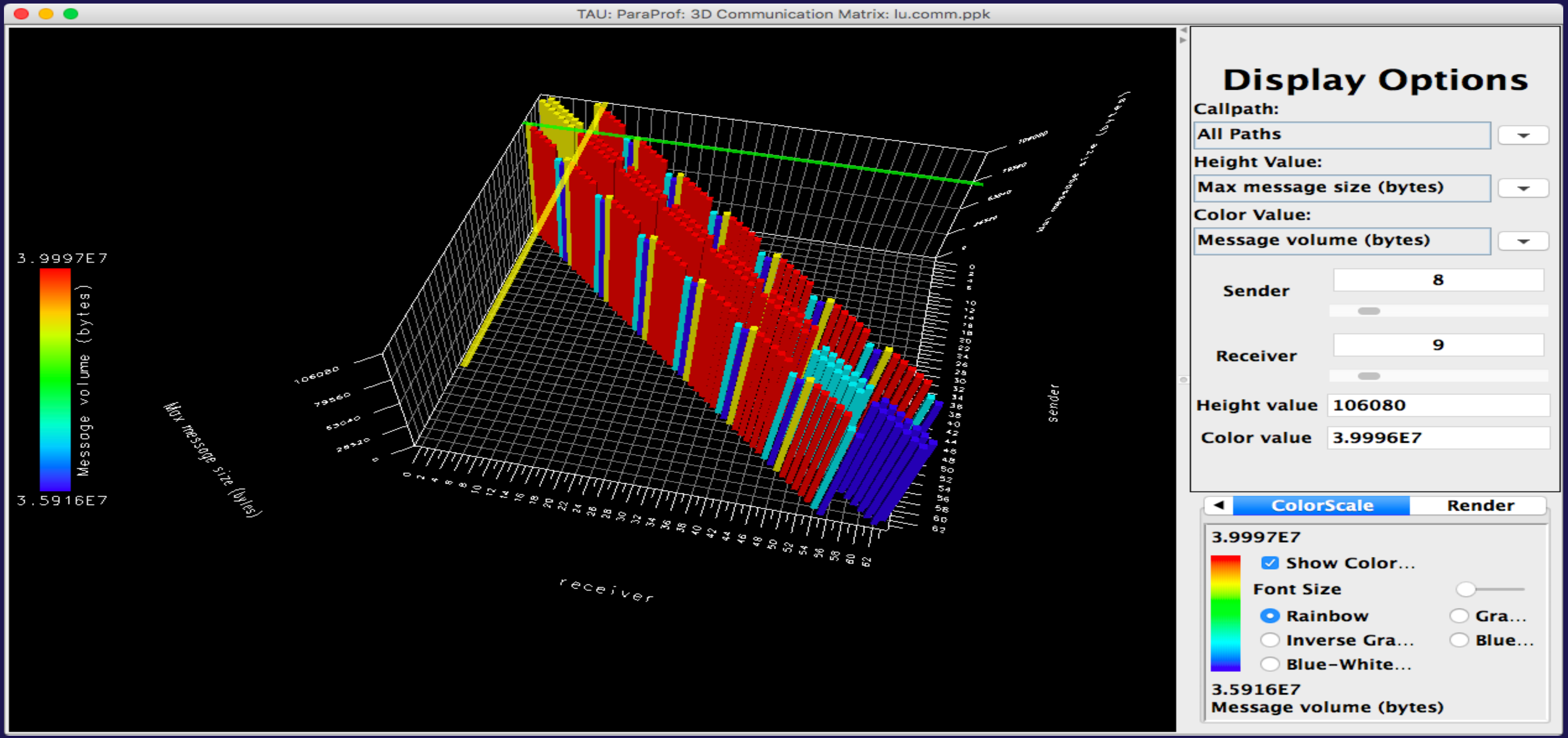
TAU – ParaProf 3D Visualization



% paraprof app.ppk

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

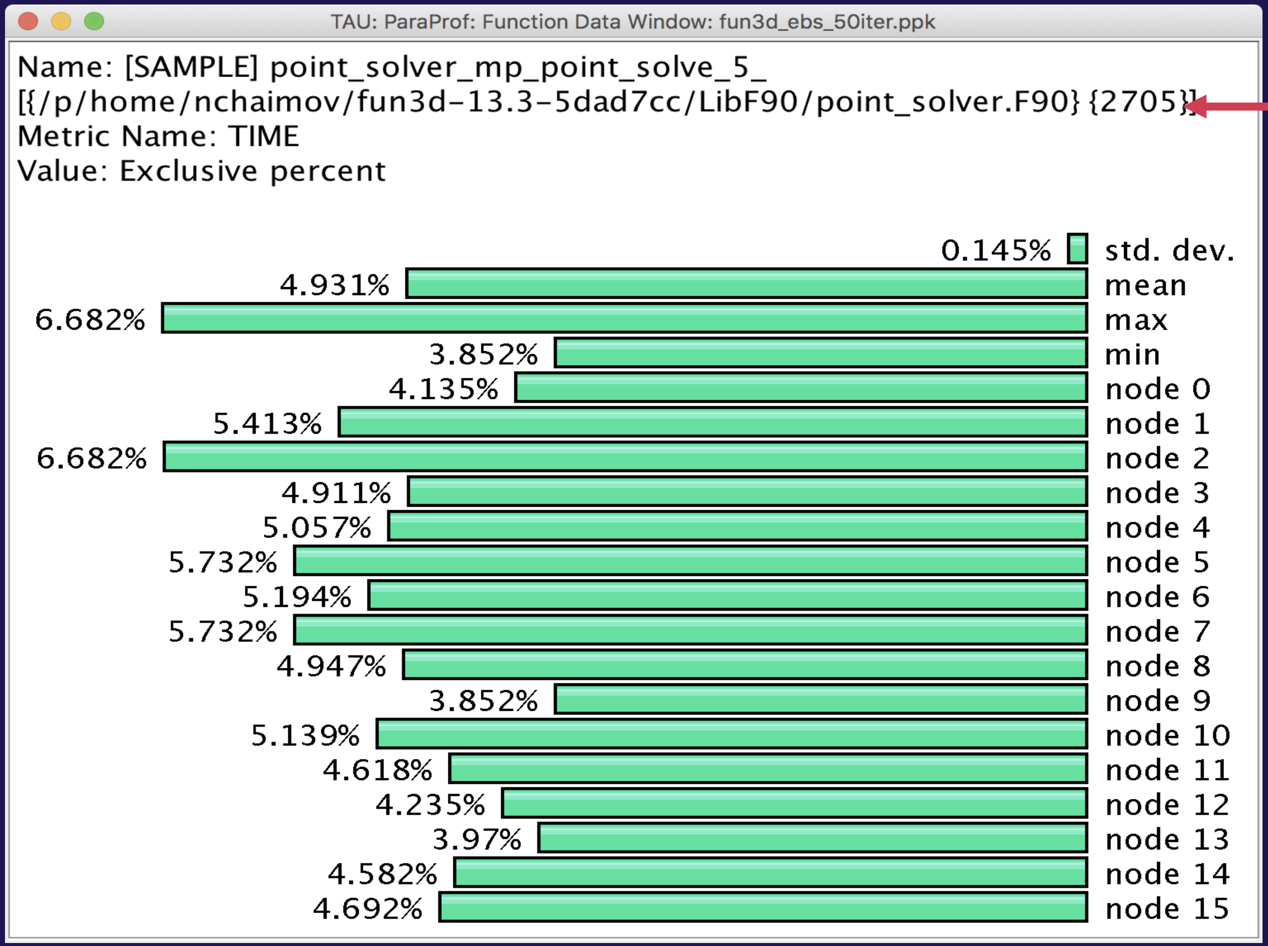
TAU – 3D Communication Window



% 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 –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
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

Tuning and Analysis Utilities

https://tau.uoregon.edu

https://e4s.io [TAU in Docker/Singularity containers]

for more information

Free download, open source, BSD license

Installing and Configuring TAU

- Installing PDT:
  - —wget tau.uoregon.edu/pdt_lite.tgz
  - —./configure –prefix=<dir>; make ; make install
- Installing TAU:
  - — wget tau.uoregon.edu/tau.tgz; tar zxf tau.tgz; cd tau-2.<ver>
  - — wget http://tau.uoregon.edu/ext.tgz ; tar xf ext.tgz
  - — ./configure -bfd=download -pdt=<dir> -papi=<dir> -mpi
    –pthread –c++=mpicxx –cc=mpicc –fortran=mpif90
    –dwarf=download –unwind=download –otf=download
    –iowrapper –papi=<dir>
  - —make install
- Using TAU for source instrumentation (not needed with tau_exec):
  - —export TAU_MAKEFILE=<taudir>/x86_64/lib/Makefile.tau-<TAGS>
  - —make CC=tau_cc.sh CXX=tau_cxx.sh F90=tau_f90.sh

Compile-Time Options

•Optional parameters for the TAU_OPTIONS environment variable:
% tau_compiler.sh

-optVerbose	Turn on verbose debugging messages
-optCompInst	Use compiler based instrumentation
-optNoCompInst	Do not revert to compiler instrumentation if source Instrumentation fails.
-optTrackIO	Wrap POSIX I/O call and calculates vol/bw of I/O operations (Requires TAU to be configured with –*iowrapper*)
-optTrackGOMP	Enable tracking GNU OpenMP runtime layer (used without –opari)
-optMemDbg	Enable runtime bounds checking (see TAU_MEMDBG_* env vars)
-optKeepFiles	Does not remove intermediate .pdb and .inst.* files
-optPreProcess	Preprocess sources (OpenMP, Fortran) before instrumentation
-optTauSelectFile="<file>"	Specify selective instrumentation file for *tau_instrumentor*
-optTauWrapFile="<file>"	Specify path to *link_options.tau* generated by *tau_gen_wrapper*
-optHeaderInst	Enable Instrumentation of headers
-optTrackUPCR	Track UPC runtime layer routines (used with tau_upc.sh)
-optLinking=""	Options passed to the linker. Typically $(TAU_MPI_FLIBS) $(TAU_LIBS) $(TAU_CXXLIBS)
-optCompile=""	Options passed to the compiler. Typically $(TAU_MPI_INCLUDE) $(TAU_INCLUDE) $(TAU_DEFS)
-optPdtF95Opts=""	Add options for Fortran parser in PDT (f95parse/gfparse) …

Compile-Time Options (contd.)

•Optional parameters for the TAU_OPTIONS environment variable:
% tau_compiler.sh

-optShared	Use TAU's shared library (libTAU.so) instead of static library (default)
-optPdtCxxOpts=""	Options for C++ parser in PDT (cxxparse).
-optPdtF90Parser=""	Specify a different Fortran parser
-optPdtCleanscapeParser	Specify the Cleanscape Fortran parser instead of GNU gfparser
-optTau=""	Specify options to the tau_instrumentor
-optTrackDMAPP	Enable instrumentation of low-level DMAPP API calls on Cray
-optTrackPthread	Enable instrumentation of pthread calls

See tau_compiler.sh for a full list of TAU_OPTIONS.

…

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 –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
TAU_EVENT_THRESHOLD	0.5	Define a threshold value (e.g., .25 is 25%) to trigger marker events for min/max

Hands-On

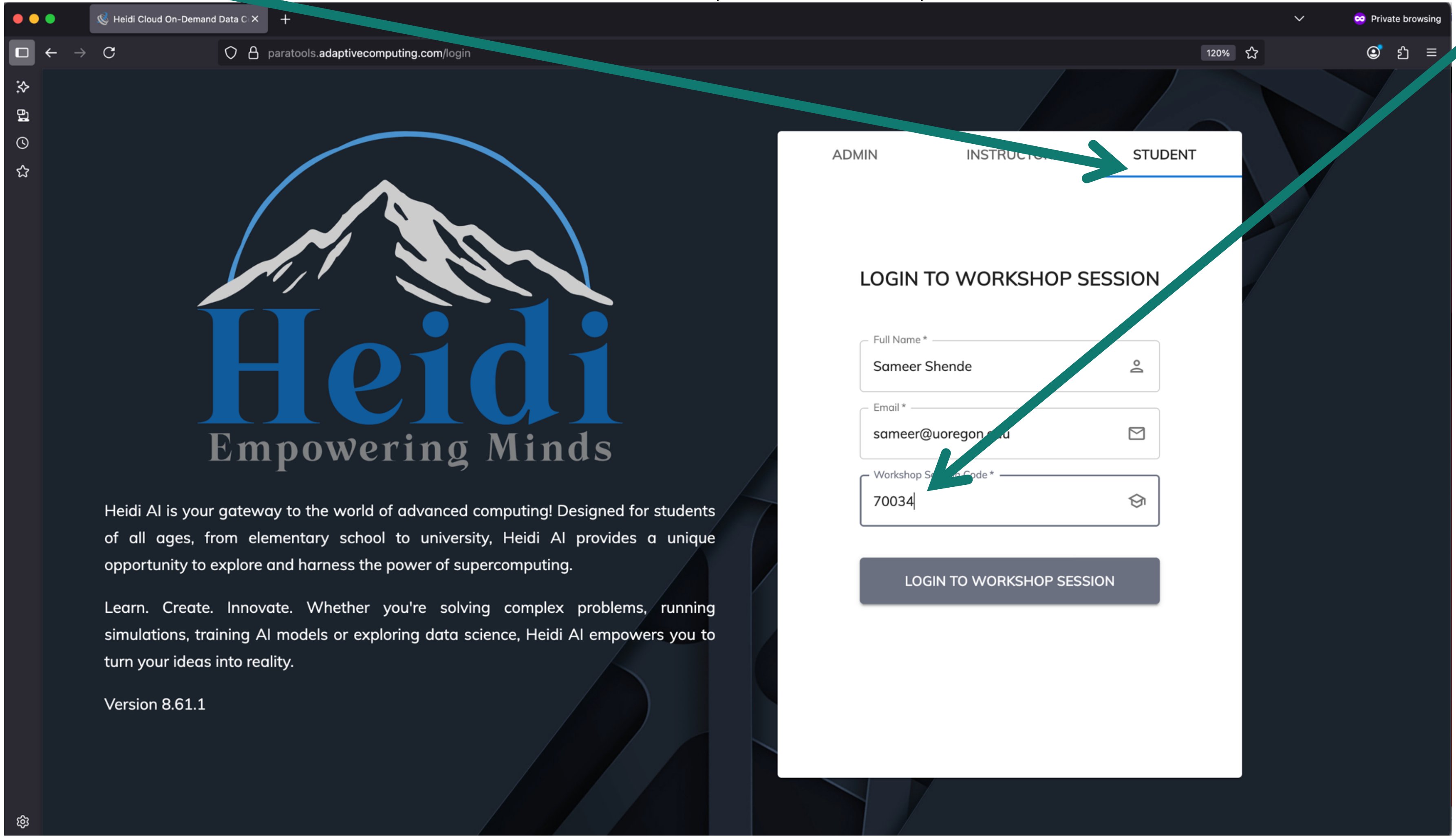
AWS

Heidi from Adaptive Computing Enterprises, Inc.

ParaTools Pro for E4S™ image

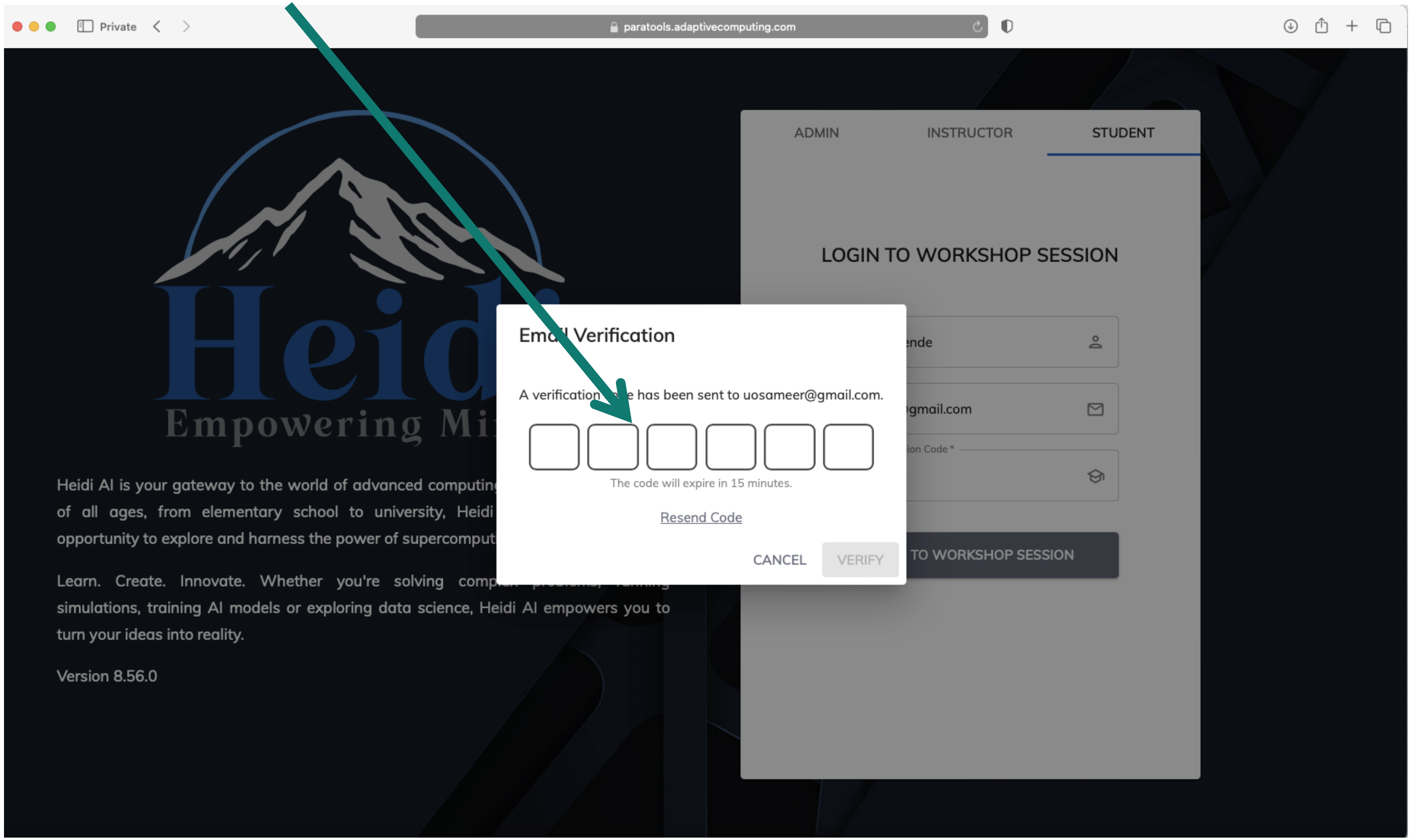
Connect to https://paratools.adaptivecomputing.com

- Use Student tab and enter name, email, session code 70034



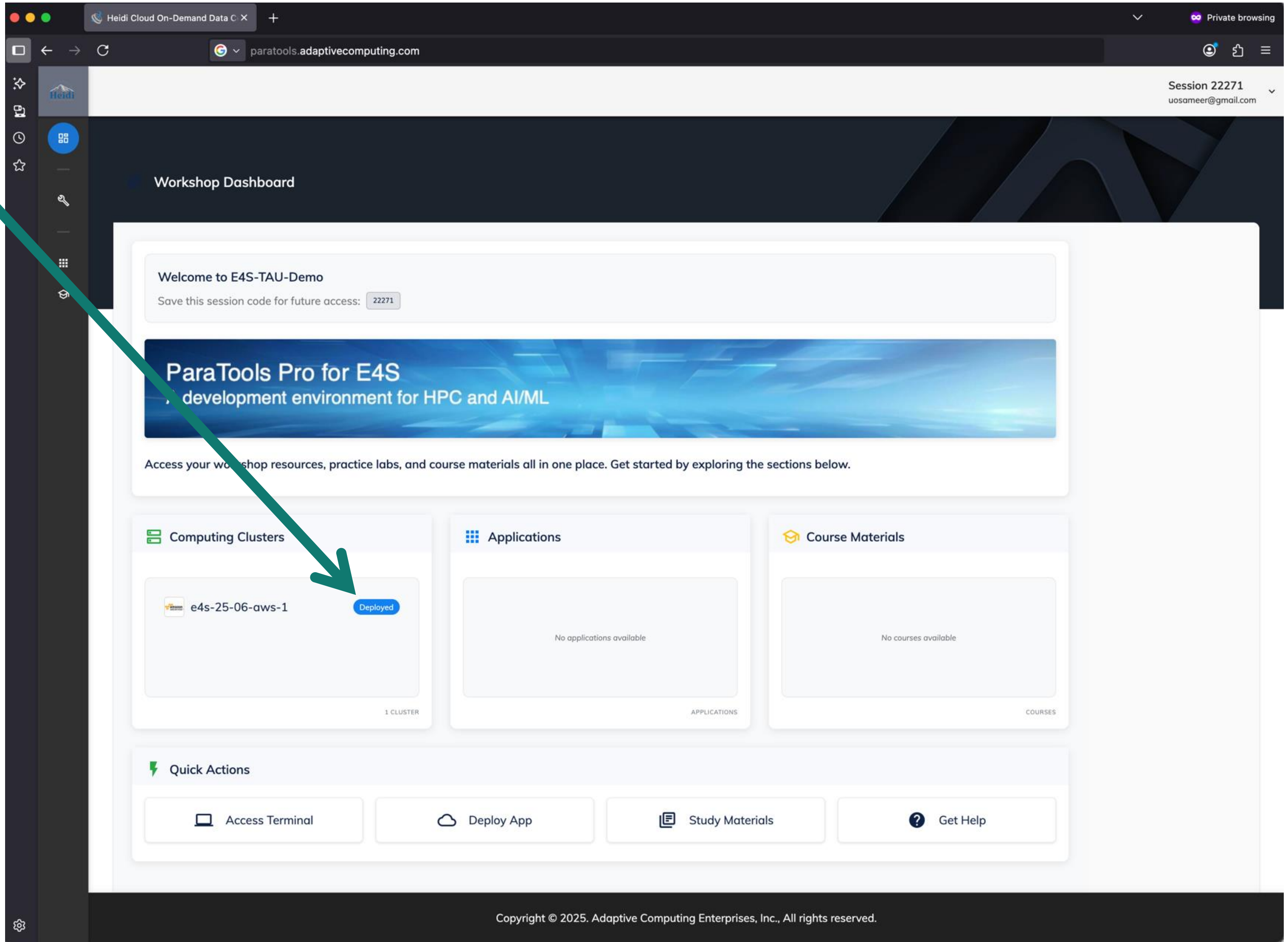
Connect to Students tab with code 70034 at https://paratools.adaptivecomputing.com

- Check your email, enter verification code.



