PLASMA TAU Guide

Parallel Linear Algebra Software for Multicore Architectures

Version 2.0

Electrical Engineering and Computer Science
University of Tennessee

Electrical Engineering and Computer Science
University of California Berkeley

Mathematical & Statistical Sciences
University of Colorado Denver

Jack Dongarra
Joshua Hoffman

Jakub Kurzak

Contents

1 Introduction

To better understand the scheduling and inner workings of PLASMA, profiles and function traces can be used to see exactly what the PLASMA libraries are doing at any point during execution. This document will guide the reader through installing TAU (Tuning and Analysis Utilities) from the University of Oregon, and how to instrument the PLASMA libraries with the TAU tool.

2 Required Files

Two separate downloads are required for the TAU tools. The first is the TAU tool itself, which is downloaded in a single tarball from the University of Oregon. Next, the tool used to automatically instrument the source code, called the PDT (Program Database Toolkit), can be downloaded from the same site. Finally, to view the slog2 traces, the Jumpshot-4 trace viewer is needed.

2.1 Downloads

All three downloads will be downloaded as tarball files, containing a root directory with their own file trees. Each package will have more detailed installation instructions included, supplementing the online documentation.

2.1.1 TAU and PDT

The TAU tools themselves, as well as the PDT, can both be downloaded from:

http://www.cs.uoregon.edu/research/tau/home.php

Near the top left corner, click download, fill out the short information form, then download the latest version of TAU, as well as the PDT, located further down the page.

2.1.2 slog2rte and Jumpshot-4

It is recommended that the slog2 trace format be used when gathering traces using TAU. To generate the traces, no extra software, other than TAU and PDT, are required. However, to view the traces, the Jumpshot-4 trace viewer is required. This guide assumes the use of the slog2 format and Jumpshot-4 viewer. The Jumpshot-4 viewer can be downloaded from:

ftp://ftp.mcs.anl.gov/pub/mpi/slog2/slog2rte.tar.gz

The viewer itself can be found in the folder (PATH_TO_SLOG2)/slog2rte/lib, and is called jumpshot.jar. A version of the java runtime environment is required to use Jumpshot. Furthermore, the Jumpshot-4 viewer can be run on any system with a graphical user interface and the java runtime environment installed.

3 Forward General Reference

Set environment variables:

```
$ export PATH=/path/to/tau/x86_64/bin:$PATH
$ export TAU_MAKEFILE=/path/to/tau/x86_64/lib/Makefile.tau.pthread-pdt-trace
```

Edit your makefile and set: `cc=tau_cc.sh; f90=tau_f90.sh; cxx=tau_cxx.sh`

Compile and link with TAU, PDT:

```
$ make
...
$ ./your_tau_linked_program.out
...
$ tau_treemerge
...
$ tau2slog2 tau.trc tau.edf -o <output>.slog2
...
```

Using the SLOG-2 Runtime:

```
$ java -jar /path/to/jumpshot/jumpshot.jar
```

4 Setup and Installation

The tools downloaded above will need to be built on a system with access to C/C++ compilers, as well as a Java SDK and runtime environment. The required Java files can be downloaded from:

http://java.sun.com/javase/downloads/index.jsp

Administrator or superuser access is not required to build or install the libraries. However, before beginning installation make sure they are in a place in where your user account has read/write/execute privileges. Also, before beginning installation, for both TAU and PDT create directories separate from their respective installation directories.

4.1 Environment Variables

The installation of the two packages and the Jumpshot-4 viewer require the modification of several environment variables. First, the location of the Java jvm must be located and added to the environment variable JAVA_HOME. For example:

```
export JAVA_HOME=/usr/lib/jvm/java-6-sun
```

Next, the path to the TAU and PDT tools must be added to the path, e.g.:

```
export PATH=$PATH:/home/user/pdt/x86_64/bin:/home/user/tau/x86_64/bin
```

Also, for some compilers (Intel, for example) the LD_LIBRARY_PATH must be changed to point to the location of your compiler's libraries ((INTEL_COMPILER_ROOT)/lib for example). And finally, after the tools have been built and installed correctly, the TAU_MAKEFILE environment variable will need to be created, pointing to the makefile containing the proper TAU settings. This will be discussed more at the end of the TAU installation section.

4.2 Building PDT

The PDT package is required for automatic instrumentation of source code to produce data with the TAU library. After downloading and extracting the PDT code, navigate to the extracted directory in the command line. The first step is to run the configure script in the top level PDT directory. The various options for the configure script are detailed in the README included with the PDT software, however three are worth noting here.

