Tau Commander: Introductory Guide

D. Mackay, ParaTools, Inc.

Contents

Introduction:

TAU Commander is a powerful product to manage performance analysis activities. Software Developers can use it to analyze software performance and determine how to optimize their software or the compute environment the software runs on. With rapid movement forward with different types of compute environments it is important for the software developer to understand how to effectively utilize the computer environment, which may include multi-core or many core, SIMD execution units, and co-processors and gpu based compute accelerators. Tau commander offers the capability to analyze MPI*, C*, C++*, Fortran*, Python*, OpenMP*, OpenCL*, CUDA, SHMEM and more. TAU Commander is one of the few products to collect performance analysis data simultaneously on both the Intel Xeon Phi (KNC) and its host simultaneously. TAU Commander operates across numerous hardware platforms, operating systems and software development environments. This guide is an introduction to its layout and how best to use. For those who just want to run a performance profile quickly, the TAU Commander Quick Start Guide is a recommended starting point and will quickly set a up developer to run a hotspots profile analysis. At the time of this writing this guide is not yet posted but expect to find it soon.

Definitions:

The basis for Tau Commander is T-A-M: Target, Application and Measurement. This is illustrated in Figure 1. All activities are associated around these three basic definitions. The first, **target**, describes the environment where data is collected. This includes the platform the code runs on, its operating system, CPU architecture, interconnectivity fabric, compilers, and installed software. The second is the **application**. The application consists of the underlying items associated with the application - whether the application uses MPI, OpenMP, threads, CUDA, OpenCL and such. The **measurements** define

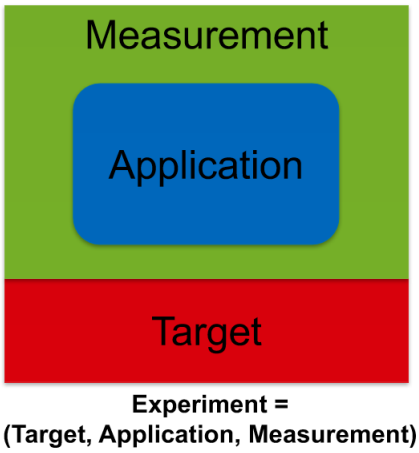


Figure 1 Basic Structure of Tau Commander

what will data will be collected and in what format. Even though an application uses OpenMP or MPI, the measurements may or may not measure those items. The format of the collected information is very important. The two basic formats are a profile or a trace. The data to be collected may include the wallclock time, hardware performance counters, network or file I/O, time spent in OpenMP or MPI run time libraries, etc. The developers choose attributes of the application and attributes of the target to define the measurements. These three components – Target, Application and Measurement – form the basis of the TAU Commander structure. In addition to that basic structure there are a couple of more components to complete the TAU Commander interface. The first is a project. A ***project*** is the container for the developers grouping of defined activities, settings and system environments. Last is the ***experiment***. An experiment consists of one target, one application and one measurement. One experiment is active and that is what will be executed when developers collect data. When an experiment is run and data is collected that completed data set is a ***trial***. Developers will typically use multiple measurement types for performance tuning, which means they will have multiple measurements and experiments defined in their project and multiple trials to analyze. Developers may have multiple applications in a project or they may create different projects for each application.

Beginning with Tau Commander

Getting TAU Commander

If TAU commander is not already on your system, developers may install it in their local directories. TAU Commander will automatically download, configure and install TAU and other appropriate utilities. When new options are activated within TAU Commander, it could take several minutes to download or build any utilities required for the new configuration. This is a one-time event and the next time that option is selected it will complete quickly. TAU Commander requires Python 2.7 or newer. The following command will retrieve it for developers:

```
git clone https://github.com/ParaToolsInc/taucmdr.git
```

This will create a taucmdr directory. Select the installation directory use that when you run make install then add the resulting bin directory to your path. On Linux this might be done with:

```
git clone https://github.com/ParaToolsInc/taucmdr.git
cd taucmdr
make install INSTALLDIR=<path>
export PATH=$PATH:<path>/bin
```

When this is completed you are ready to begin with TAU Commander. It is recommended to add the taucmdr/bin to the environment setup each time you log in so that TAU Commander is always in your path.

TAU Commander setup

The first step is to define a project. Let's begin with one of the examples. From the taucmdr directory move to `./taucmdr/examples/mm`. The easiest way to begin is to enter `tau initialize` or simply `tau init`. This first initialization will take quite a bit of time. Not only is this command creating a project it is also downloading and building the TAU Performance System® and associated libraries that it depends on. Let this run and check for successful completion. When it completes it displays the basics of the project – shown in five tables. This display can be shown at any time by entering `tau dashboard`. An example of this is shown in Figure 2.