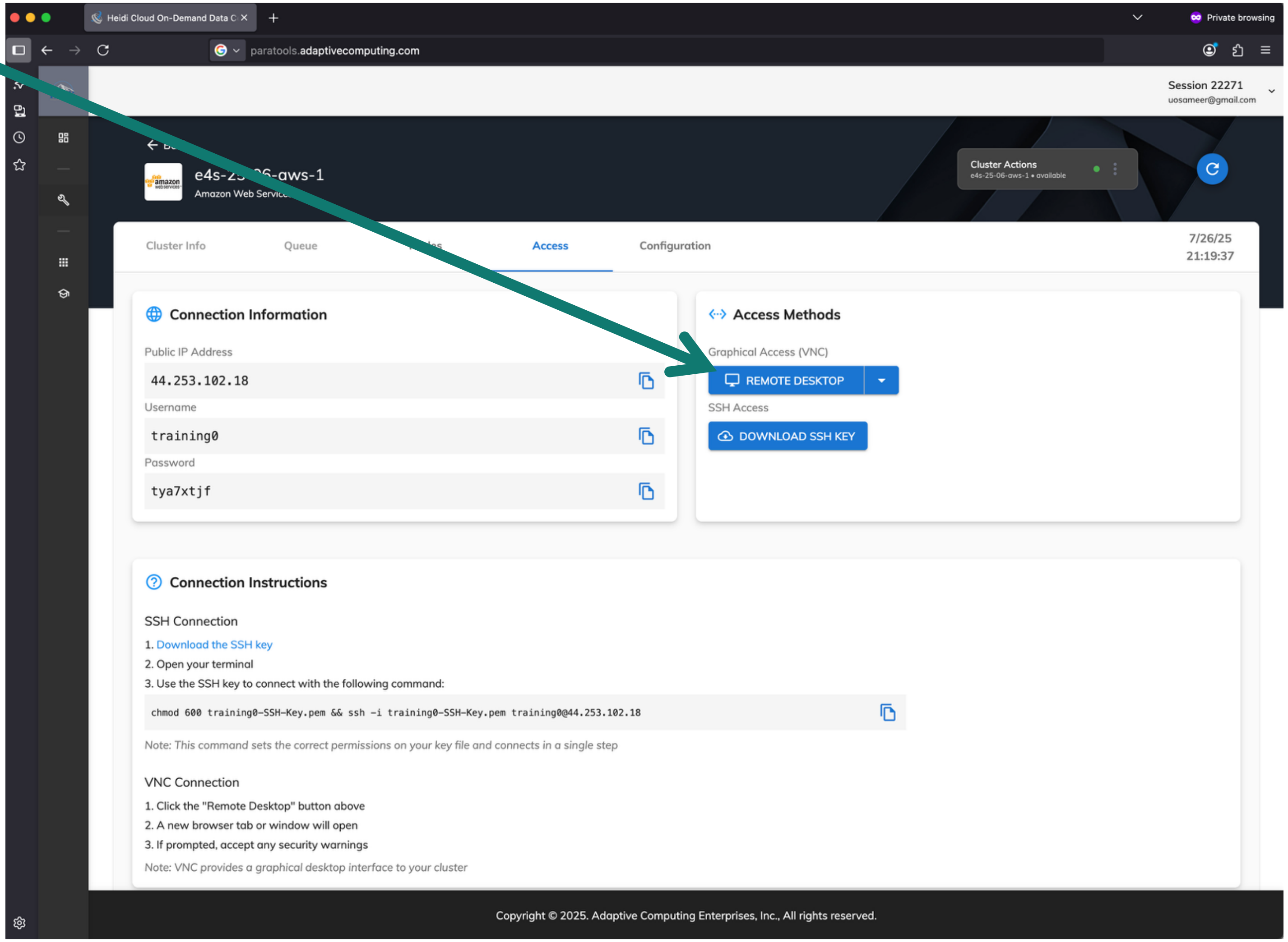
Connect to Students tab with code 70034 at https://paratools.adaptivecomputing.com

- Click cluster



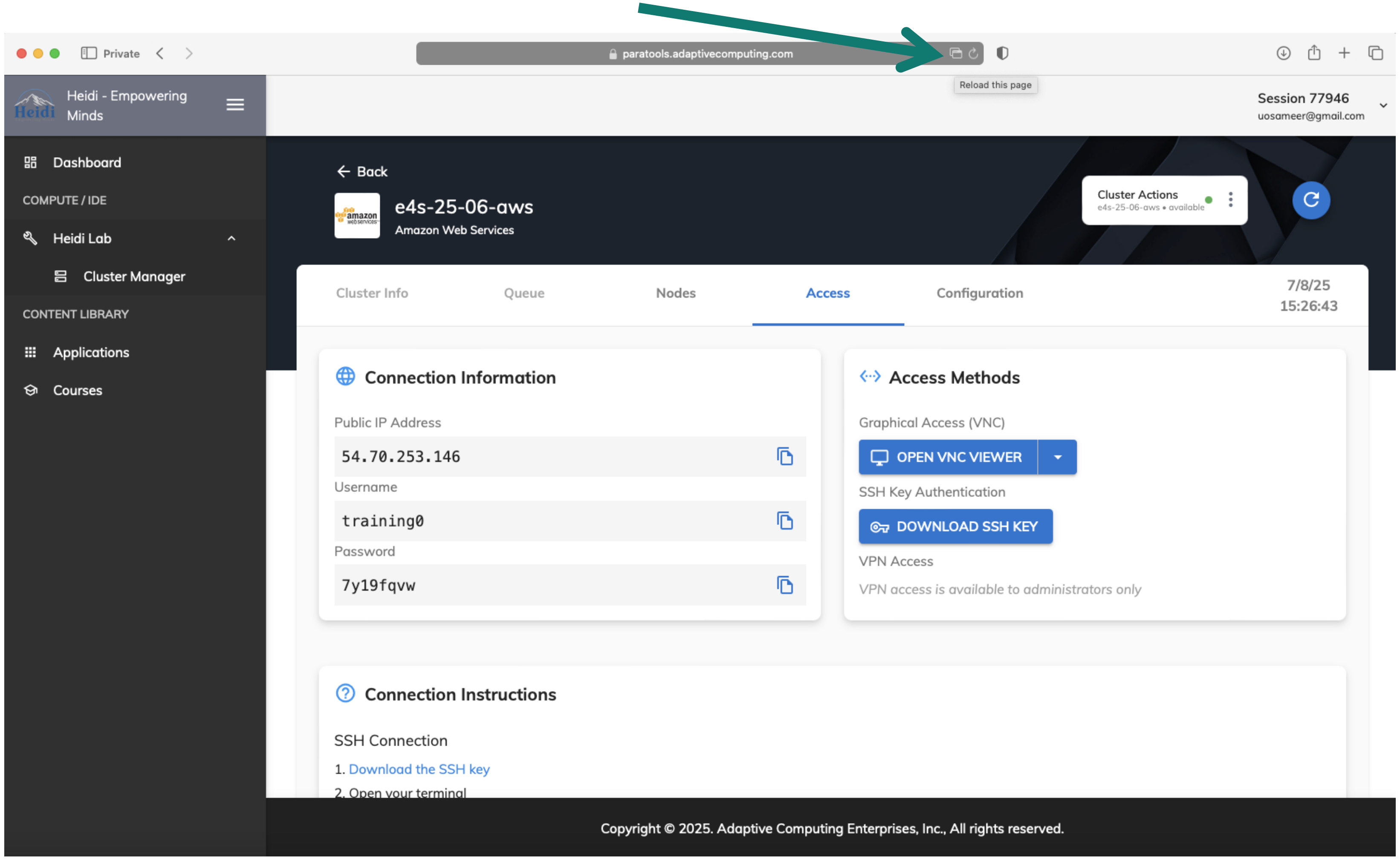
Connect to Students tab with code 70034 at https://paratools.adaptivecomputing.com

- Click Remote Desktop



Connect to Students tab with code 70034 at https://paratools.adaptivecomputing.com

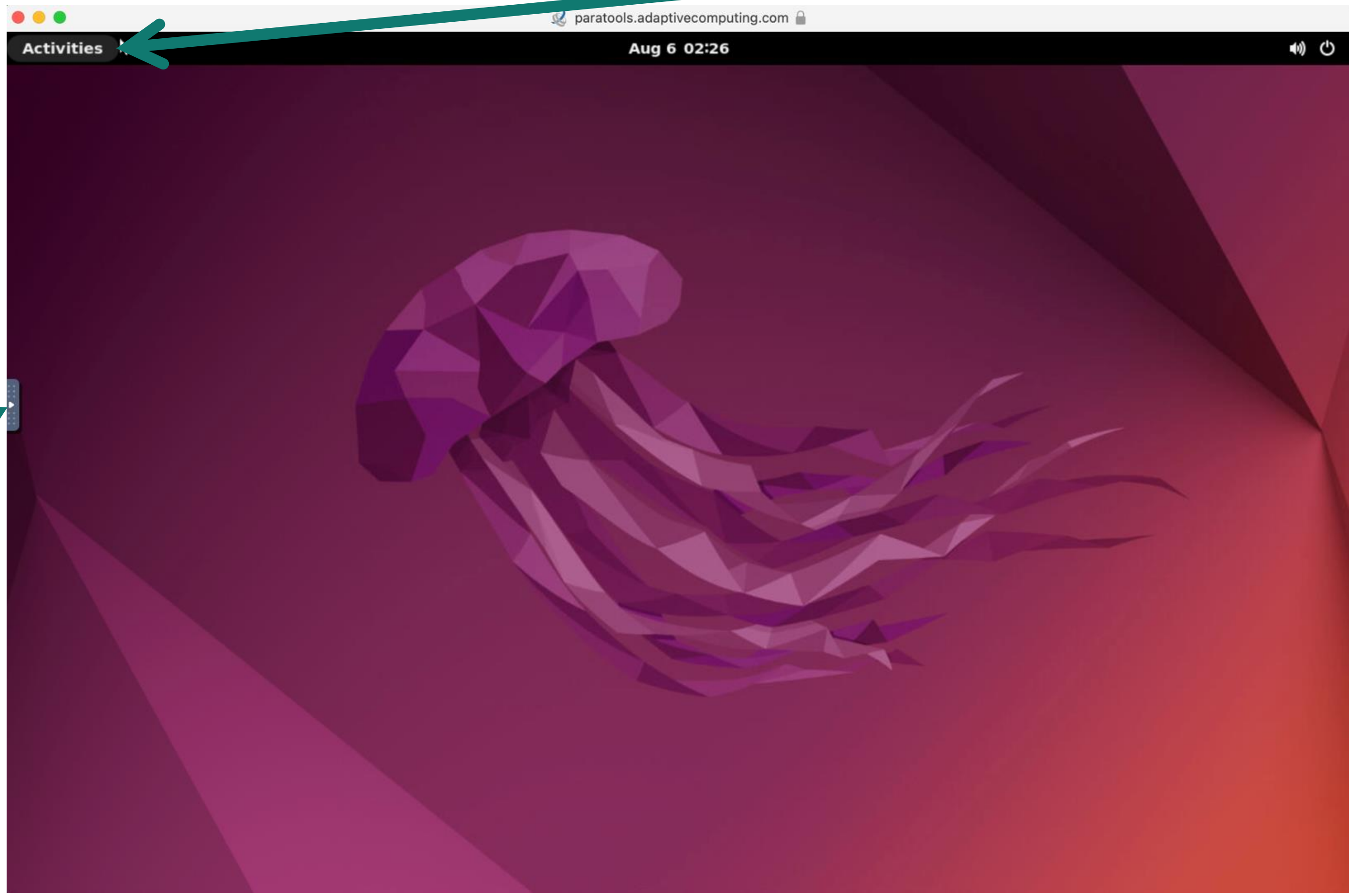
- You may have to enable pop-up windows and accept



Connect to Students tab with code 70034 at https://paratools.adaptivecomputing.com

- You should see this jellyfish. Click on Activities.

To copy text from other windows click on the this button to access the clipboard



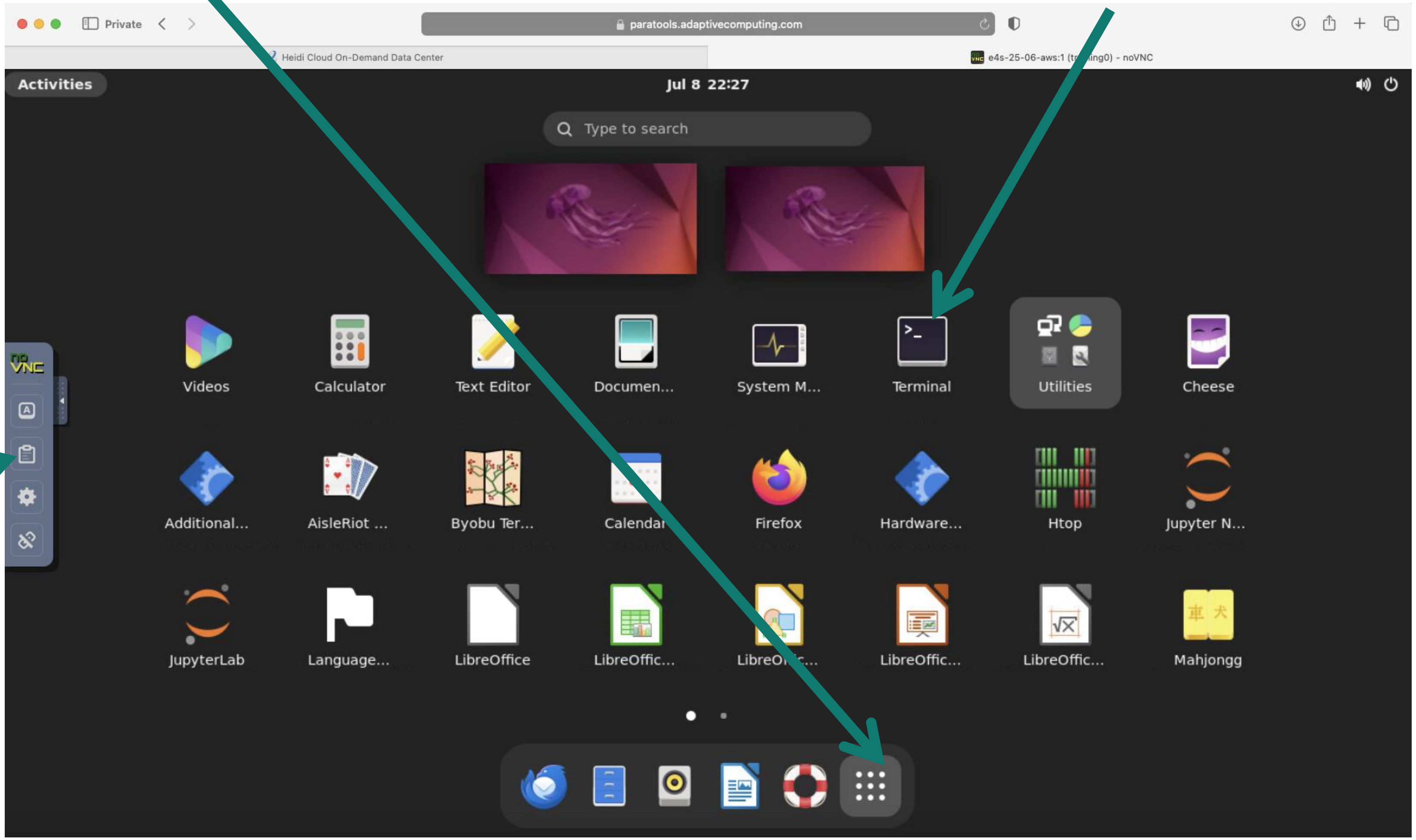
If this doesn't work for you!

- Go to https://uooddc.nic.uoregon.edu student tab, enter 25724 as the code. Try again.

