ALCF Hands on HPC Workshop 2024

TAU

http://tau.uoregon.edu

Sameer Shende
Research Professor and Director, Performance Research Laboratory
OACISS, U. Oregon
President and Director, ParaTools, Inc.
Track 1, TCS 1404, 4:00 – 5:15pm, Wednesday, October 30, 2024
sameer@cs.uoregon.edu

https://www.alcf.anl.gov/events/2024-alcf-hands-hpc-workshop

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.
  - http://tau.uoregon.edu
- It is getting harder to install our HPC and AI/ML tools.

Motivation: Improving Productivity

- TAU Performance System®:
  - Deliver a scalable, portable, performance evaluation toolkit for HPC and AI/ML workloads
  - http://tau.uoregon.edu
- Extreme-scale Scientific Software Stack (E4S):
  - Delivering a modular, interoperable, and deployable software stack
  - Deliver expanded and vertically integrated software stacks to achieve full potential of extreme-scale computing
  - Lower barrier to using software technology (ST) products from ECP
  - Enable uniform APIs where possible
  - https://e4s.io

TAU Performance System®

- Versatile profiling and tracing toolkit that supports:
  - MPI, DPC++/SYCL (Level Zero), OpenCL, and OpenMP (OpenMP Tools Interface for Target Offload)
- 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 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
- http://tau.uoregon.edu
- module load tau
  - to load TAU on Sunspot and other ALCF systems

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.
- 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?
- 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?

UNIVERSITY OF OREGON

Using TAU on Polaris at ALCF

```
% qsub -I -l select=1 -l filesystems=home:eagle -l walltime=1:00:00 -q R2035675 -A ATPESC2024 -X
```

```
% module use /soft/modulefiles; module load tau
% wget http://tau.uoregon.edu/workshop_atpesc24.tgz; tar workshop_atpesc24.tgz
% cd workshop; cat README; cd TeaLeaf_CUDA; make; cd bin; ./run.sh; pprof -a
% cd ../../petsc-tau; ./clean.sh; ./compile.sh; ./run.sh
% paraprof --pack petsc_ex19.ppk ;  <SCP to AWS>; paraprof petsc_ex19.ppk
```

- **Un-instrumented run with MPI** **% aprun –n N ./a.out**
- **Profiling an un-instrumented application (use tau_exec –ebs with any of the following for event-based sampling):**
- `MPI without GPUs:` `% aprun -n N tau_exec -ebs ./a.out`
- `CUDA with MPI:` `% aprun -n N tau_exec –T cupti,mpi –cupti –ebs ./a.out`

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

Tracing:

- `Vampir:` `% export TAU_TRACE=1; export TAU_TRACE_FORMAT=otf2`
  `% aprun -n N tau_exec [options] ./a.out; vampir traces.otf2 &`
- `Chrome:` `% export TAU_TRACE=1; aprun -n N 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 https://Perfetto.dev`
- `Jumpshot:` `% export TAU_TRACE=1; aprun -n N tau_exec [Options]./a.out;`
  `% tau_treemerge.pl; tau2slog2 tau.trc tau.edf –o app.slog2; jumpshot app.slog2`

TAU: Quickstart Guide for Sunspot.alcf.anl.gov

```
% tar xf /soft/perftools/tau/tar/workshop.tgz; cd workshop; cat README
% qsub -I -l select=<N> -l walltime=1:00:00 -A <Project_id>
% module use /soft/modulefiles; module load tau
```

- Un-instrumented run with MPI `% mpirun –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:` `% mpirun -np N tau_exec -ebs ./a.out`
- `DPC++/SYCL with MPI:` `% mpirun –np N tau_exec –T level_zero -l0 ./a.out`
- `DPC++/SYCL without MPI:` `% TAU_SET_NODE=0 tau_exec –T level_zero –l0 ./a.out`
- `OpenMP with MPI:` `% mpirun -np N tau_exec –T ompt -ompt ./a.out`
- `OpenMP without MPI:` `% TAU_SET_NODE=0 tau_exec –T ompt –ompt./a.out`
- `OpenCL with MPI:` `% mpirun -np N tau_exec –T level_zero -opencl ./a.out`
- `OpenCL without MPI:` `% TAU_SET_NODE=0 tau_exec –T level_zero -opencl ./a.out`

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

Tracing:

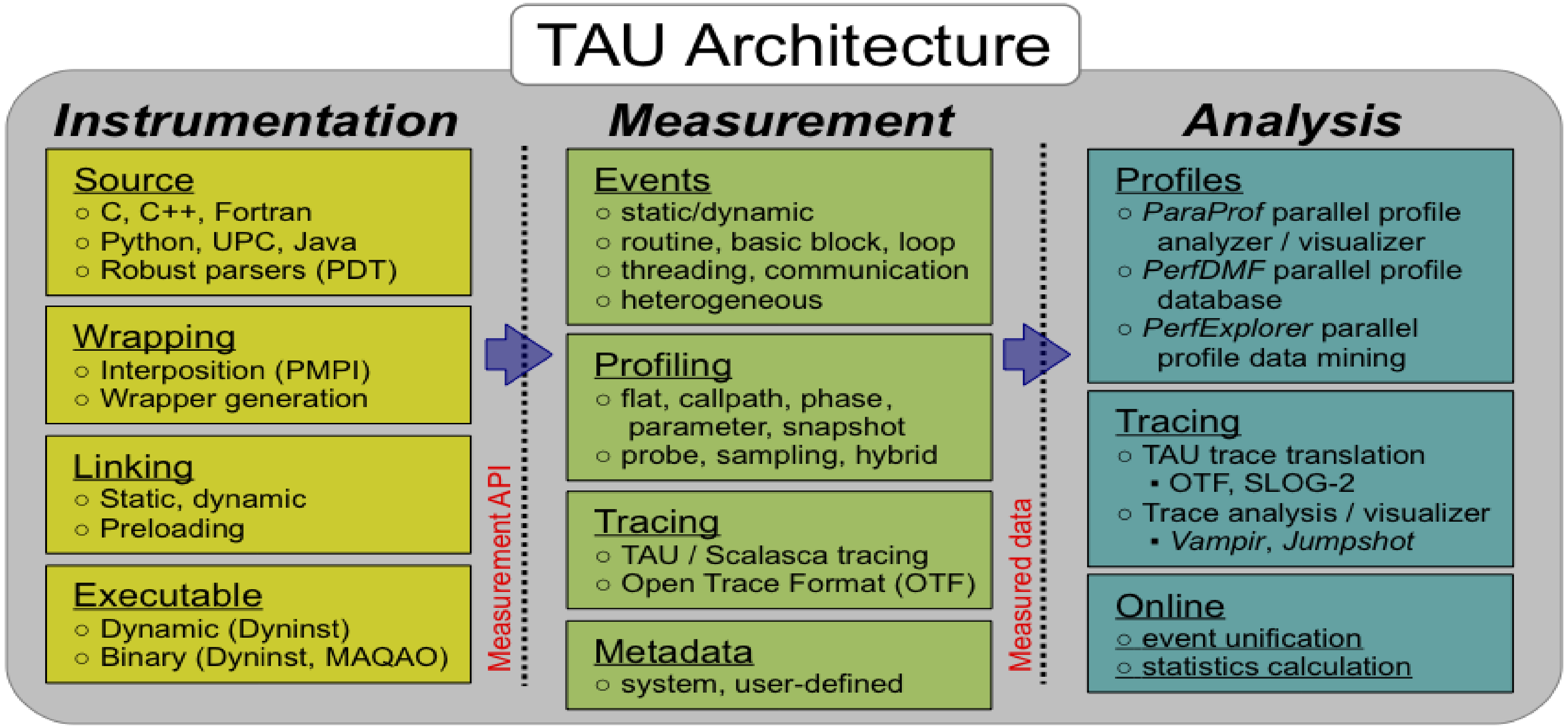
- `Vampir:`
```
% export TAU_TRACE=1; export TAU_TRACE_FORMAT=otf2
% mpirun -np N tau_exec [Options] ./a.out; vampir traces.otf2 &
```
- `Chrome:`
```
% export TAU_TRACE=1; mpirun –np N tau_exec ./a.out; tau_treemerge.pl;
% tau_trace2json tau.trc tau.edf –chrome –ignoreatomic –o app.json
```
  `Chrome browser: chrome://tracing (Load -> app.json) or https://Perfetto.dev`
- `Jumpshot:`
```
% export TAU_TRACE=1; mpirun –np N tau_exec [Options]./a.out;
% tau_treemerge.pl; 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

Tuning and Analysis Utilities



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 **mpirun tau_exec ./app** with options.

Configure TAU on Polaris

Support for MPI and CUDA

- ./configure -c++=CC -cc=cc -fortran=ftn -cuda=/opt/nvidia/hpc_sdk/Linux_x86_64/23.9/cuda/12.2 -pdt=/soft/perftools/tau/pdtoolkit-3.25.2 -bfd=download -otf=download -dwarf=download -iowrapper -mpi -papi=/opt/cray/pe/papi/7.0.1.2/ -pthread
- make install –j

  Builds craycnl/lib/Makefile.tau-nvidia-papi-mpi-cupti-pdt
  and craycnl/lib/shared-nvidia-papi-mpi-cupti-pdt/libTAU.so

- We can build multiple configurations of TAU with PrgEnv-nvhpc and PrgEnv-gnu

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
```

Using TAU

- TAU supports several measurement and thread options
  - Phase profiling, profiling with hardware counters, MPI library, CUDA…
  - 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 load tau
% export TAU_MAKEFILE=/soft/perftools/tau/tau-2.33.2/craycnl/lib/Makefile.tau-gnu-papi-mpi-pthread-cupti-pdt

% 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)**

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

```
$ 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 –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)

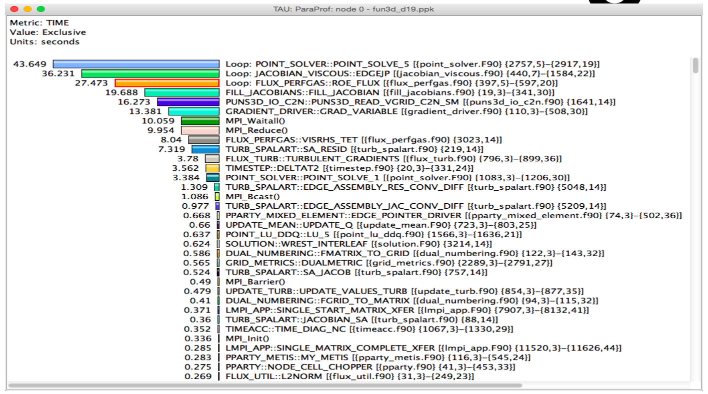
Use event based sampling (compile with –g)

% mpirun –np 256 tau_exec –ebs ./a.out

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

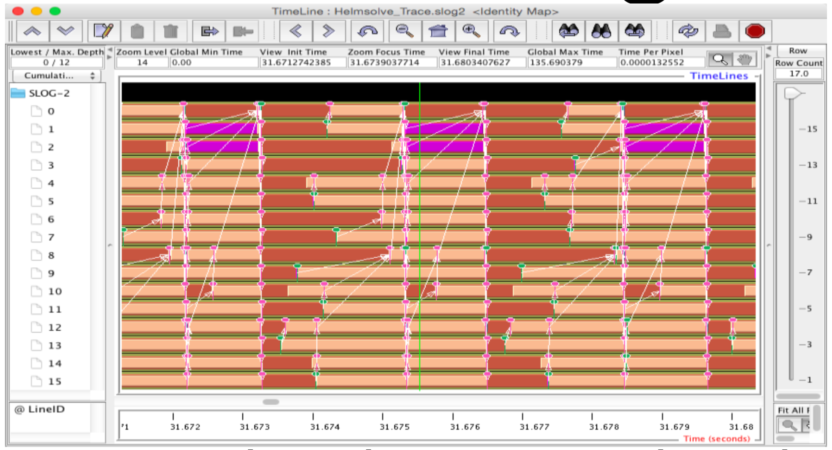
Profiling and Tracing

Profiling



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

Tracing



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

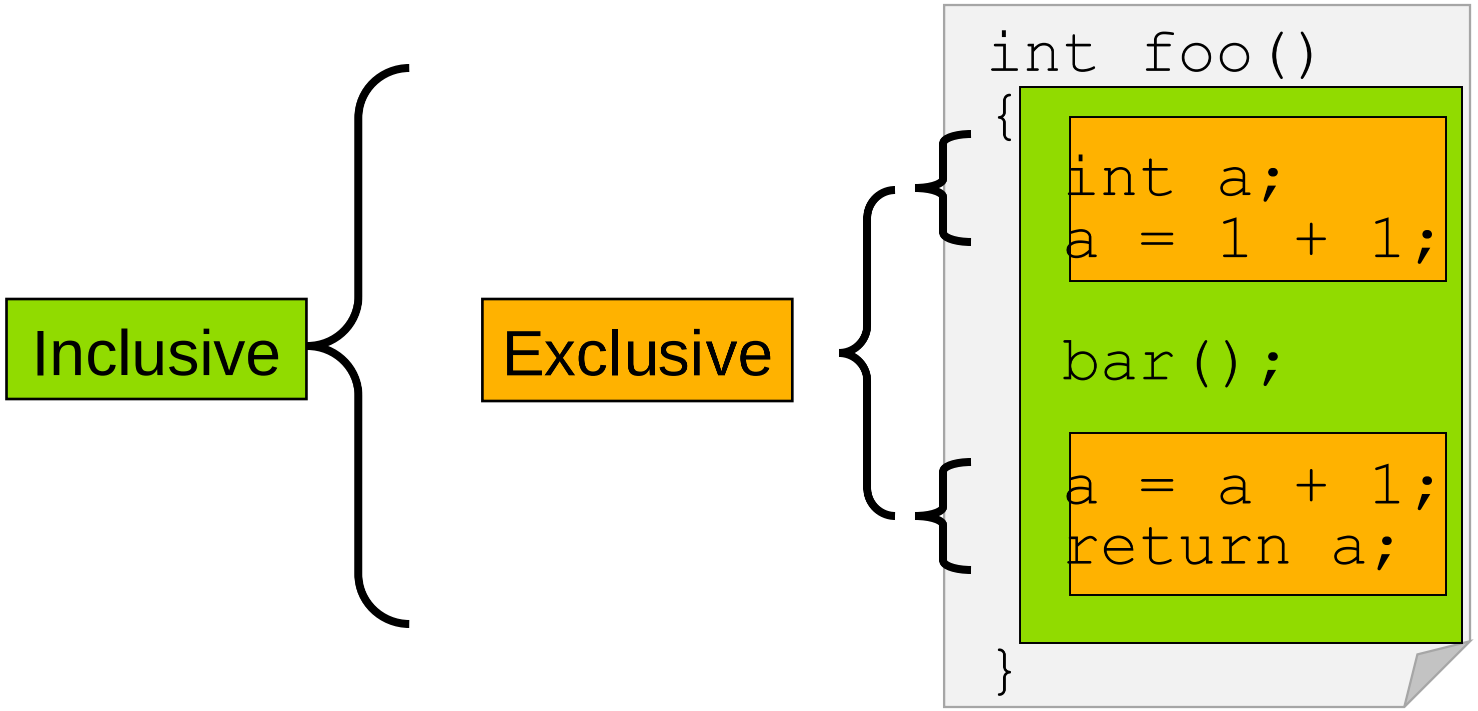
- Profiling and tracing

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

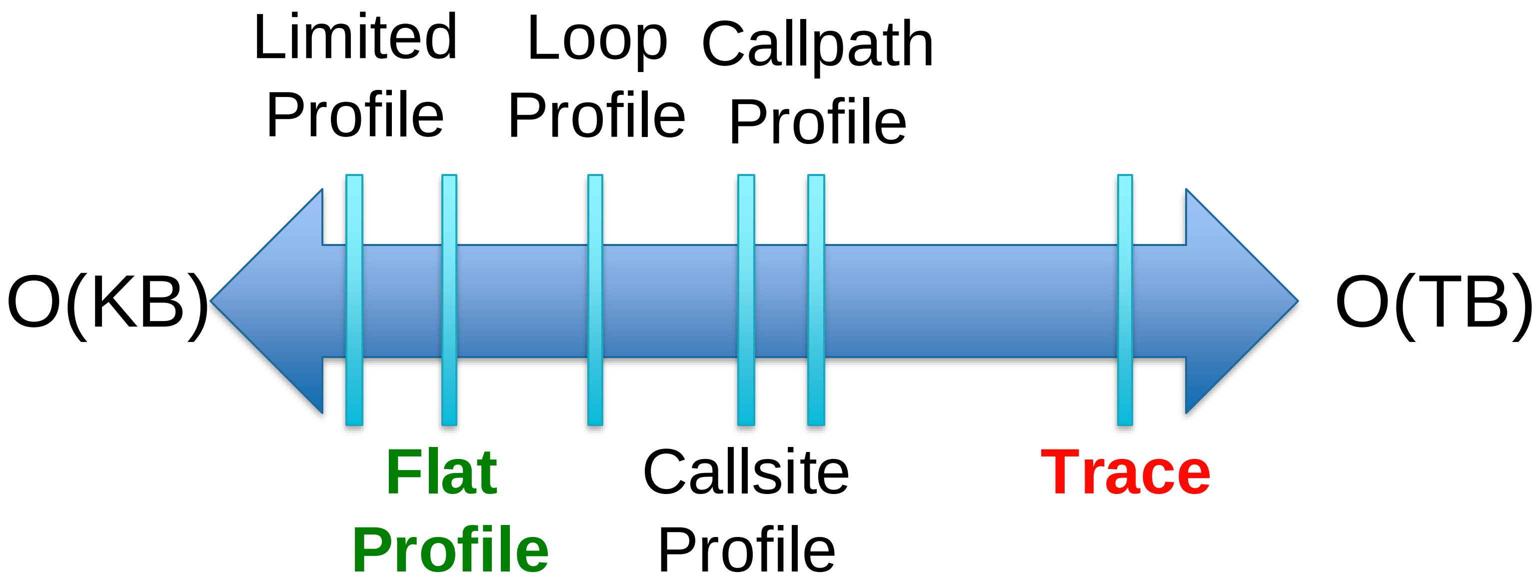
  Tracing shows you **when** the events take place on a timeline

Inclusive vs. Exclusive values

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



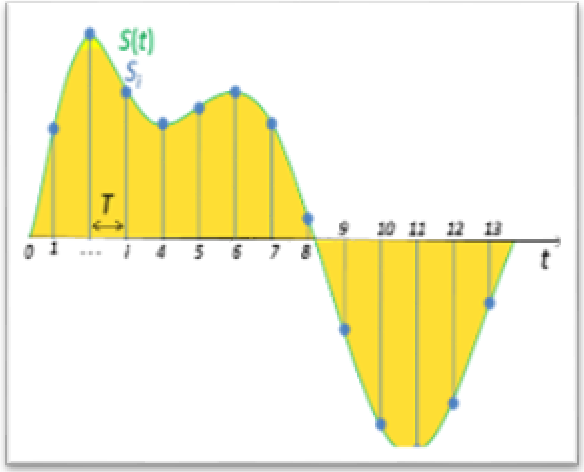
How much data do you want?



Performance Data Measurement

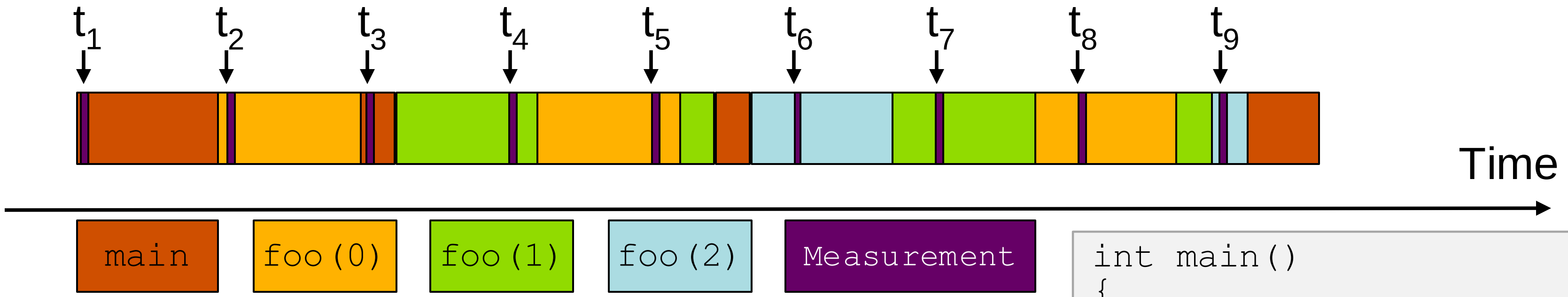
```
Call START('potential')
// code
Call STOP('potential')
```

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



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

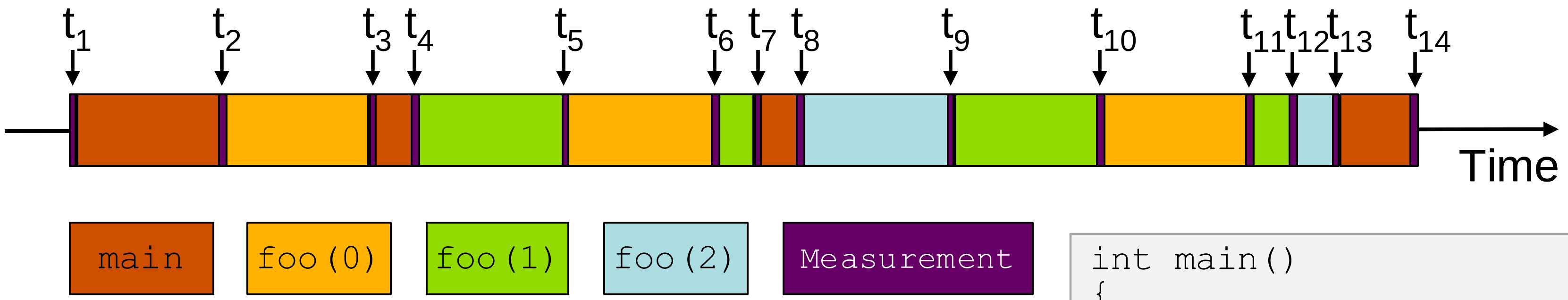
Sampling



- 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);

}
```