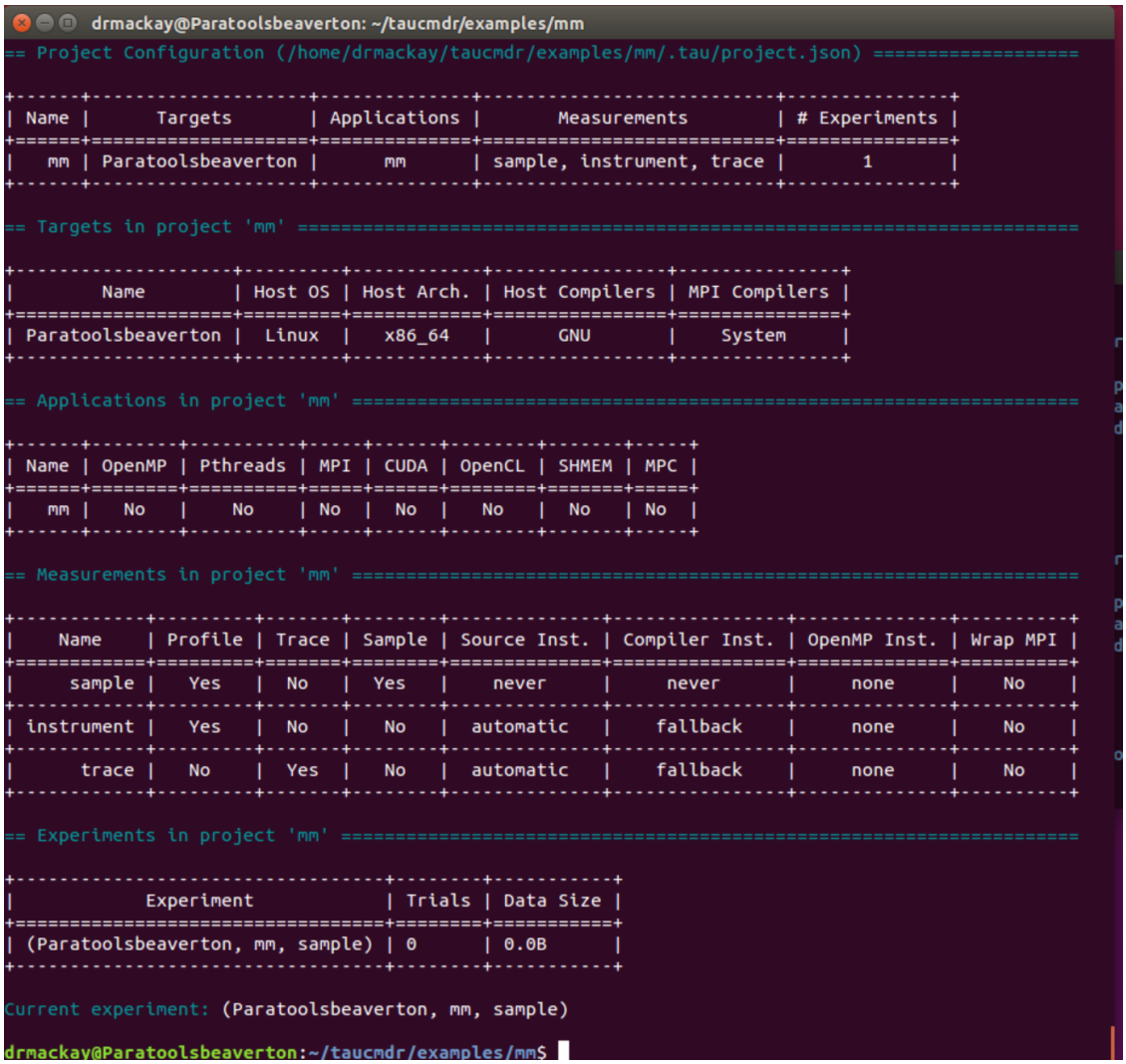


Figure 2: Tau dashboard displayed after first project initialized.

Many parameters may be defined at initialization. For example, to define a project named myname enter: `tau init --project myname`. To add additional projects to your system you may enter: `tau project create yourname`. This creates a project called `yourname`. If you type `tau dashboard` Tau Commander will show you the details of the active project (e.g. `myname)`. To see details of your new project `yourname` you must select it by entering `tau project select yourname`. Now when you enter `tau dashboard` the details of project `yourname` will be displayed. If you do this notice that project `yourname` did not default to any targets, applications or measurements – these need to be added to the project through tau create commands. It is frequently easier to define application parameters at initialization rather than editing later (fewer key strokes). For example, if your application uses MPI and OpenMP you may initialize Tau Commander by entering: `tau init --project myname --openmp T –mpi T`. Typical application parameters to define at initialization are: CUDA, linkage (static/dynamic), MPI, OpenCL, OpenMP, pthreads, SHMEM, and TBB. To get a full list of options for the Tau Commander version installed on your system just type: `tau application edit --help` and a complete print out of options will be printed.

Editing TAU Commander Applications

It is common to edit the parameters within a TAU Commander project. By default, TAU Commander uses the name of the directory where it is initialized as the application name. This is the name of the application within the TAU Commander project. The executable name may be entirely different. An application name might be mm while the executable binary is a.out. This is just fine. If multiple binaries are to be analyzed for performace they may be all be part of the current application or the analysis of the different binaries can be kept in a different application. One can create a new application with: `tau application create <new_application_name>`, or copy an existing application with: `tau application copy <existing_application_name> <new_application_name>`. Applications can also be deleted or edited. Most common is editing of an application property. This is easy to edit just type: `tau application edit --<name of application property> <setting (typically T or F)>`. A list of application properties that may be activated are: CUDA, linkage (static/dynamic), MPI, OpenCL, OpenMP, pthreads, SHMEM, TBB. To get a full list options on your installations just type: `tau application edit --help` and a complete print out of options will be printed.

Editing TAU Commander Measurements:

The chief considerations for measurement is type of data, quantity and how it is collected. A simple profile may just show hotspots – where most execution time is spent. A more complete profile may include callpath data showing sequence of what routine calls which routine. A trace gives specific order of events including point to point transactions. The amount of data associated with each is illustrated in Figure 3 below.

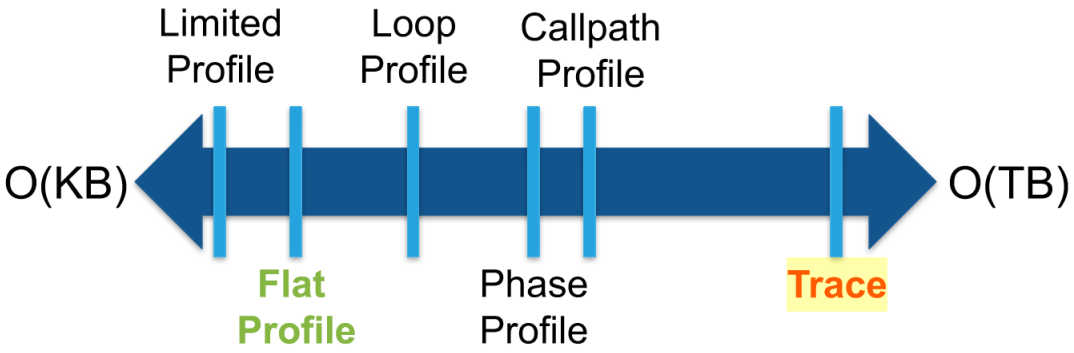


Figure 3: measurement options and size of data files associated with that option.

TAU Commander supports both profiling and tracing. To create a new measurement with one of these options merely enter `tau measurement create myprofile --profile` or `tau measurement create yourprofile --trace`. The data that appears in the profile or trace may be gathered by one or more data measurement methods including sampling, source instrumentation, or compiler instrumentation. The default is sampling, where TAU Commander will use the symbols in the binary when built with -g to decipher code information. Source instrumentation relies on the Program Database Toolkit (PDT) to add information to track and the last is utilization of the compiler to instrument the source code. There are several TAU commands that can explicitly be added to the code which is beyond the scope of this introductory manual. Those interested can find more information in the TAU User Guide (https://www.cs.uoregon.edu/research/tau/docs/newguide/index.html). The application section defined several important parameters which may be active in the binary and thus can be activated. It is important to specify which of those items are to be active in the measurement definition – these include: OpenMP, CUDA, I/O, MPI and SHMEM. For example, for MPI this can be done by setting the MPI Boolean to True with this command: `tau measurement edit <measurement_name> --mpi T`. Substitute measurement of interest in place of MPI T (e.g. --cuda T).

TAU Commander Experiments

Create experiments by entering:

```
tau select [target] [application] [measurement]
```

The select command accepts the names of exactly one target, application, and/or measurement objects given in any order. For example, the following commands are equivalent:

```
tau select my_profile my_target my_measurement

tau select my_target my_profile my_measurement
```

If the target, application, or measurement name can be implied then it may be omitted. For example, if you have only one target and only one application in your project then only the measurement name must be specified:

```
tau select my_profile
```

The select command will name the new experiment based on the names of the selected objects. You may rename the new experiment with the `tau experiment edit` command, or take full control of experiment creation by explicitly specifying each parameter:

`tau experiment create my_experiment --application my_application --measurement my_profile --target my_target.` The entities my_experiment, my_application, my_profile as well as my_target must already be defined. The subsections above discuss creating and editing those entities.

Build and Run

Now that the TAU Commander project is well defined you are ready to build your application and run. Typically, this can be done by adding the taucmdr/bin directory to your path and then adding "tau" in front of your usual compiler and link commands – that is "tau cc" or "tau f77". It is easy to edit a Makefile to add tau prefixes in front of compiler and link commands. When you build, the active experiment or "Selected Experiment" defined in the Tau Commander project will determine the TAU features activated during the build. The last line of the `tau dashboard` output will show the selected experiment. Changing the selected experiment typically will require that you rebuild your application binaries – depending on what changes. TAU Commander will remind you to rebuild your binary executable.