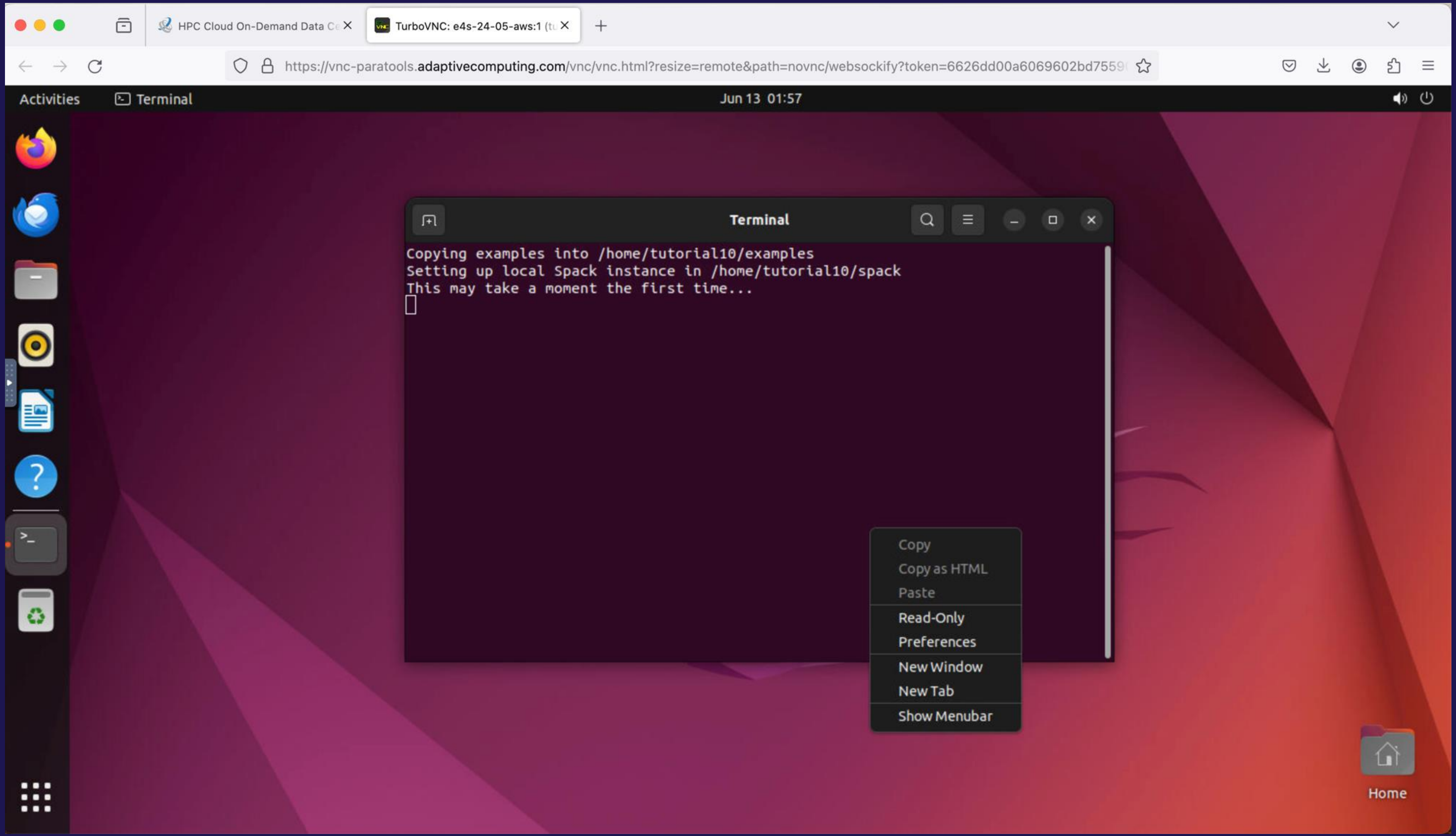
Connect to Students tab with code 70034 at https://paratools.adaptivecomputing.com

- Click on Activities, nine dots, and then select the Terminal application

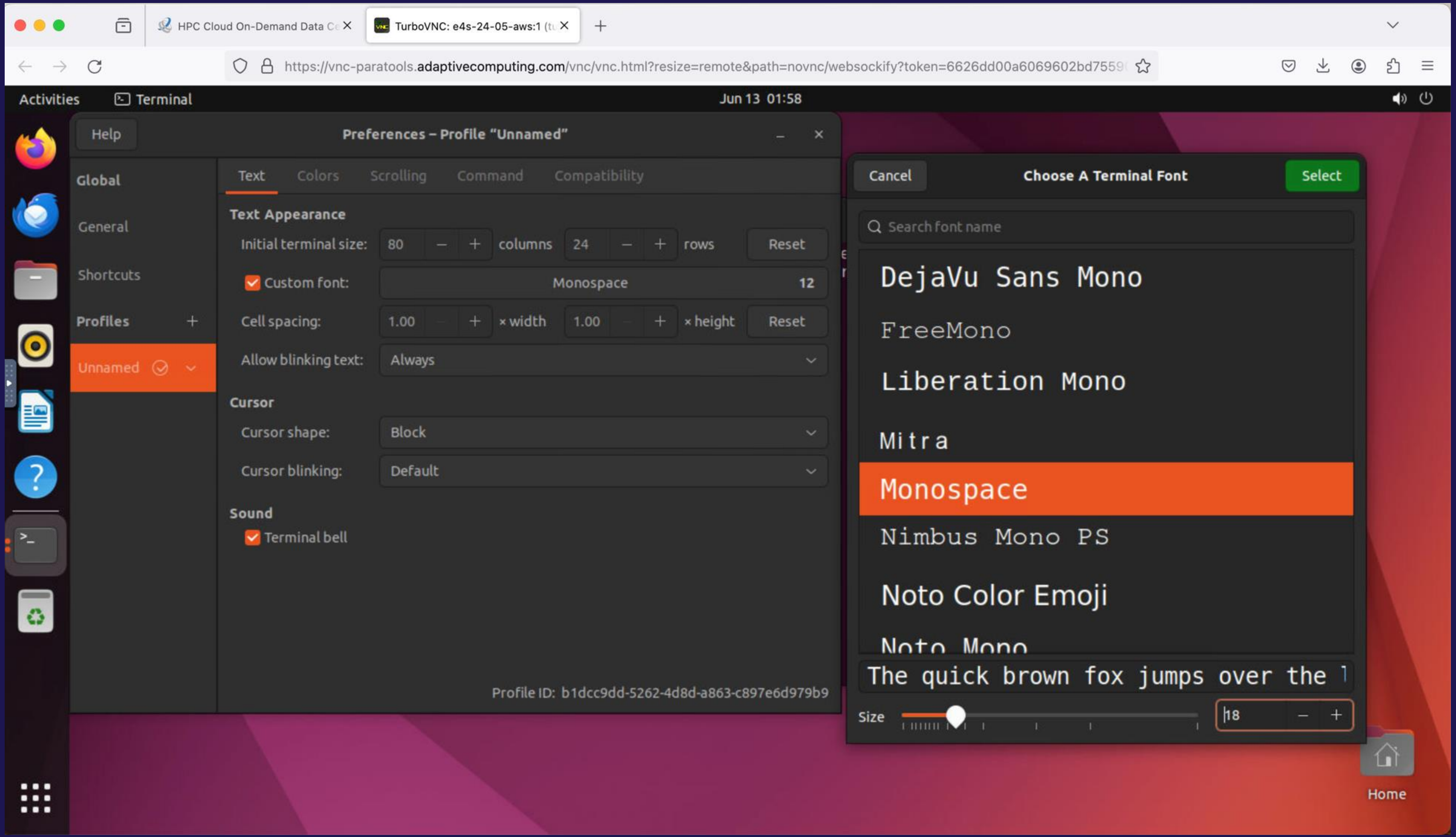
To copy text from other windows click on the clipboard



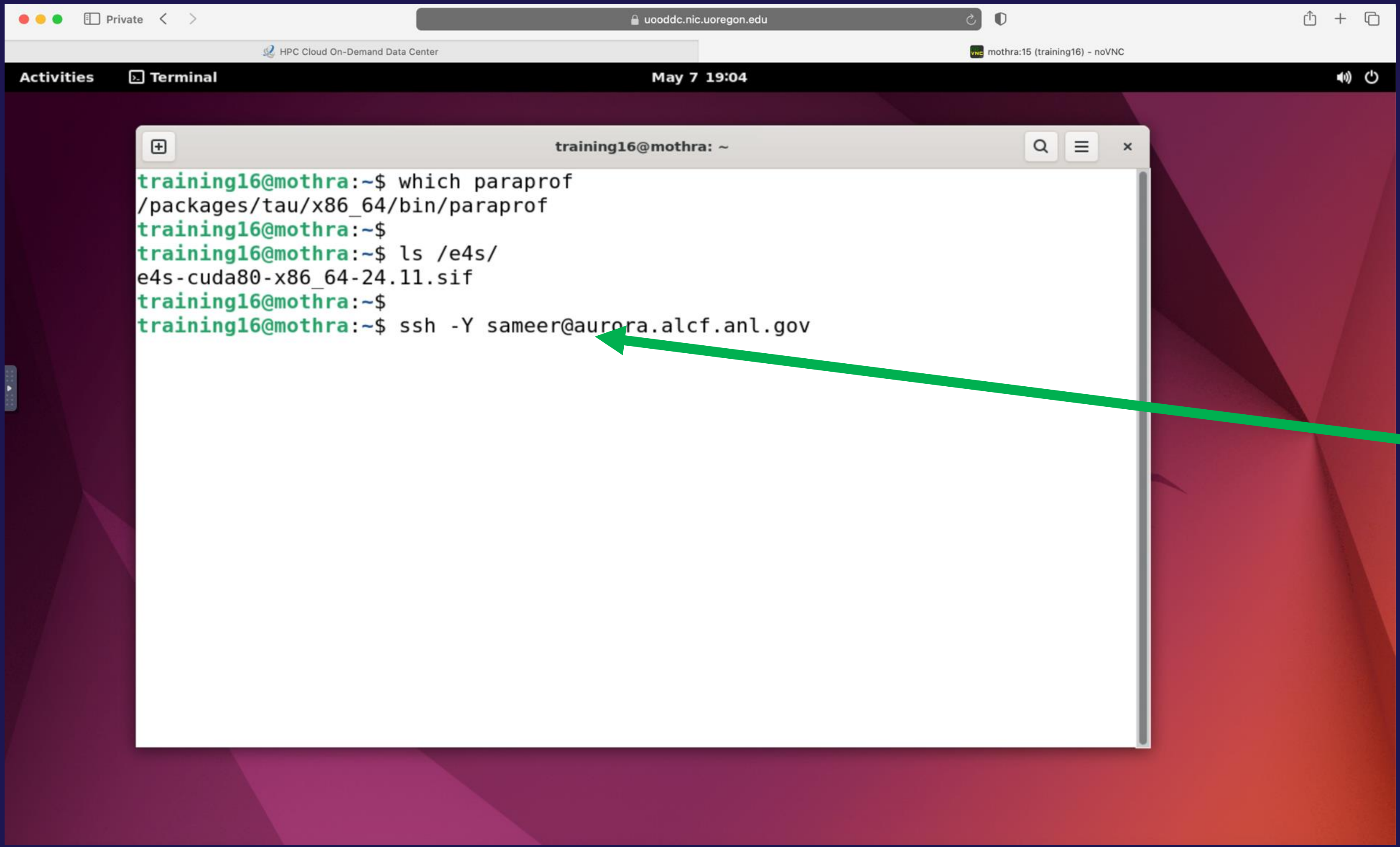
To increase font size right click and choose preferences



Choose font size after clicking Custom Font for Terminal

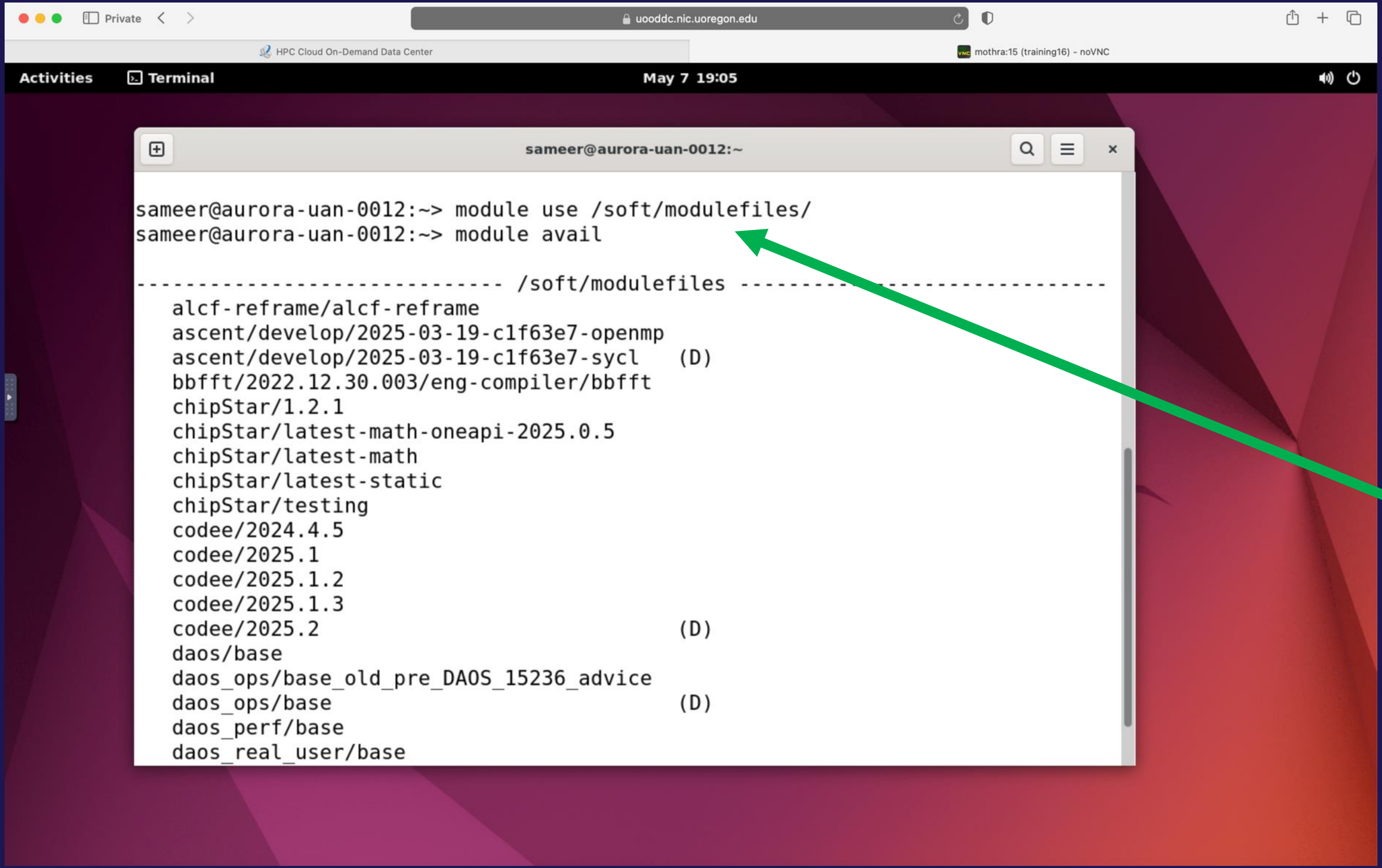


Adaptive Computing's Heidi: Logging into Aurora



ssh using -Y

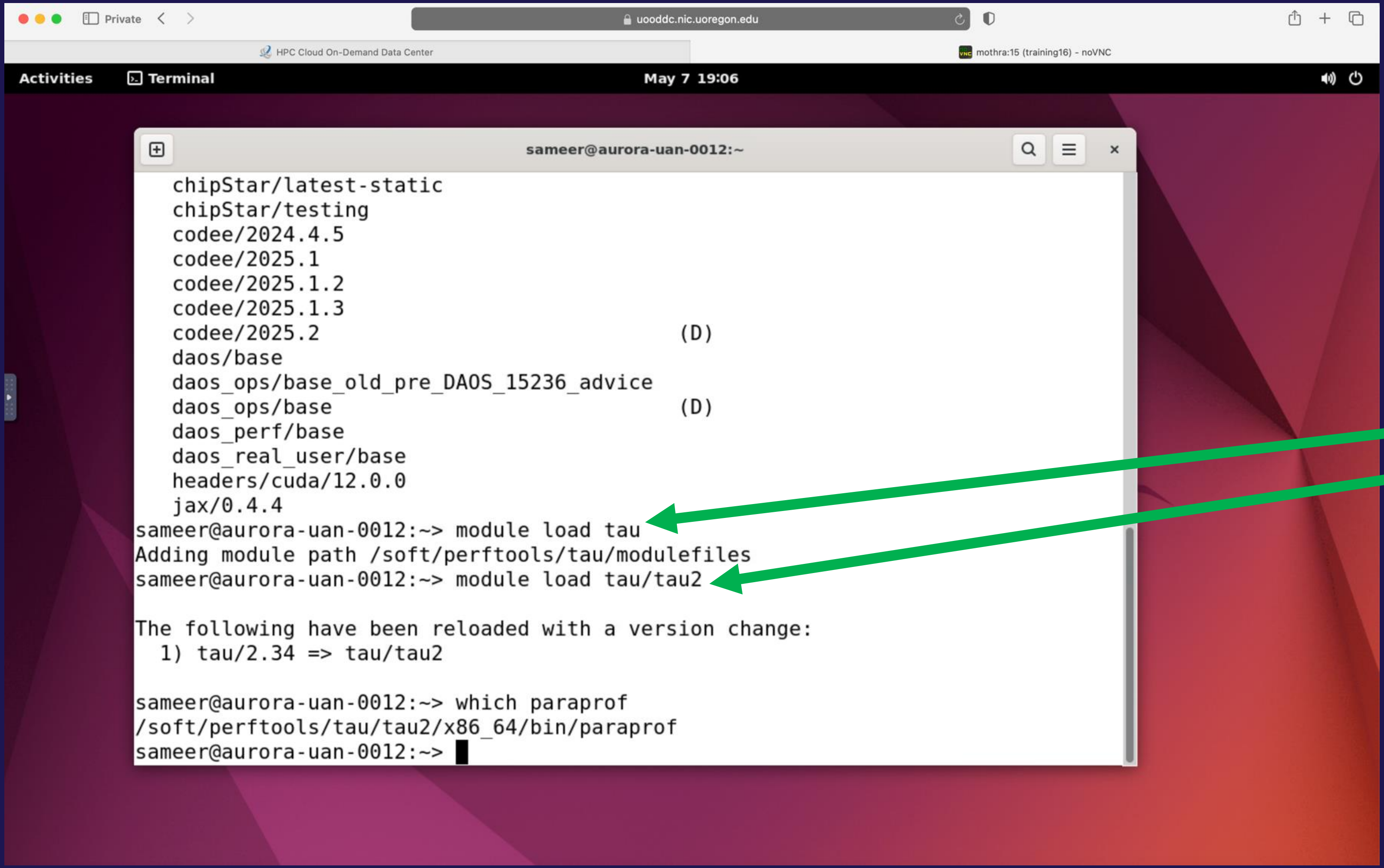
Aurora: Add a directory to the default module path



module use /soft/modulefiles

Argonne NATIONAL LABORATORY

Aurora: Loading TAU



Load TAU using
module load tau
module load tau/tau2

Aurora: Setting up workshop examples

```
sameer@aurora-uan-0012:~/workshop> qsub -I  -l walltime=0:29:00 -lfilesystems=home:flare -A tools -q debug -l select=1 -X
qsub: waiting for job 4683648.aurora-pbs-0001.hostmgmt.cm.aurora.alcf.anl.gov to start
qsub: job 4683648.aurora-pbs-0001.hostmgmt.cm.aurora.alcf.anl.gov ready

sameer@x4106c0s5b0n0:~> module use /soft/modulefiles
sameer@x4106c0s5b0n0:~> module load tau
Adding module path /soft/perftools/tau/modulefiles
sameer@x4106c0s5b0n0:~> module load tau/tau2

The following have been reloaded with a version change:
  1) tau/2.34 => tau/tau2

sameer@x4106c0s5b0n0:~> module load frameworks
(/opt/aurora/24.347.0/frameworks/aurora_nre_models_frameworks-2025.0.0) sameer@x4106c0s5b0n0:~> which python
/opt/aurora/24.347.0/frameworks/aurora_nre_models_frameworks-2025.0.0/bin/python
(/opt/aurora/24.347.0/frameworks/aurora_nre_models_frameworks-2025.0.0) sameer@x4106c0s5b0n0:~> which tau_exec
/soft/perftools/tau/tau2/x86_64/bin/tau_exec
(/opt/aurora/24.347.0/frameworks/aurora_nre_models_frameworks-2025.0.0) sameer@x4106c0s5b0n0:~> which paraprof
/soft/perftools/tau/tau2/x86_64/bin/paraprof
(/opt/aurora/24.347.0/frameworks/aurora_nre_models_frameworks-2025.0.0) sameer@x4106c0s5b0n0:~> mpiexec -n 4 tau_exec -l0 -ebs ./a.out
```

Use -X for GUIs

Use your allocation alcf_training and queue alcf_training

Load TAU on compute nodes

Using TAU on Aurora

```
tar xf /soft/perftools/tau/tar/workshop.tgz
cd workshop; cat README

qsub -I  -l walltime=1:29:00 -lfilesystems=home:flare -A alcf_training
                        -q alcf_training -l select=1 -X

module use /soft/modulefiles
module load tau
module load tau/tau2                # Uses the latest git head version
module load frameworks

mpiexec -n 4 ./a.out
mpiexec -n 4 tau_exec -l0 -ebs ./a.out

mpiexec -n 4 tau_exec -l0 -ebs python ./foo.py
pprof -a  | more
paraprof
For large scale runs: export TAU_PROFILE_FORMAT="merged" and use
paraprof tauprofile.xml &
```