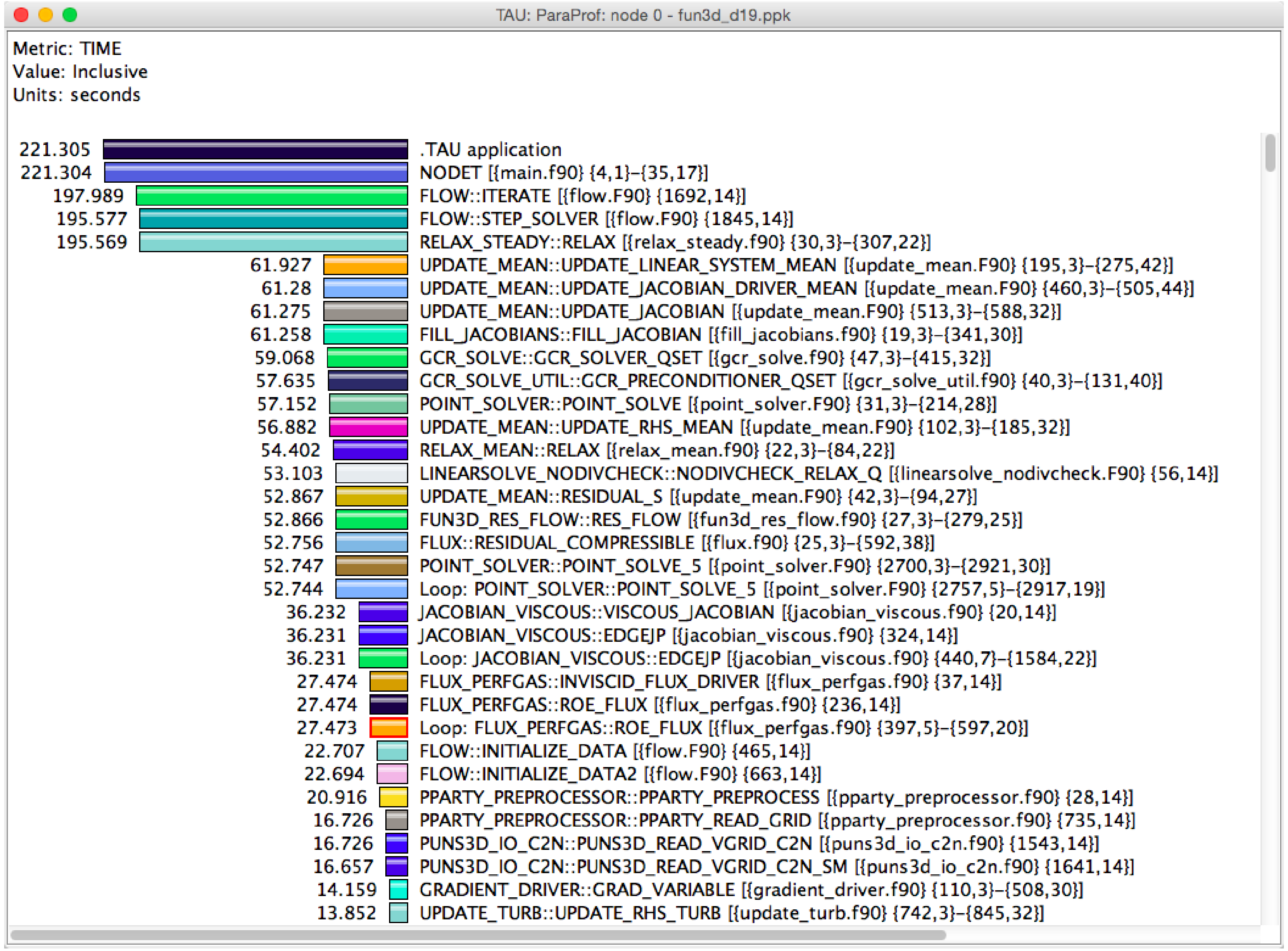
Instrumentation



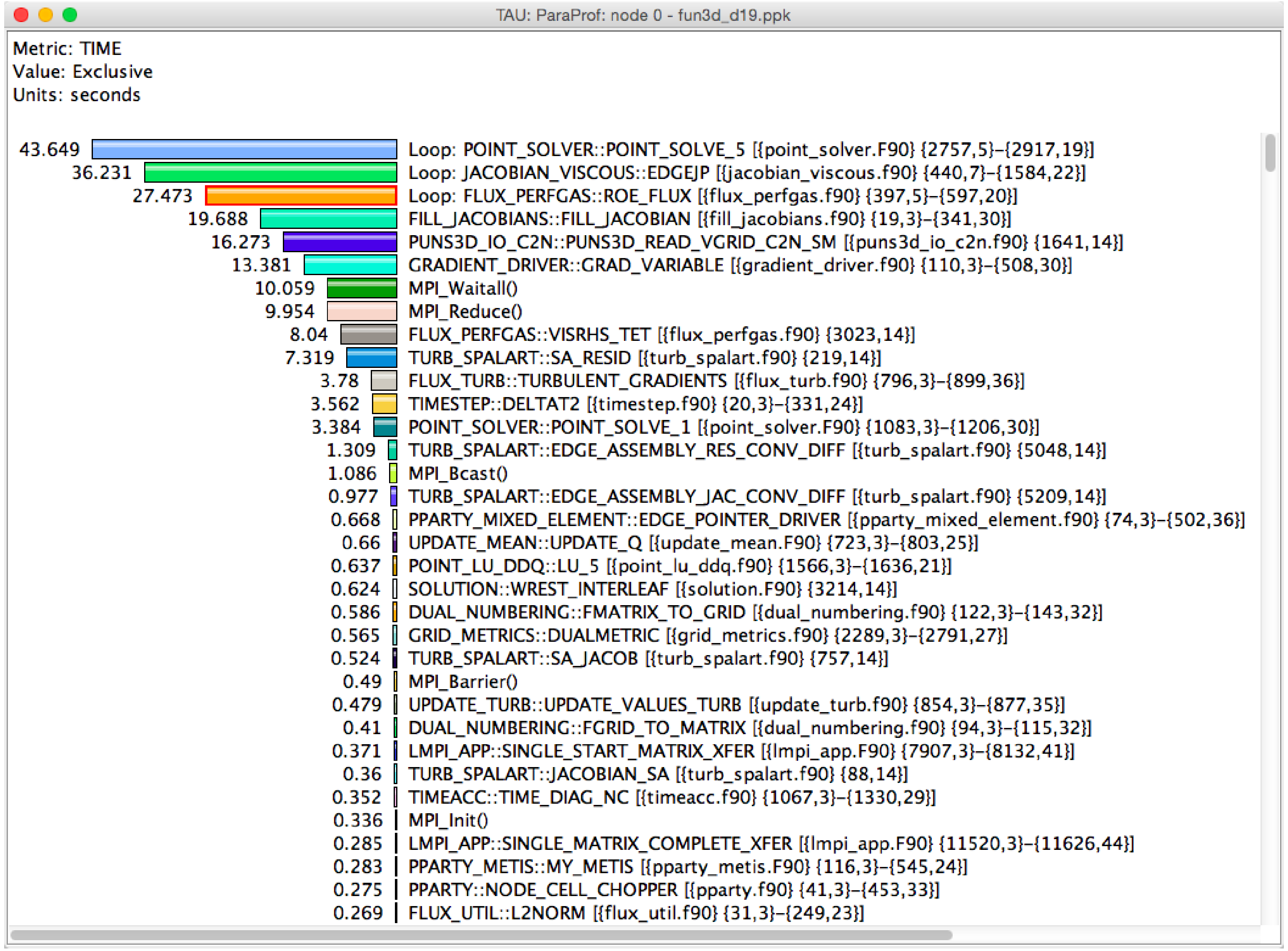
- Measurement code is inserted such that every event of interest is captured directly
  - Can be done in various ways
- Advantage:
  - Much more detailed information
- Disadvantage:
  - Processing of source-code / executable necessary
  - Large relative overheads for small functions

```
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



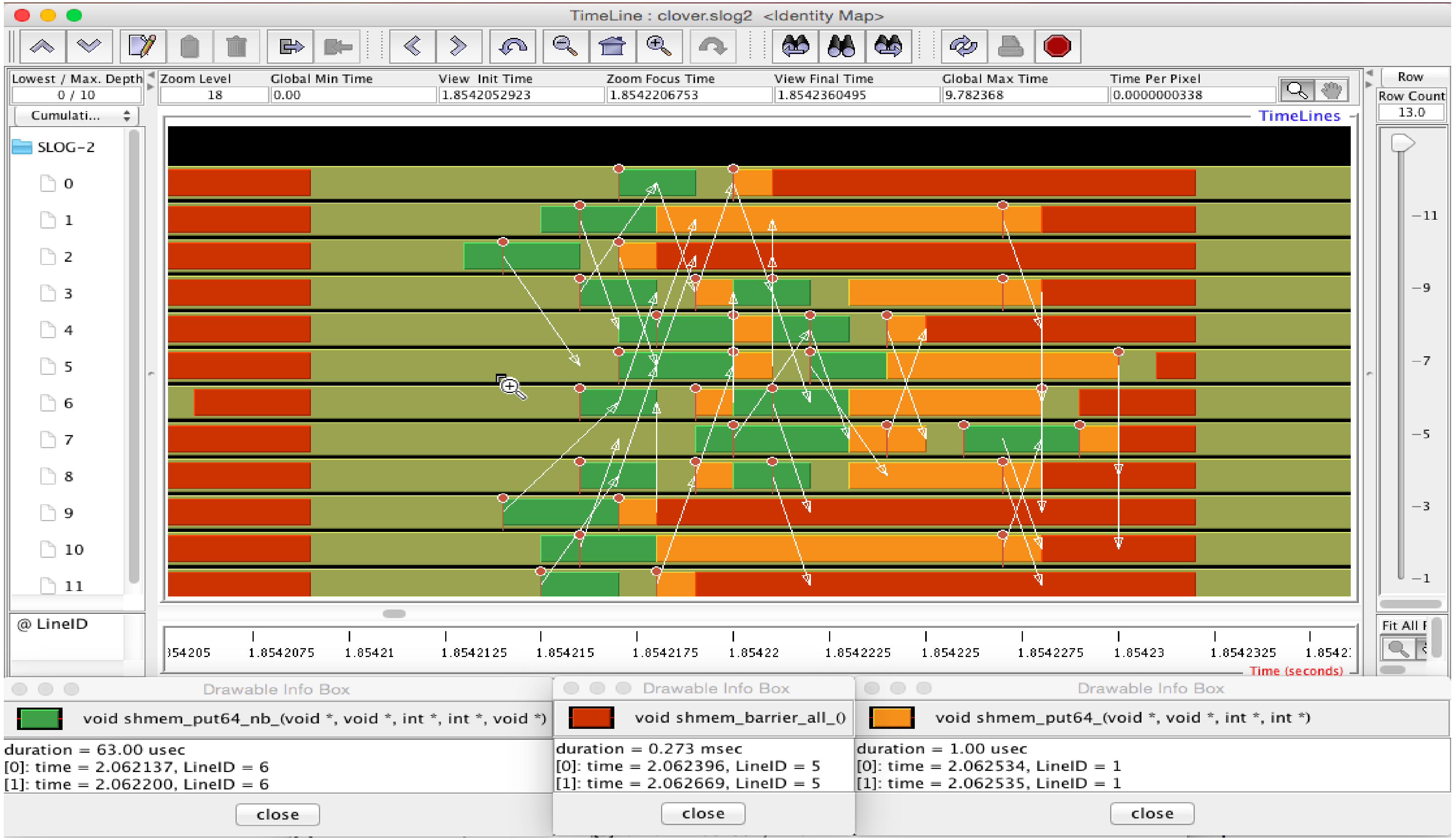
Exclusive Time



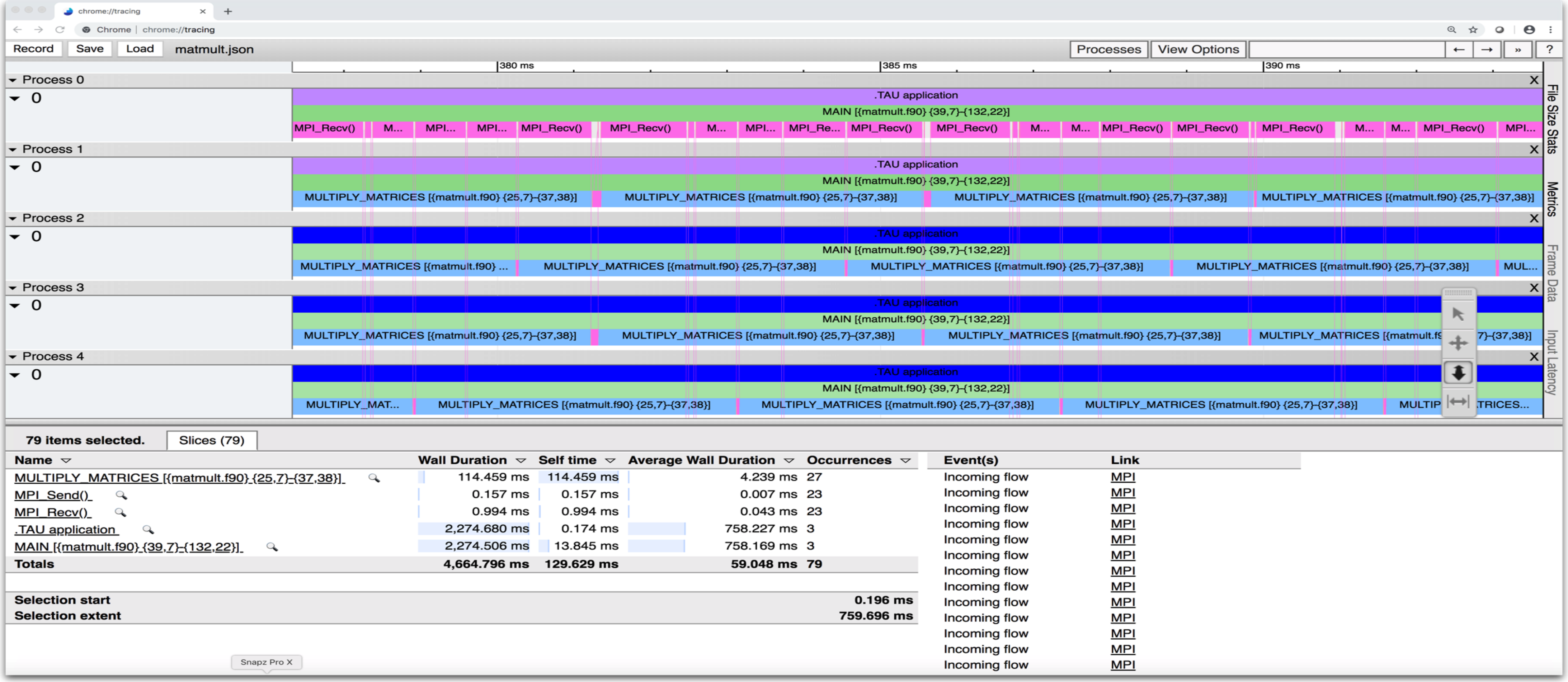
TAU’s Runtime Environment Variables

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 (ships with TAU)



Tracing: Chrome Browser

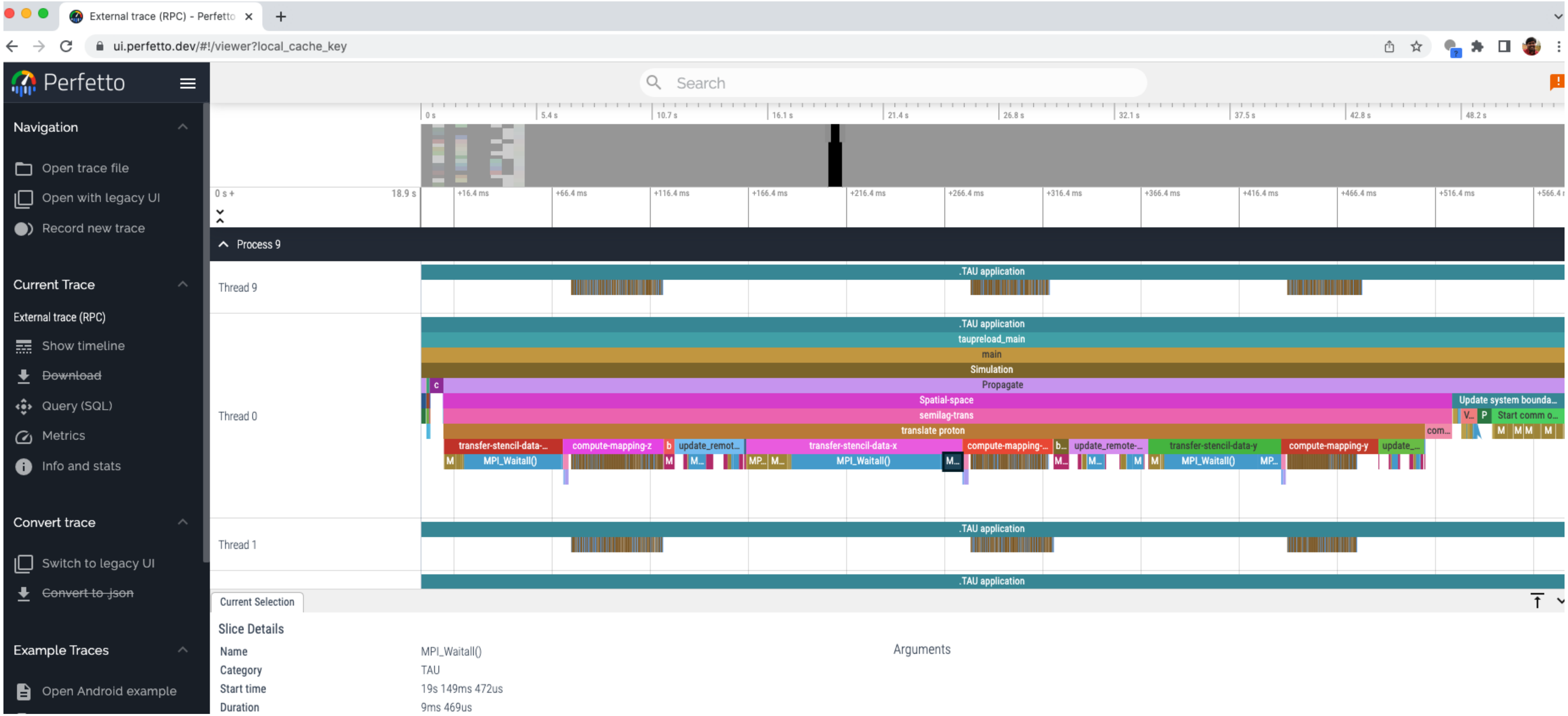


% export TAU_TRACE=1
% mpirun –np 256 tau_exec ./a.out
% tau_treemerge.pl; tau_trace2json tau.trc tau.edf –chrome –ignoreatomic –o app.json

Chrome browser: chrome://tracing (Load -> app.json)

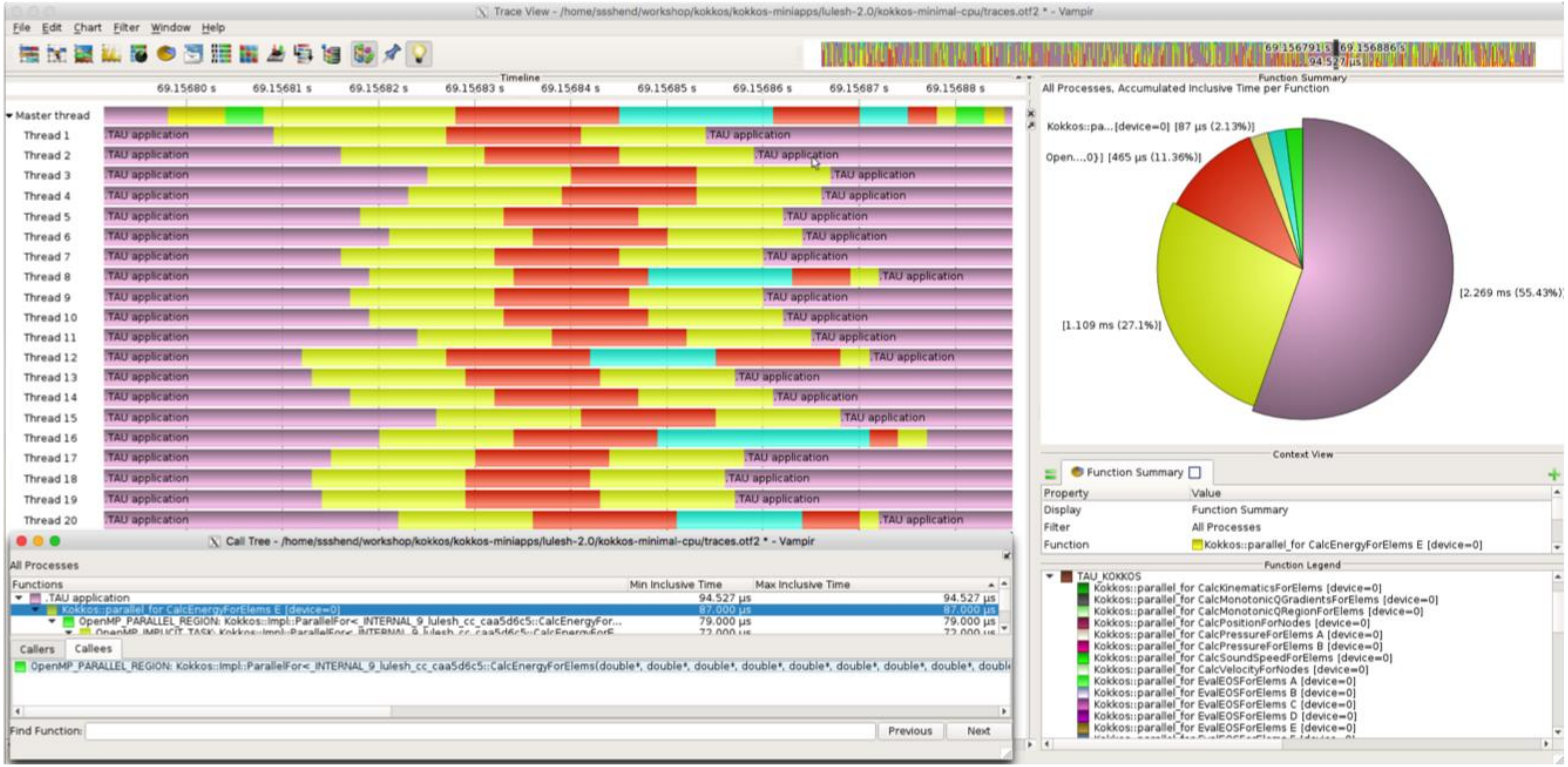
UNIVERSITY OF OREGON

Perfetto.dev



UNIVERSITY OF OREGON

Vampir [TU Dresden] Timeline: Kokkos



% export TAU_TRACE=1; export TAU_TRACE_FORMAT=otf2

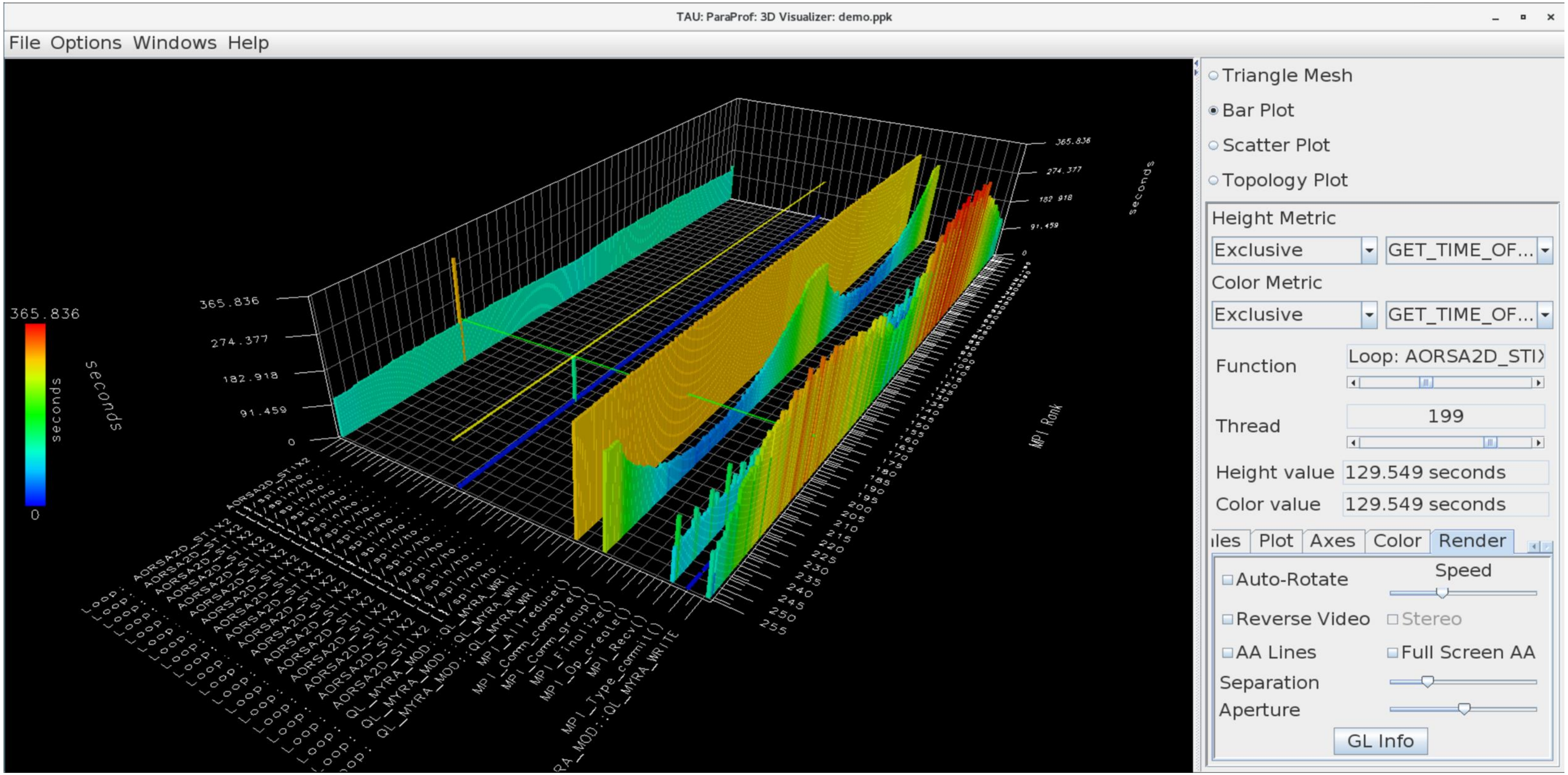
% tau_exec -T ompt –ompt ./a.out

% vampir traces.otf2 &

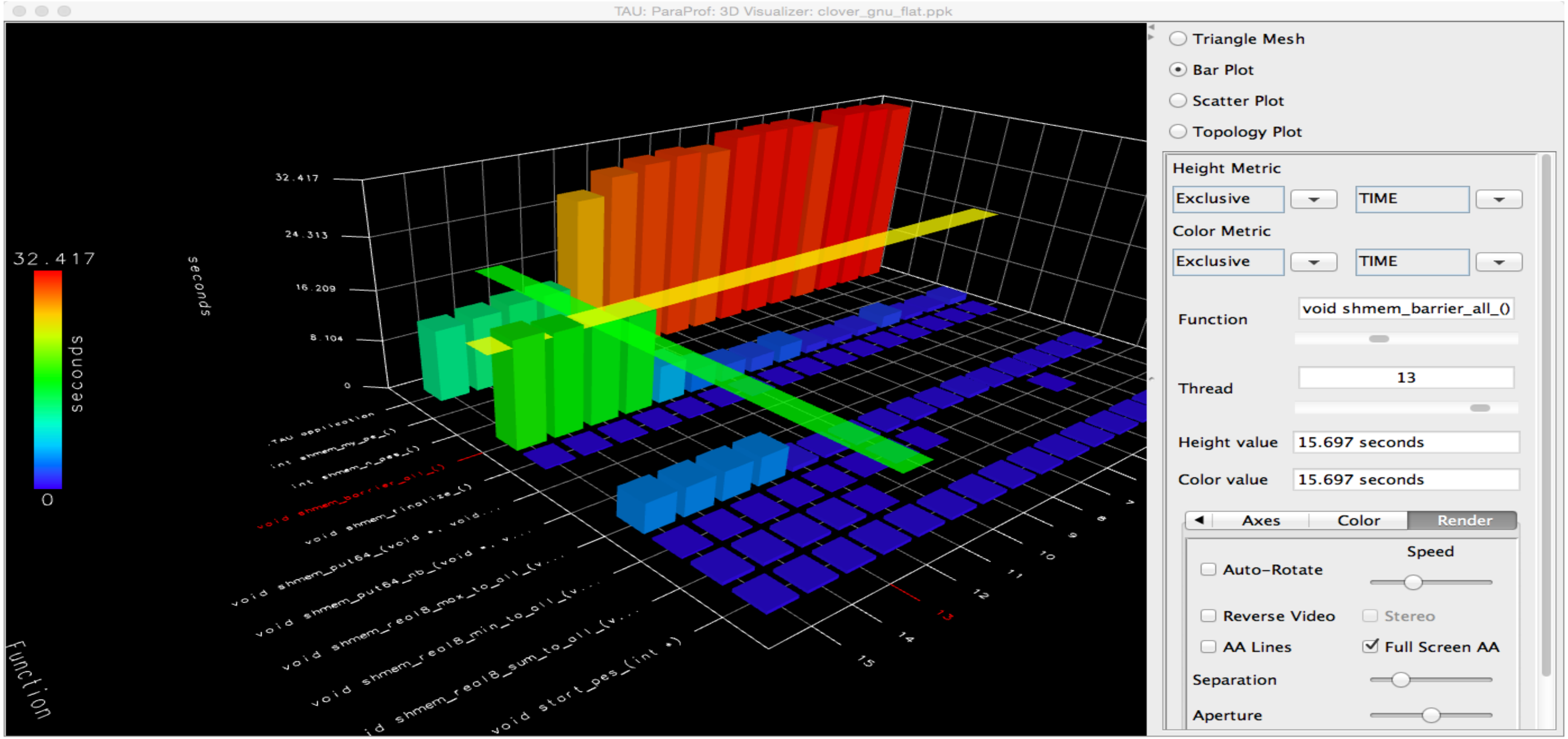
ParaProf Profile Browser