Now that the project is well defined and you rebuilt your executable binary, it is ready to run. This is done by preceding the executable binary with "tau". So instead of simply `./a.out` one would enter: `tau ./a.out.` The binary a.out will be executed and the TAU Commander specified data will be collected. This will become a trial.

Viewing Data

Enter `tau show` and TAU Commander will open up the appropriate display window to graphically show the data of the selected trial. This will be the last trial collected with the active or selected experiment. To see data for a different trial for this experiment just enter the trial number after show (e.g. `tau trial show 0`). Please note that trial numbers begin with 0. The first trial is trial number 0. If tau dashboard shows 3 trials those 3 trials are numbered 0, 1, 2. To show data for a different experiment enter `tau select <desired_experiment>`, then `enter tau show #` where # is the trial number of <desired_experiment> whose data you want to view graphically.

Examples

This section will illustrate several different experiments. The basis will be the sample code packaged with TAU Commander in examples/sc15. You may use any code you like to reproduce similar views.

Profile-hotspot

A basic hot spot profile is shown first.
Install TAU Commander and define path as shown in the beginning of this document. Then go to taucmdr/examples/sc15/serial
Enter `tau init`
Enter `make`
Enter `tau ./matmult`
Enter `tau show`
TAU Commander automatically invokes paraprof. The view should be something like that shown in Figure 4.

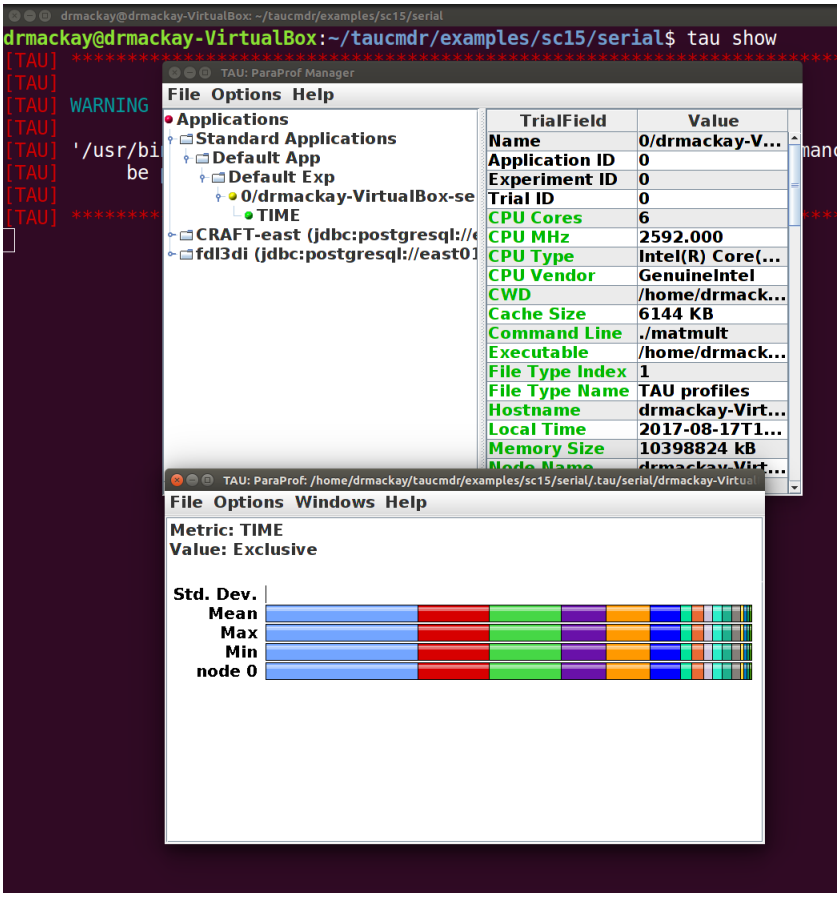


Figure 4:Initial paraprof view from tau show.

Now select “windows” from the top menu bar and scroll down to “thread” and select “Bar Chart” in the pop menu that appears as shown in Figure 5.

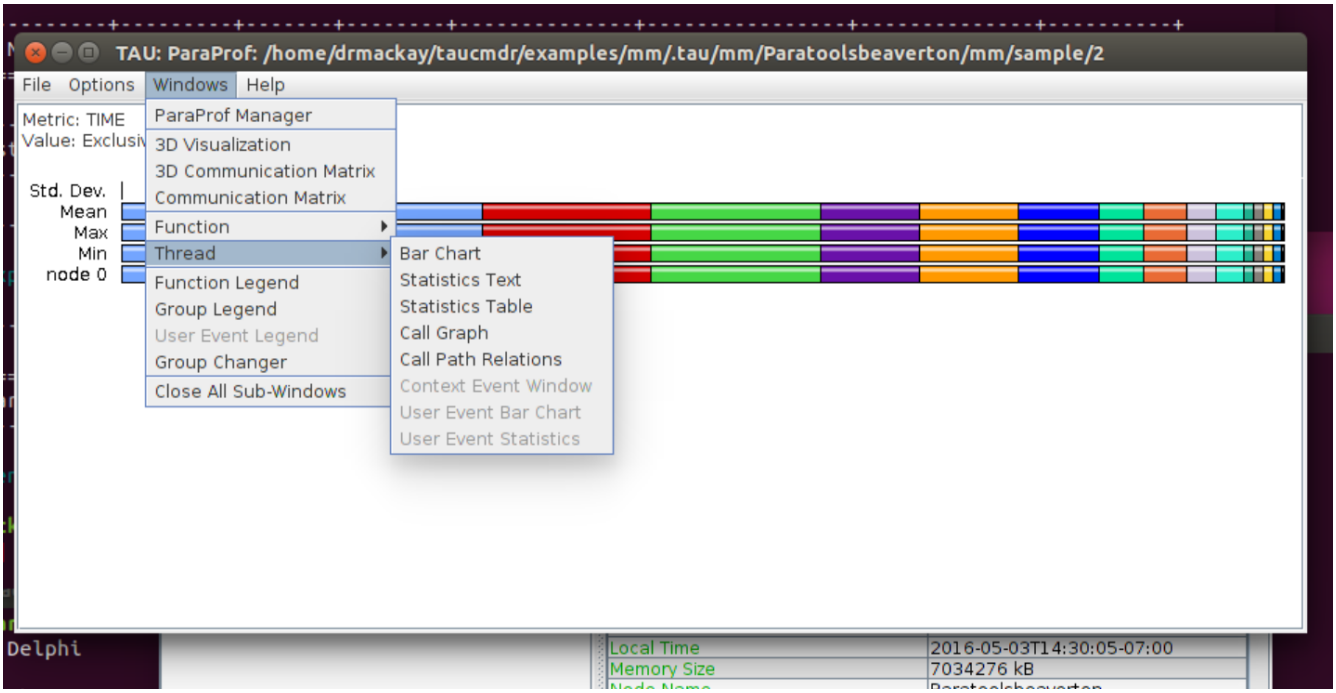


Figure 5: Thread options pop up menu.

Then select the top thread (n,c,t.0,0.0) in the box that appears as shown in Figure 6.