Using TAU on Aurora

```
# After allocating a node and adding TAU to your path with the module commands
cat workshop/README; cd workshop/pti-gpu/samples
cd mpi_dpc_gemm; ./run.sh; cd build; paraprof & ; cd ../..
cd mpi_omp_gemm; ./run.sh; cd build; paraprof & ; cd ../..
# Similarly execute run.sh in dpc_gemm, omp_gemm, ze_gemm directories

# AI examples.
cd ~/workshop/aurora-callbacks; ./run.sh; paraprof & ; cd ..
cd pytorch; ./run.sh; paraprof &; cd ..
cd tensorflow; ./run.sh; paraprof &
```

Using TAU on Polaris natively

- Setup preferred program environment compilers (check instructions)

```
% ssh -Y <login>@polaris.alcf.anl.gov
% module load tau
% tar zxf /soft/perftools/tau/tar/workshop.tgz; cd workshop
% paraprof demo.ppk &

If you are on a Mac with Xquartz, you may need:
% paraprof -fix-xquartz demo.ppk &
In the directory where profile.* files are created. Xquartz 2.7.4 works well without this.
Please do not use paraprof on the compute nodes. You may also use uooddc.nic.uoregon.edu or
AWS (paratools.adaptivecomputing.com) or install TAU locally on your laptop.
```

TAU Hands-on Session – CUDA with MPI on Polaris

- Setup preferred program environment compilers (check instructions)

```
% ssh -Y <login>@polaris.alcf.anl.gov
% module load tau
% tar zxf /soft/perftools/tau/tar/workshop.tgz
% cd workshop/TeaLeaf_CUDA;
% make clean
% make; cd bin
% qsub -I -q alcf_training -t 60 -n 1 -A alcf_training
% ./run.sh
% pprof -a | more
% paraprof --pack app.ppk
You may use paraprof --dump app.ppk to write out the profile.* files.
Bring ppk file to your desktop:
% paraprof app.ppk &
```

Setup: Installing TAU on Laptops

- Prerequisites: Java in your path
- Microsoft Windows
  - Install Java from Oracle.com
    - http://tau.uoregon.edu/tau.exe
    - Install, click on a ppk file to launch paraprof
- macOS (arm64, Apple Silicon M series)
  - http://tau.uoregon.edu/java_arm64.dmg
  - http://tau.uoregon.edu/tau_arm64.dmg
- macOS (x86_64)
  - Install Java 11.0.3:
    - Download and install 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:$PATH
  - Download and install TAU (copy to /Applications from dmg):
    - http://tau.uoregon.edu/tau.dmg
    - export PATH=/Applications/TAU/tau/apple/bin:$PATH
    - paraprof app.ppk &
- Linux (http://tau.uoregon.edu/tau.tgz)
  - ./configure; make install; export PATH=<taudir>/x86_64/bin:$PATH; paraprof app.ppk &

Using TAU in E4S Containers. Download from https://e4s.io

Private https://e4s.io

E4S: An HPC-AI Software Ecosystem for Science

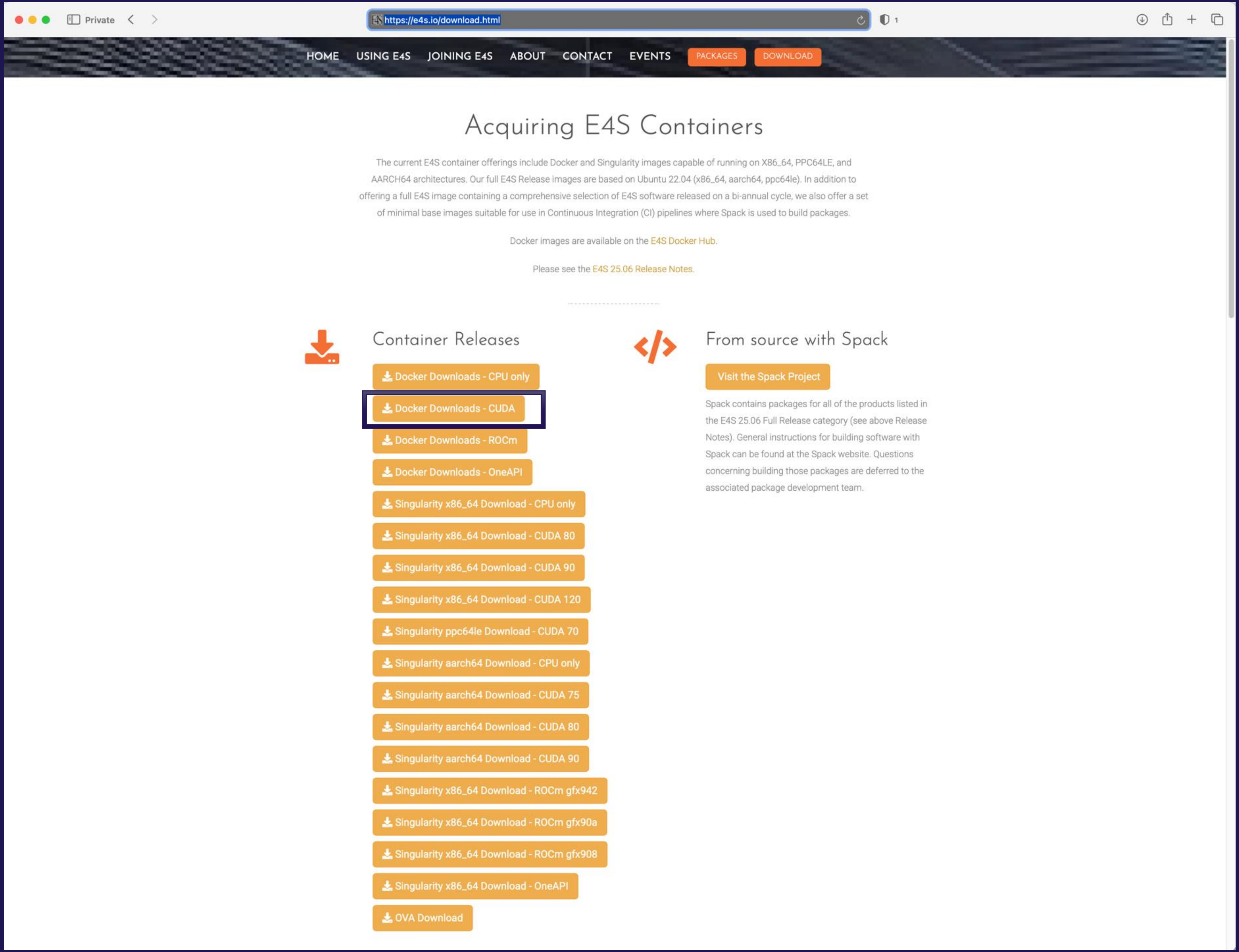
HOME USING E4S JOINING E4S ABOUT CONTACT EVENTS PACKAGES DOWNLOAD

E4S is a community effort to provide and support an open-source software ecosystem for science. E4S provides a **curated collection of scientific libraries and tools (packages) that form the foundation for hundreds of the world's most advanced scientific applications.**

E4S packages support developing, deploying and running scientific applications on high-performance computing (HPC) and AI platforms sponsored by the **US Department of Energy (DOE) Office of Advanced Scientific Computing Research**. E4S is also used as a foundation for applications on leadership-class computing systems at the **US Department of Defense, US National Science Foundation, and other federal agencies.** It is used on numerous high-performance computing systems at universities and at collaborating international organizations.

E4S provides from-source builds, containers, and pre-installed versions of a broad collection of HPC and AI software packages (E4S 25.06 release announcement). E4S includes contributions from many organizations, including national laboratories, universities, and industry. E4S is one of the key legacies of the US Exascale Computing Project (ECP), a collaborative effort of the US Department of Energy Office of Advanced Scientific Computing Research and the National Nuclear Security Administration.

E4S Container Download from https://e4s.io



- Separate full featured Singularity images for 3 GPU architectures
- GPU full featured images for
  - x86_64 (Intel, AMD, NVIDIA)
  - ppc64le (NVIDIA)
  - aarch64 (NVIDIA)
- Full featured images available on Dockerhub
- 130+ products on 3 architectures

Download E4S 25.06 GPU Container Images: AMD, Intel, and NVIDIA

https://e4s.io/download.html

Note on Container Images

Container images contain binary versions of the Full Release packages listed above. Full-featured GPU-enabled container images are available from Dockerhub:

```
# docker pull ecpe4s/e4s-cuda:25.06
# docker pull ecpe4s/e4s-rocm:25.06
# docker pull ecpe4s/e4s-oneapi:25.06
# docker pull ecpe4s/e4s-cpu:25.06
```

E4S Full GPU Images

These images contain a full Spack-based deployment of E4S, including GPU-enabled packages for NVIDIA, AMD, or Intel GPUs.

These images also contain TensorFlow, PyTorch, and TAU.

AMD ROCm (x86_64)

- ecpe4s/e4s-rocm:25.06 docker
- e4s-rocm942-x86_64-25.06.sif mirror 1
- e4s-rocm90a-x86_64-25.06.sif mirror 1
- e4s-rocm908-x86_64-25.06.sif mirror 1

NVIDIA CUDA (X86_64, PPC64LE, AARCH64)

- ecpe4s/e4s-cuda:25.06 docker
- e4s-cuda80-x86_64-25.06.sif mirror 1
- e4s-cuda90-x86_64-25.06.sif mirror 1
- e4s-cuda70-ppc64le-25.06.sif mirror 1
- e4s-cuda75-aarch64-25.06.sif mirror 1
- e4s-cuda80-aarch64-25.06.sif mirror 1
- e4s-cuda90-aarch64-25.06.sif mirror 1

Intel OneAPI (x86_64)

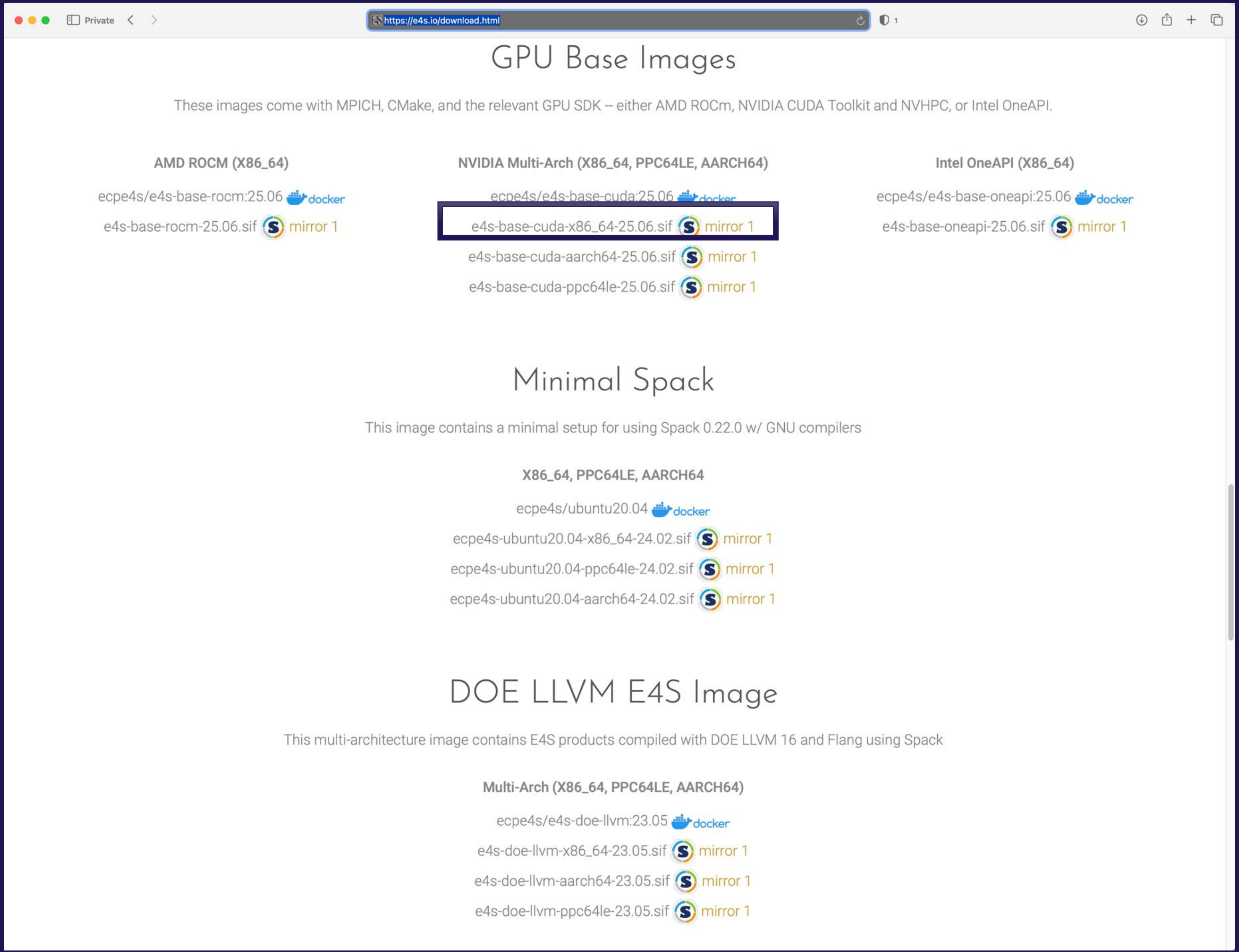
- ecpe4s/e4s-oneapi:25.06 docker
- e4s-oneapi-x86_64-25.06.sif mirror 1

CPU-only (x86_64, aarch64)

- ecpe4s/e4s-cpu:25.06 docker
- e4s-cpu-x86_64-25.06.sif mirror 1
- e4s-cpu-aarch64-25.06.sif mirror 1

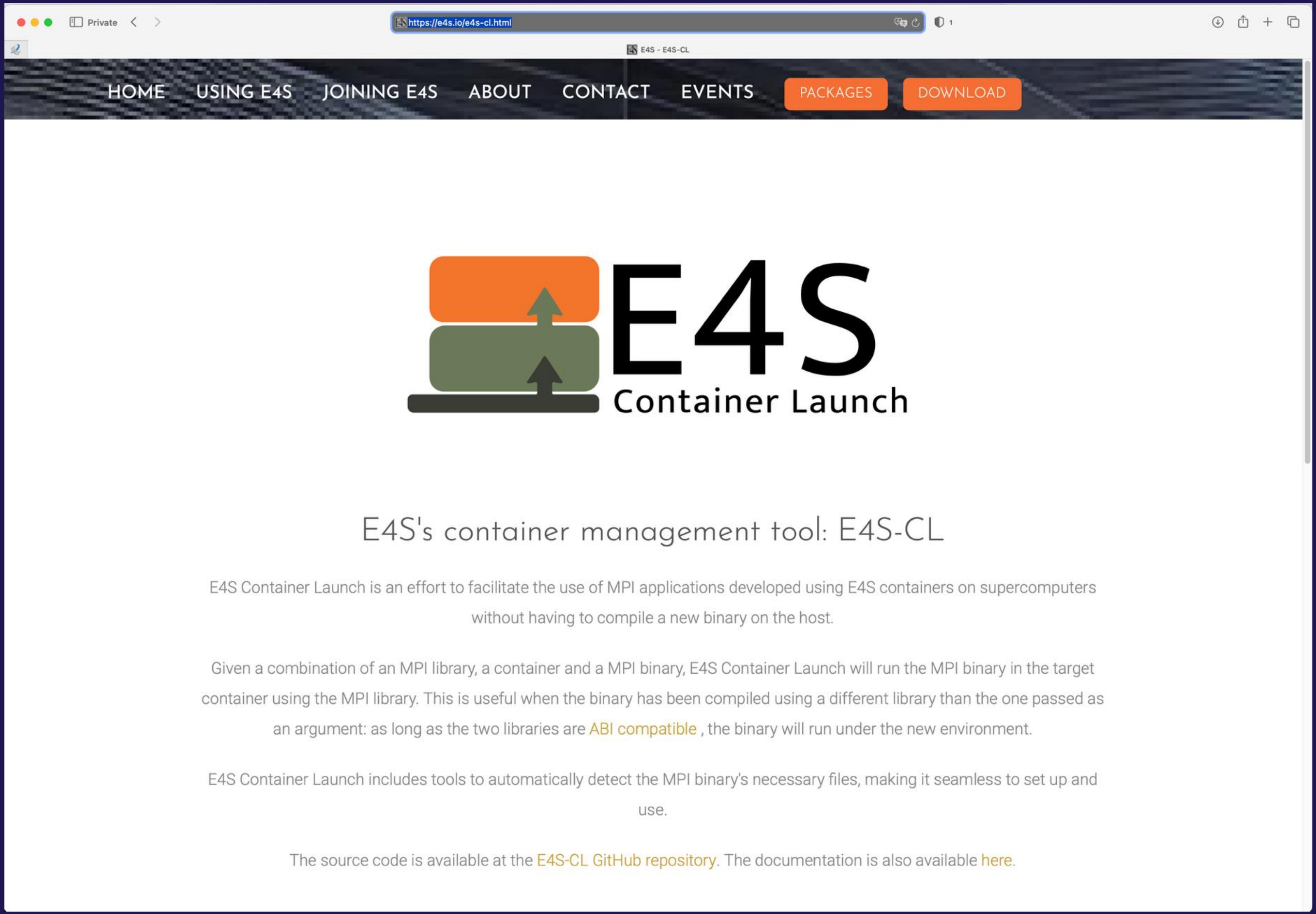
https://e4s.io

E4S base container images allow users to customize their containers



- Intel oneAPI
- AMD ROCm
- NVIDIA CUDA

E4S Tools: e4s-cl: Container Launch tool for MPI applications



- Distribute your MPI application as a binary with an E4S image
- While deploying on a system substitute the embedded containerized MPI in applicatio with the system/vendor MPI
- Use inter-node network interfaces efficiently for near native performance!