```
% paraprof
```

ParaProf 3D Profile Browser

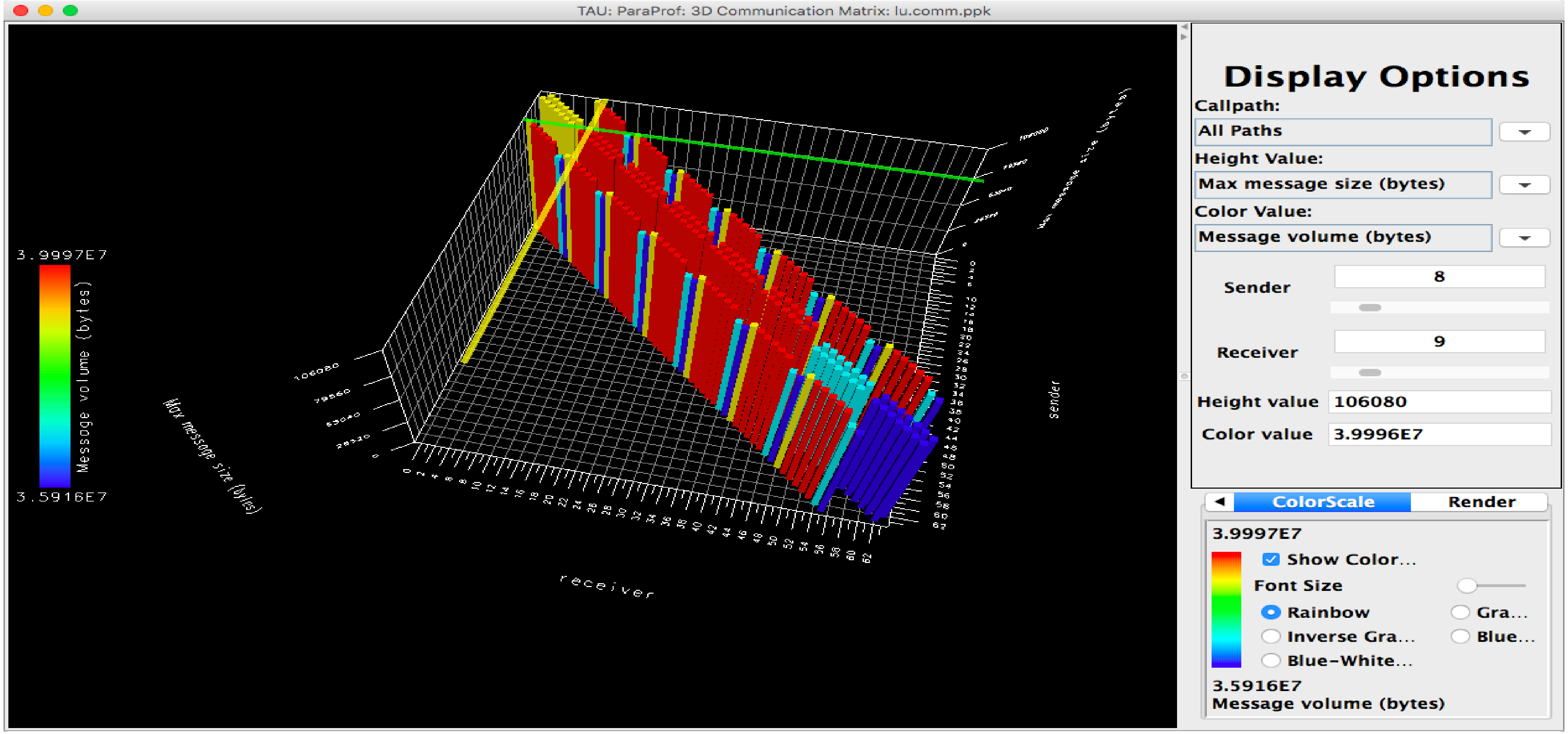


TAU – ParaProf 3D Visualization



% paraprof app.ppk
Windows -> 3D Visualization -> Bar Plot (right pane)

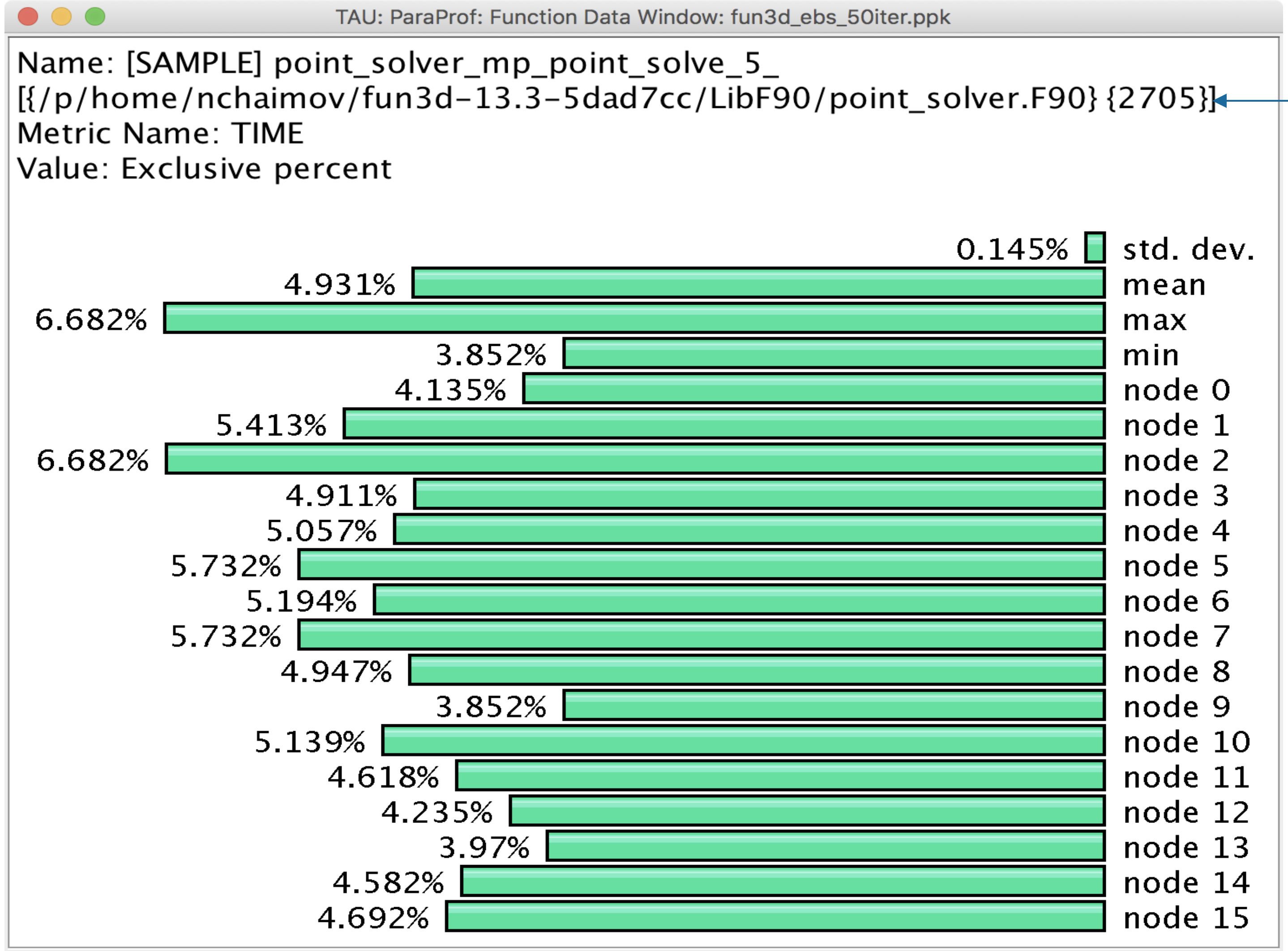
TAU – 3D Communication Window



% 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 –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

http://tau.uoregon.edu

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

for more information

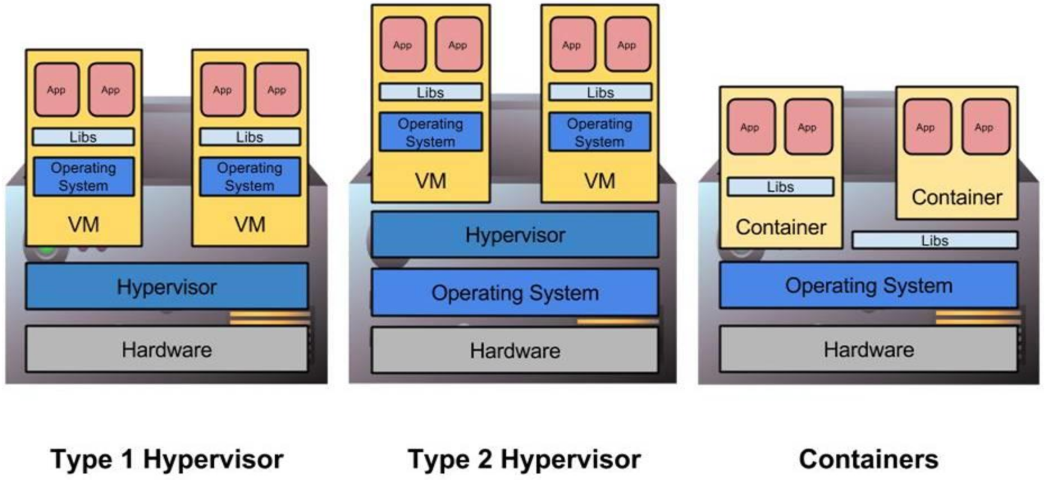
Free download, open source, BSD license

What are containers

- A lightweight collection of executable software that encapsulates everything needed to run a single specific task
  - Minus the OS kernel
  - Based on Linux only
- Processes and all user-level software is isolated
- Creates a portable* software ecosystem
- Think `chroot` on steroids
- Docker most common tool today
  - Available on all major platforms
  - Widely used in industry
  - Integrated container registry via Dockerhub

Hypervisors and Containers

- Type 1 hypervisors insert layer below host OS
- Type 2 hypervisors work as or within the host OS
- Containers do not abstract hardware, instead provide “enhanced chroot” to create isolated environment
- Location of abstraction can have impact on performance
- All enable custom software stacks on existing hardware



E4S: Extreme-scale Scientific Software Stack

- E4S is 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, curated software stack based on Spack [https://spack.io] 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 120+ HPC (OpenFOAM, Gromacs, Nek5000, LAMMPS), EDA (e.g., Xyce), and AI/ML packages (e.g., TensorFlow, PyTorch, TorchBraid, Scikit-Learn, Pandas, JAX, Horovod, and LBANN).
- 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 GPUs from NVIDIA, AMD, and Intel
    - HPC and AI/ML packages are optimized for GPUs and CPUs.
- Container images on DockerHub and E4S website of pre-built binaries of ECP ST products.
- Base images and full featured containers (with GPU support) and DOE LLVM containers.
- Commercial support for E4S through ParaTools, Inc. for installation, maintaining an issue tracker, and ECP AD engagement.
- E4S for commercial cloud platforms: Adaptive Computing's ODDC with ParaTools Pro for E4S image with support for AWS.
    - https://adaptivecomputing.com
- 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.

PESO

https://e4s.io

E4S Community Policies Version 1

A Commitment to Quality Improvement

- Will serve as membership criteria for E4S
  - Membership is not required for *inclusion* in E4S
  - Also includes forward-looking draft policies
- Purpose: enhance sustainability and interoperability
- Topics cover building, testing, documentation, accessibility, error handling and more
- Multi-year effort led by SDK team
  - Included representation from across ST
  - Multiple rounds of feedback incorporated from ST leadership and membership
- Modeled after xSDK Community Policies
- https://e4s-project.github.io/policies.html

P1 *Spack-based Build and Installation* Each E4S member package supports a scriptable *Spack* build and production-quality installation in a way that is compatible with other E4S member packages in the same environment. When E4S build, test, or installation issues arise, there is an expectation that teams will collaboratively resolve those issues.

P2 *Minimal Validation Testing* Each E4S member package has at least one test that is executable through the E4S validation test suite (*https://github.com/E4S-Project/testsuite*). This will be a post-installation test that validates the usability of the package. The E4S validation test suite provides basic confidence that a user can compile, install and run every E4S member package. The E4S team can actively participate in the addition of new packages to the suite upon request.

P3 *Sustainability* All E4S compatibility changes will be sustainable in that the changes go into the regular development and release versions of the package and should not be in a private release/branch that is provided only for E4S releases.

P4 *Documentation* Each E4S member package should have sufficient documentation to support installation and use.

P5 *Product Metadata* Each E4S member package team will provide key product information via metadata that is organized in the *E4S DocPortal* format. Depending on the filenames where the metadata is located, this may require *minimal setup*.

P6 *Public Repository* Each E4S member package will have a public repository, for example at GitHub or Bitbucket, where the development version of the package is available and pull requests can be submitted.

P7 *Imported Software* If an E4S member package imports software that is externally developed and maintained, then it must allow installing, building, and linking against a functionally equivalent outside copy of that software. Acceptable ways to accomplish this include (1) forsaking the internal copied version and using an externally-provided implementation or (2) changing the file names and namespaces of all global symbols to allow the internal copy and the external copy to coexist in the same downstream libraries and programs. This pertains primarily to third party support libraries and does not apply to key components of the package that may be independent packages but are also integral components to the package itself.

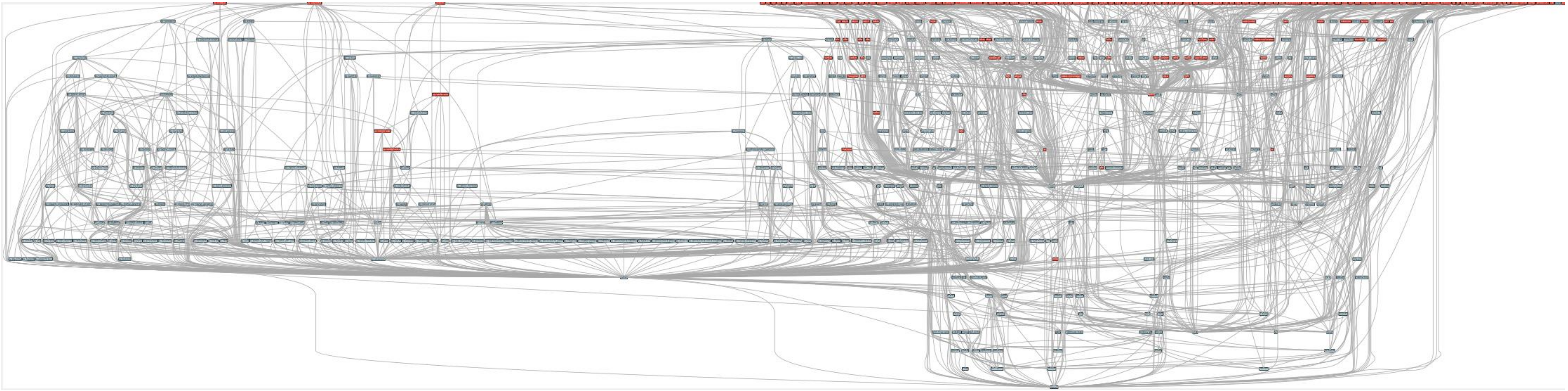
P8 *Error Handling* Each E4S member package will adopt and document a consistent system for signifying error conditions as appropriate for the language and application. For e.g., returning an error condition or throwing an exception. In the case of a command line tool, it should return a sensible exit status on success/failure, so the package can be safely run from within a script.

***P9** Test Suite* Each E4S member package will provide a test suite that does not require special system privileges or the purchase of commercial software. This test suite should grow in its comprehensiveness over time. That is, new and modified features should be included in the suite.

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.

Managing large software installations: E4S



- Red boxes are the packages in it (about 100)
- Blue boxes are what *else* you need to build it (about 600)
- It’s infeasible to build and integrate all of this manually

Spack enables software distribution for HPC

No installation required: clone and go

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

$ spack install tau	unconstrained
$ spack install tau@2.33.2	@ custom version
$ spack install tau@2.33.2 %gcc@11.2.0	% custom compiler
$ spack install tau@2.33.2 %gcc@11.2.0 +mpi+python+pthreads	+/- build option
$ spack install tau@2.33.2 %gcc@11.2.0 +mpi ^mvapich2@2.3~wrapperrpath	^ 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

github.com/spack/spack

PESO

The Spack community is growing rapidly

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

Visit spack.io

github.com/spack/spack

@spackpm

PESO

Download E4S 24.05 Container for Intel oneAPI: Docker or Singularity

https://e4s-project.github.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:24.05
```

```
# docker pull ecpe4s/e4s-rocm:24.05
```

```
# docker pull ecpe4s/e4s-oneapi:24.05
```

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:24.05 docker

e4s-rocm90a-x86_64-24.05.sif mirror 1

e4s-rocm908-x86_64-24.05.sif mirror 1

NVIDIA CUDA (X86_64, PPC64LE, AARCH64)

ecpe4s/e4s-cuda:24.05 docker

e4s-cuda80-x86_64-24.05.sif mirror 1

e4s-cuda90-x86_64-24.05.sif mirror 1

e4s-cuda70-ppc64le-24.05.sif mirror 1

e4s-cuda75-aarch64-24.05.sif mirror 1

e4s-cuda80-aarch64-24.05.sif mirror 1

e4s-cuda90-aarch64-24.05.sif mirror 1

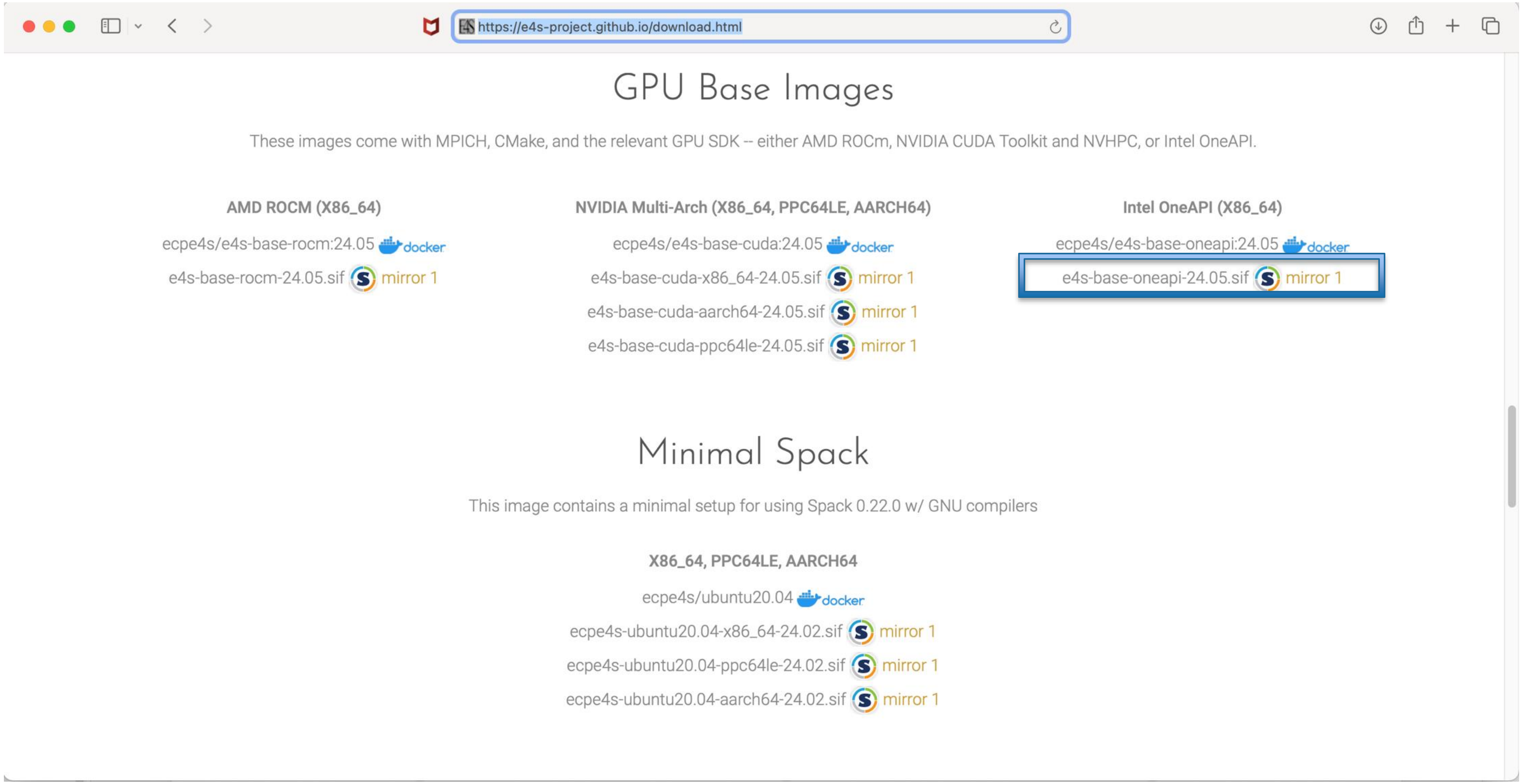
Intel OneAPI (x86_64)

ecpe4s/e4s-oneapi:24.05 docker

e4s-oneapi-x86_64-24.05.sif mirror 1

https://e4s.io

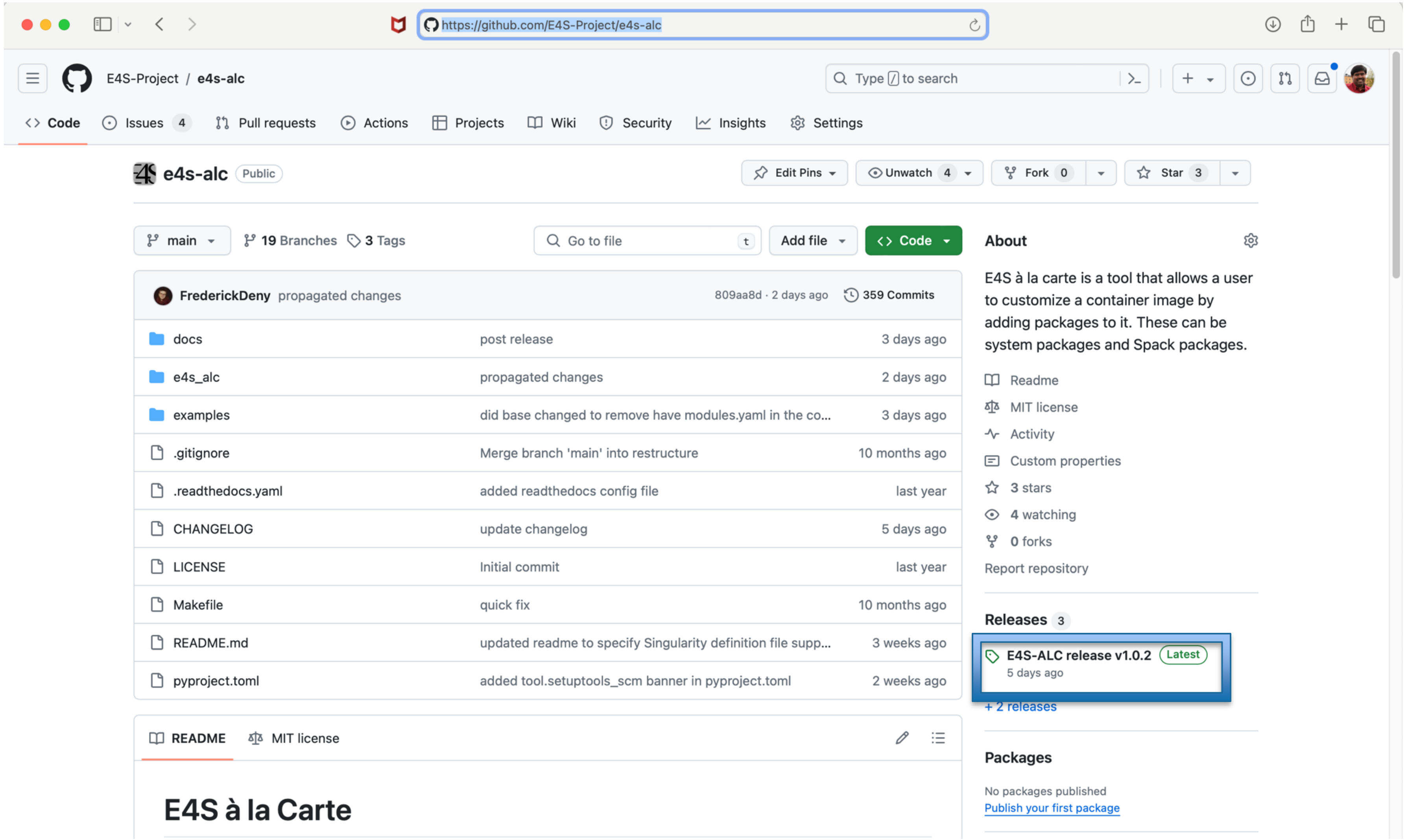
E4S base container images allow users to customize their containers