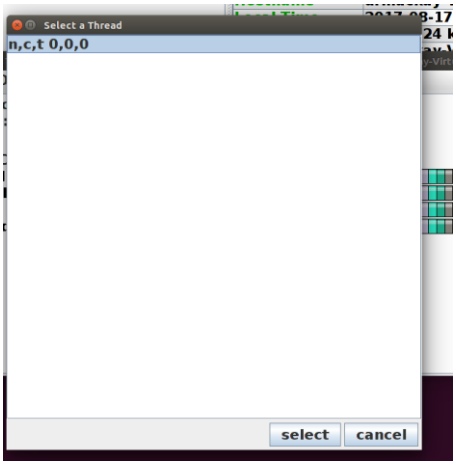


Figure 6: Final thread selection pop up box.

This will now display a list of functions and time spent in each one. The view should be something like that shown below in Figure 7.

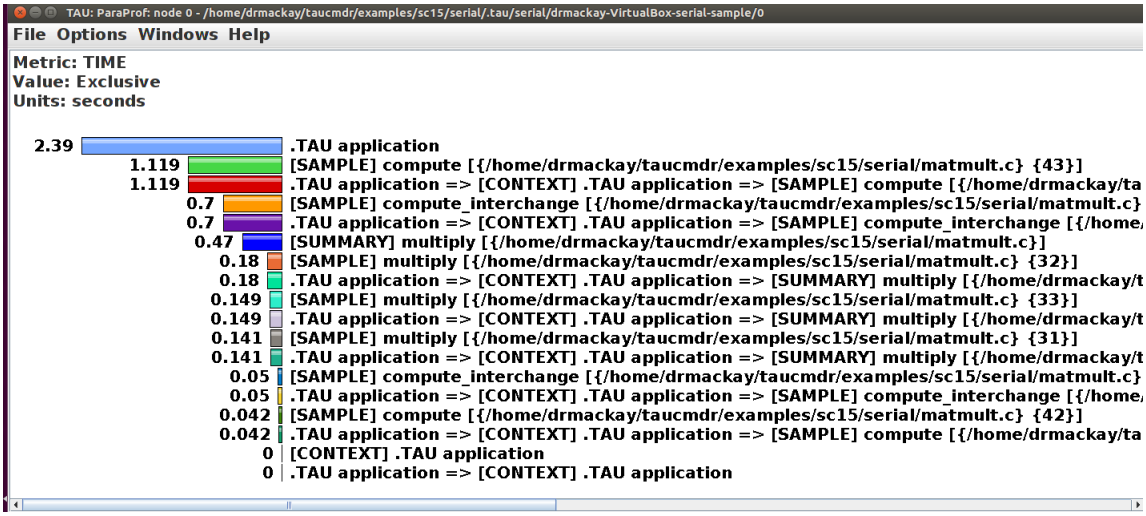


Figure 7: Bar chart - showing time spent in each function of thread 0.

In order to see callpath information select “Call Graph” or “Call Path Relations” instead of “Bar Chart” as illustrated in Figure 5. The “Call Graph” option will provide a graphical interface with boxes and arrows (you can pull on the text boxes in the graphics to make them legible). The “Call Path Relations” options provides a textual list of routines with data showing which routines call which routines and the number of times called.

3D Visualization

Another common view is a 3D visualization of each MPI rank and a metric (e.g. time) for each routine it calls. To see this view begin by selecting the windows menu item as shown below in Figure 8. Select the 3D Visualization option at the top just below Paraprof Manager. This will open the 3D image shown in Figure 9. In Figure 9 there is one axis with a data entry line for each MPI rank (8 in this case). Along another axis are the routines called by the program and along the third access is the time spent in each of those routines. This data was collected on an oversubscribed condition – which leads to imbalance in MPI routines which was expected. This view can be quite helpful in looking for work/load balance issues with TAU Commander.