First, the configure script must be told which C++ compiler version to use via a command line option. The PDT software supports 10 compiler packages, which are listed in the README, however two of the more common ones are for the Intel package and the GNU compilers; -icpc and -GNU, respectively. The next command line options are optional, however using them will help keep everything organized. Using the -prefix=<dir> option will allow you to specify a target directory for the installation instead of installing into the source directories. Also, the -compdir=<name> will help keep the files organized. An example command would look like:

```
./configure -icpc -prefix=/home/user/PDT -compdir=intel
```

After the configure script has been executed, run make, then make install, and the PDT tools will be built and installed in the directory chosen above.

4.3 Building slog2sdk

With the slog2 run time environment, nothing new needs to be built. All of the files come as executable Java .jar files.

4.4 Building TAU

Configuring TAU is slightly more involved than building the PDT. Before running the configure script for TAU, make sure the environment variables above (other than TAU_MAKEFILE for now) are set. Also, know the location of the top level PDT directory. Finally, read the INSTALL file included in the extracted TAU files to get an idea of which options best suit your needs.

The following command will configure TAU to produce slog2 traces using Intel compilers and install TAU into the directory pointed to by the -prefix=<directory> directory. Programs will be built using the pthreads threading library as well. Programs built with the TAU compiler scripts will be automatically instrumented using the PDT built above.

```
./configure -pdt=/home/user/pdt -pdtcompdir=intel -cc=icc
    -c++=icpc -fortran=intel -pthread -slog2 -TRACE - PROFILE
    -prefix=/home/user/tau
```

Once the script has completed, run

```
make install
```

in the top level TAU directory, and upon successful completion, the TAU libraries will be built and installed under your system's architecture folder (x86_64 for example) in the directory specified by the -prefix= option. The only thing left is to set the TAU_MAKEFILE environment variable. From the top level TAU installation directory, change directories to the folder named after your system's architecture, and then the lib directory. In the lib directory, find the makefile which best describes the options used in the configure script(e.g. Makefile.tau-icpc-pthread-pdt-profile-trace). Now, set the environment variable TAU_MAKEFILE to the FULL PATH to this file, including file name.

In the directory pointed to by the -prefix= option above, a folder called 'examples' will have been created as well. Inside are directories containing over 50 example programs and makefiles to test the various functions of TAU. Refer to the file examples/README for a detailed description of each example.

4.5 Instrumenting Code

Once TAU and PDT are built and installed correctly, and the environment variables are set properly, it is a good idea to run several tests before using the tools on complicated code. In the top level of the TAU installation directory, there is an examples directory containing over 50 different programs to test the tools on. Simply change directories to one of these examples and run make to build the example. The pthreads TAU example is a good example to start with. It will pause for 5 seconds during execution, so do not panic.

To instrument your own projects with TAU, your code will need to be compiled with the compiler scripts created by TAU. For C files, the tau_cc.sh compiler should be used, and for tau_f90.sh for FORTRAN. For simplicity, the compiler option inside your makefiles can be changed to the necessary TAU compilers, and make can be run as usual. A simple makefile might look like:

```
CC     = tau_cc.sh
INC    = -I/home/user/intel/mkl/include
LIB    = -mkl_em64t -lguide
LIBDIR = -L/home/user/intel/mkl/lib/em64t

all:
    $(CC) $(INC) $(LIBDIR) $(LIB) test.c -o test
```

Notice the makefile is a normal makefile, with just the compiler variable changed.

4.5.1 Select File Instrumentation

Because of the overhead associated with profiling with TAU, PLASMA performance will suffer. However, this can be avoided by only tracing important functions, reducing the performance hit caused by TAU. This can be done at compile time, by appending -optTauSelectFile=<path_to_select_file> to the TAU_OPTIONS environment variable. The file pointed to by <path_to_select_file> is the select file PDT will use when instrumenting the program.

Select files are a powerful tool, and only a brief overview of their capabilities will be discussed here. For more information regarding select files, refer to the TAU online documentation.

The TAU select file tells PDT at compile time which functions to instrument and which ones to skip. Not only that, but it can also specify entire files to skip or include.

The select file itself is a simple text file, with any file extension; however for convenience this guide will refer to it as select.tau.

Creating a select file is fairly simple; it consists of lists of files or functions to include or exclude and the flags to mark the beginning or end of a list. To include or exclude files, the lists are surrounded by: BEGIN_FILE_INCLUDE_LIST / END_FILE_INCLUDE_LIST and BEGIN_FILE_EXCLUDE_LIST / END_FILE_EXCLUDE_LIST markers respectively.