https://e4s.io/e4s-cl.html

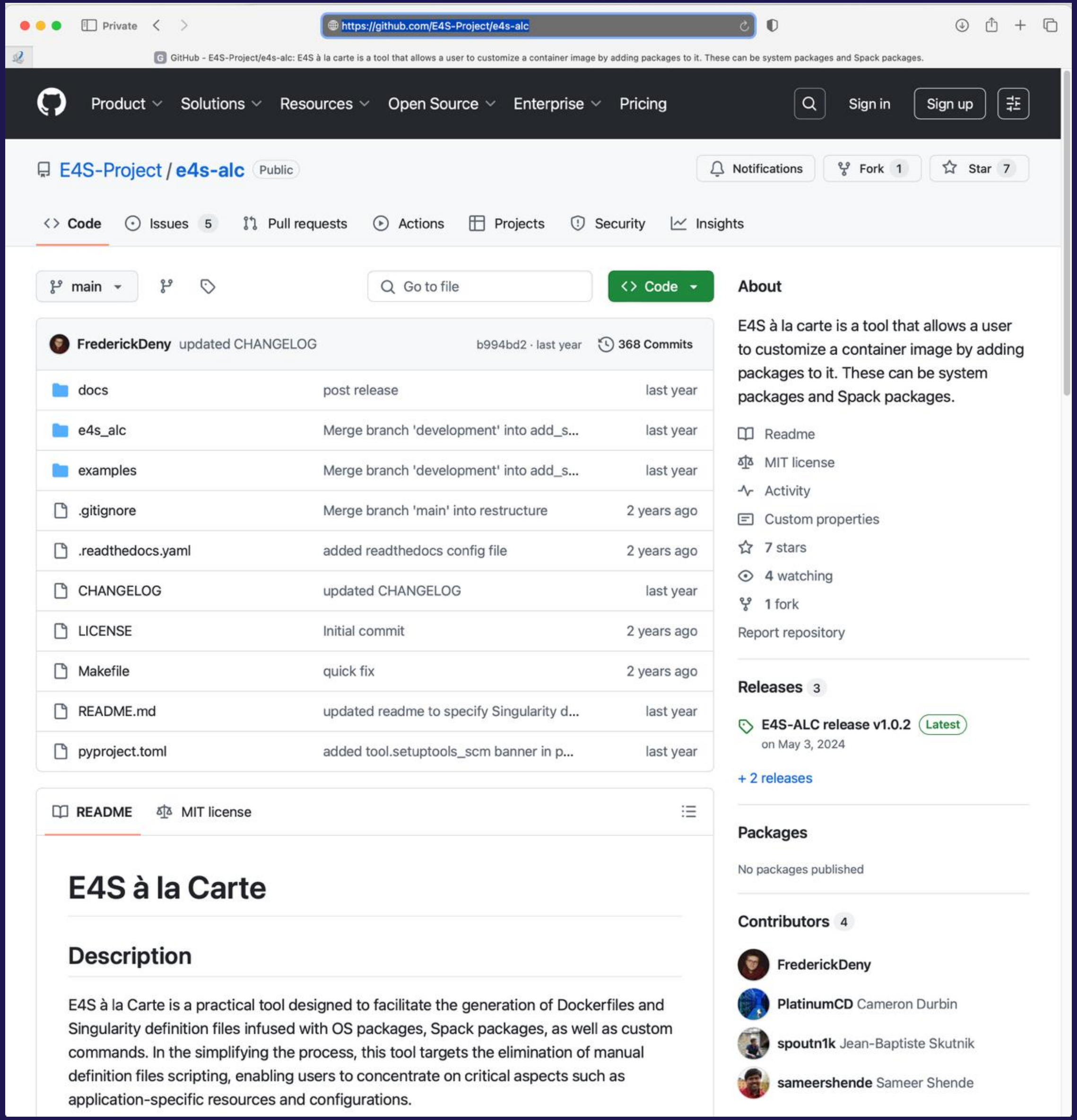
e4s-cl: A tool to simplify the launch of MPI jobs in E4S containers

- E4S containers support replacement of MPI libraries using MPICH ABI compatibility layer and Wi4MPI [CEA] for OpenMPI replacement.
- Applications binaries built using E4S can be launched with Singularity using MPI library substitution for efficient inter-node communications.
- e4s-cl is a new tool that simplifies the launch and MPI replacement.
  - e4s-cl init --backend [singularity|shifter|docker] --image <file> --source <startup_cmds.sh>
  - e4s-cl mpirun -np <N> <command>
- Usage:

```
% e4s-cl init --backend singularity --image ~/images/e4s-gpu-x86.sif --source ~/source.sh
% cat ~/source.sh
   . /spack/share/spack/setup-env.sh
   spack load trilinos+cuda cuda_arch=90
% e4s-cl mpirun -np 4 ./a.out
```

https://github.com/E4S-Project/e4s-cl

E4S Tools: E4S à la carte or e4s-alc: Customize container images



- Add new system packages
- Add new Spack packages
- Add new tarballs
- Customize the container image
- Start with a base image
- Add packages
- Create a new container image!

https://github.com/E4s-Project/e4s-alc

Spack

- E4S uses the Spack package manager for software delivery
- Spack provides the ability to specify versions of software packages that are and are not interoperable.
- Spack is a build layer for not only E4S software, but also a large collection of software tools and libraries outside of ECP ST.
- Spack supports achieving and maintaining interoperability between ST software packages.
- https://spack.io

Spack is a flexible package manager for HPC

- How to install Spack (works out of the box):

```
$ git clone https://github.com/spack/spack
$ . spack/share/spack/setup-env.sh
```

- How to install a package:

```
$ spack install tau
```

- TAU and its dependencies are installed within the Spack directory.

- Unlike typical package managers, Spack can also install many variants of the same build.
  — Different compilers
  — Different MPI implementations
  — Different build options

Visit spack.io

github.com/spack/spack

HPSF
HIGH PERFORMANCE SOFTWARE FOUNDATION

Spack provides the *spec* syntax to describe custom configurations

```
$ git clone https://github.com/spack/spack
$ . spack/share/spack/setup-env.sh
$ spack compiler find                                   # set up compilers
$ spack external find                                   # set up external packages
```

```
$ spack install tau                                                  unconstrained
$ spack install tau@2.34.1                                           @ custom version
$ spack install tau@2.34.1 %gcc@12.4.0                               % custom compiler
$ spack install tau@2.34.1 %gcc@12.4.0 +rocm                         +/- build option
$ spack install tau@2.34.1 %gcc@12.4.0 +mpi ^mvapich2@4.0            ^ dependency information
```

- Each expression is a ***spec*** for a particular configuration
  - Each clause adds a constraint to the spec
  - Constraints are optional – specify only what you need.
  - Customize install on the command line!
- Spec syntax is recursive
  - Full control over the combinatorial build space

The Spack community is growing rapidly

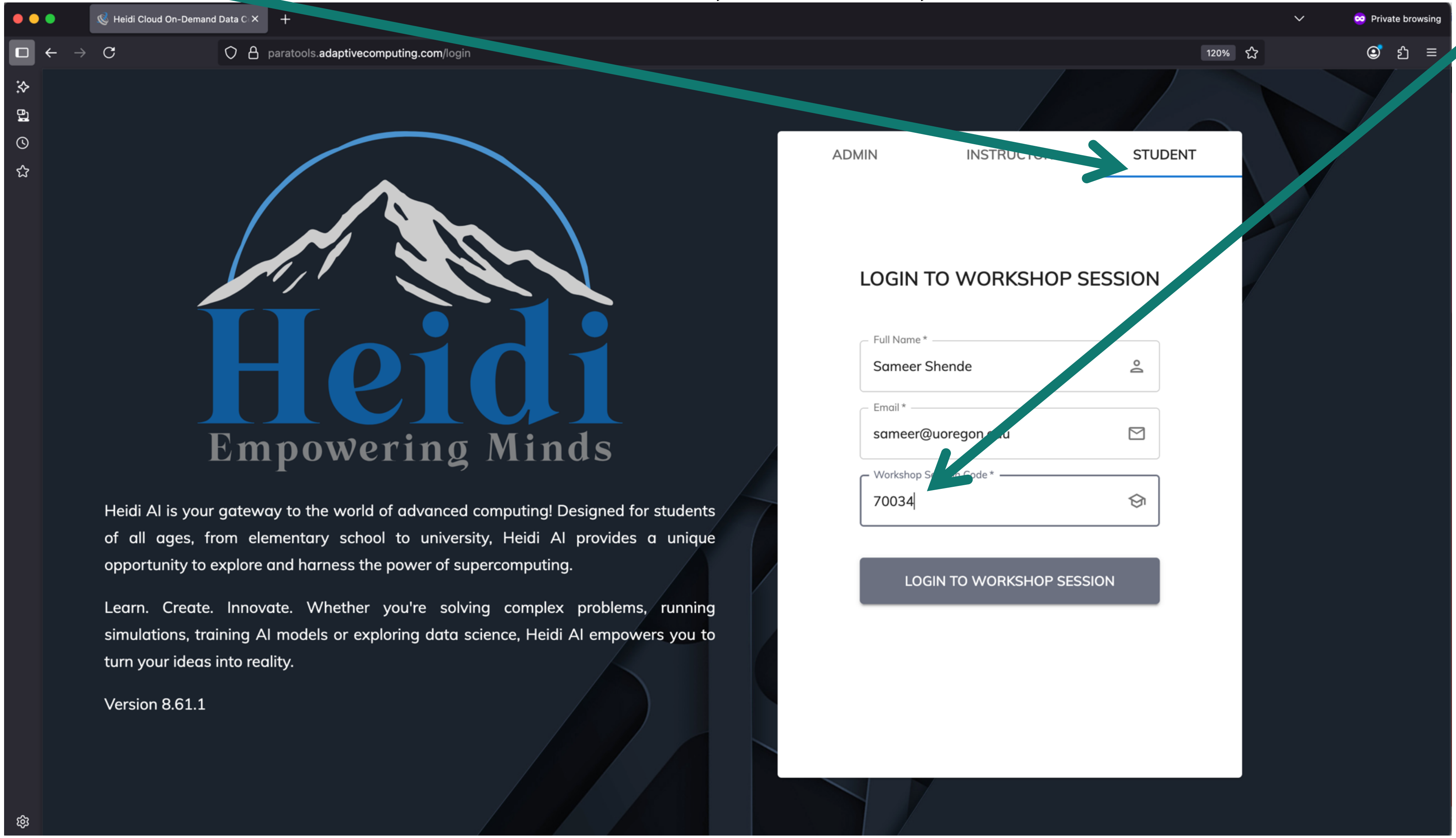
- **Spack simplifies HPC software for:**
  - Users
  - Developers
  - Cluster installations
  - The largest HPC facilities
- **Spack is central to HPSF's software strategy**
  - Enable software reuse for developers and users
  - Allow the facilities to consume the entire E4S
- **The roadmap is packed with new features:**
  - Building the software distribution
  - Better workflows for building containers
  - Stacks for facilities
  - Chains for rapid dev workflow
  - Optimized binaries
  - Better dependency resolution

Visit spack.io
hpsf.io

github.com/spack/spack

Using TAU on AWS. Connect to https://paratools.adaptivecomputing.com

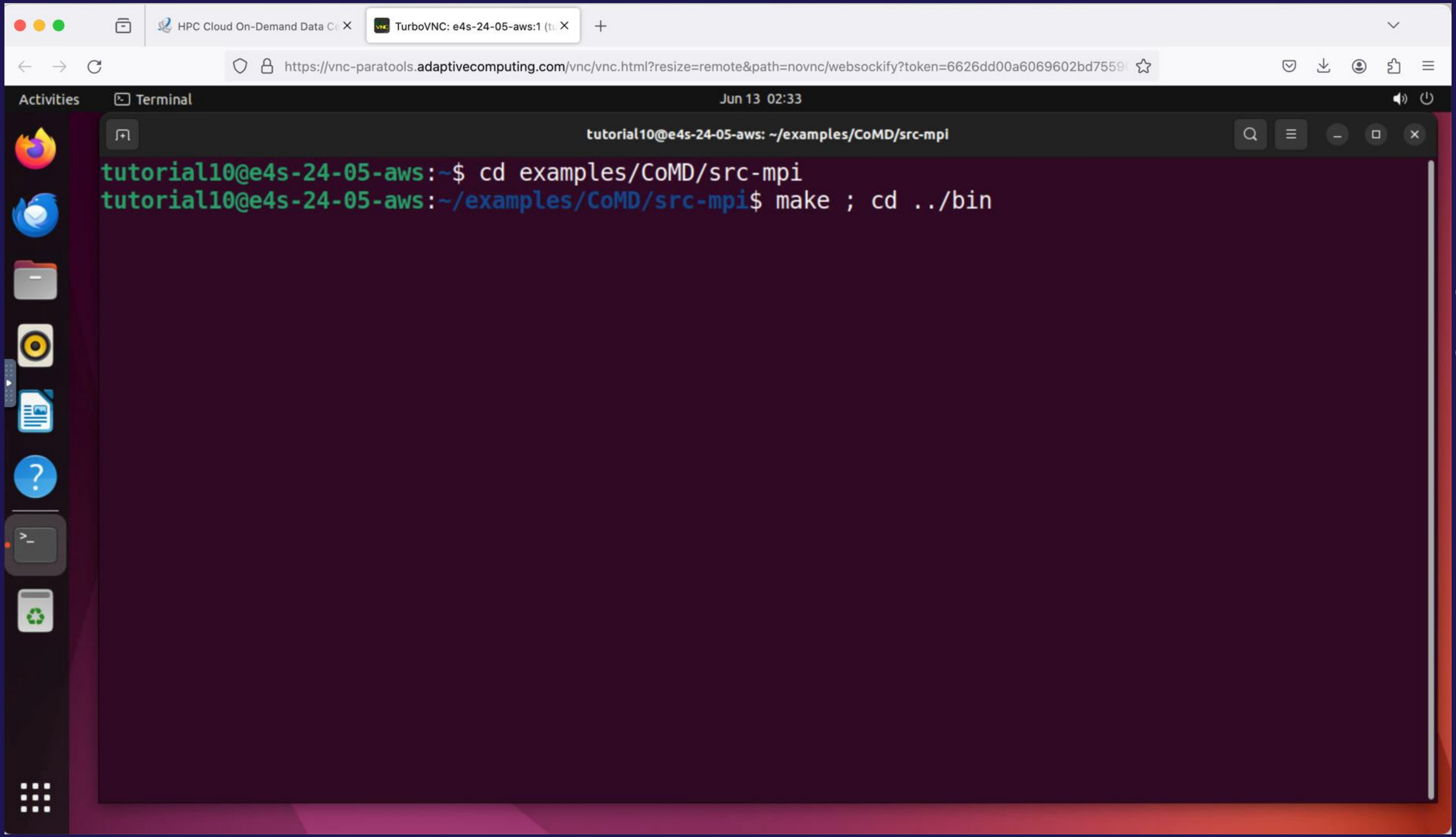
- Use Student tab and enter name, email, session code 70034



Running your first MPI application on the allocated cluster

```
% cd ~/examples/mpi-procname
% ./compile.sh
% ./run-single-node.sh      # on the login node
% cat mpiprocname.qsub
% qsub mpiprocname.qsub
% qstat –u $USER
% cat mpiprocname.o*

% cd ~/examples/osu-benchmarks
% cat bw.qsub
% qsub bw.qsub
% cat bw.o*                 # How close did you get to 50Gbps? At what message size? Multiply MB/s x 8 …
```

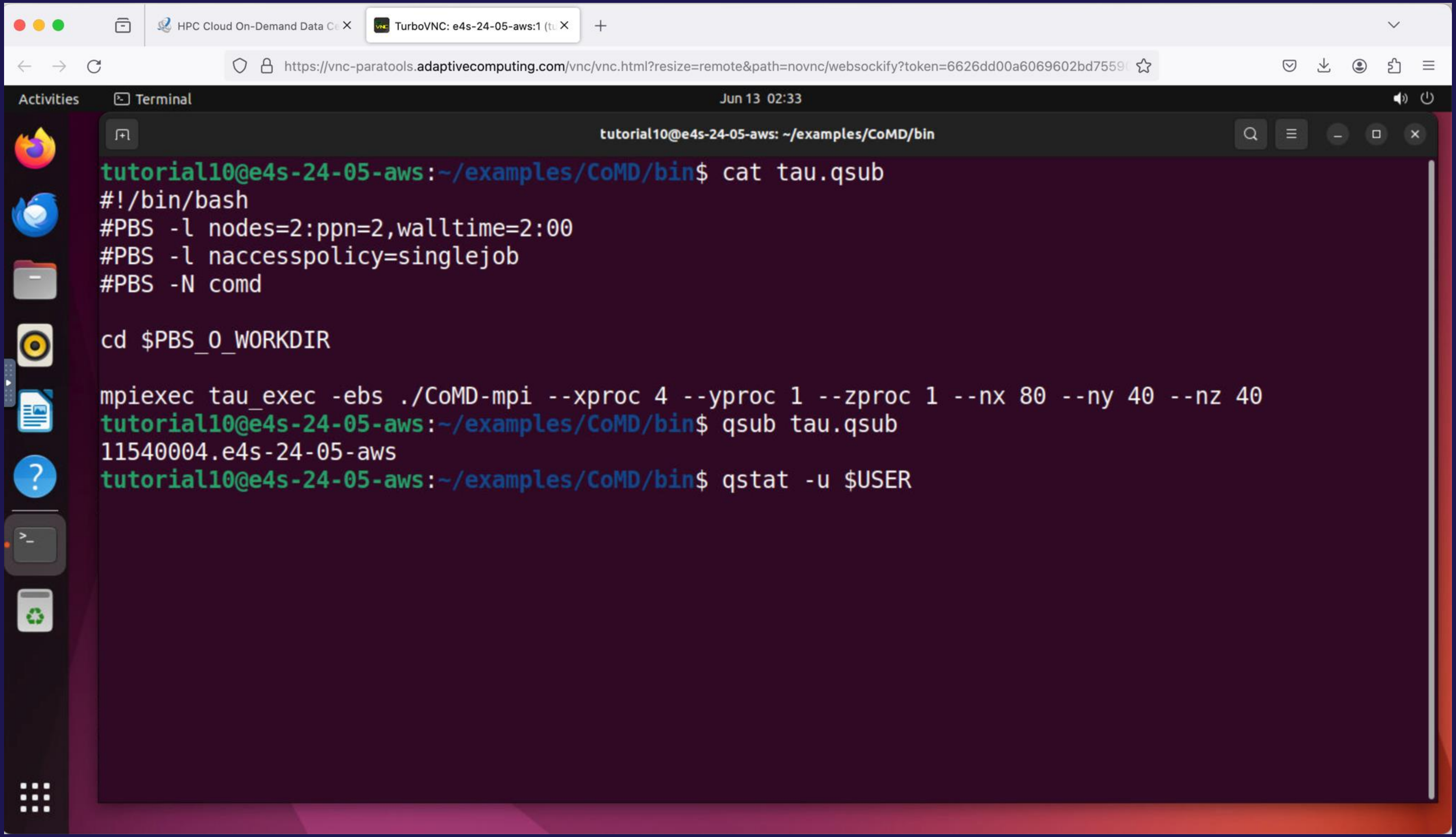
CoMD: TAU with event-based sampling (EBS)



% cd examples/CoMD/src-mpi

% make; cd ../bin

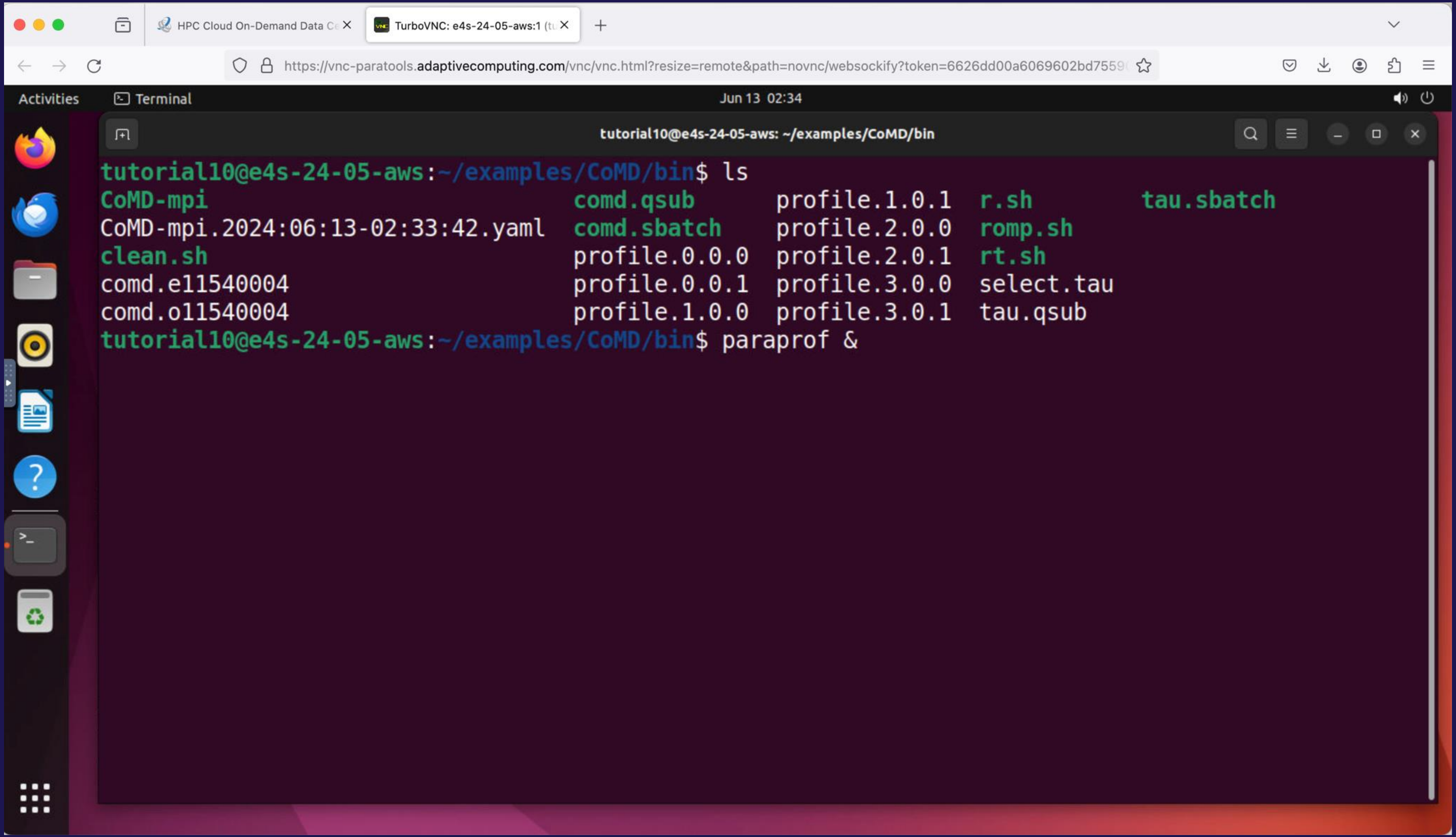
CoMD: TAU with event-based sampling (EBS)