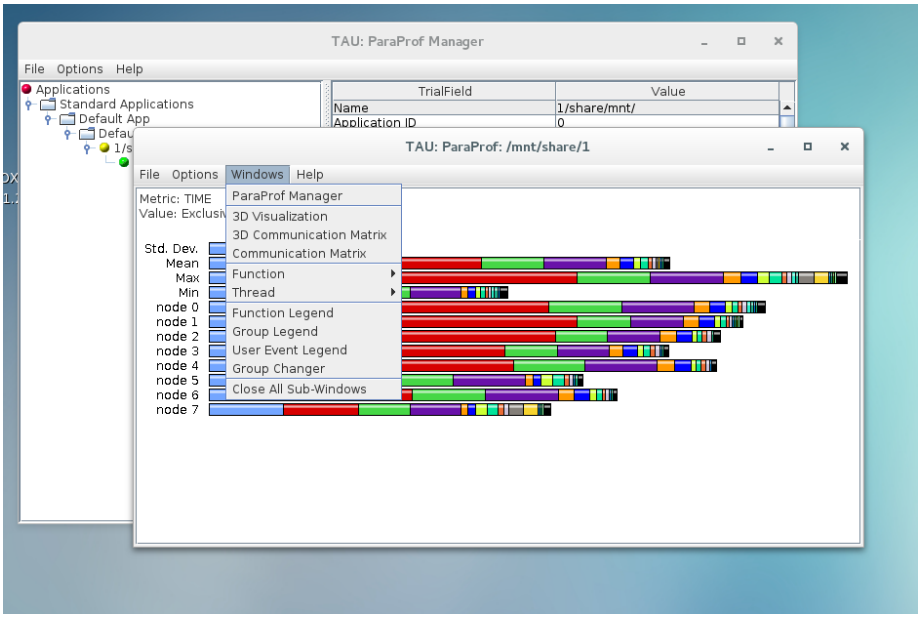


Figure 8: Window selection options with an MPI run

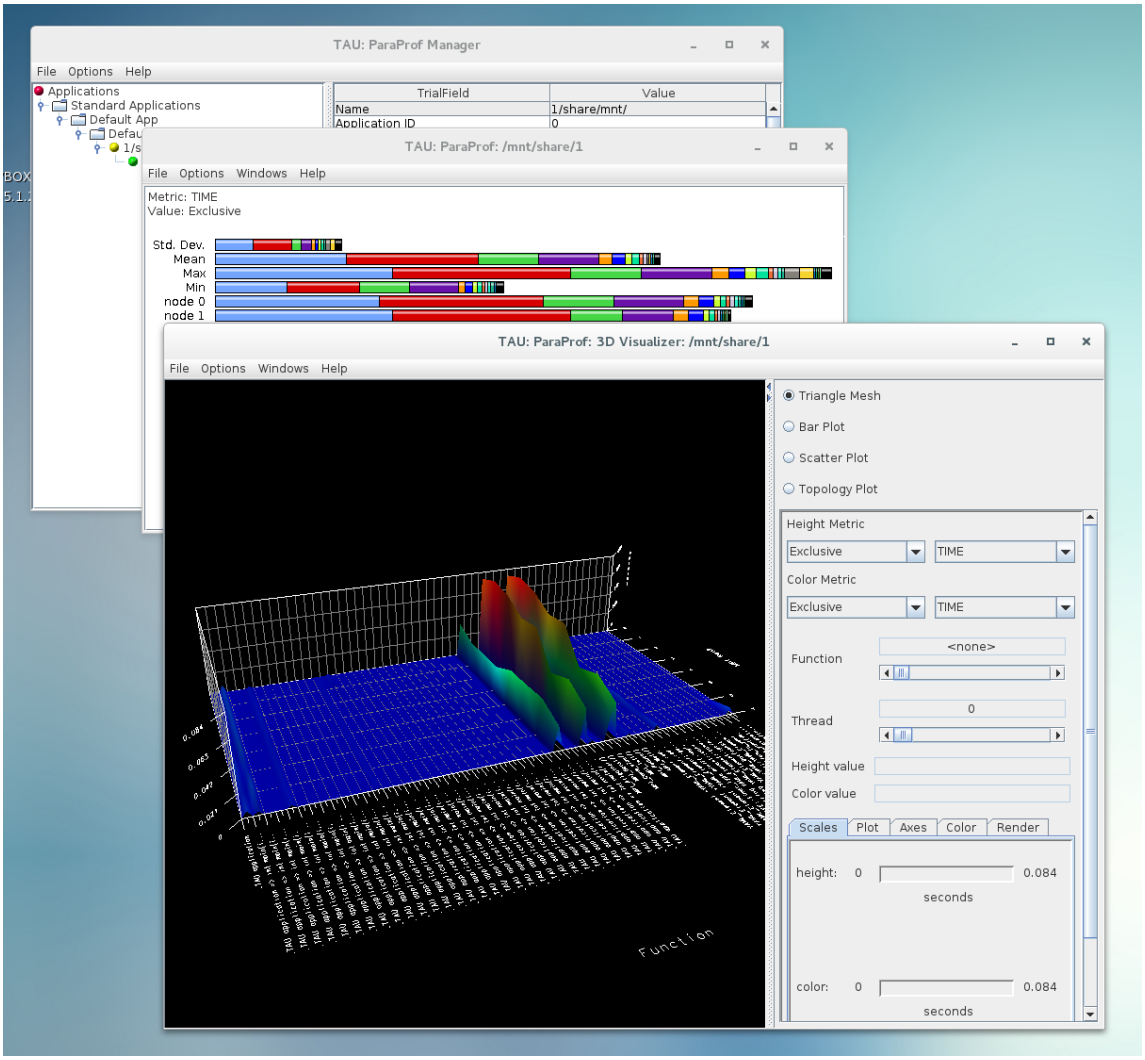


Figure 9: 3D visualization of 8 MPI ranks and routines called.

*Names and trademarks belong to their respective owners

Hardware counters with PAPI*

TAU Commander can also be configured to collect hardware events using PAPI and then create customer metrics based on those collections. One example is shown here of setup and configuration. Please check whether papi is already installed on your system. By default TAU Commander will download and install PAPI. If it is already on your system (please check) you can use the PAPI libraries already installed. On the system used in this illustration papi is installed as linux modules and is accessed by loading the appropriate module (`module load papi/5.4.3`). To use the already installed PAPI module let TAU Commander know where it is – on the system for this example that would be /packages/papi/5.4.3 (this directory contains the bin and lib subdirectories as well as include, man and chare). This example was done by the following – first create a new measurement to include PAPI events. This was done by entering: `tau measurement copy sample samplehwc`. Then inform TAU Comammander where to find the PAPI utilities: `tau target edit <target_name> --papi /packages/papi/5.4.3.` Next is to define PAPI events. To see a list of PAPI events you may run `papi_avail`. In this example the first PAPI event is selected: PAPI_L1_DCM. To select this metric enter: `tau measurement edit samplehwc --metrics PAPI_L1_DCM`. Next the executable binary is rebuilt and run: `tau ./matmult` and displayed: `tau show`. This will invoke paraprof in a view similar to that shown for the basic hotspot profile. This new display is shown in Figure 10, notice that there is now additional items in the main view with green colored bullets next to it for the PAPI events (TIME and PAPI_L1_DCM). Frequently it is desired to derive a metric based on the events collected. To derive an event select the options menu from the main ParaProf window and a pop up menu appears, just select the top option "Show Derived Metric Panel" and it will appear as shown in Figure 11. This creates a new item as a green bullet for the derived metric below the papi events. Double clicking on one of the green bullets will open a new window based on that papi event or derived metric. The same options are available for this new window just as for a profile hotspot. Figure 12 shows the bar chart for the PAPI event PAPI_L1_DCM.

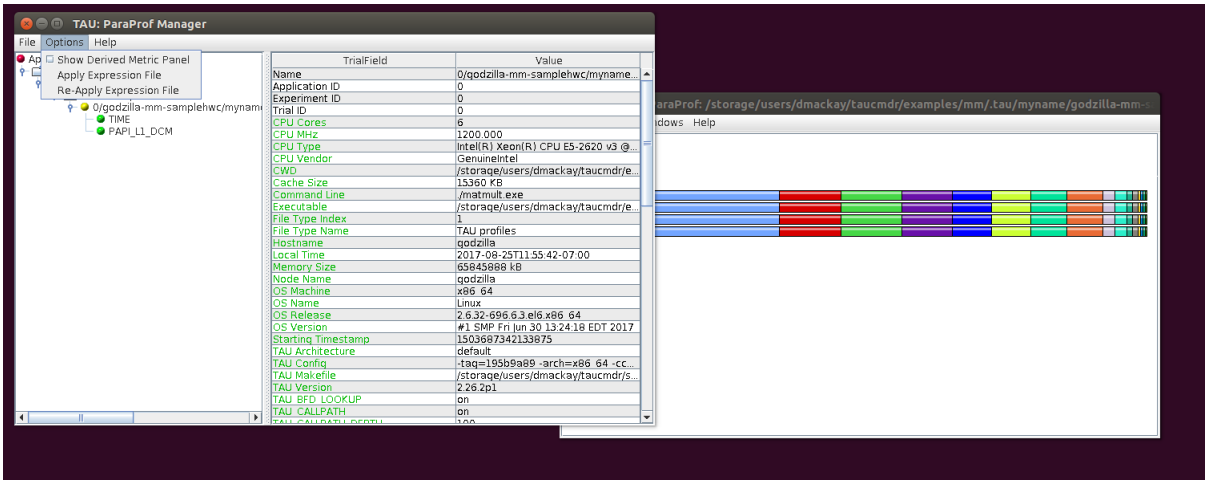


Figure 10 View after collecting PAPI events.*

*Names and trademarks belong to their respective owners

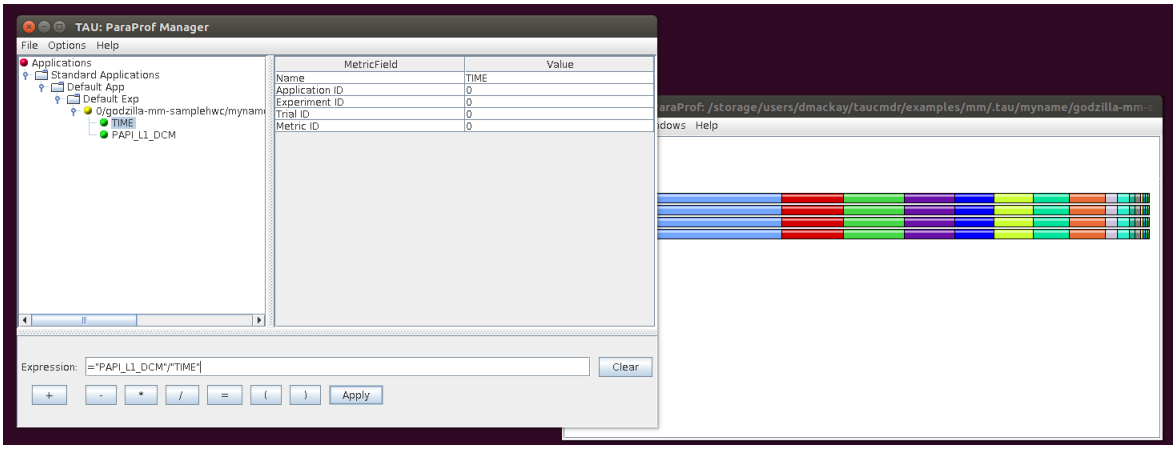


Figure 11 Derived Metric Panel shown at bottom of ParaProf Manager window.