- Intel oneAPI

https://e4s.io

e4s-alc: a new tool to customize container images. Version 1.0.2

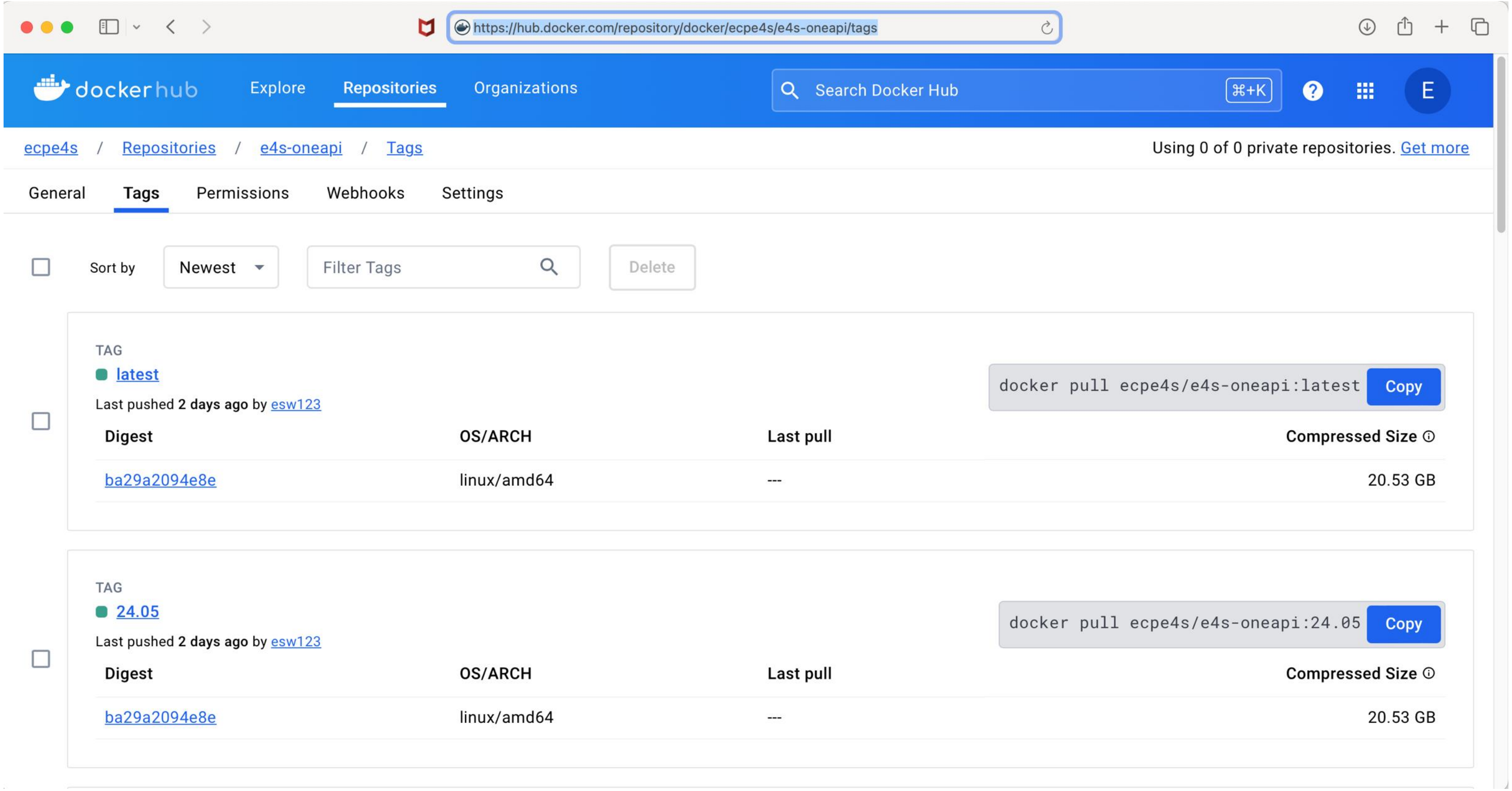


Add to a base image:

- Spack packages
- OS packages
- Tarballs
- Can create a Dockerfile
- Can create Singularity definition file

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

E4S 24.05 oneAPI release on Dockerhub



docker pull ecpe4s/e4s-oneapi

Using E4S on Cloud Platforms

Considerations while deploying HPC/AI workloads to the cloud

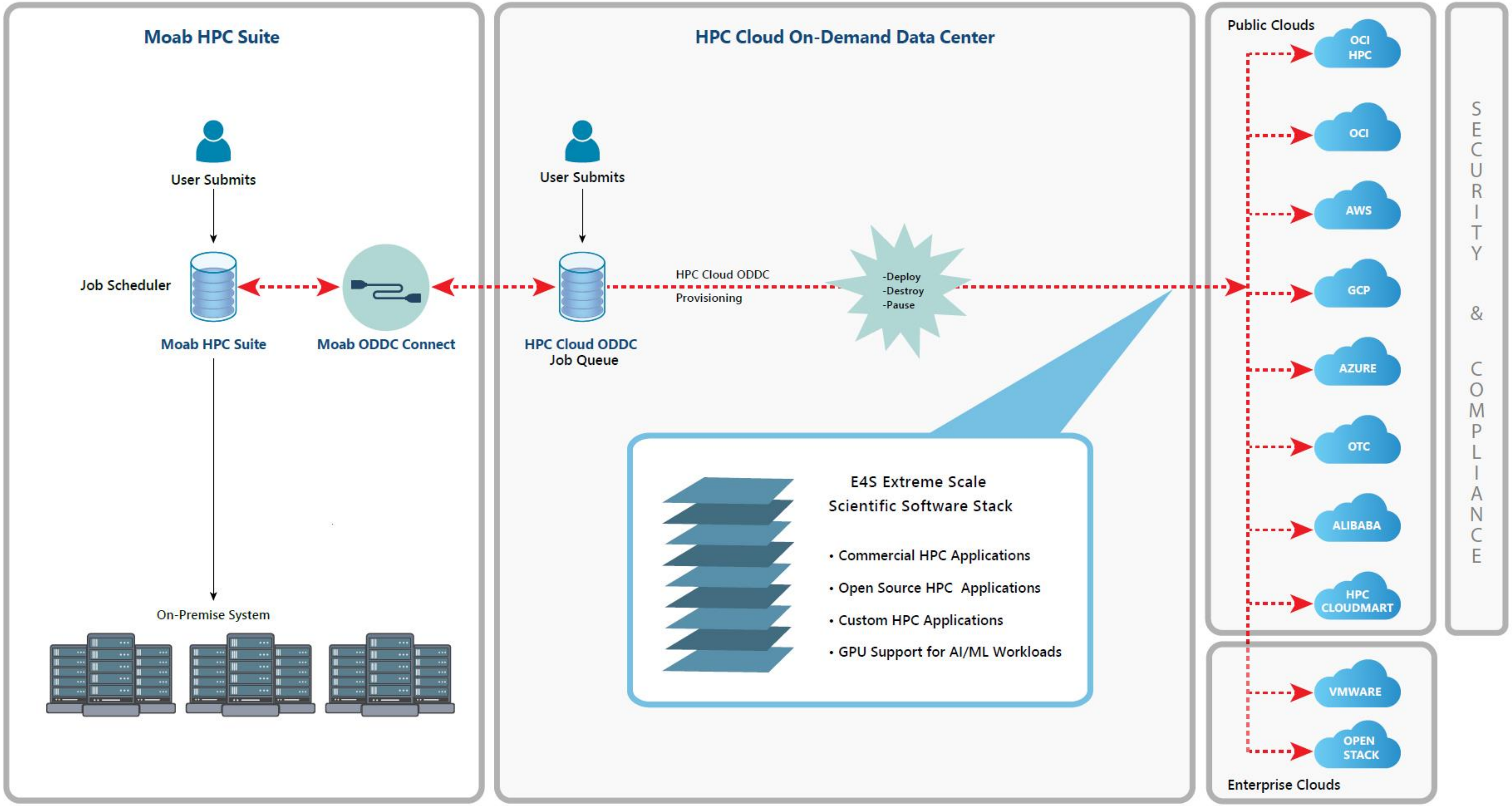
- Which cloud provider?
  - AWS, OCI, GCP, Azure, …
  - Why not all?
- HPC and AI/ML workloads need low latency, high bandwidth
  - Which MPI?
- Which image?
  - Base Ubuntu without HPC tools or libraries? Too steep a learning curve
- Provisioning and building the image on different cloud providers
  - Command line interfaces can be cumbersome to use
- Bursting to the cloud from on-prem clusters using batch submission scripts?

PESO

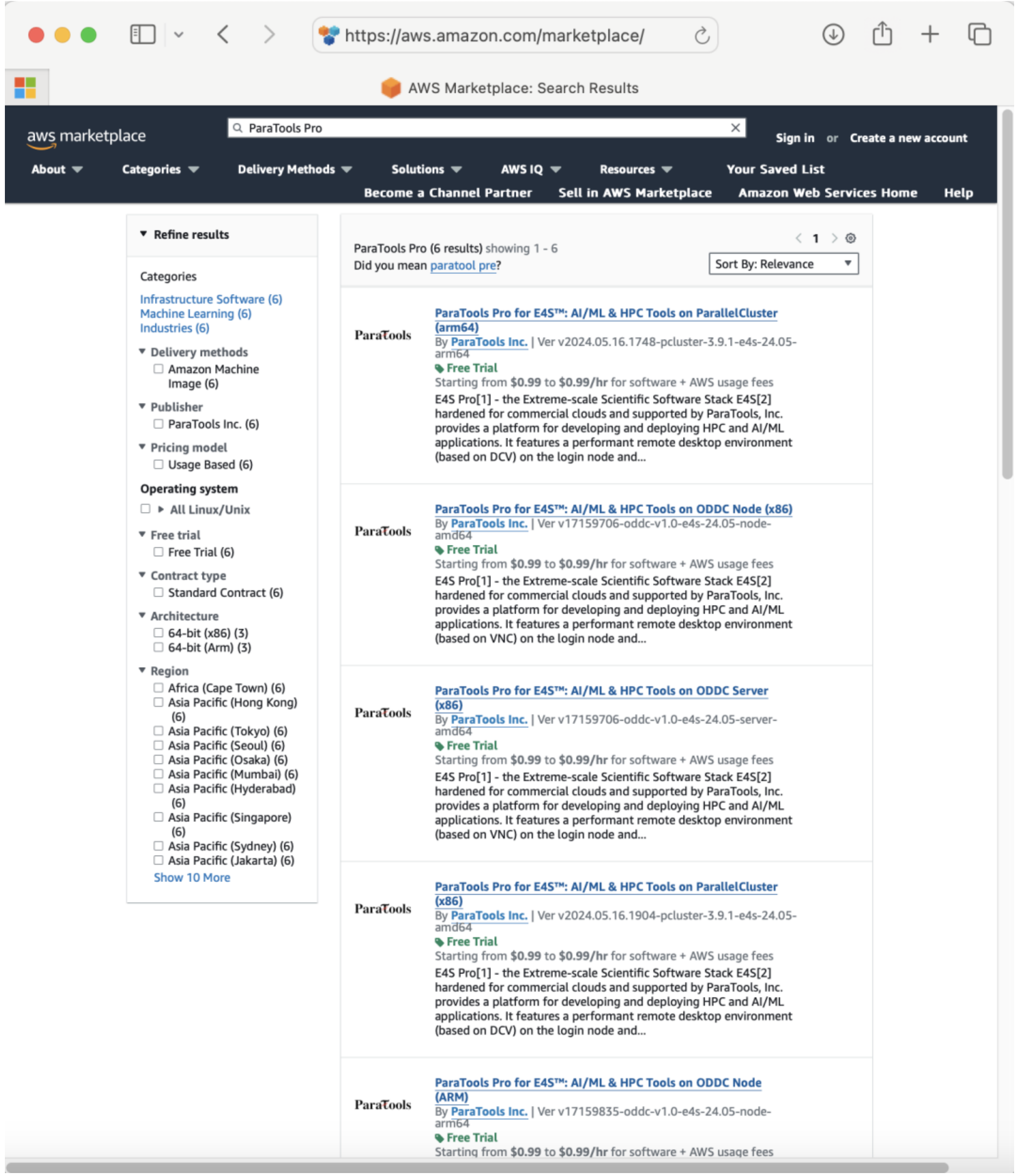
Key considerations for cloud-based deployment for E4S

- MPI - the core inter-node communication library has several implementations
  - Intel MPI, MVAPICH2-X, OpenMPI
  - Interfacing MPI with the job scheduling package (MOAB, Torque, SLURM)
- Cloud providers have different inter-node network adapters:
  - Elastic Fabric Adapter (EFA) on AWS
  - Infiniband on Azure
  - Mellanox Connect-X 5 Ethernet (ROCE) on Oracle Cloud Infrastructure (OCI)
  - IPU on Google Cloud (GCP)
- Intra-node communication with XPMEM (driver and kernel module support is critical)
- GPU Direct Async (GDR) support for communication between GPUs in MVPICH-Plus release
- ParaTools, Inc. building E4S optimized with MVAPICH-Plus for AWS, OCI, GCP, and Azure
- Using Adaptive Computing’s ODDC interface to launch E4S jobs on multiple cloud providers!

Adaptive Computing's ODDC with ParaTools Pro for E4S



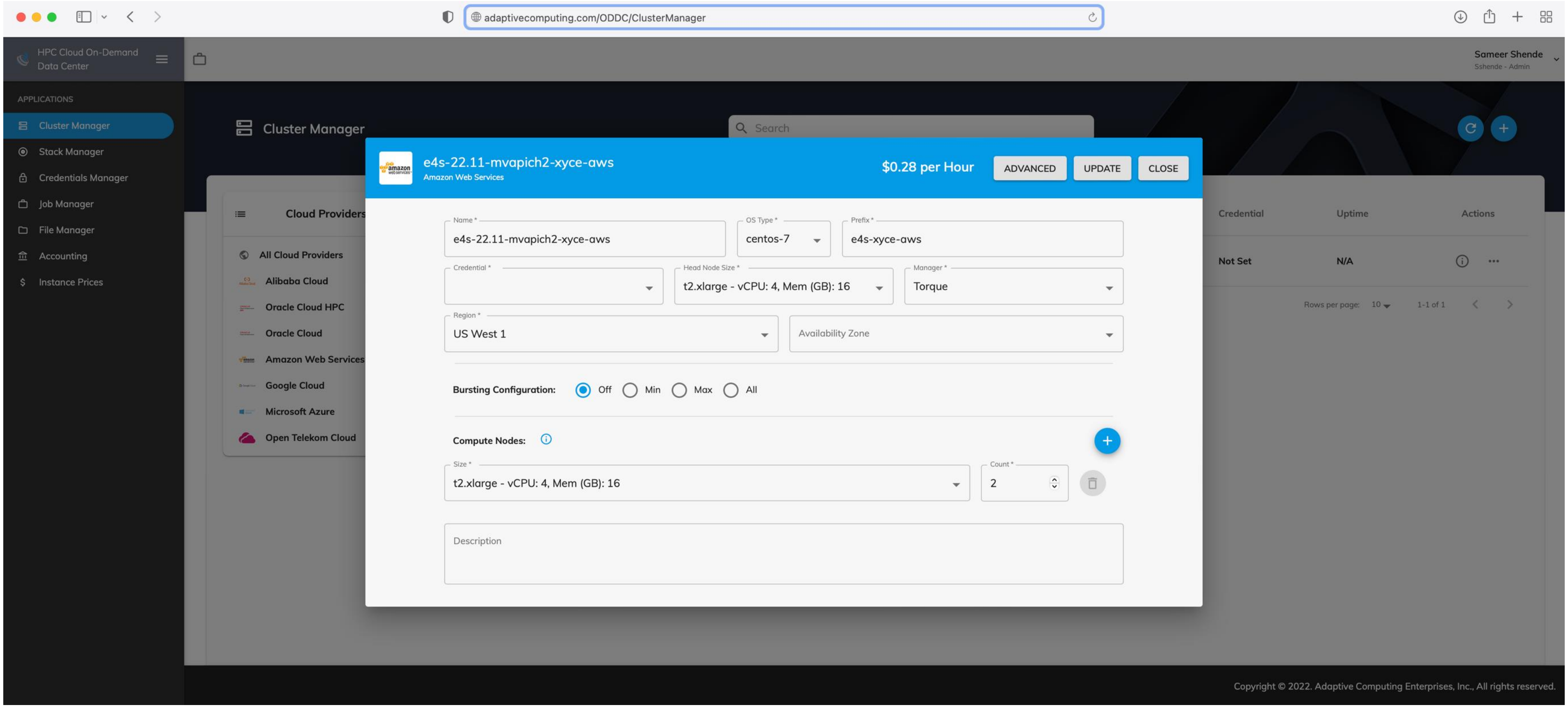
ParaTools Pro for E4S on AWS Marketplace



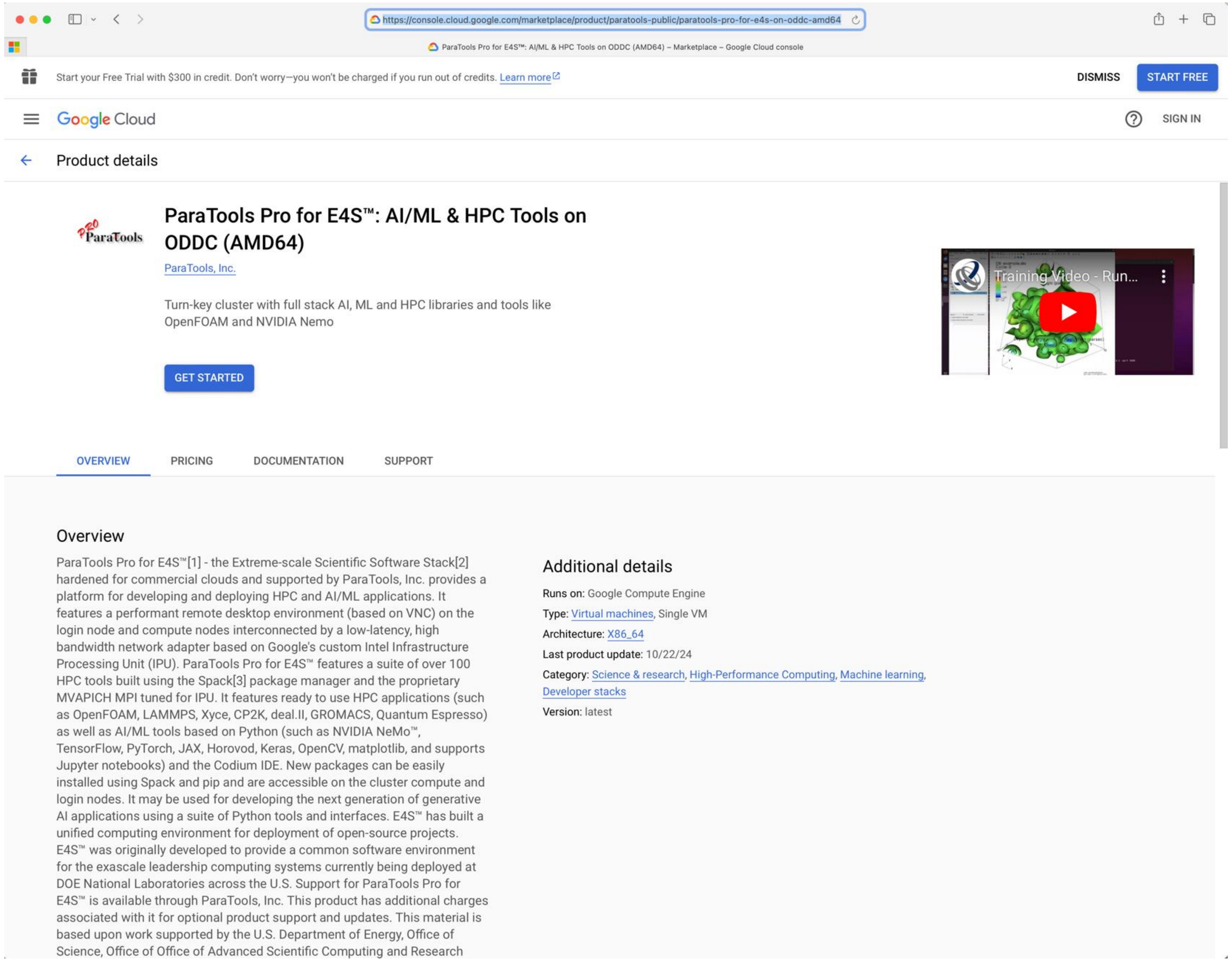
ParaTools

PESO

Choosing an instance on AWS to run the image



ParaTools Pro for E4S™ on Google Cloud Marketplace



https://console.cloud.google.com/marketplace/product/paratools-public/paratools-pro-for-e4s-on-oddc-amd64

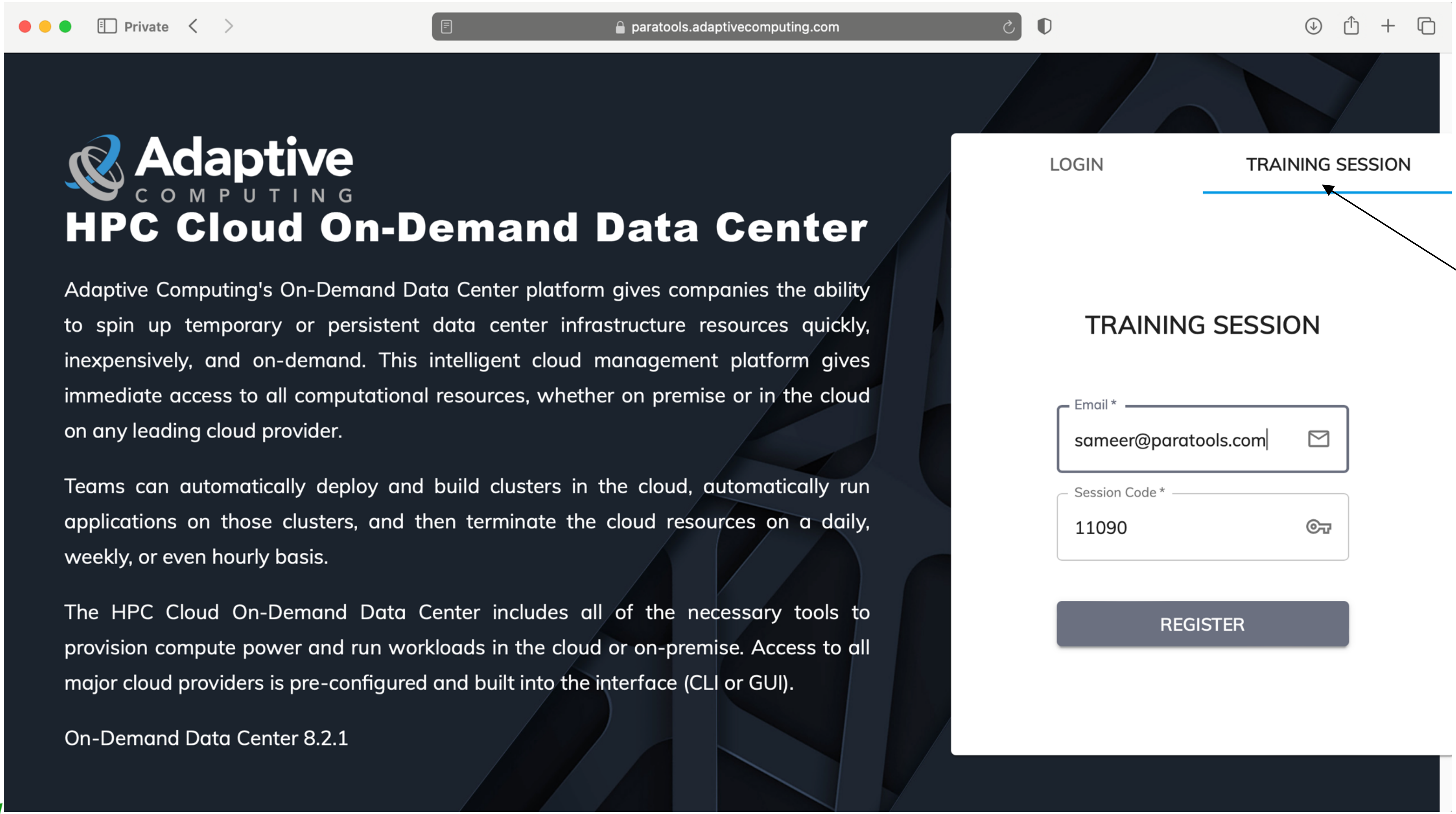
E4S Exercise:

TAU

Using ParaTools Pro for E4S image on AWS with Adaptive Computing's On-Demand Data Center (ODDC)

STEP 1: Click on Training tab at:
https://paratools.adaptivecomputing.com
Firefox recommended.