% qsub tau.qsub

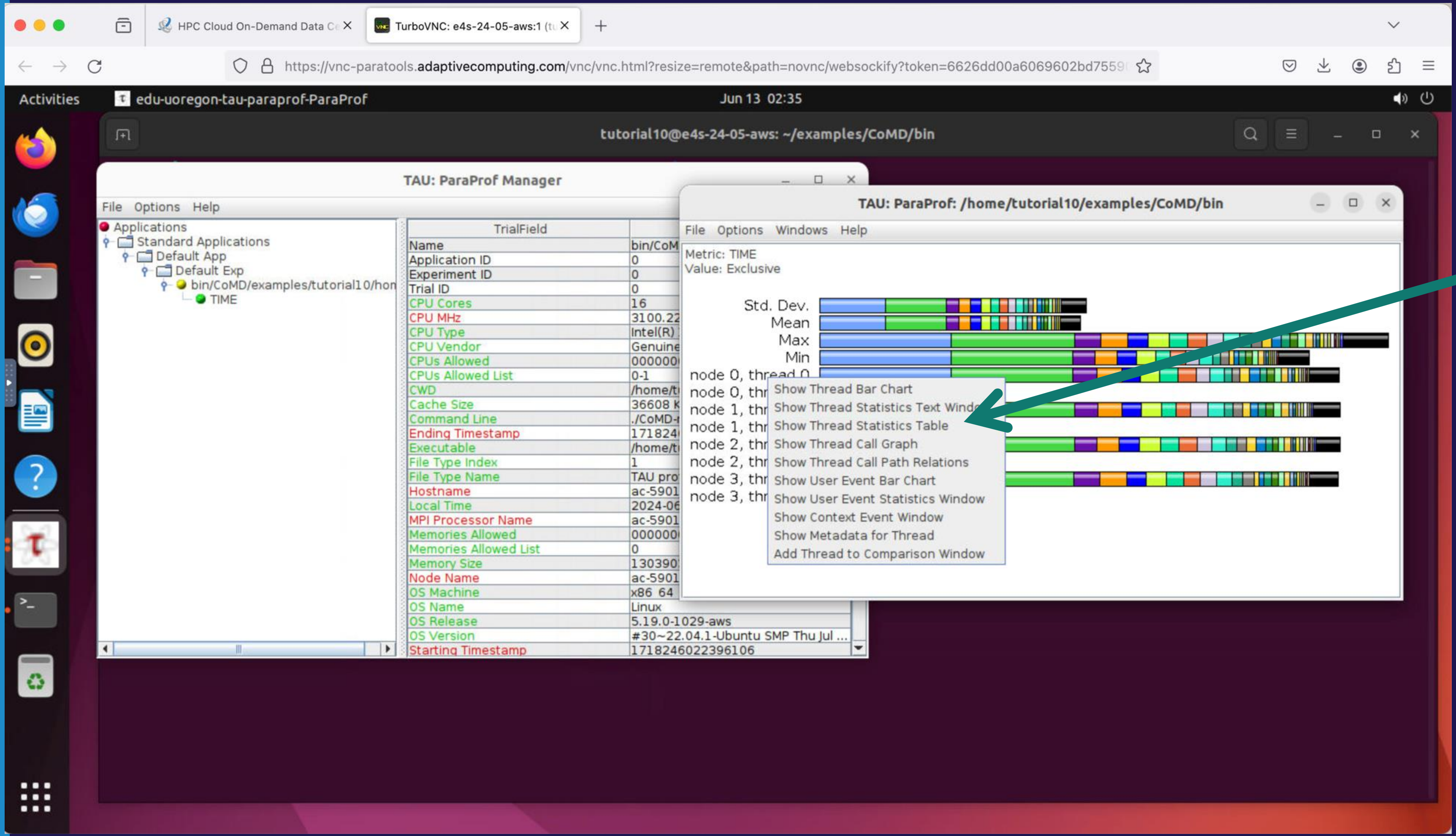
% qstat –u $USER

CoMD: TAU's paraprof visualizer



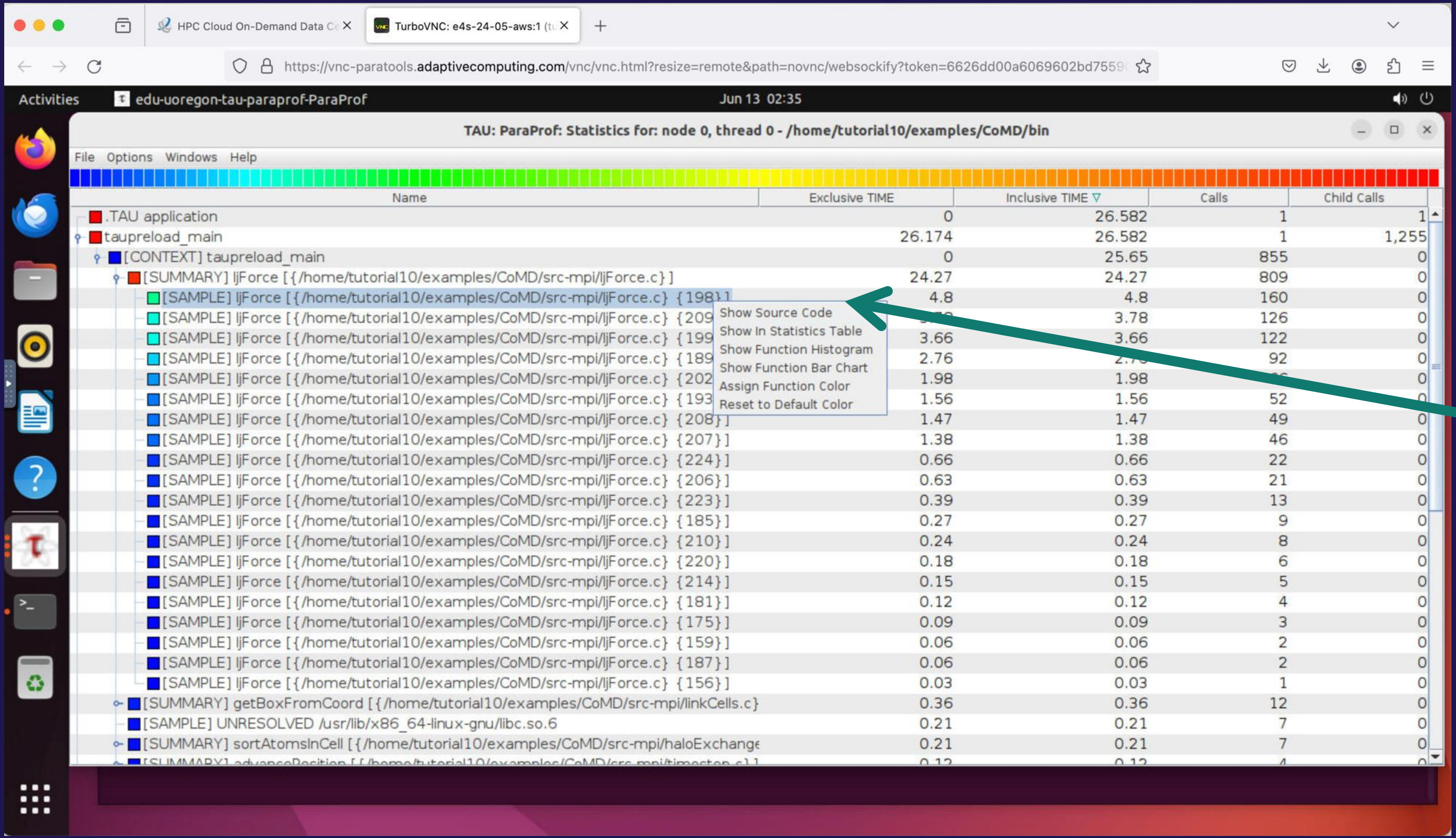
% paraprof &

CoMD: TAU's paraprof visualizer



Right click on Node 0, Thread 0 and choose Show Thread Statistics Table (third option)

TAU's ParaProf Profile Browser: Thread Statistics Table



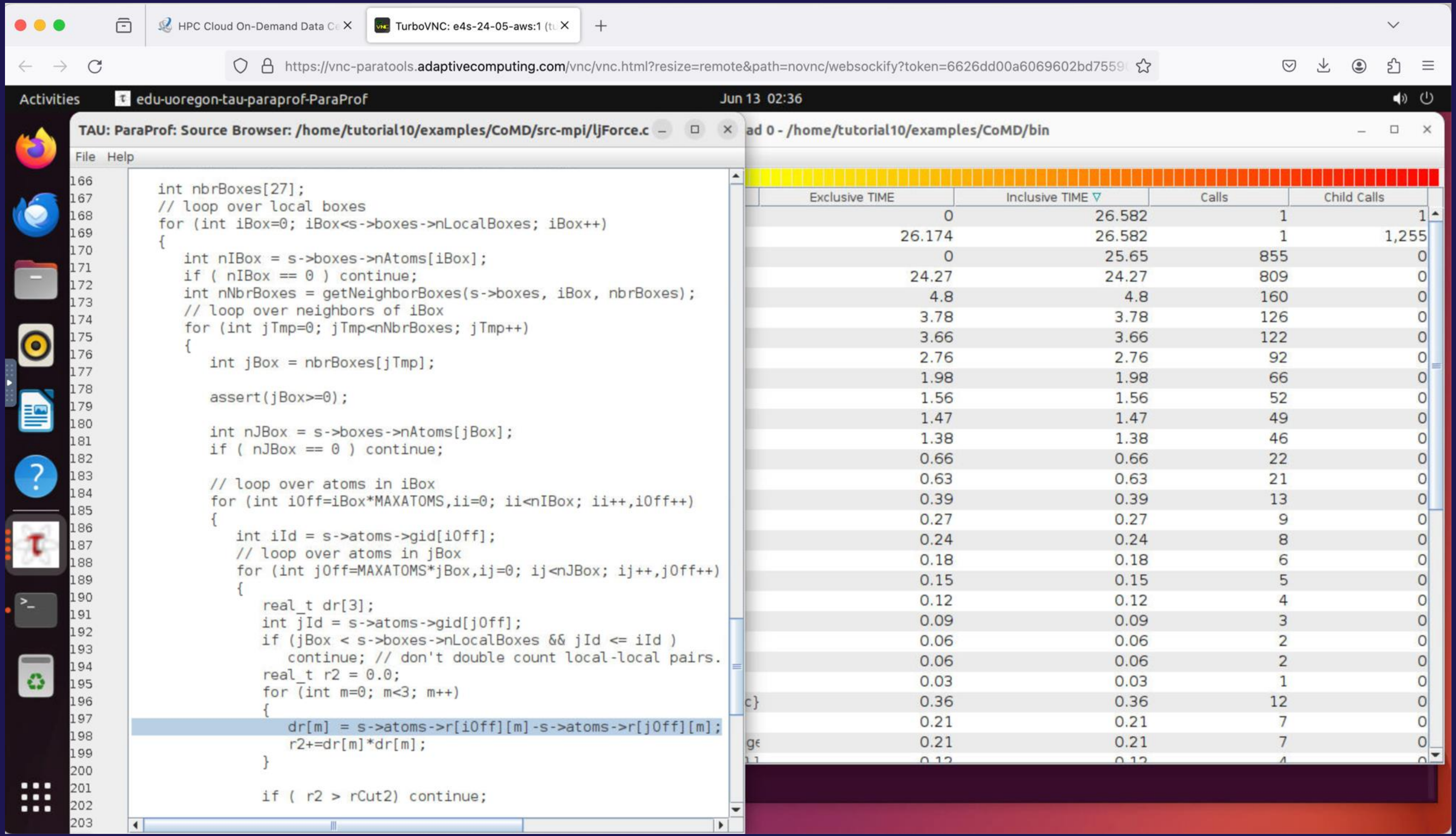
Click on columns to sort (e.g., Inclusive)

Expand nodes and right click on a sample and

Select "Show Source Code"

Argonne NATIONAL LABORATORY

TAU's ParaProf Profile Browser: Source Code Browser



The application spent

4.8 seconds at line 198 in ljForce.c in MPI rank 0. TAU collected 160 samples at this line of code.

It is within five levels of for loops!

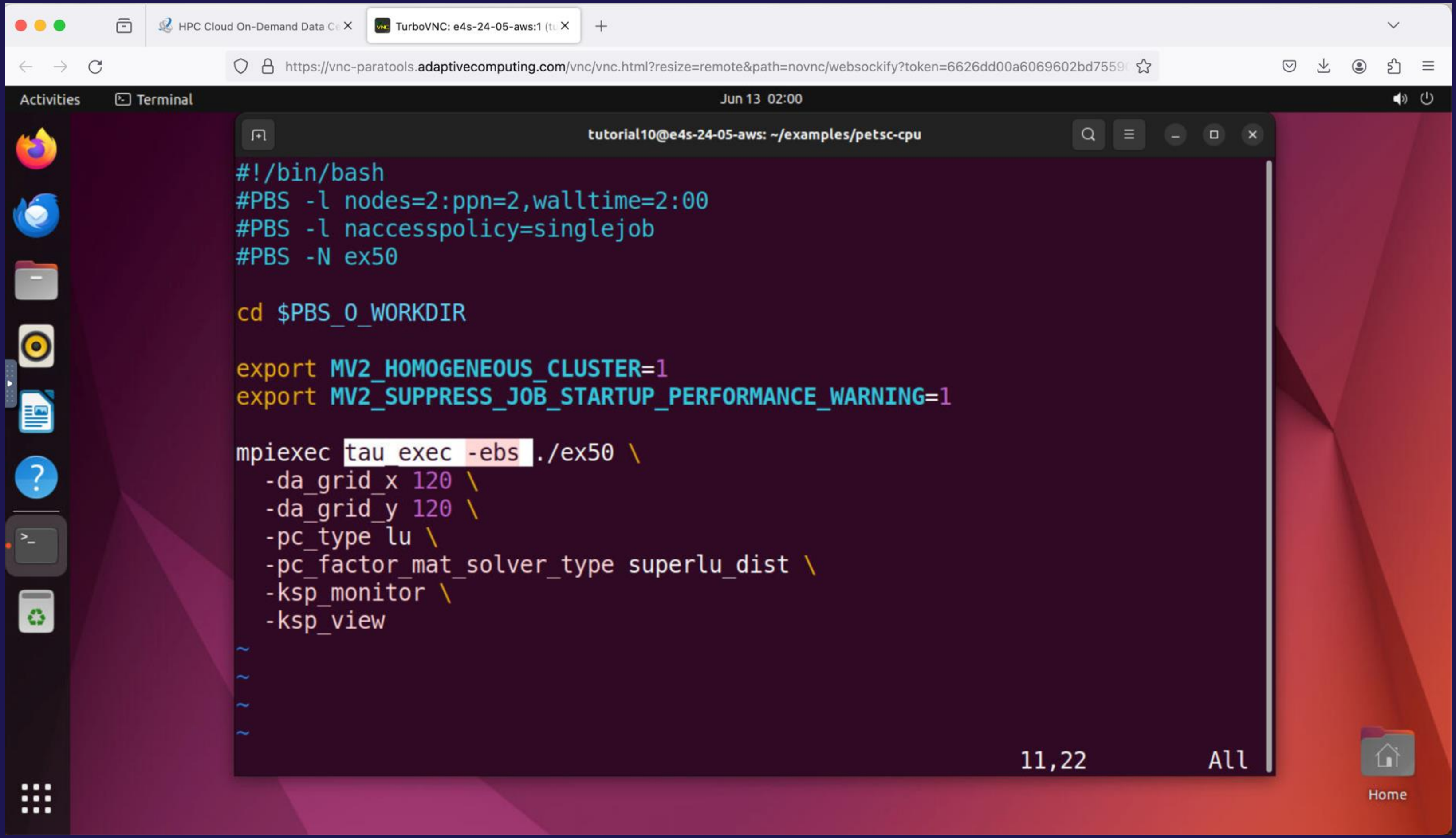
There was no change to source code,
build system, or the application binary!

Argonne NATIONAL LABORATORY

TAU Exercise #2:

Instrumenting PETSc application using TAU's Perfstubs interface

Launching the binary using tau_exec –ebs



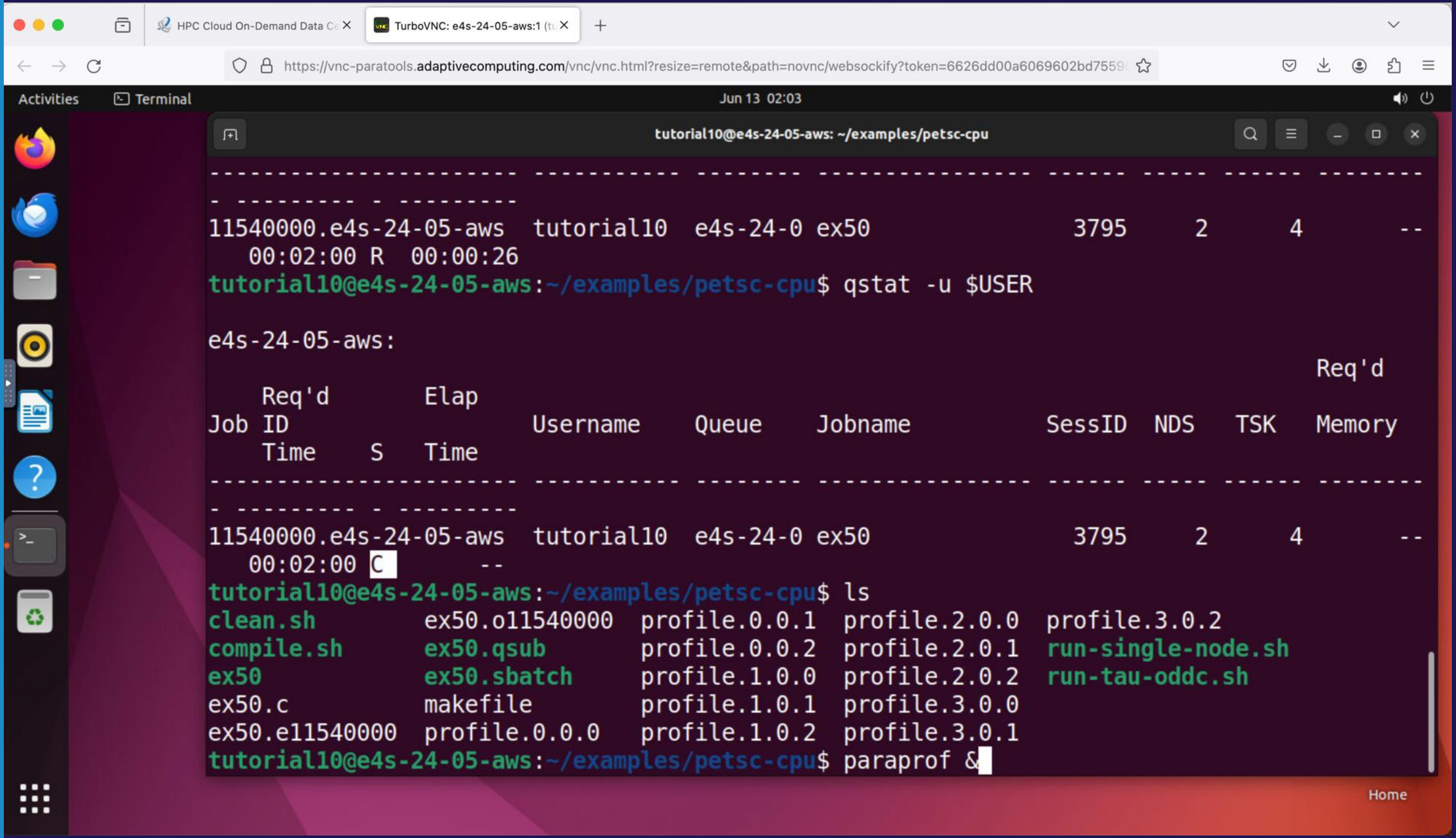
cd ~/examples/petsc-cpu
vi ex50.qsub

Add tau_exec –ebs before ./ex50 in the launch command. Save the file.