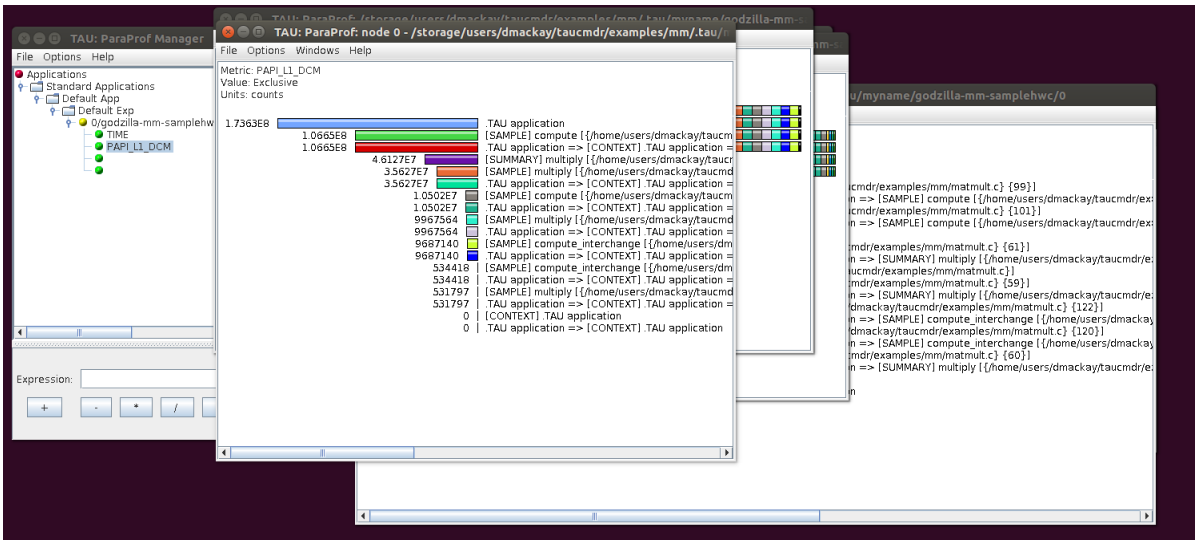


Figure 12 Display of PAPI event PAPI_L1_DCM

Traces

Another popular format is traces so two examples are presented. The first example uses the default otf2 format. This example was generated by first entering: `tau init --mpi T -trace otf2` (otf2 is optional as it is the default trace format in tau commander). After initializing `tau select trace` was entered the Makefile was modified to use “tau mpic++” and “tau mpicc” instead of “mpicxx” and “mpicc”. The binary (minife) was built and executed (tau mpirun -np 8 ./miniFE.x 120 120 120). The command `tau dash` will show the trial with the data collected in otf2 format. If vampir is on the same system you can enter `tau show` and this will invoke vampir to display the results . Otherwise you may export the files to view on another system (see section on trial export). TAU Commander will not automatically install Vampir. Figure 13 shows the data collected. In this view within Vampir the following selection was made – File->preferences->appearance-> (expand MPI events and select all) right click and select random colors then click on apply and close. This provides different colors for different MPI tasks – Allreduce, Wait, Send, ... TAU Commander was also configured with callsite 1 for this data collection. In Figure 13 the different MPI events are visible as well as the point to point message MPI_Send events from one MPI rank to another. As of the writing

of this document TAU Commander needs to be configured to use a nightly build of TAU in order to produce the otf2 traces. Expect this to change very soon.

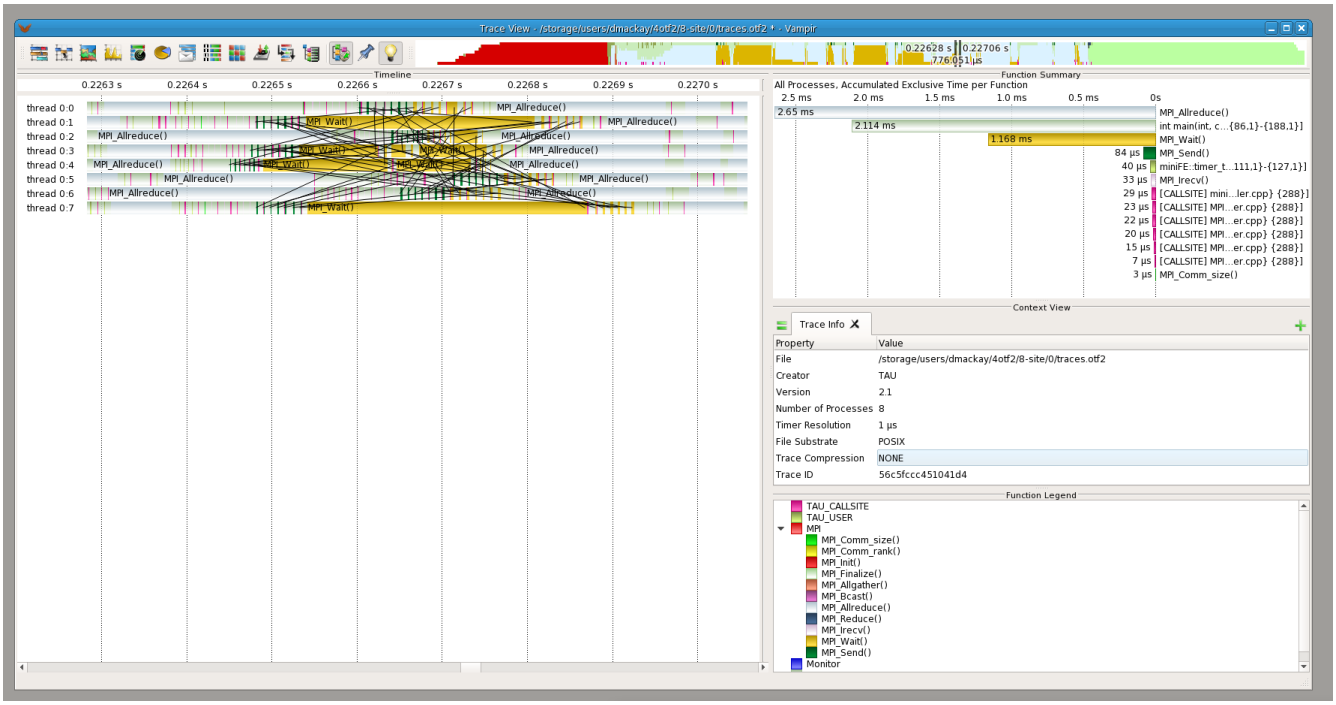


Figure 13: Vampir display of otf2 data collected with TAU Commander.