To make selecting file easier, TAU supports the use of two wildcards (* and ?) in file include/exclude lists. The first, *, matches any number of characters while ? only matches one. Also, # can be used for comment lines. An example section of a select file which has an exclude file list would look like:

```
BEGIN_FILE_EXCLUDE_LIST

# exclude all files with a .f extension
*.f

# exclude files beginning with main.
# with a single letter file extension
main.?

END_FILE_EXCLUDE_LIST
```

Similarly, the function include/exclude lists are encased by BEGIN_INCLUDE_LIST / END_INCLUDE_LIST and BEGIN_EXCLUDE_LIST / END_EXCLUDE_LIST markers, respectively. The function include/exclude lists are slightly more difficult than the file lists because the functions must have an argument type list as well as a return type. Also, if using mixed FORTRAN and C code as in PLASMA, a capital letter 'C' is required at the end of each function listing in the select file. Unlike the file lists above, the function lists only have one wildcard symbol '#'. However, the wildcard only matches against function names; a correct argument list and return type are still required. If used at the beginning of the line, '#' will still mark a commented line. For example, the following is a snippet from a select file excluding several functions.

```
BEGIN_EXCLUDE_LIST

double get_current_time(void) C

# the next line excludes all functions beginning with sort_
# which return void and accept int *
void sort_#(int *) C

# FORTRAN subroutines only require the name of the subroutine
CORE_DGESSM
```

```
END_EXCLUDE_LIST
```

Select file instrumentation also supports an instrument section, where specific functions may be instrumented differently, as well as tracking different memory allocations/deallocations and loop specific instrumentation. However, these are beyond the scope of this document; please refer to the TAU documentation for more information about select file instrument sections.

4.5.2 Compiling PLASMA with TAU

To compile plasma with TAU, make sure the make.inc.example file is correctly renamed 'make.inc'. Then, edit the make.inc file so that the lines

```
CC     = icc
FC     = ifort
LINKER = ifort
```

are changed to

```
CC     = tau_cc.sh
FC     = tau_f90.sh
LINKER = tau_f90.sh
```

Also, ensure that the INC, LIB, and LIBDIR variables correspond to your BLAS. Then, run make as usual. If the build is successful, test PLASMA and TAU using the files included in the 'testing' directory.

5 Using TAU

After the executables have been built using the TAU compilers and are properly instrumented, all that remains is to run the code on a sufficiently large data set, to gain a clear picture of what the code is doing; for PLASMA code a matrix of at least 1000x1000 is recommended.

To localize profile and trace output from TAU, two environment variables can be set to point to a directory where the corresponding output will be dumped. It is recommended to set a different output location for each program run, and for runs you wish to keep separated; TAU output files will overwrite previous data. The two environment variables are PROFILEDIR and TRACEDIR. Set these to their target directories, and TAU will automatically put the output files here at run time.

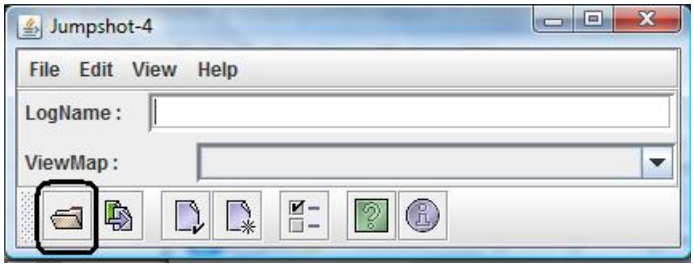

Figure 1: Button to open file select dialog.

5.1 Trace Format Conversion

Once the code is finished executing, several files will be created in the current directory. These files are events.0.edf, and tautrace.0.0.*.trc, where * is a specific processor number. Each processor core used will have it's own file created. Before these files can be converted to slog2 format, they first must be merged into a single file. This is simply done by running `tau_treemerge` from the command line (assuming the PATH is set properly) in the directory in which the files are located. This creates two new files, tau.edf and tau.trc. The next step is to convert these two new files into slog2 format. Just run `tau2slog2 tau.trc tau.edf -o <output>.slog2` from the command line, where <output> is the desired name of the trace file.

5.2 Viewing slog2 Trace

With the TAU trace converted to slog2 format, open the Jumpshot viewer by executing the Java .jar file, jumpshot.jar. Click the "Select new logfile" button in the bottom left hand corner of the viewer (Figure 1), then navigate to the .slog2 file output by tau2slog2, click it, and then select open in the dialog box. After choosing and opening the .slog2 file, three windows should appear (Figure 2).