OR

qsub tau.qsub

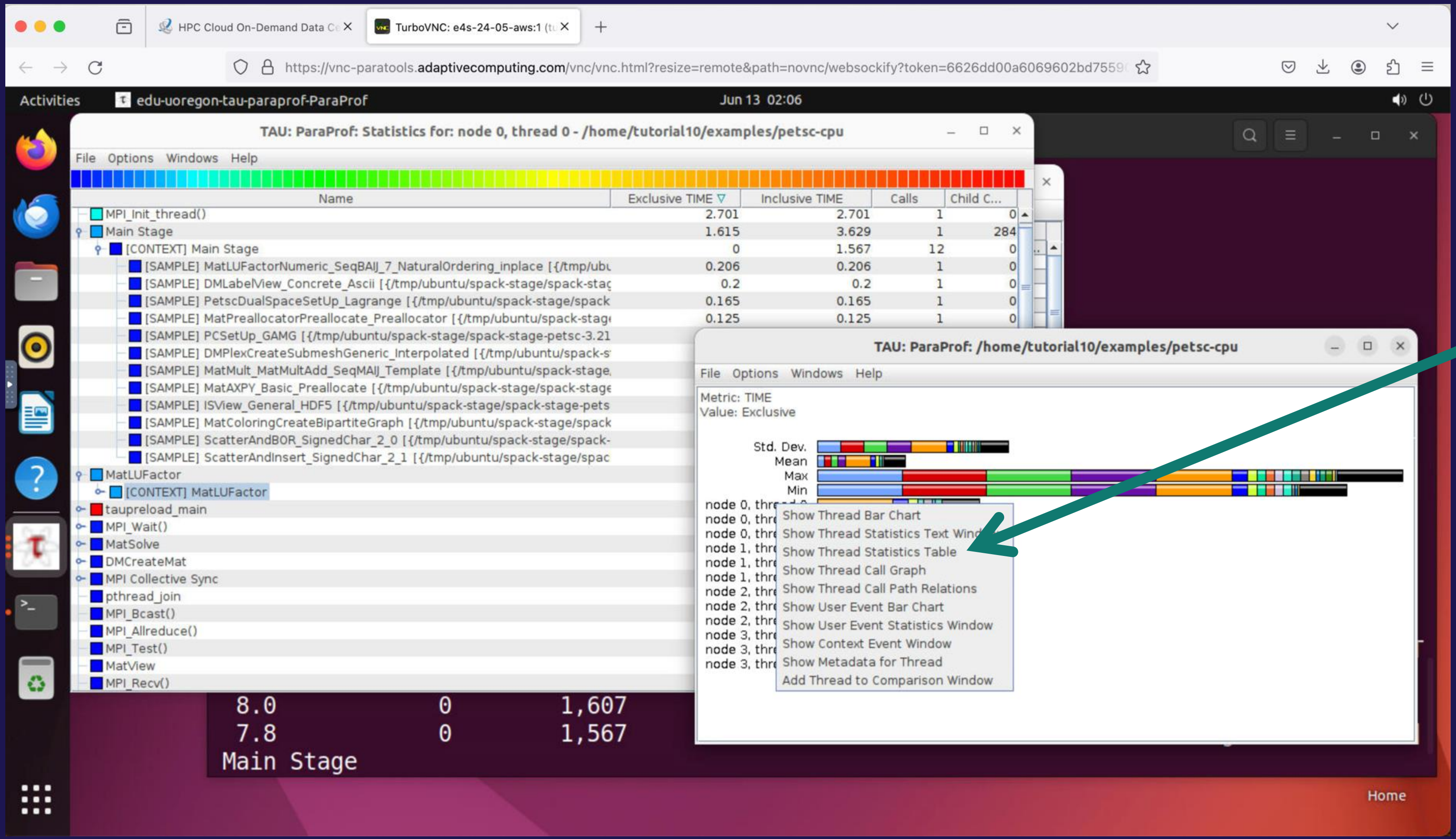
TAU's ParaProf Profile Browser: Source Code Browser



qsub ex50.qsub
qstat –u $USER

After it completes
ls
paraprof &

TAU's paraprof browser with PETSc performance profile

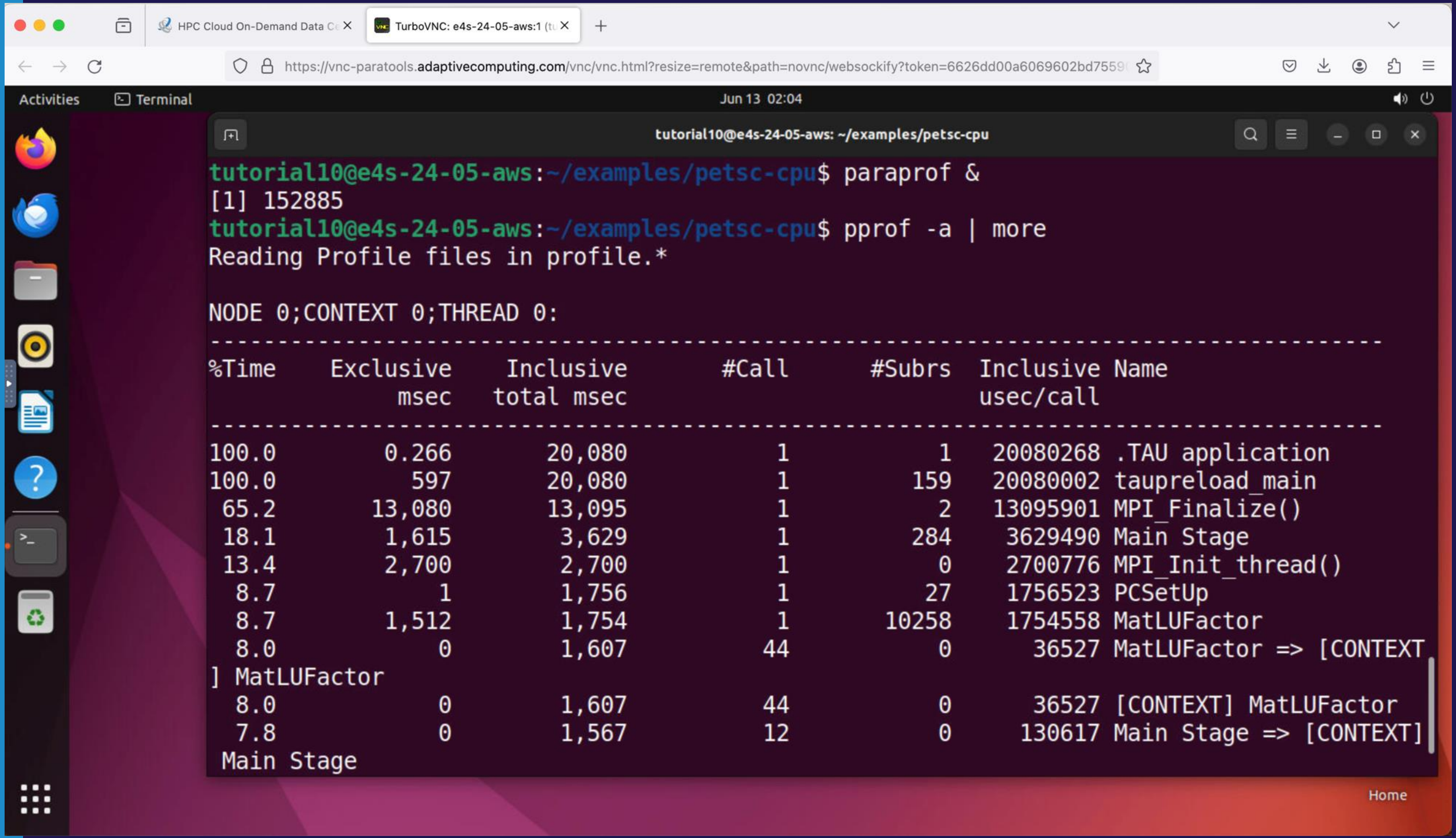


paraprof

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

Argonne NATIONAL LABORATORY

Using pprof: TAU's text based profile browser



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!

Argonne NATIONAL LABORATORY

TAU Exercise #3:

Instrumenting PETSc application using TAU's Perfstubs interface and generating callpath profiles

Generating callpath profiles

```
Edit ex50.qsub
# add
export TAU_CALLPATH=1
export TAU_CALLPATH_DEPTH=100

export TAU_PROFILE_FORMAT=merged
mpirun …

% qsub ex50.qsub
% paraprof tauprofile.xml
```

TAU Exercise #4:

Instrumenting PETSc application using TAU's Perfstubs interface and generating traces

Generating Traces

```
# cd ~/examples/petsc-cuda; vi ex50.qsub
# Comment out previous CALLPATH options
export TAU_TRACE=1


% qsub ex50.qsub
% tau_treemerge.pl
% tau_trace2json tau.trc tau.edf -chrome \
 -ignoreatomic -o ex50.json
Open Firefox, load Perfetto.dev
trace visualizer and open ex50.json
wasd keys to widen/shrink/left/right
```


Argonne NATIONAL LABORATORY

PETSc CUDA: What it was doing

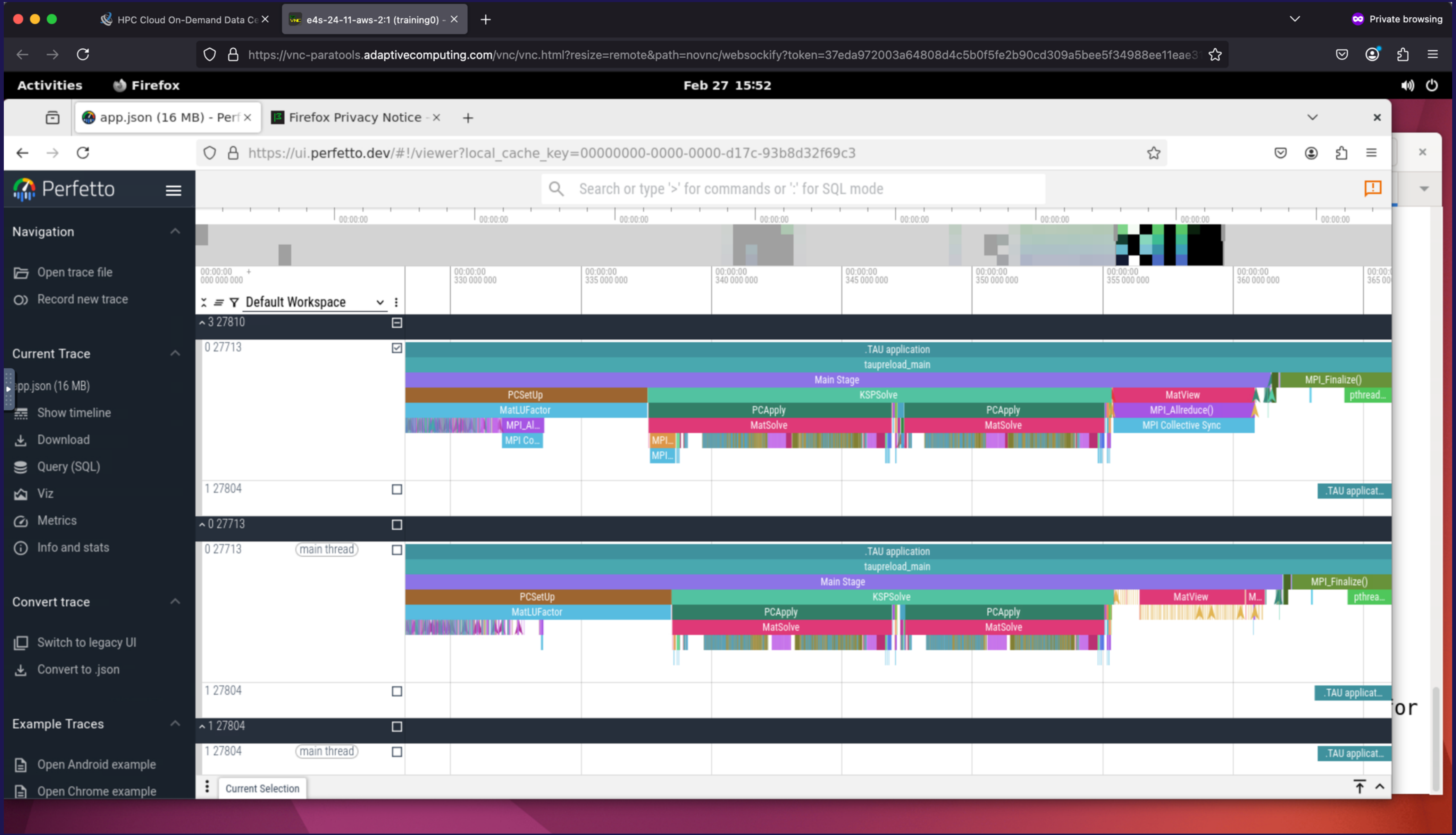
```
% cd ~/examples/petsc-cuda
% ./compile.sh
Edit the qsub file, increase wallclock
time to 5 mins:
spack load tau+mpi+cuda
export TAU_PROFILE_FORMAT=merged
mpiexec tau_exec -T cupti,mpi -cupti -ebs ./ksp…
% qsub *.qsub

% paraprof tauprofile.xml &
```

PETSc CUDA:

```
training0@e4s-25-06-aws-1:~/examples/petsc-cuda$ cat tau.qsub
#!/bin/bash
#PBS -l nodes=2:ppn=2,walltime=5:00
#PBS -l naccesspolicy=singlejob
#PBS -N bench_kspsolve

cd $PBS_O_WORKDIR

export MV2_HOMOGENEOUS_CLUSTER=1
export MV2_SUPPRESS_JOB_STARTUP_PERFORMANCE_WARNING=1

spack load tau+mpi+cuda
mpiexec tau_exec -T cupti,mpi -cupti -ebs ./bench_kspsolve \
  -n 128 \
  -its 1000 \
  -matmult \
  -mat_type aijcusparse \
  -use_gpu_aware_mpi 0
training0@e4s-25-06-aws-1:~/examples/petsc-cuda$
```