The alternate format for traces is slog2. This example was generated by first entering: `tau init --mpi T --trace slog2`. If TAU Commander is already initialized and no traces have been collected yet you may edit the trace measurement, otherwise it is easier to copy a trace setting using commands like this: `tau measurement copy trace traceslog2` followed by: `tau measurement edit traceslog2 --trace slog2`. After initializing `tau select trace (or traceslog2)` was entered the Makefile was modified to use “tau mpic++” and “tau mpicc” instead of “mpicxx” and “mpicc”. The binary (minife) was built and executed then `tau show` was run to display the trace. TAU Commander automatically installs and invokes jumpshot for viewing SLOG2 trace files without any extra configuration options. The image in Figure 14 is a zoomed in region of the trace from running minife on 2 MPI ranks. Please note that the SLOG2 traces are 3 to 4 times larger than the otf2 traces and SLOG2 and jumpshot do not scale well beyond 1024 processes.

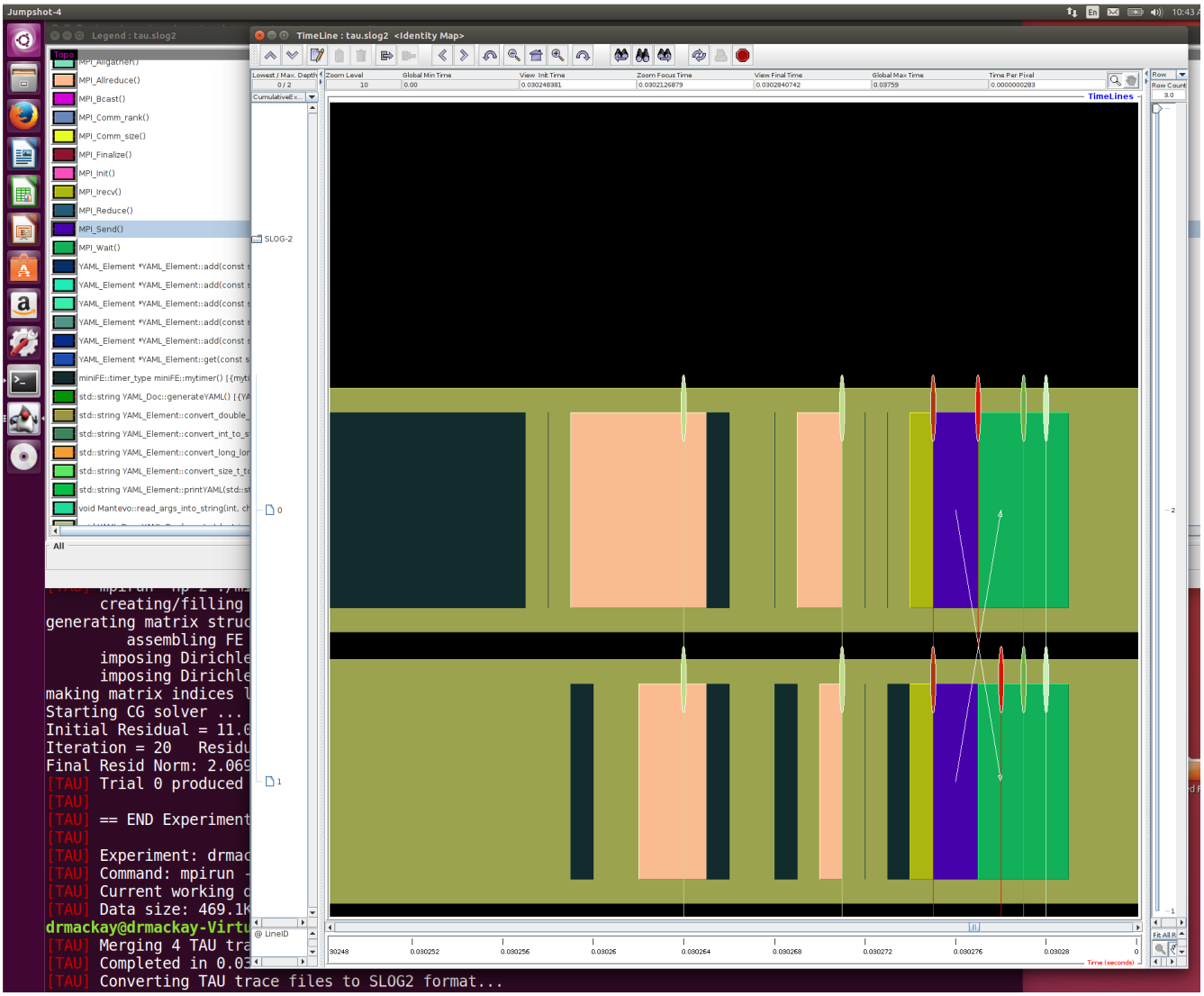
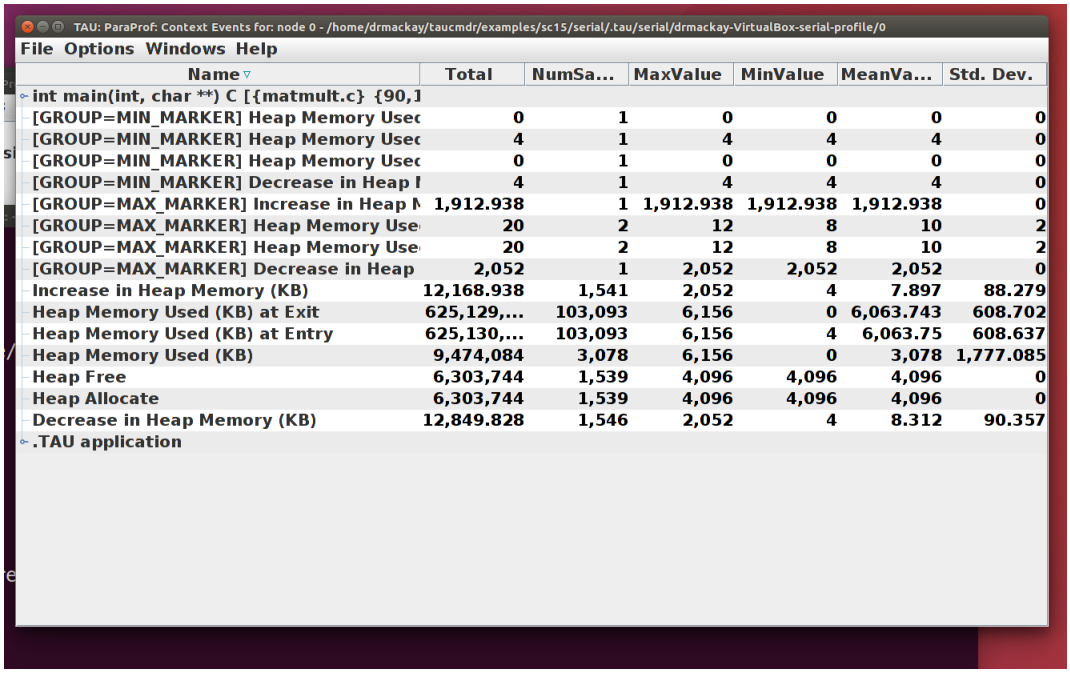


Figure 14: MPI Trace showing MPI_Allreduce and MPI_Send commands.