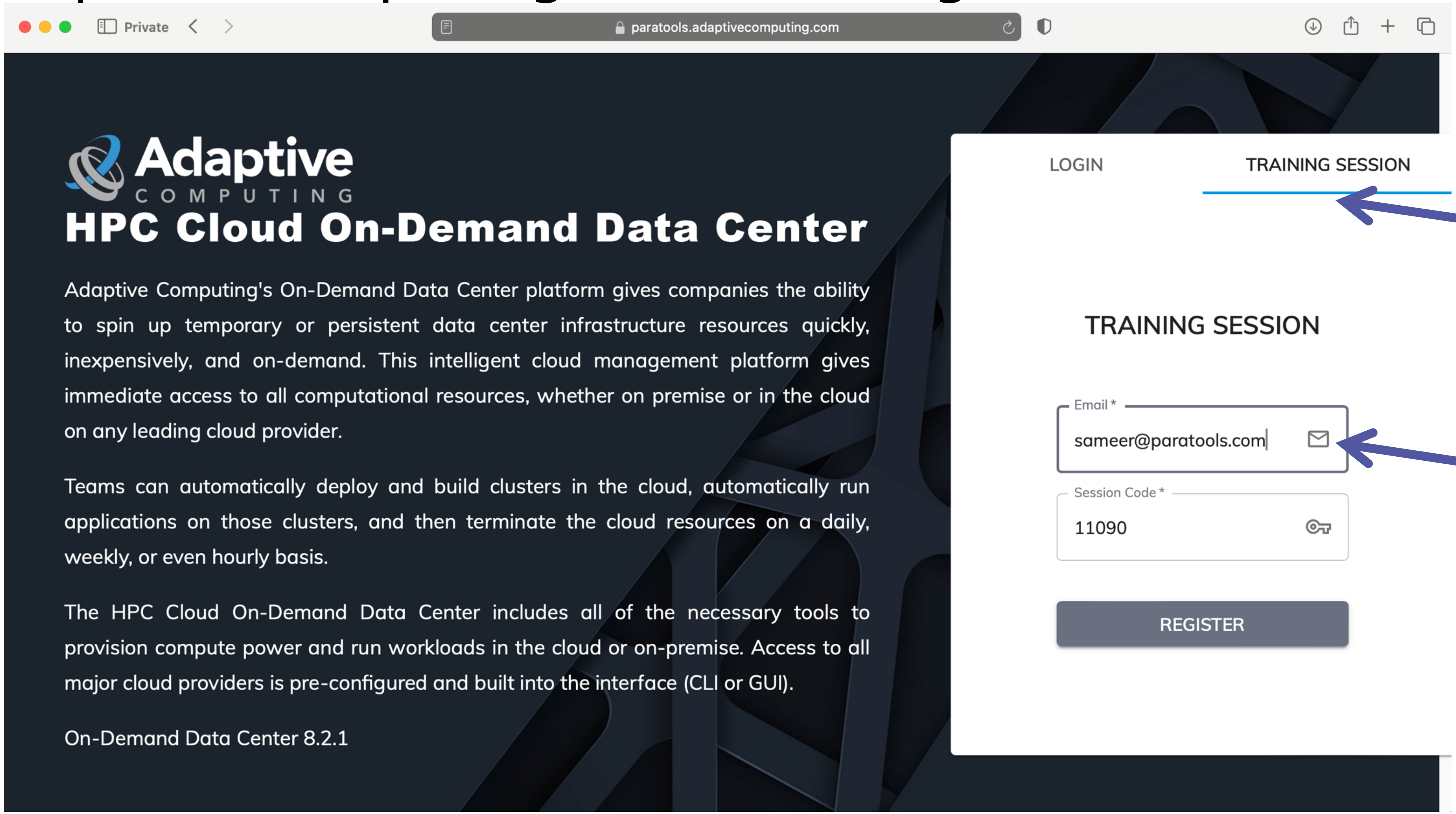
Adaptive Computing's ODDC: Go to Training session tab and enter email and 11090 as session id



Click here

https://paratools.adaptivecomputing.com

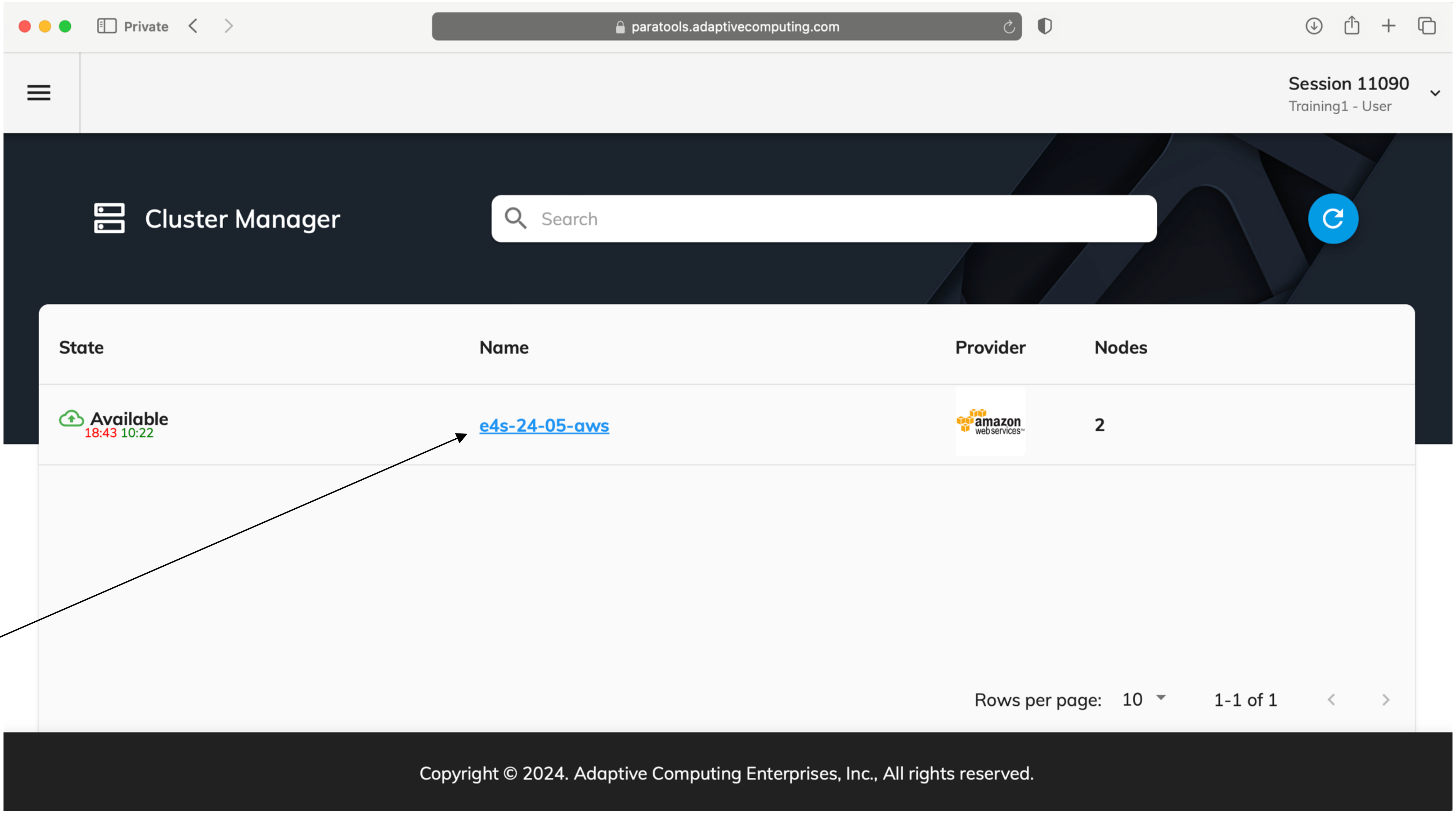
Adaptive Computing’s ODDC: Login with session code



Click on Training Session

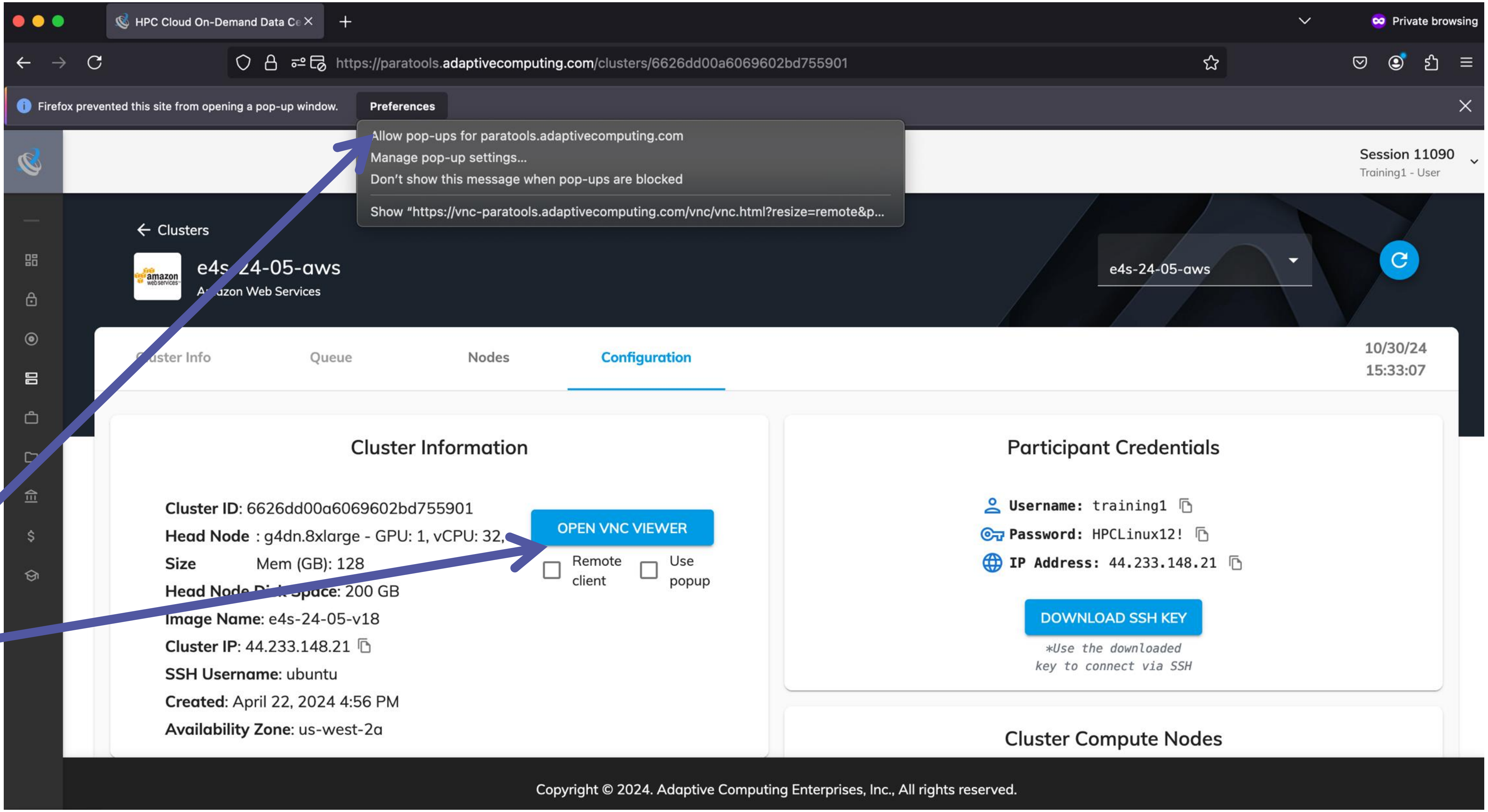
Enter email and session Code 11090

Launch VNC Viewer from ODDC's Configuration Tab



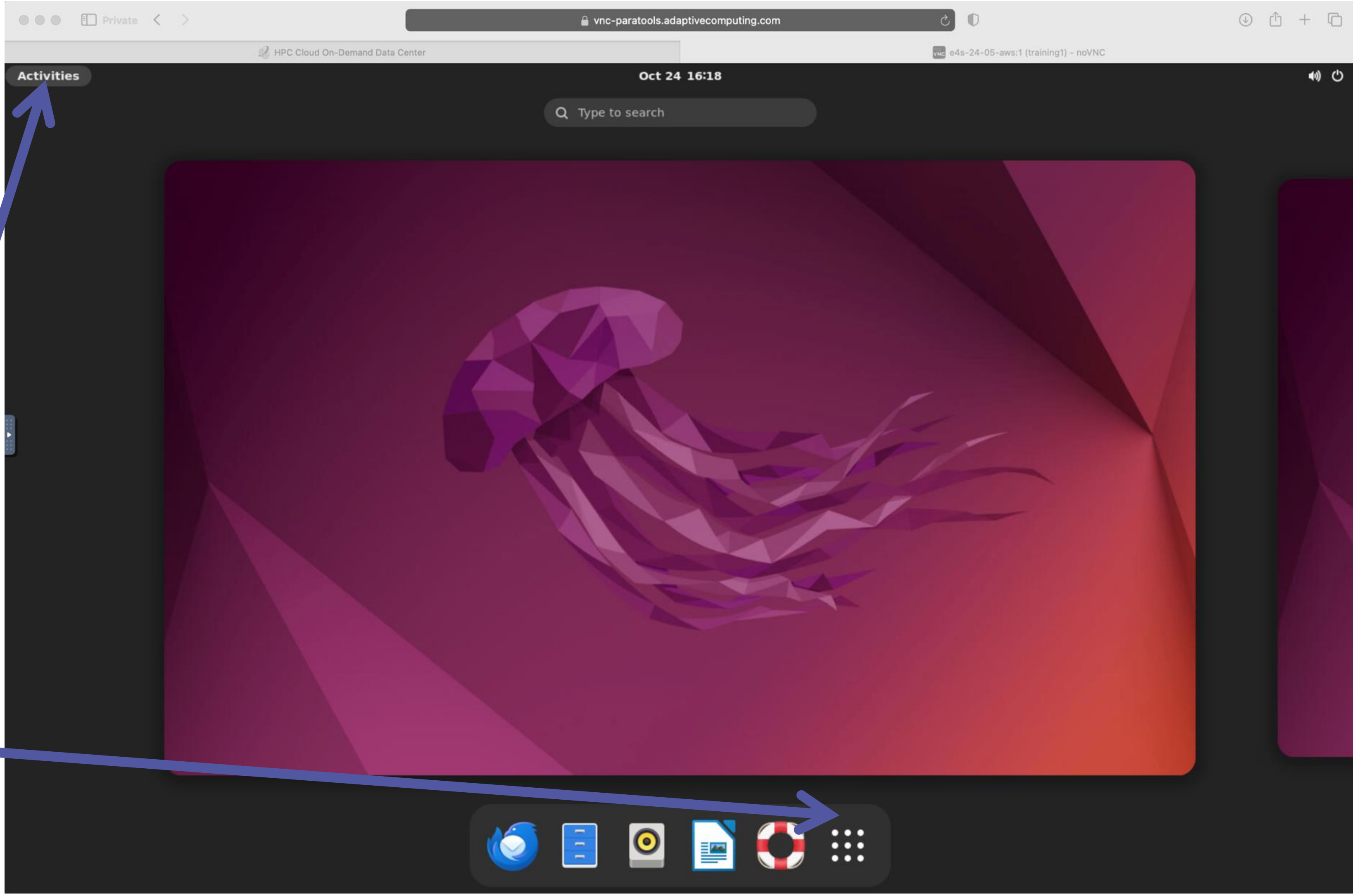
Click here

Launch VNC Viewer from ODDC and allow popups



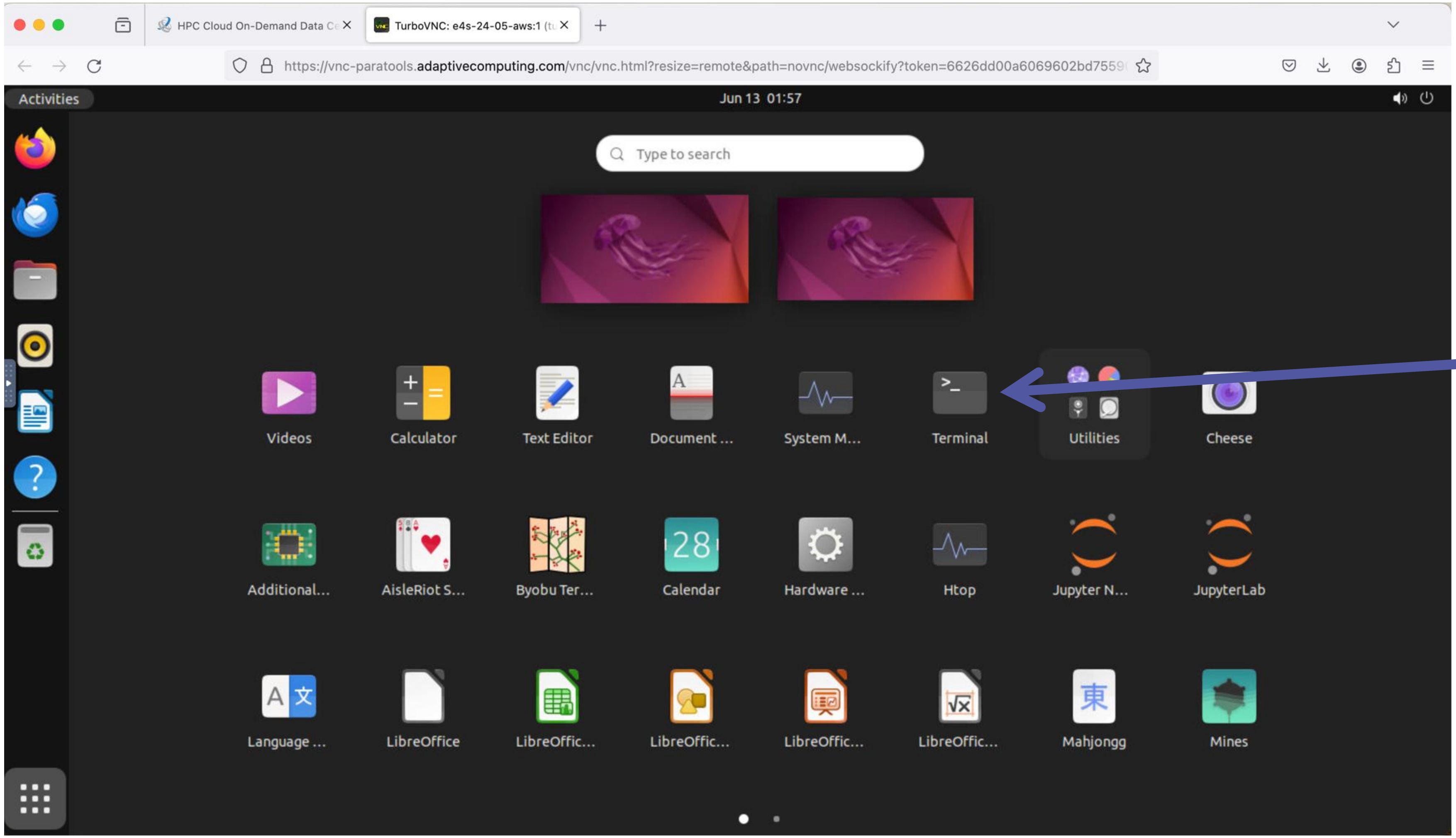
Click here

Remote Desktop from the ParaTools Pro for E4S™ image on AWS and GCP



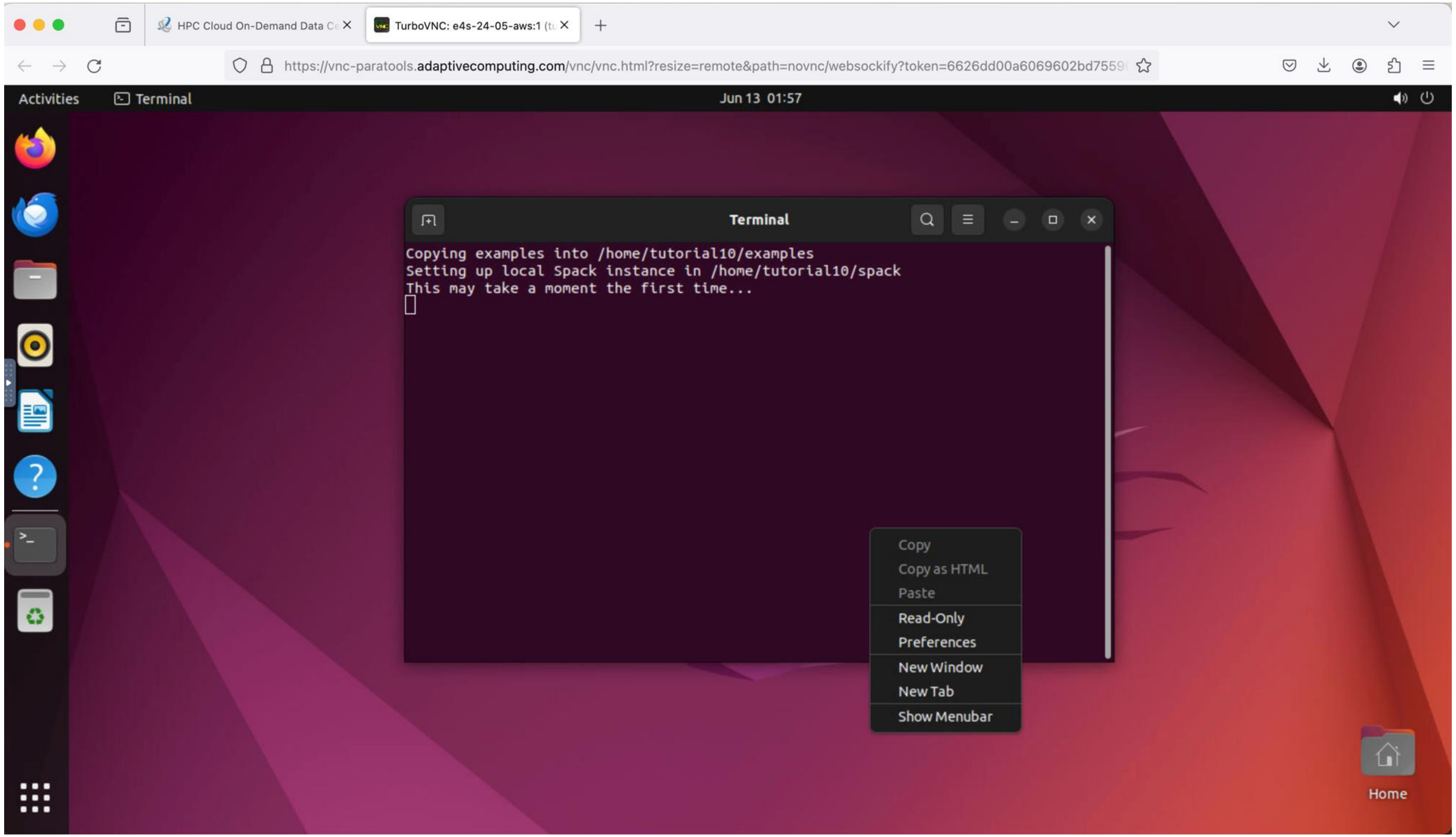
Click here

Launch Terminal in Remote Desktop

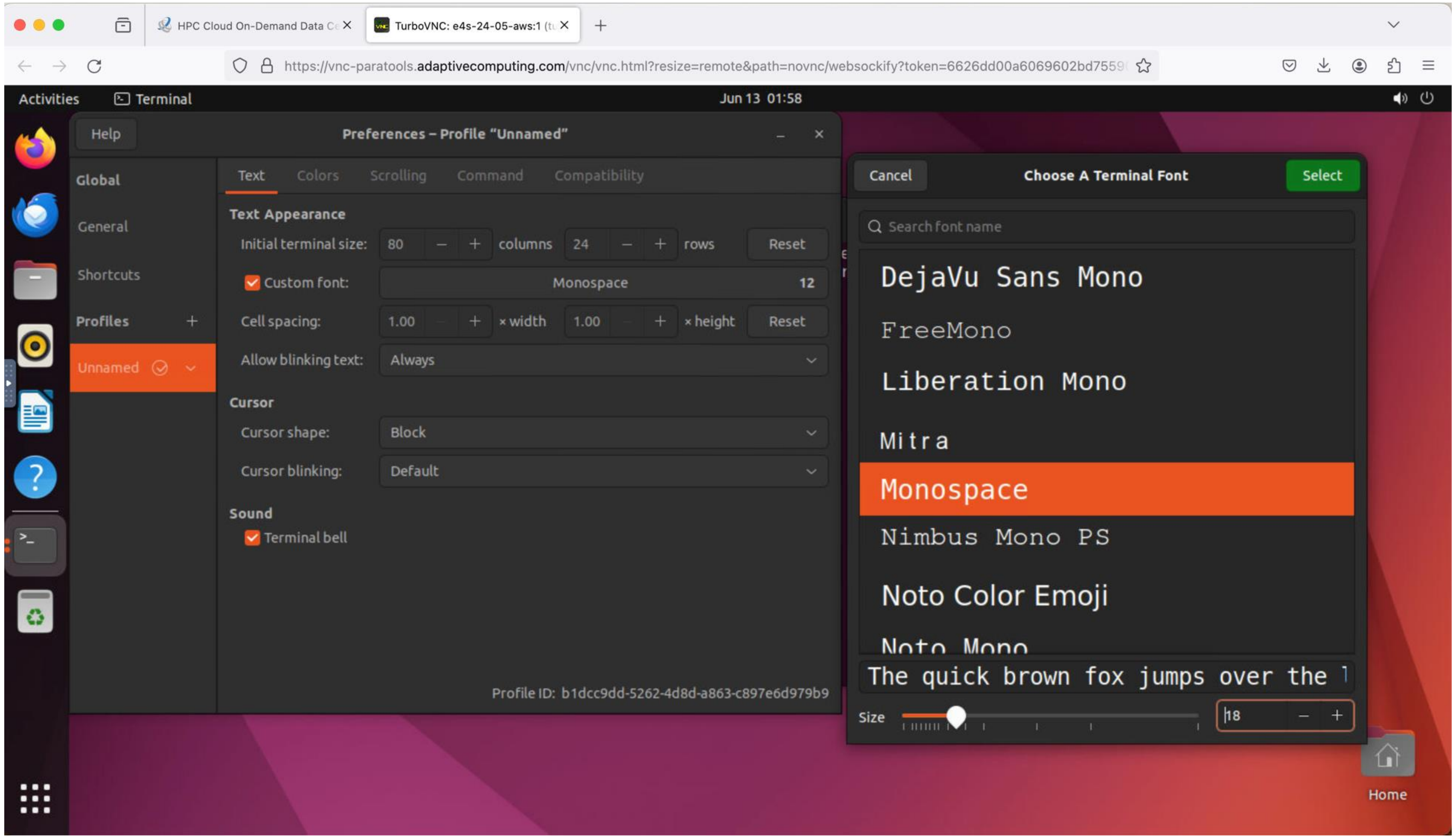


Click here

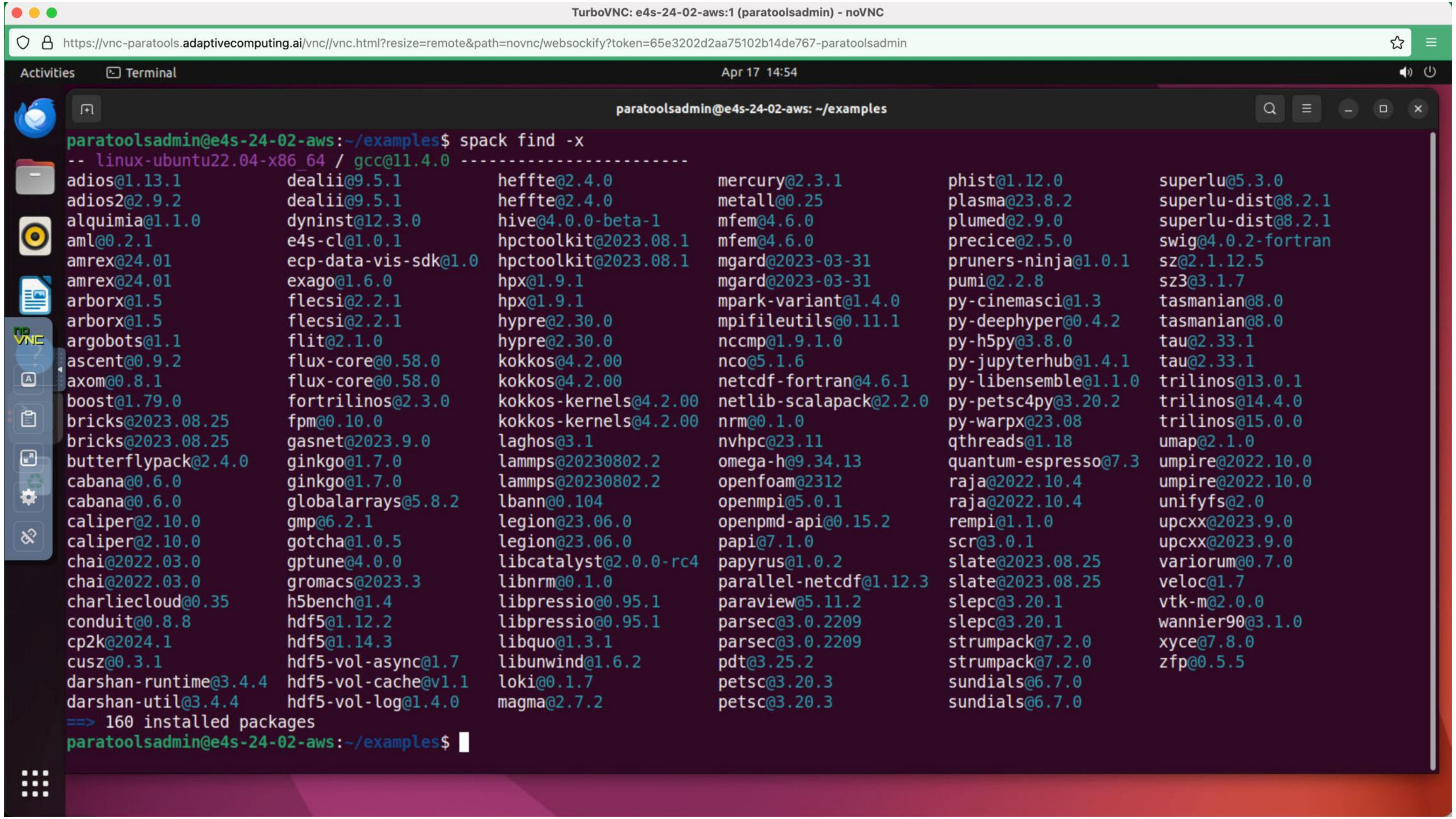
To increase font size right click and choose preferences