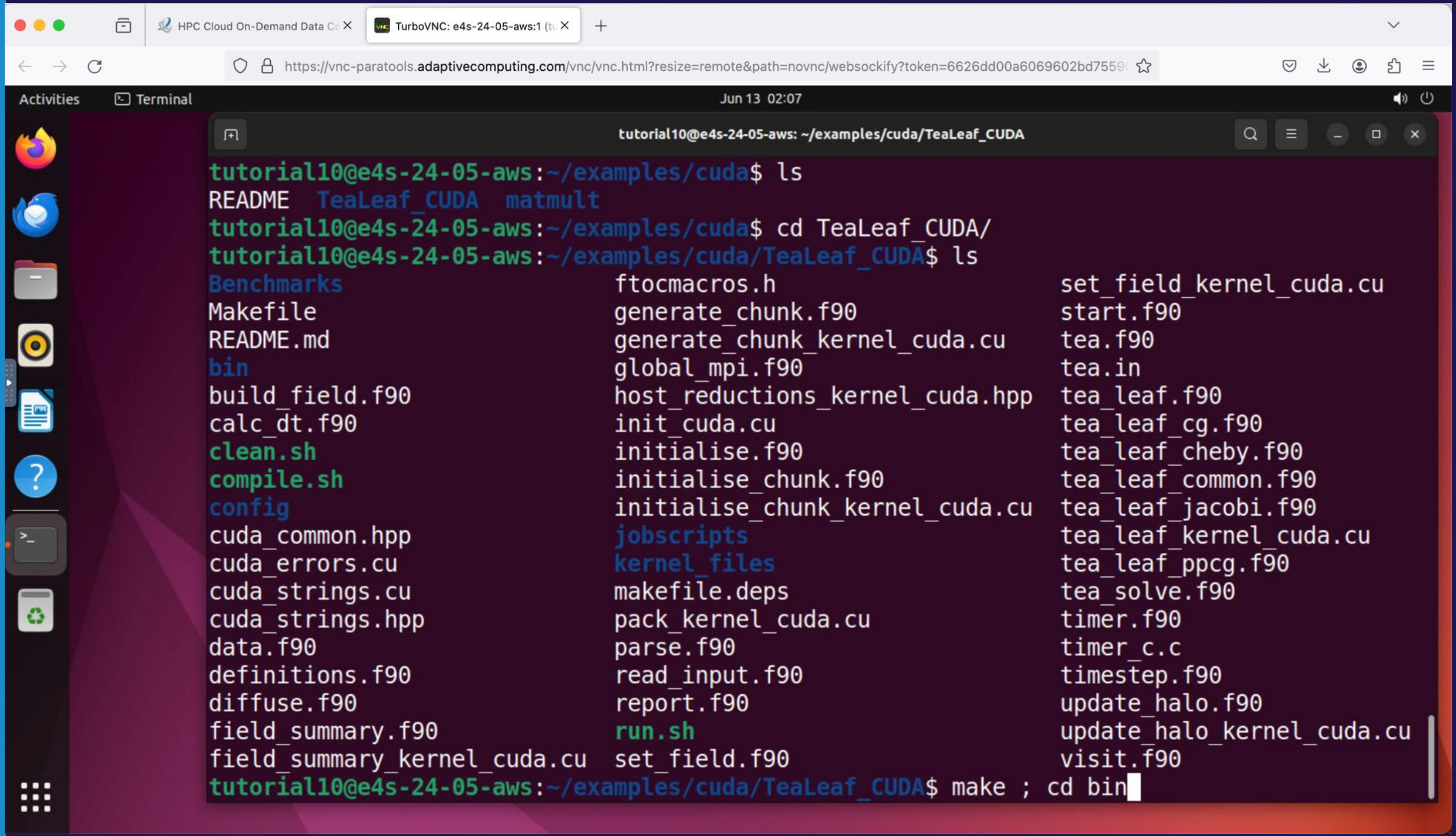
Increase wallclock time to
5 mins

Visualizing Traces with https://Perfetto.dev



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

Compiling TeaLeaf_CUDA application



cd ~/examples/cuda

cd TeaLeaf_CUDA

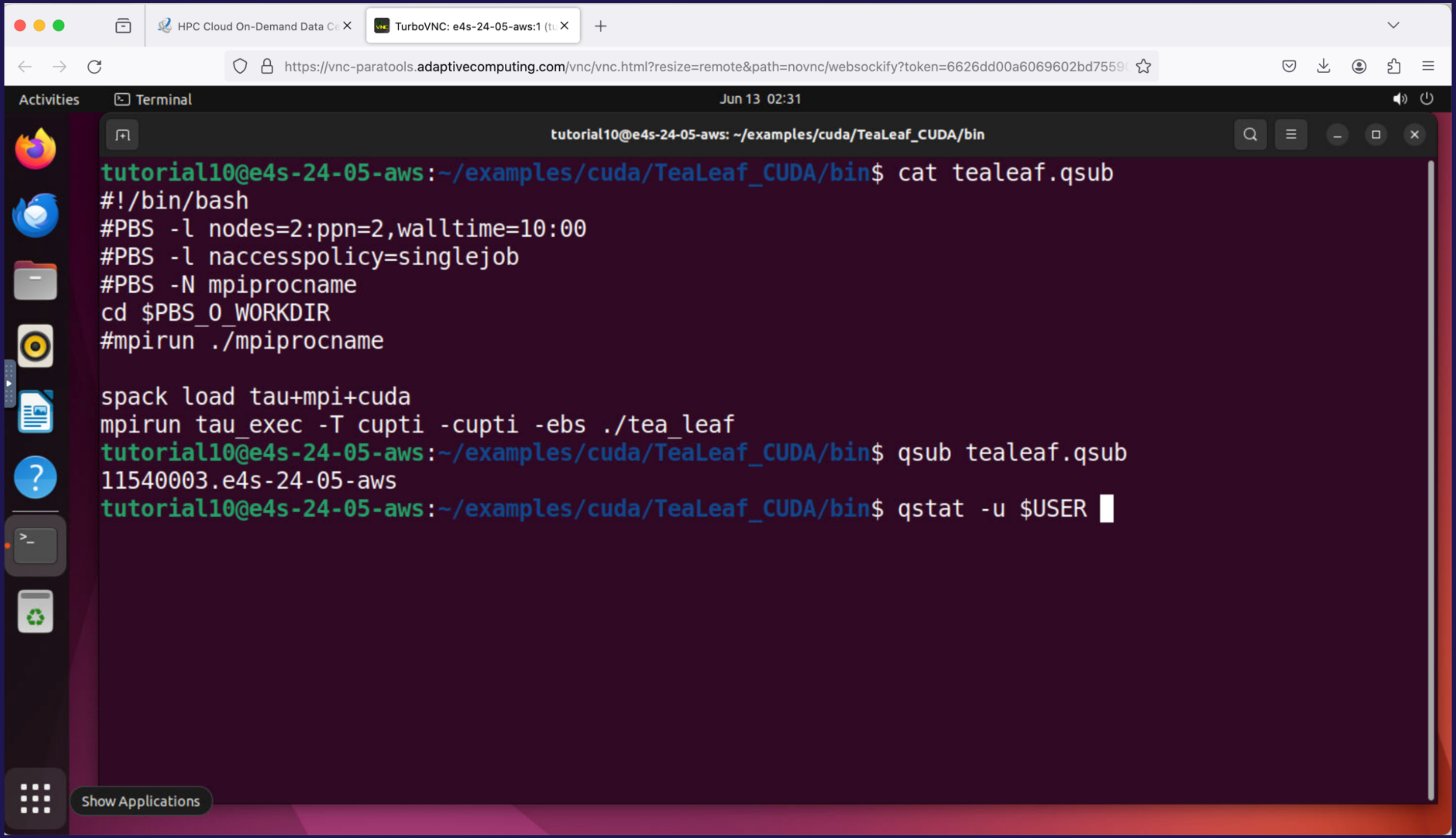
ls

make

cd bin

Argonne NATIONAL LABORATORY

Submit the CUDA job to run on four MPI ranks



qsub tealeaf.qsub

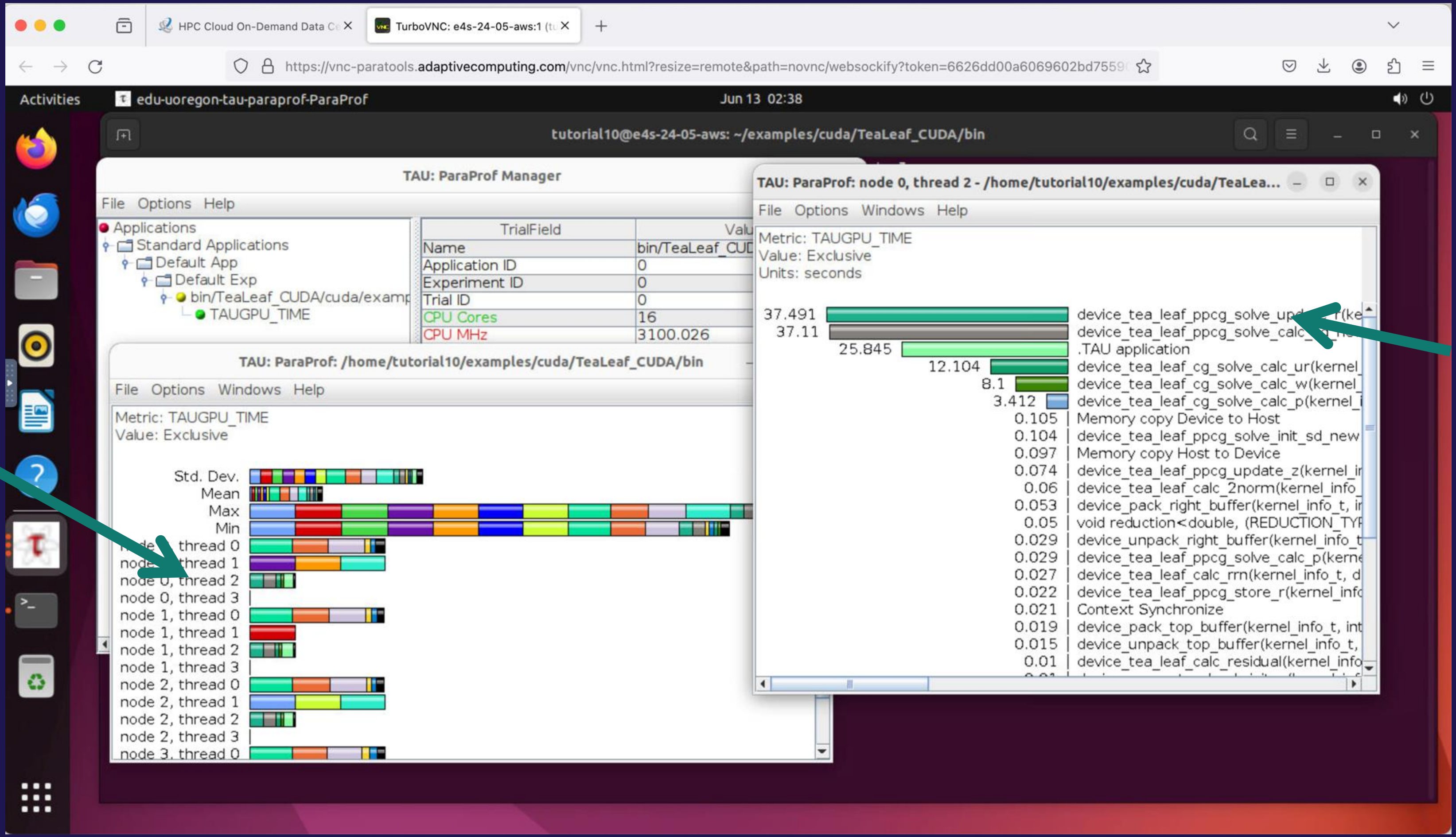
qstat –u $USER

After it runs

paraprof &

TAU's paraprof shows the time spent in individual CUDA kernels

Left click node 0, thread 2



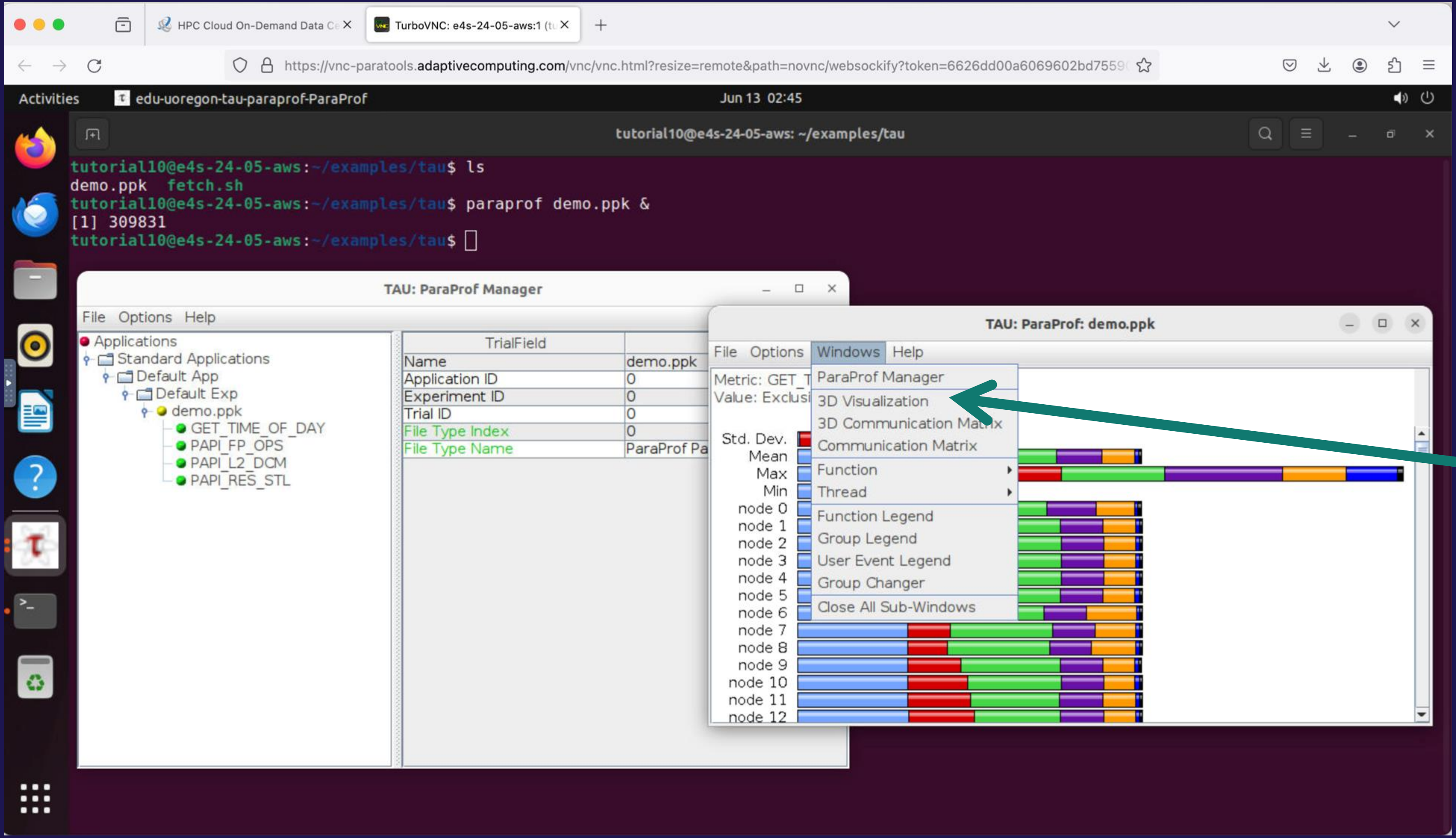
CUDA kernels

Argonne NATIONAL LABORATORY

TAU Exercise #6:

paraprof 3D display

TAU paraprof

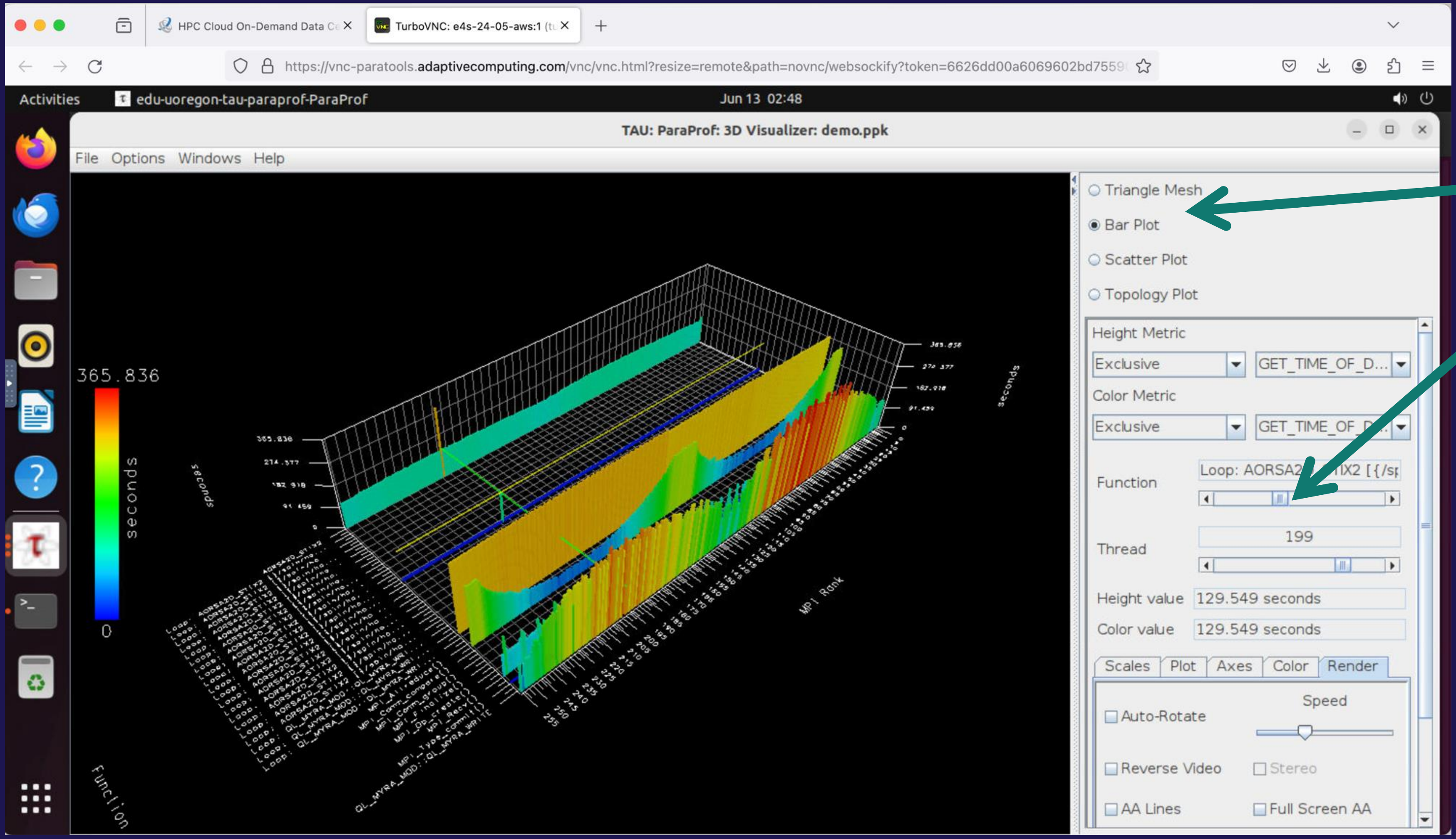


cd ~/examples/tau

paraprof demo.ppk &

Choose 3D Visualization

TAU paraprof 3D visualization



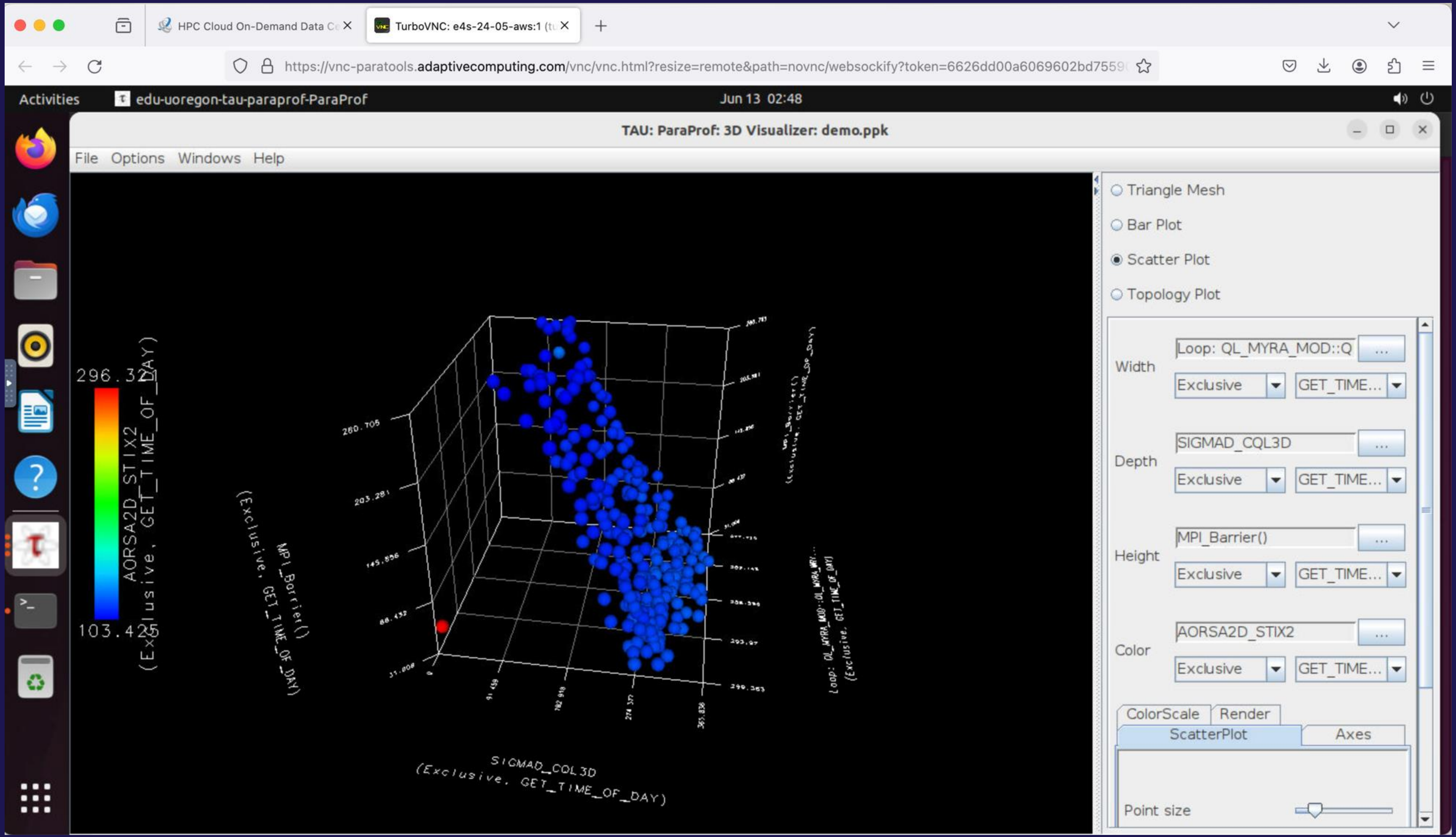
Choose Bar Plot and move

Function and Thread Sliders

First mouse button to rotate
Second mouse button to translate (left to right)
Scroll wheel (or +/- keys) to zoom in.

Try Scatter plot next

TAU paraprof: 3D Scatter Plot



Argonne NATIONAL LABORATORY

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
- CEA, France
- Partners:
  - —University of Oregon
  - —The Ohio State University
  - —ParaTools, Inc.
  - —University of Tennessee, Knoxville
  - —T.U. Dresden, GWT
  - —Jülich Supercomputing Center

Acknowledgment

- *This work was supported by the U.S. Department of Energy, Office of Science, Advanced Computing Research, through the Next-Generation Scientific Software Technologies (NGSST) under contract DE-AC02-AC05-00OR22725 and DOE SBIR DE-SC0022502.*

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

Thank you

https://www.exascaleproject.org

This research was supported by the Exascale Computing Project (17-SC-20-SC), a joint project of the U.S. Department of Energy's Office of Science and National Nuclear Security Administration, responsible for delivering a capable exascale ecosystem, including software, applications, and hardware technology, to support the nation's exascale computing imperative.

EXASCALE COMPUTING PROJECT

Thank you to all collaborators in the ECP and broader computational science communities. The work discussed in this presentation represents creative contributions of many people who are passionately working toward next-generation computational science.