Memory usage

Memory usage is a common metric of interest. Tau Commander can help developers track this too. Developers can create a new measurement or copy an existing measurement and edit it. The options to activate are heap-usage and memory-alloc. Activation would be done like this: `tau measurement edit profile --heap-usage T --memory-alloc T.` If you already have a measurement named profile if not, please substitute your measurement name in place of "profile". Rebuild (make or tau g++ file.cp) and run as normal (`tau ./a.out`) and enter `tau show`. This time select "Context Event Window" from the menu shown in Figure 5 (note in Figure 5 this option is greyed out as an invalid option – when you set heap-usage and memory-alloc to True and run, this menu option will no longer be greyed out as it becomes a valid option). The display will be something like that shown in Figure 15.



Name	Total	NumSa...	MaxValue	MinValue	MeanVa...	Std. Dev.
int main(int, char **) C [{matmult.c} {90,1						
[GROUP=MIN_MARKER] Heap Memory Usec	0	1	0	0	0	0
[GROUP=MIN_MARKER] Heap Memory Usec	4	1	4	4	4	0
[GROUP=MIN_MARKER] Heap Memory Usec	0	1	0	0	0	0
[GROUP=MIN_MARKER] Decrease in Heap I	4	1	4	4	4	0
[GROUP=MAX_MARKER] Increase in Heap M	1,912.938	1	1,912.938	1,912.938	1,912.938	0
[GROUP=MAX_MARKER] Heap Memory Use	20	2	12	8	10	2
[GROUP=MAX_MARKER] Heap Memory Use	20	2	12	8	10	2
[GROUP=MAX_MARKER] Decrease in Heap	2,052	1	2,052	2,052	2,052	0
Increase in Heap Memory (KB)	12,168.938	1,541	2,052	4	7.897	88.279
Heap Memory Used (KB) at Exit	625,129,...	103,093	6,156	0	6,063.743	608.702
Heap Memory Used (KB) at Entry	625,130,...	103,093	6,156	4	6,063.75	608.637
Heap Memory Used (KB)	9,474,084	3,078	6,156	0	3,078	1,777.085
Heap Free	6,303,744	1,539	4,096	4,096	4,096	0
Heap Allocate	6,303,744	1,539	4,096	4,096	4,096	0
Decrease in Heap Memory (KB)	12,849.828	1,546	2,052	4	8.312	90.357
.TAU application						

Figure 15: Memory display for profile collection.

IO

Collecting IO is similar to collecting memory information. The first step is to activate io as a measurement parameter. This is done by: `tau measurement edit profile --io T` or `tau measurement edit trace --io T`. Once again use your measurement names – profile and trace are used here as these measurement definitions are created by default. Rebuild the executable binary and run to create a new trial. Now enter `tau show`. The results should be similar to what is shown below in Figure 16

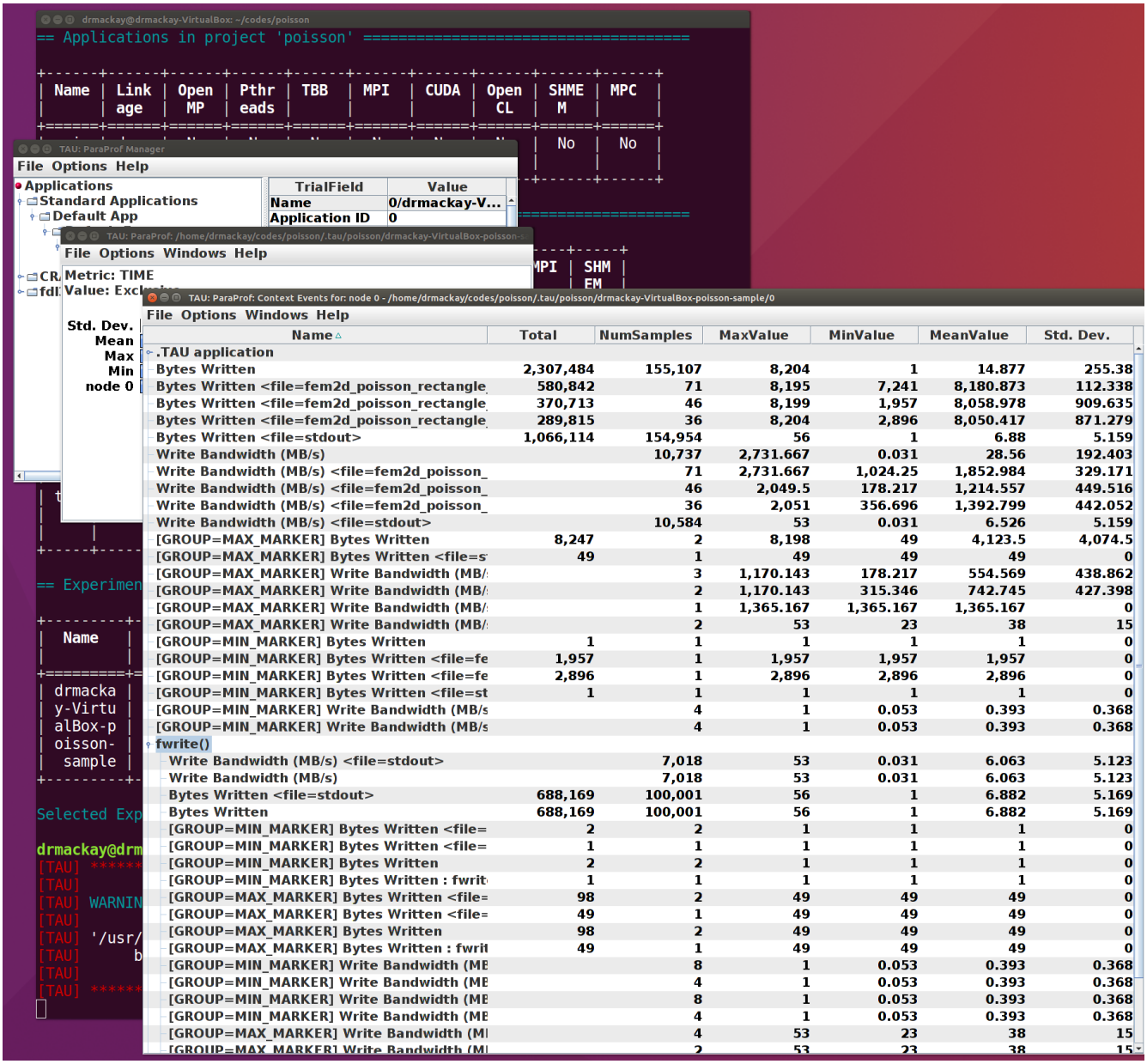


Figure 16: Display of IO profile collected with Tau Commander.

ParaTools encourages the reader to continue to explore the capabilities of TAU Commander. The wealth of information provided will assist developers identify bottlenecks and see where they can tune their software to improve performance.

See:
TAU Commander Quick reference card