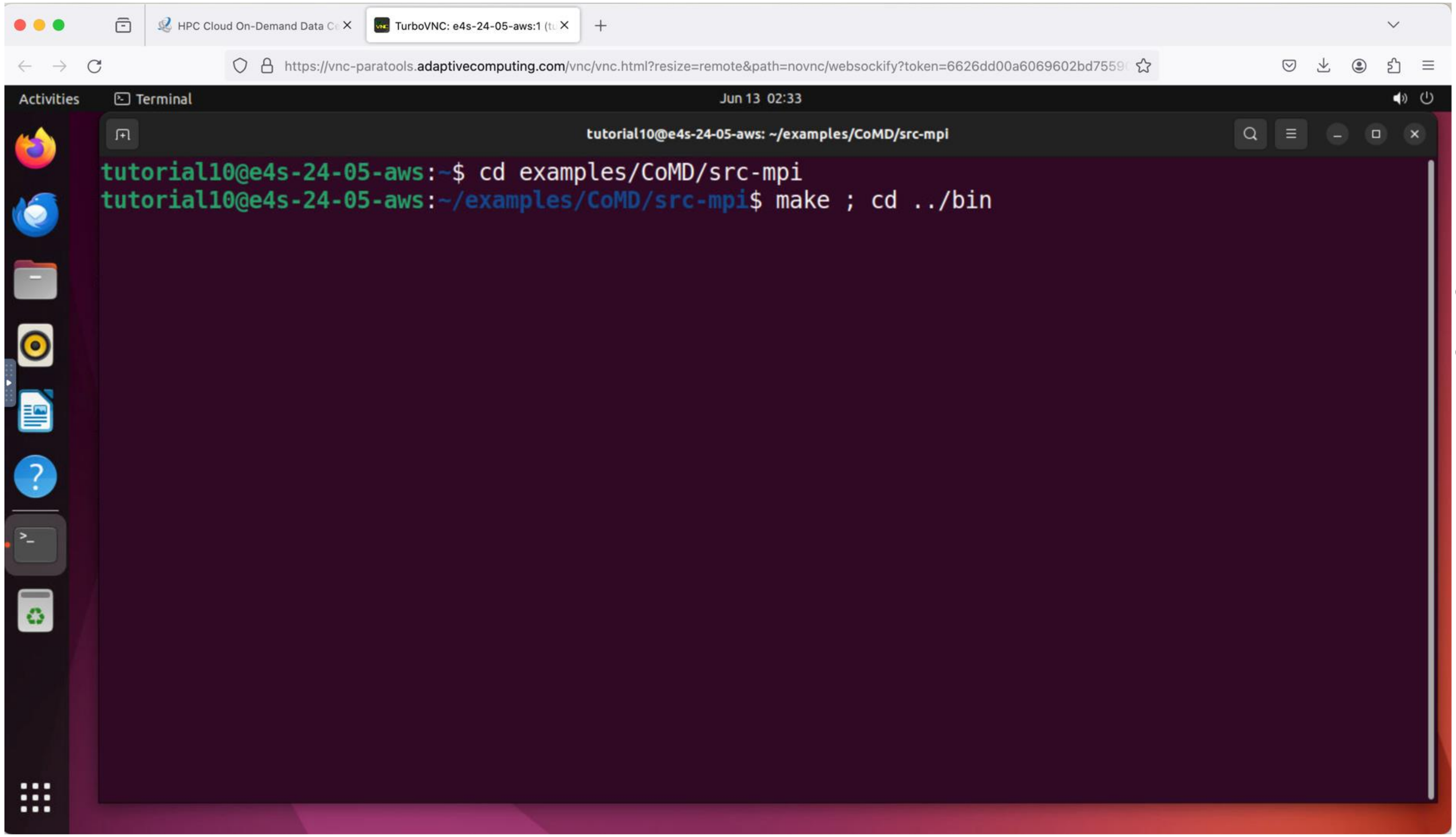
Choose font size after clicking Custom Font for Terminal



Spack Package Manager



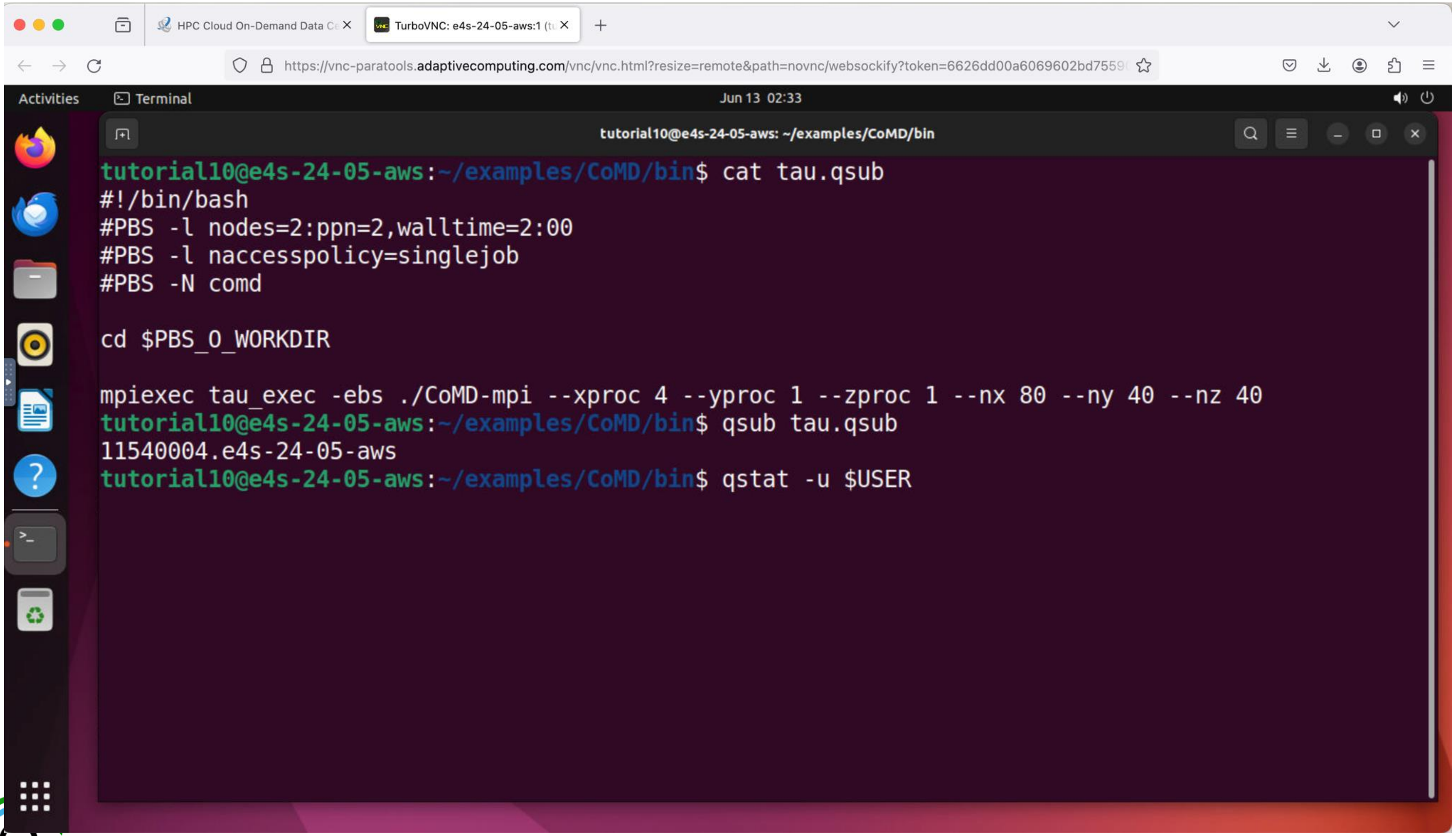
CoMD: TAU with event-based sampling (EBS)



% cd examples/CoMD/src-mpi

% make; cd ../bin

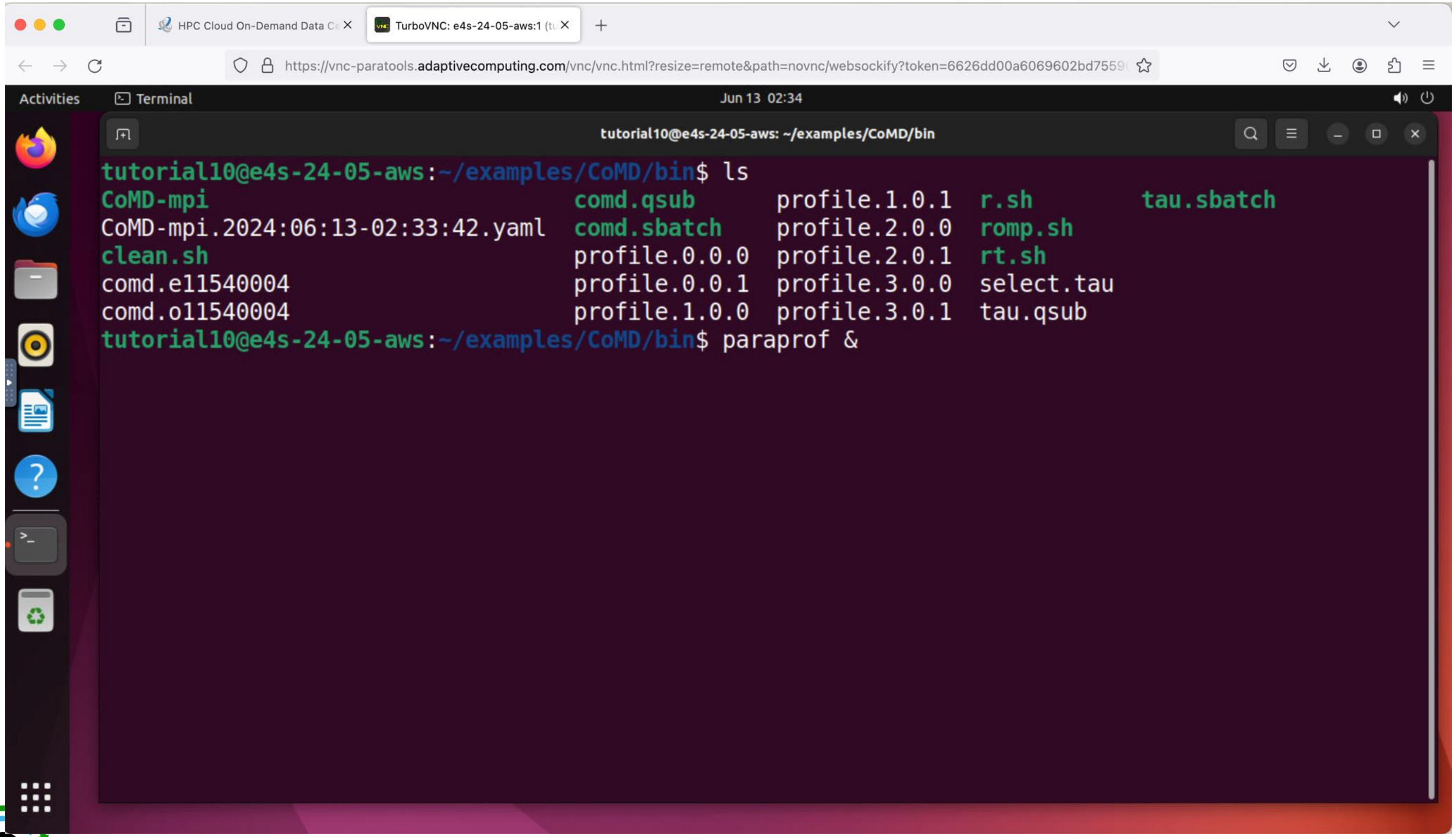
CoMD: TAU with event-based sampling (EBS)



% qsub tau.qsub

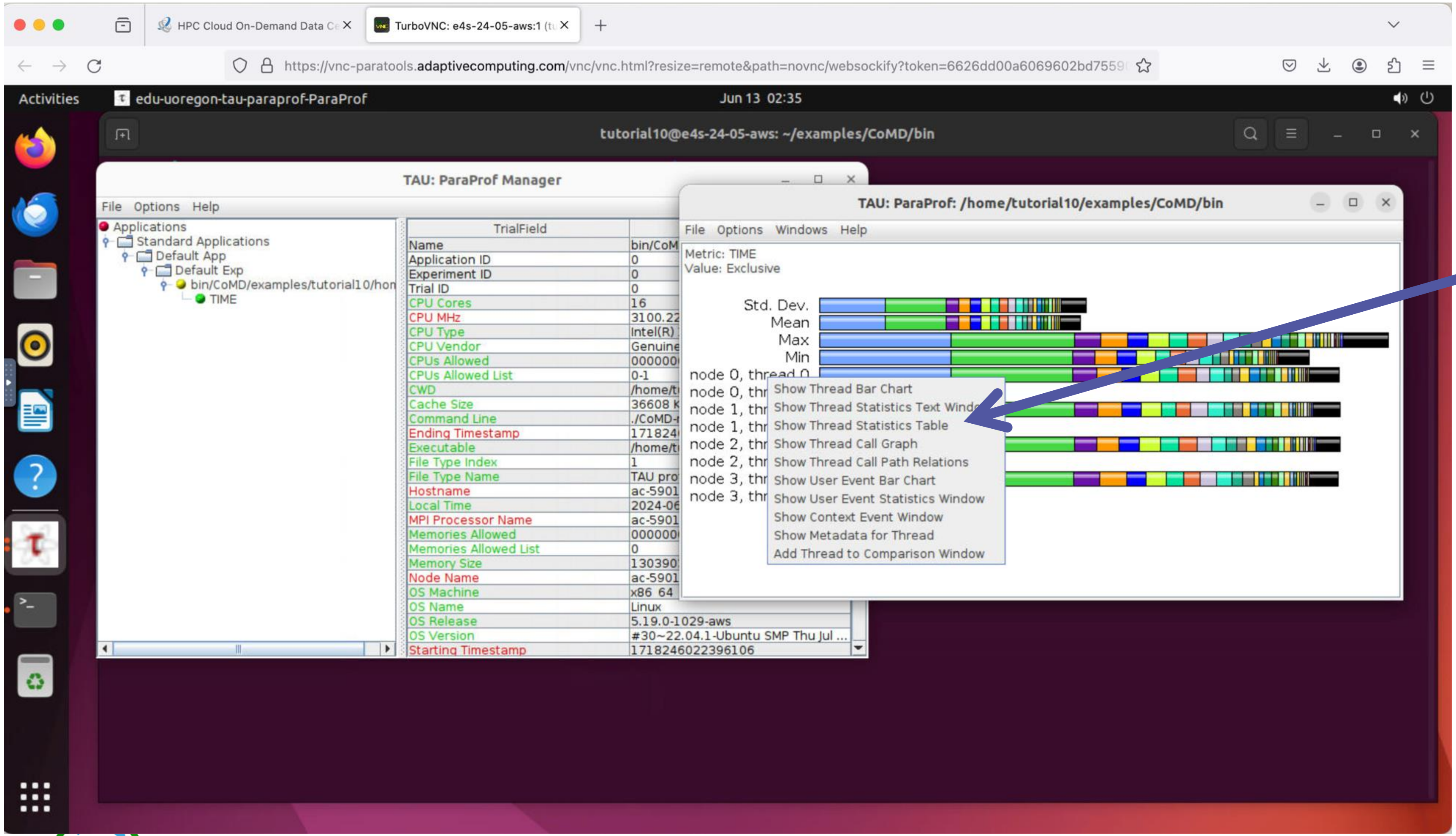
% qstat –u $USER

CoMD: TAU’s paraprof visualizer



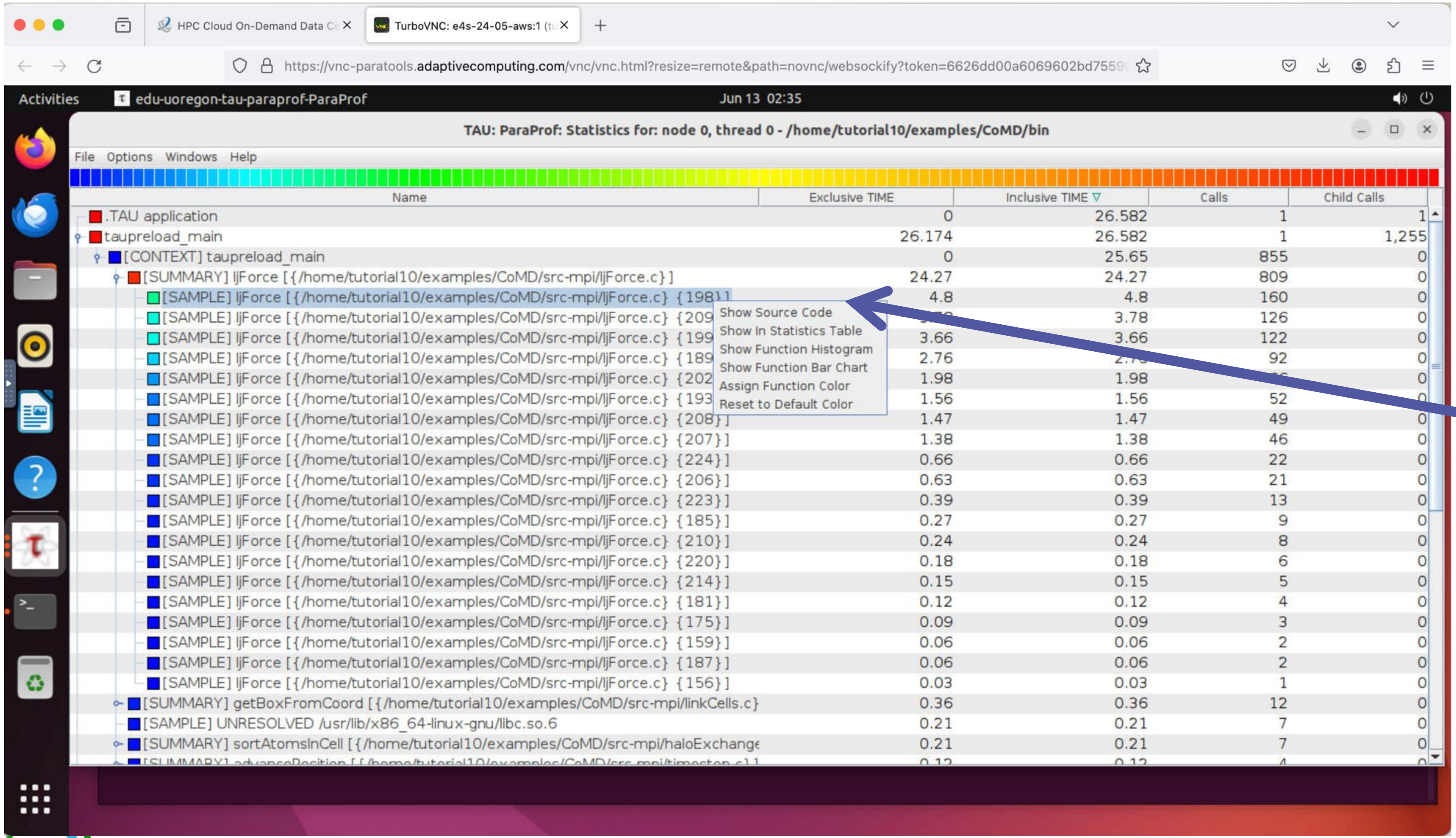
% paraprof &

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

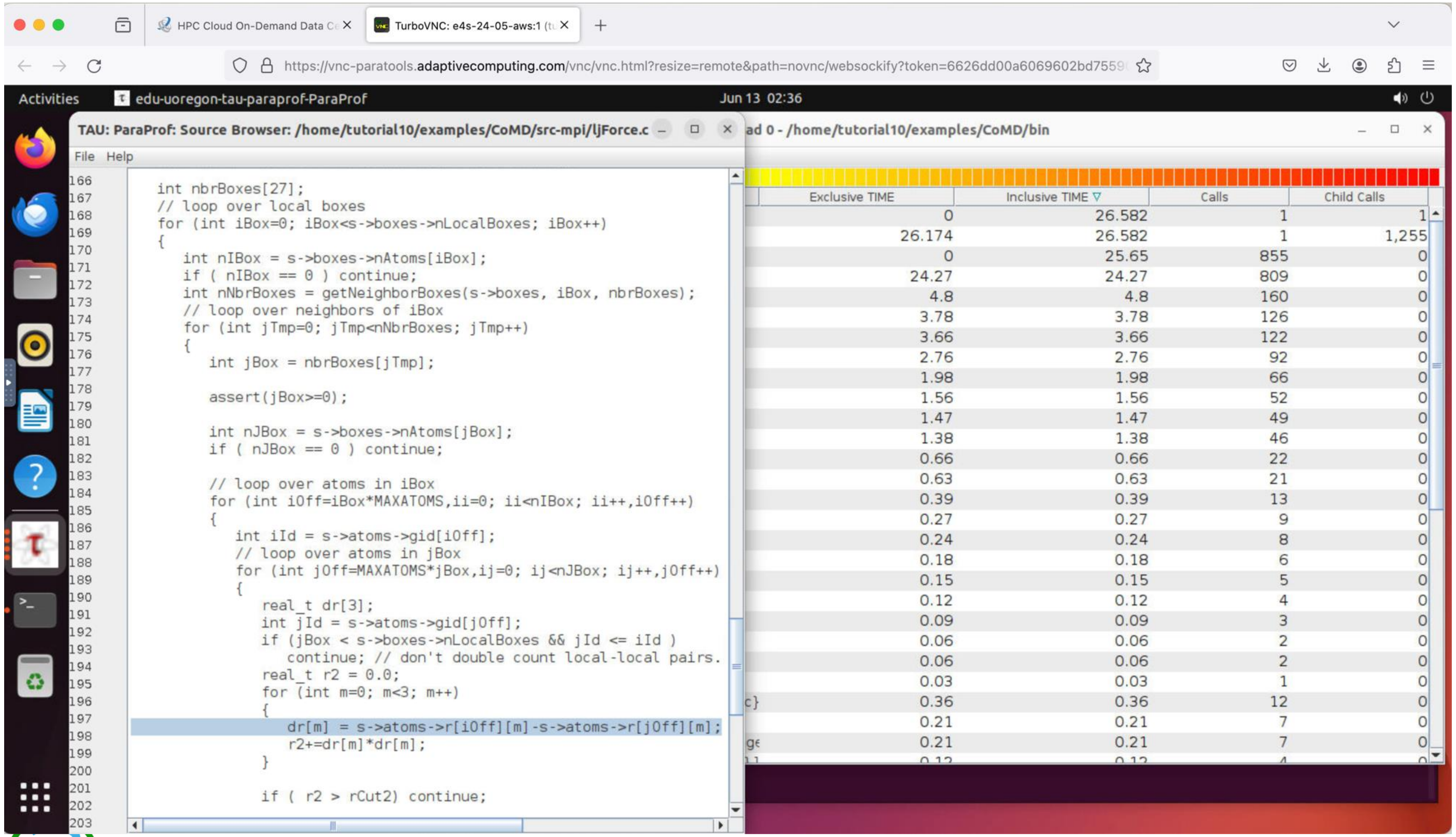


Click on columns to sort (e.g., Inclusive)

Expand nodes and right click on a sample and

Select “Show Source Code”

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!

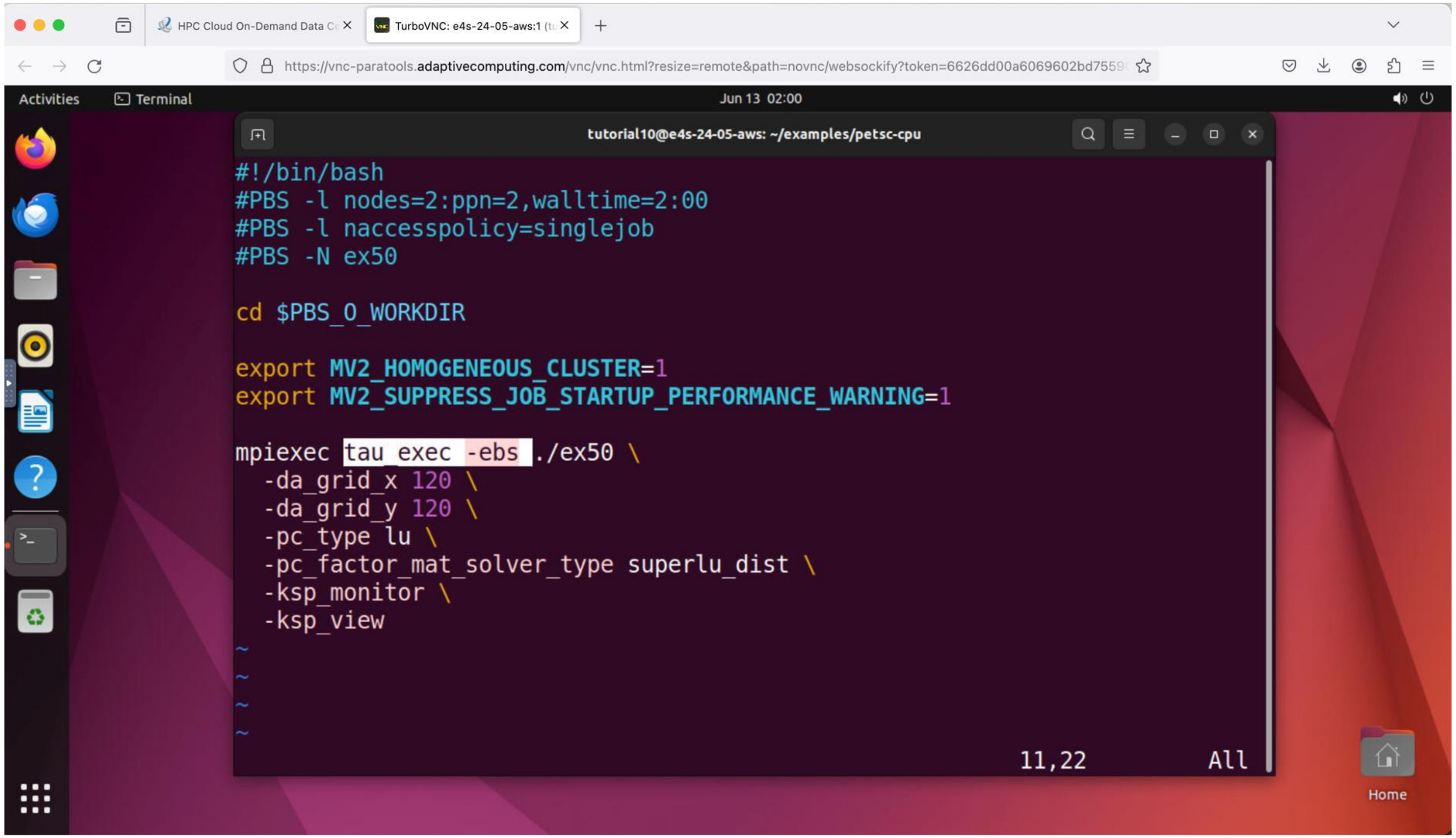
PESO

TAU Exercise #2:

Instrumenting PETSc application using TAU’s Perfstubs interface

PESO

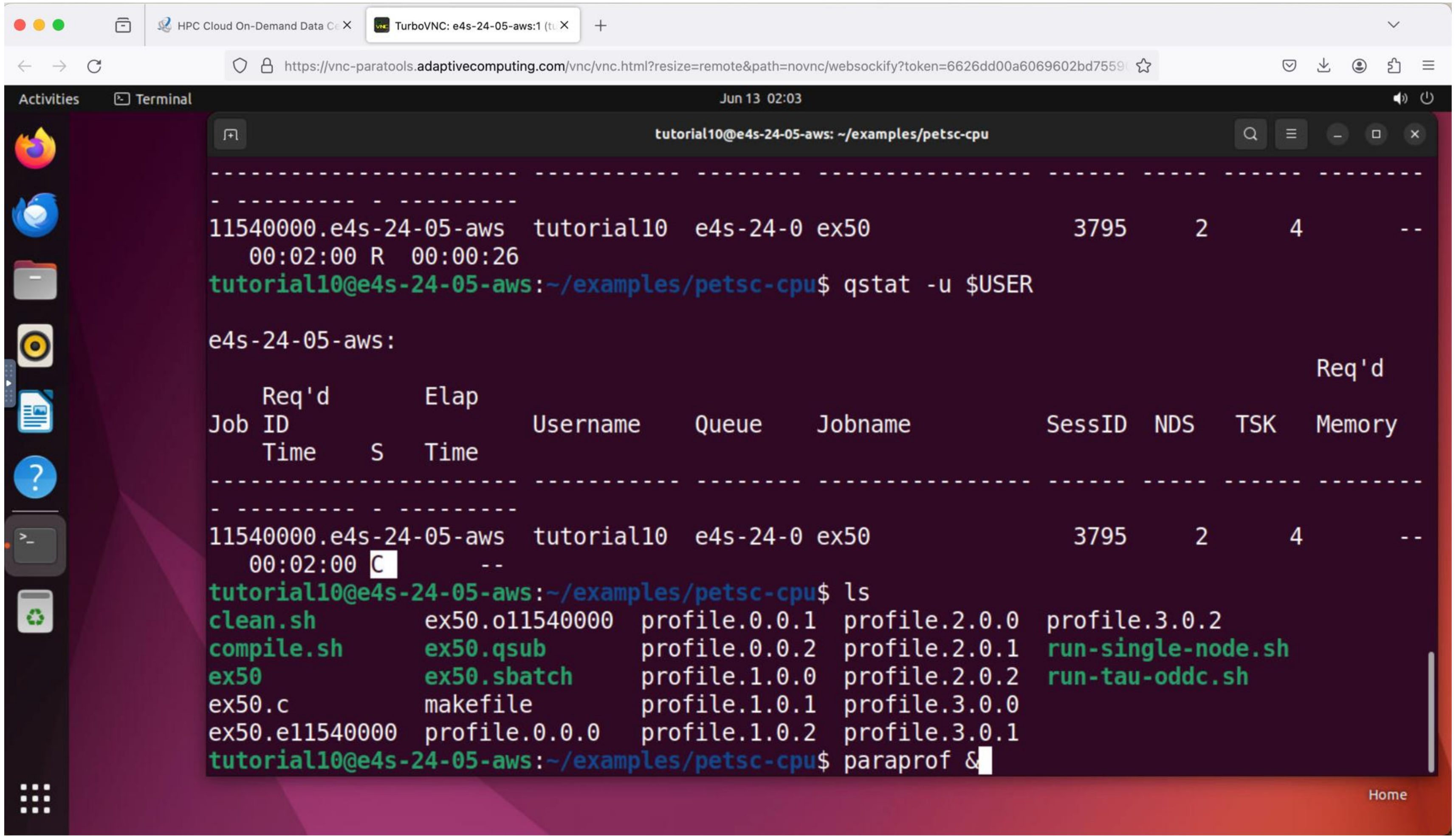
Launching the binary using tau_exec –ebs



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

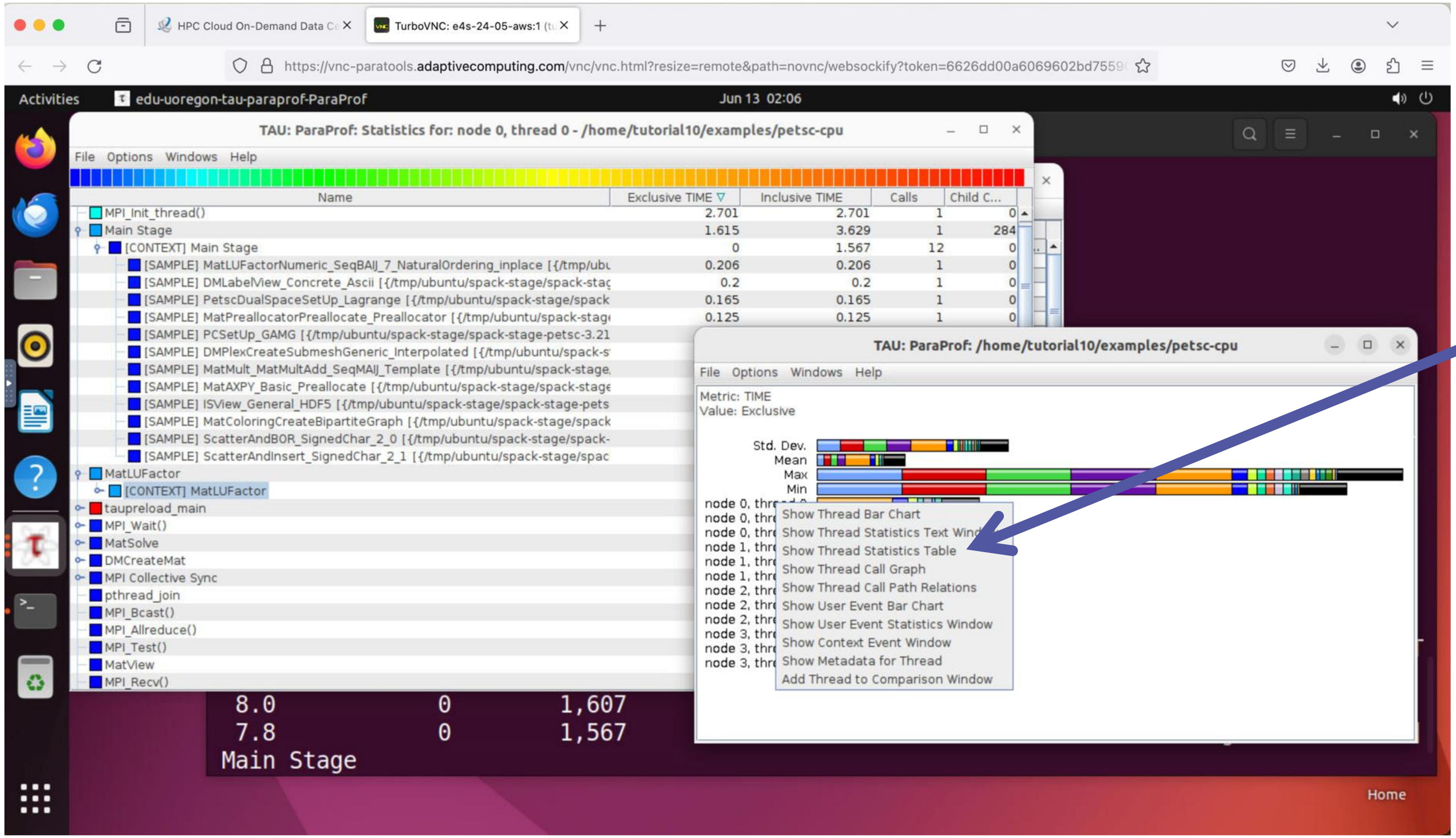
Add tau_exec –ebs before ./ex50 in the launch command. Save the file.

TAU’s ParaProf Profile Browser: Source Code Browser




```
qsub ex50.qsub
qstat –u $USER

# After it completes
ls
paraprof &
```

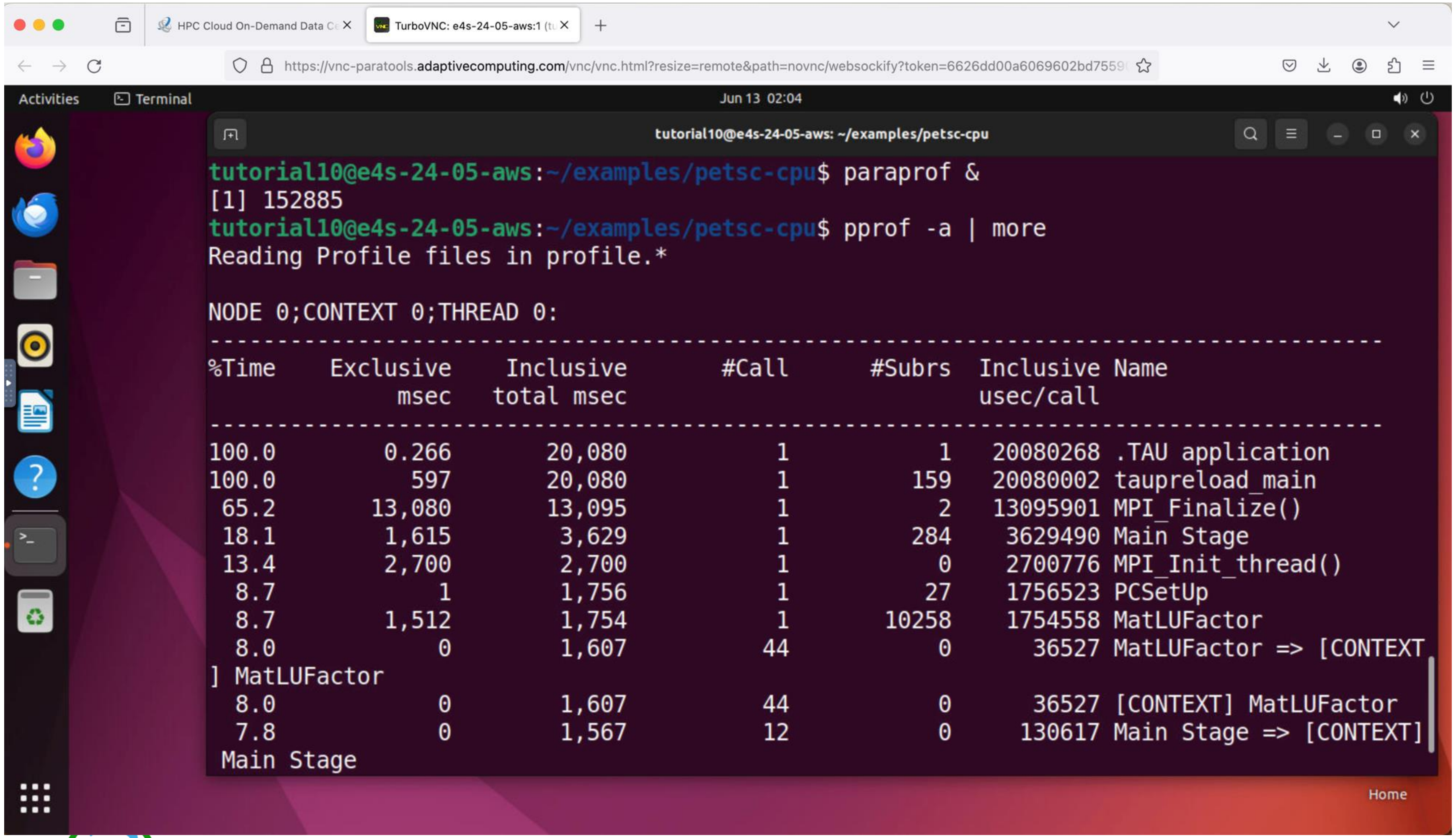
TAU’s paraprof browser with PETSc performance profile



paraprof

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

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!

PESO

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
```

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
```

PESO

Using TAU on Polaris at ALCF

```
% qsub -I -l select=1 -l filesystems=home:eagle -l walltime=1:00:00 -q R2035675 -A ATPESC2024 -X

% module use /soft/modulefiles; module load tau

% wget http://tau.uoregon.edu/workshop_atpesc24.tgz; tar xf workshop_atpesc24.tgz

% cd workshop; cat README; cd TeaLeaf_CUDA; make; cd bin; ./run.sh

% cd ../../petsc-tau; ./clean.sh; ./compile.sh; ./run.sh

% paraprof --pack petsc_ex19.ppk ;  <SCP to AWS>; paraprof petsc_ex19.ppk
```

Un-instrumented run with MPI % aprun –n N ./a.out

Profiling an un-instrumented application (use tau_exec –ebs with any of the following for event-based sampling):

```
MPI without GPUs:          % aprun –n N tau_exec -ebs ./a.out
CUDA with MPI:             % aprun –n N tau_exec –T cupti,mpi –cupti –ebs ./a.out
```

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

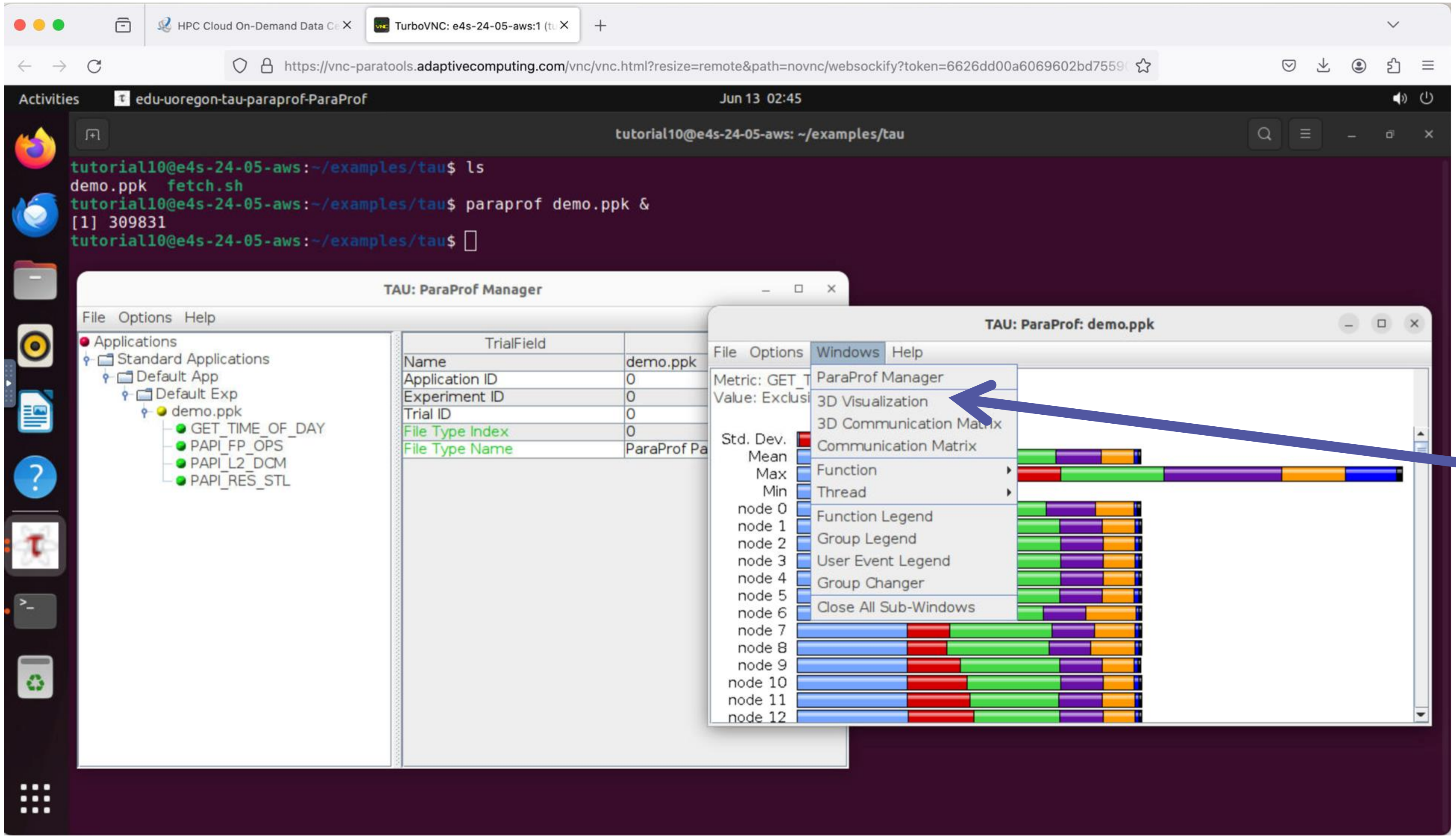
Tracing:

```
Vampir:          % export TAU_TRACE=1; export TAU_TRACE_FORMAT=otf2
                 % aprun –n N tau_exec [options] ./a.out; vampir traces.otf2 &
Chrome:          % export TAU_TRACE=1; aprun –n N 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 https://Perfetto.dev
Jumpshot:        % export TAU_TRACE=1; aprun –n N tau_exec [Options]./a.out;
                 % tau_treemerge.pl; tau2slog2 tau.trc tau.edf –o app.slog2; jumpshot app.slog2
```

TAU Exercise #3:

paraprof 3D display

TAU paraprof

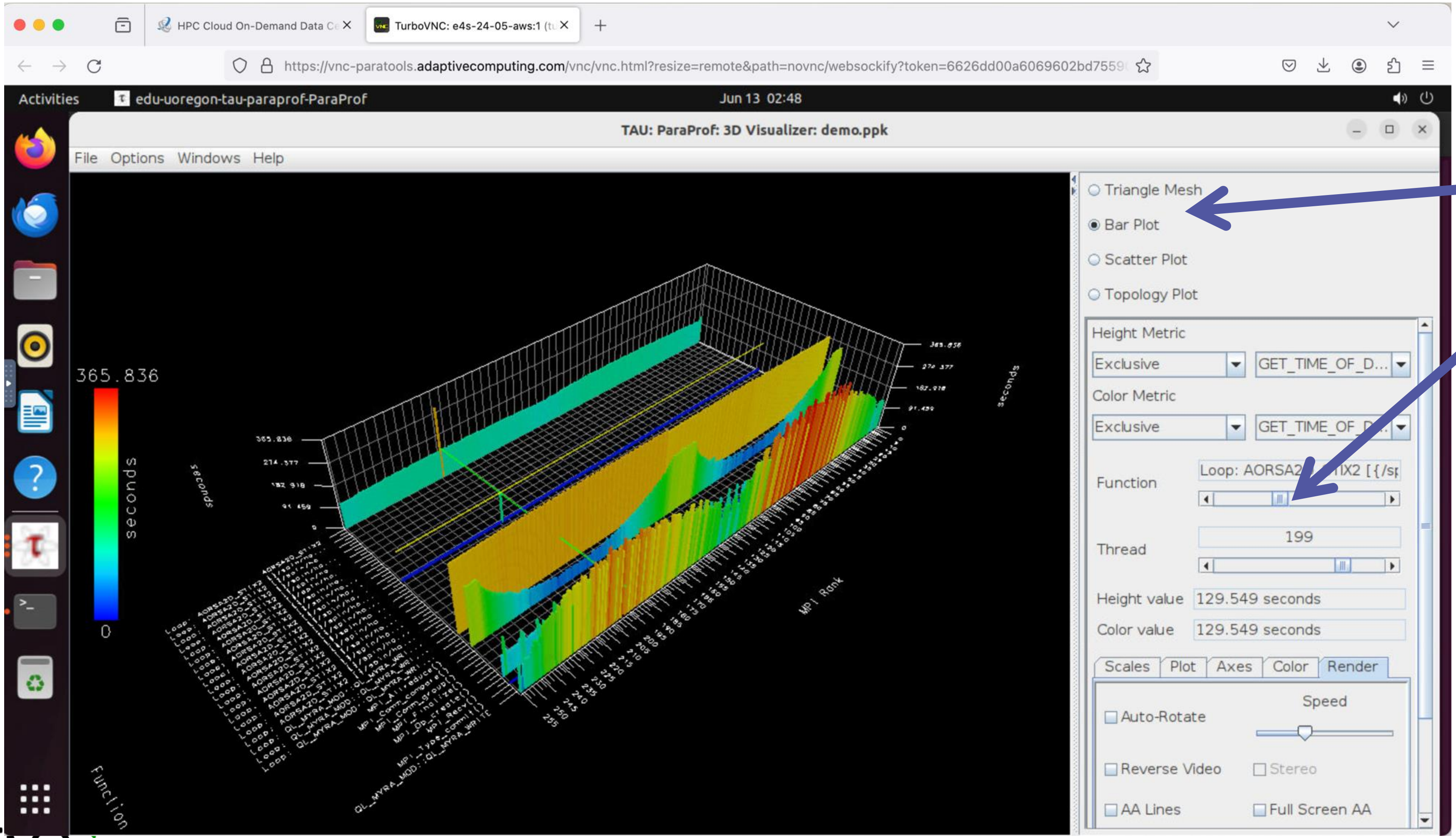


cd ~/examples/tau

paraprof demo.ppk &

Choose 3D Visualization

TAU paraprof 3D visualization



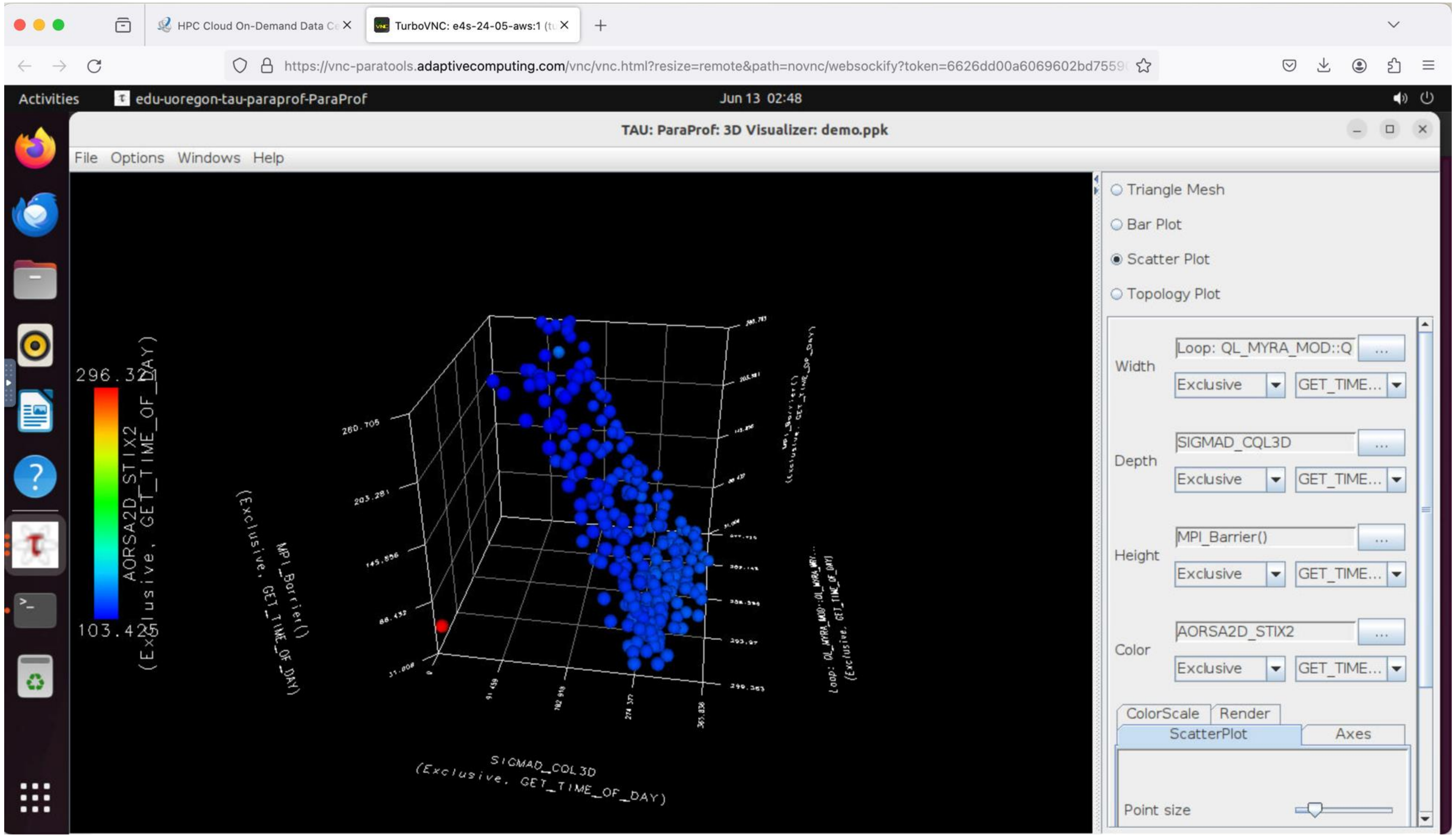
Choose Bar Plot and move

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

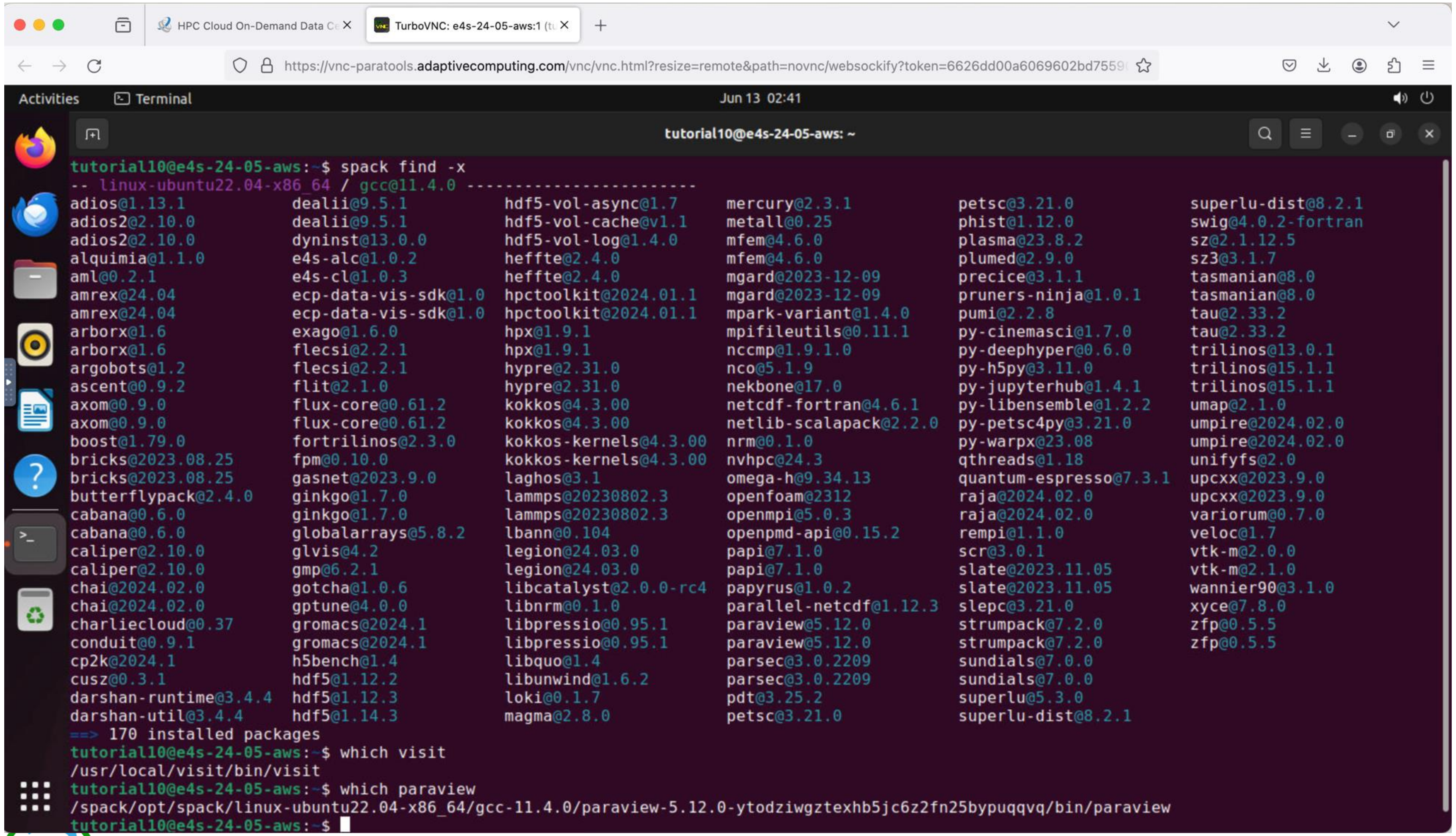
TAU paraprof: 3D Scatter Plot



Exercise #7

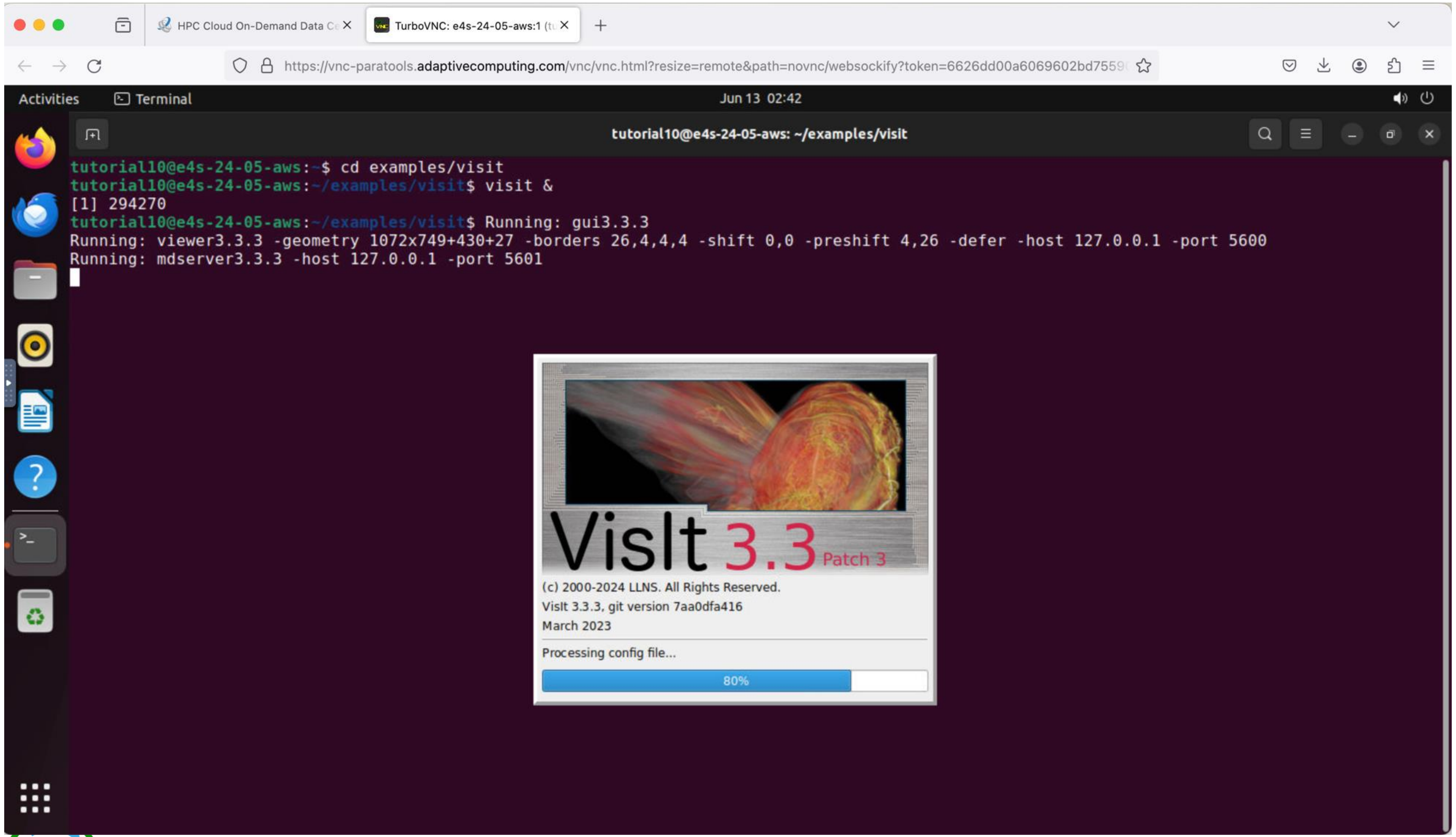
E4S: Extreme-scale Scientific Software Stack

Spack package manager [https://spack.io]



spack find –x
spack find

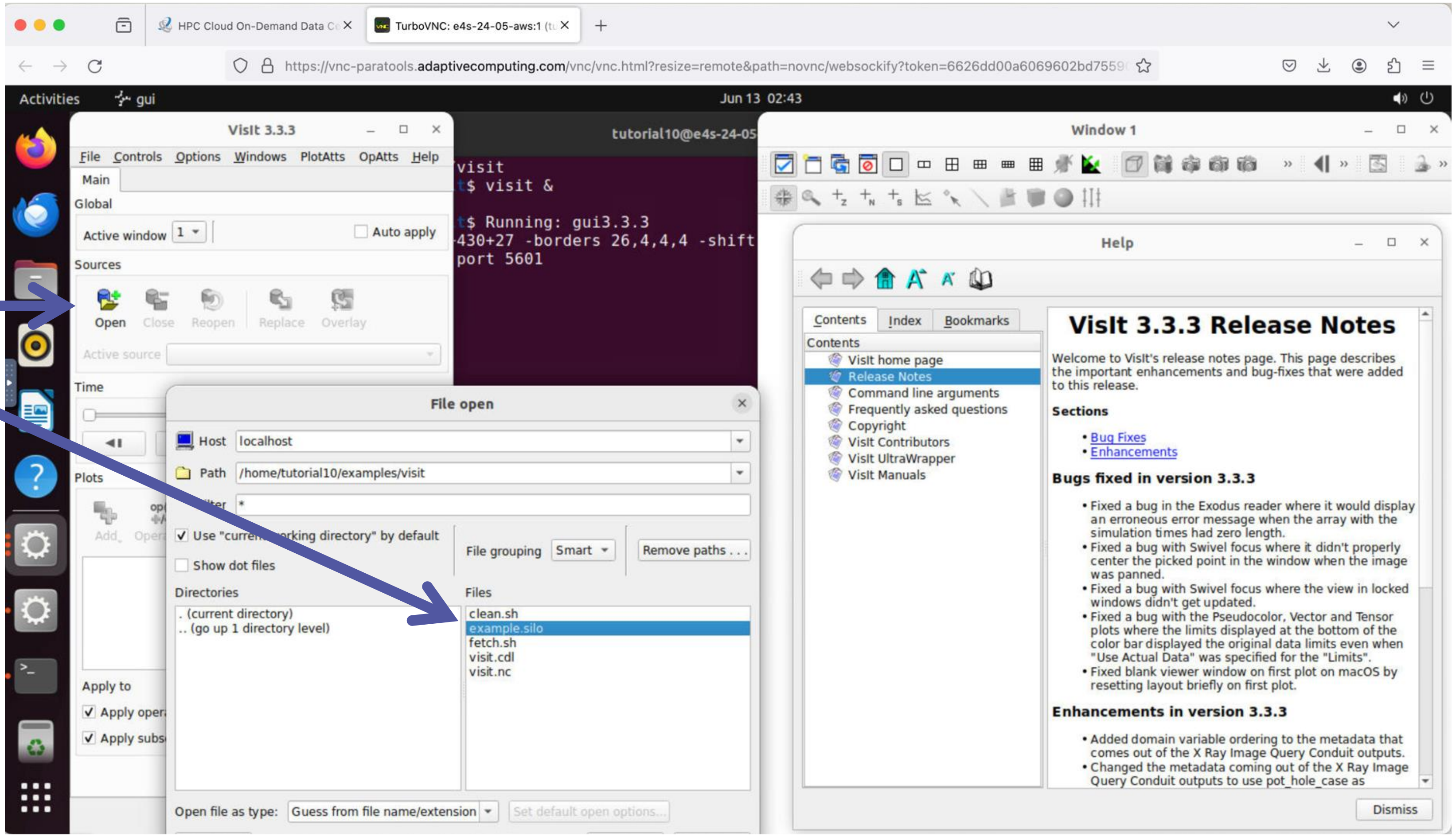
VisIt visualizer



cd ~/examples/visit
visit &

VisIt visualizer

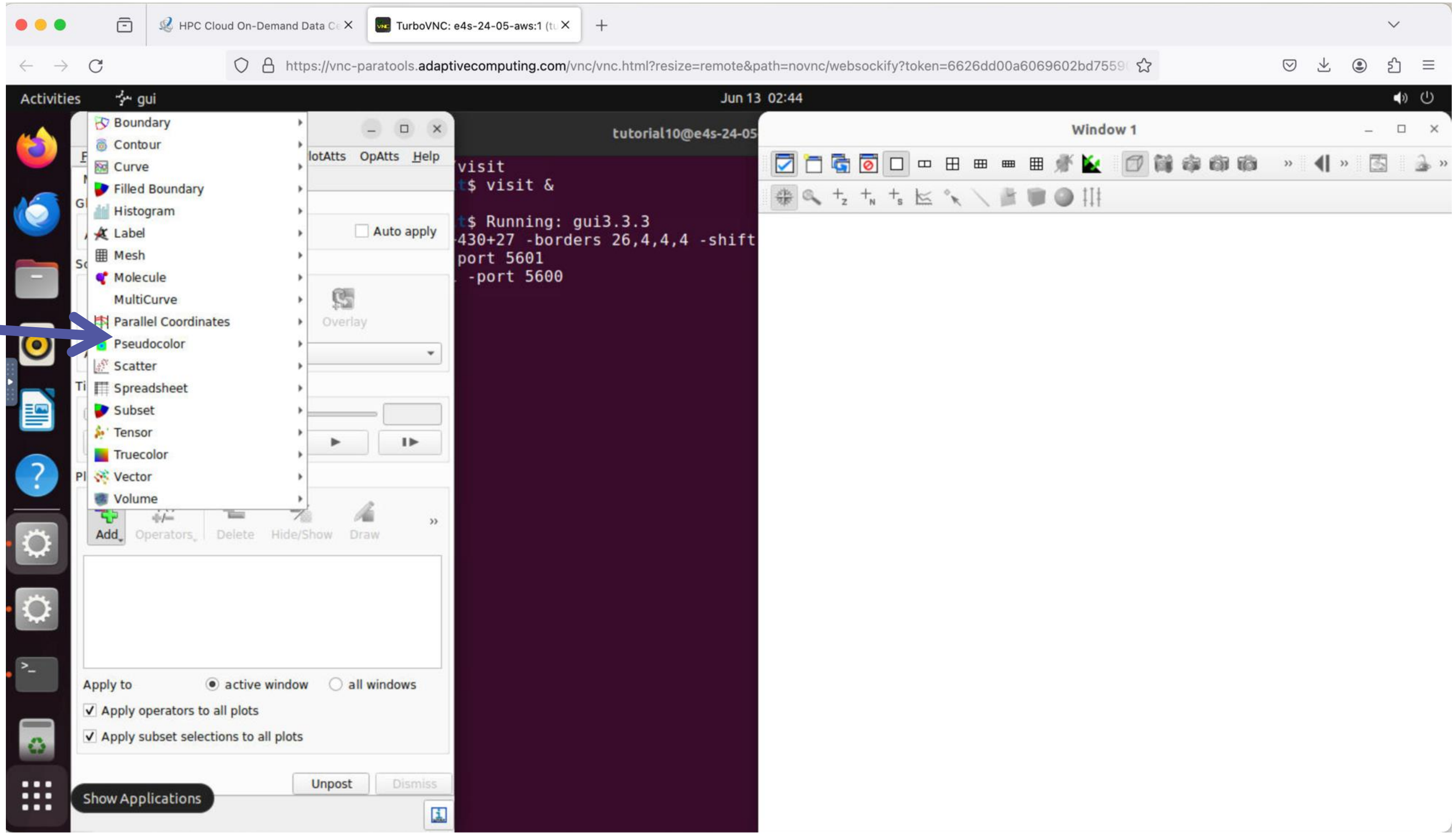
Open example.silo



PESO

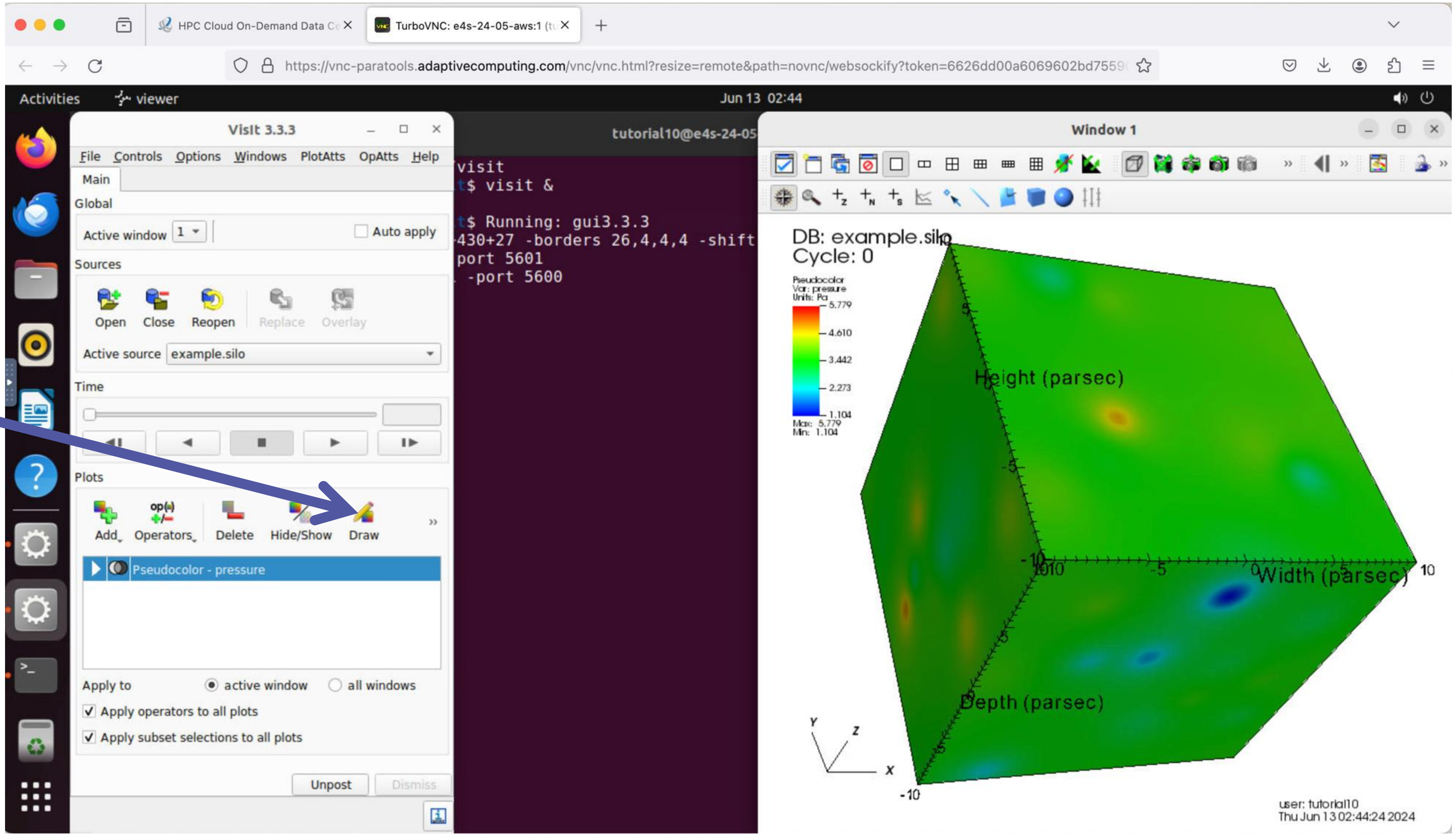
VisIt visualizer

Add Pseudocolor -> Pressure



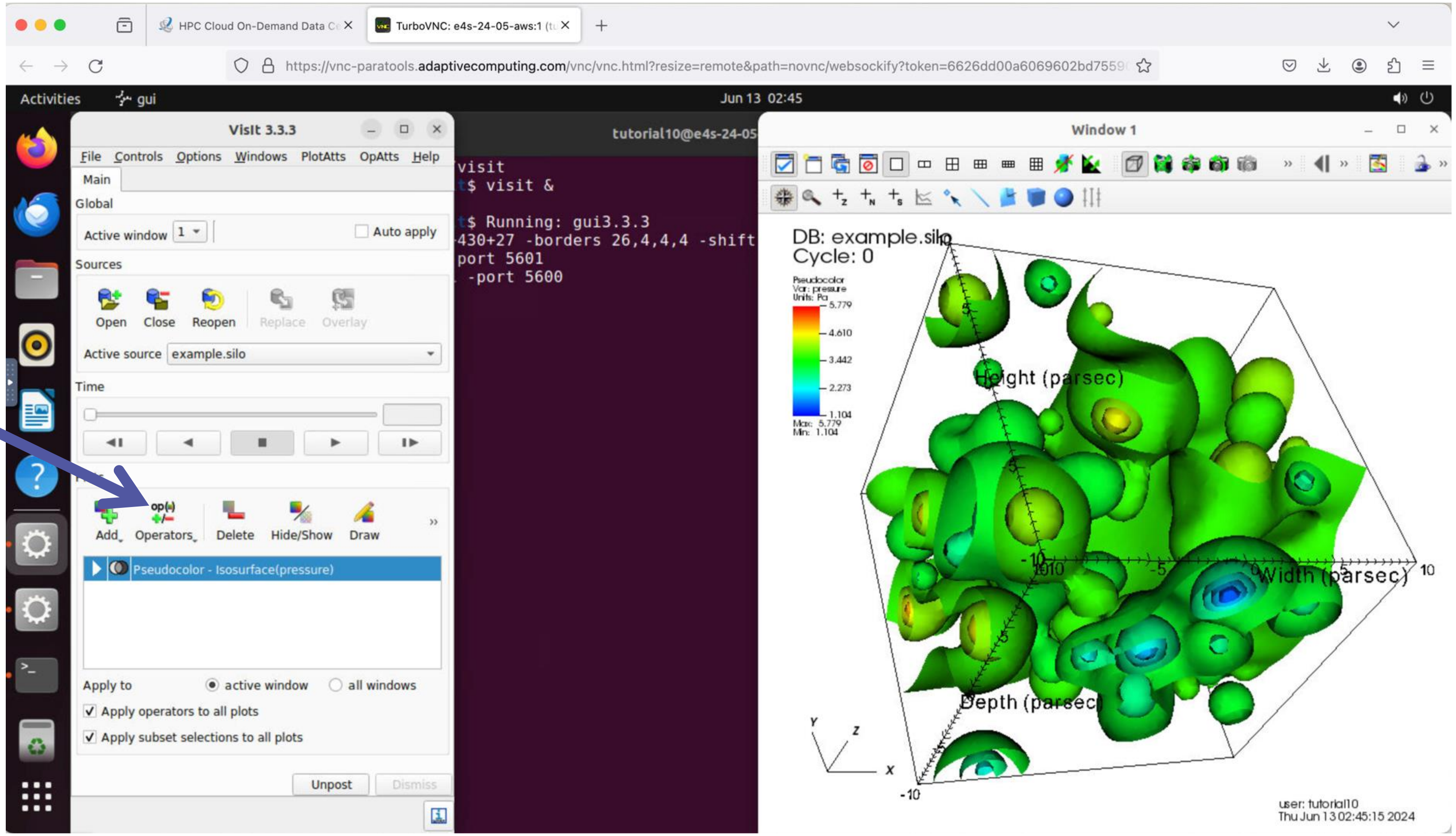
VisIt visualizer

Add Pseudocolor -> Pressure
Click Draw
Rotate image



VisIt visualizer

Add Operators ->
Isosurface
Click Draw
Rotate image



Reference

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="" $(TAU_DEFS)	Options passed to the compiler. Typically $(TAU_MPI_INCLUDE) $(TAU_INCLUDE)
-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 (default)	Use TAU's shared library (libTAU.so) instead of static library
-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>)

PESO

Runtime Environment Variables

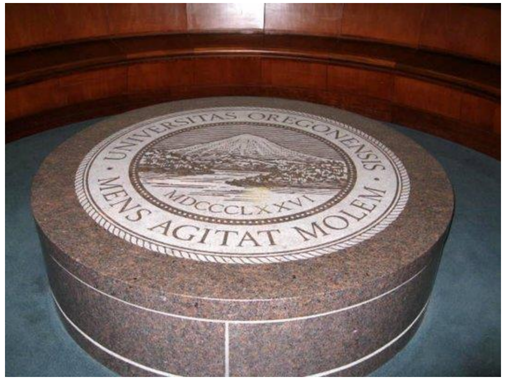
Environment Variable	Default	Description
TAU_TRACE	0	Setting to 1 turns on tracing
TAU_TRACE_FORMAT	Default	Setting to “otf2” turns on TAU’s native OTF2 trace generation (configure with –otf=download)
TAU_EBS_UNWIND	0	Setting to 1 turns on unwinding the callstack during sampling (use with tau_exec –ebs 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

PESO

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

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.

ECP

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.

PESO

Acknowledgment

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research (ASCR).

U.S. DEPARTMENT OF ENERGY | Office of Science

https://science.osti.gov/ascr
https://pesoproject.org
https://ascr-step.org

PESO

STEP SOFTWARE TOOLS ECOSYSTEM PROJECT

UNIVERSITY OF OREGON