Window 1 is where the trace will be displayed. Window 2 is the legend, where the colors of the trace window are related to the functions traced in the program. Window 3 is the Jumpshot viewer window, where a new file can be opened and Jumpshot options modified.

To view the function trace for each thread, find the "ViewMap" drop-down box on the Jumpshot-4 viewer, then select Identity Map. This changes the trace window to show the activity of each thread. The trace window should change to look like the one on Figure 3.

Depending on your program, this view might contain a large amount of information that can be difficult to interpret. However, using the legend, extraneous functions can be removed to give a clearer view of the important information. First, with the legend window active, uncheck boxes in the 'V' column that correspond to functions to be removed (Figure 4).

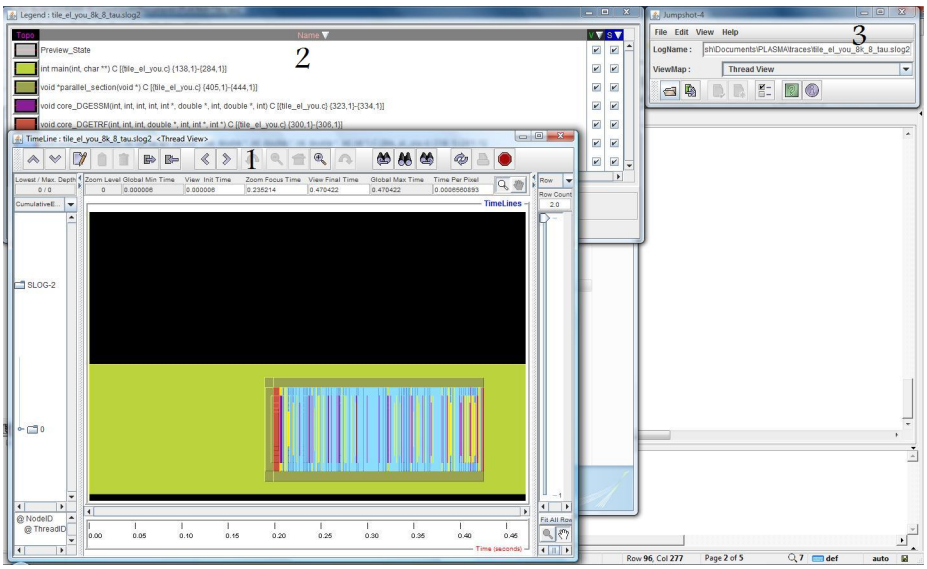

Figure 2: Trace window (1), legend window(2), and viewer window (3).

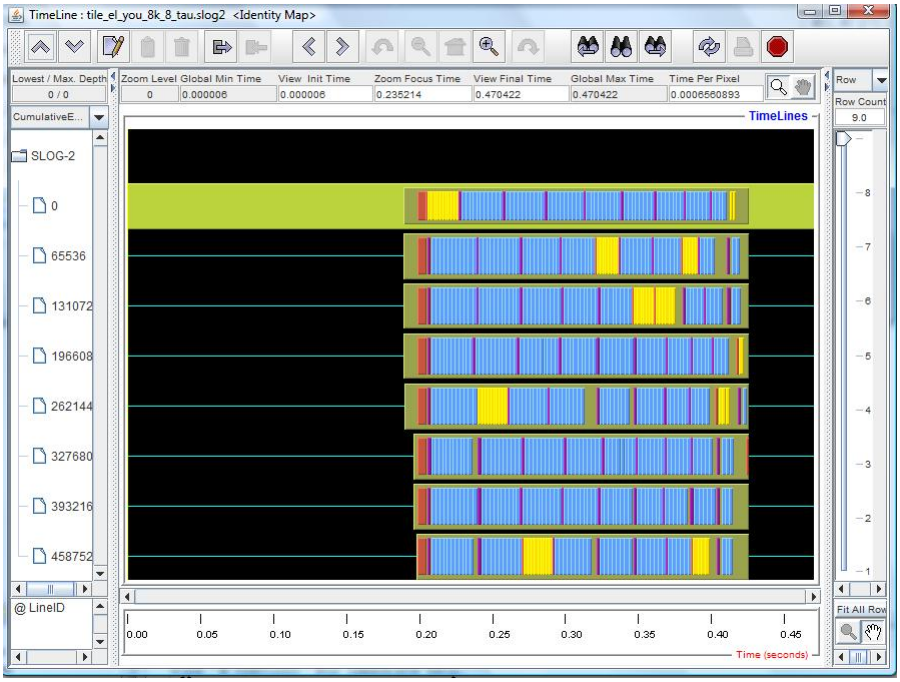

Figure 3: Trace view window after Identity Map has been selected.

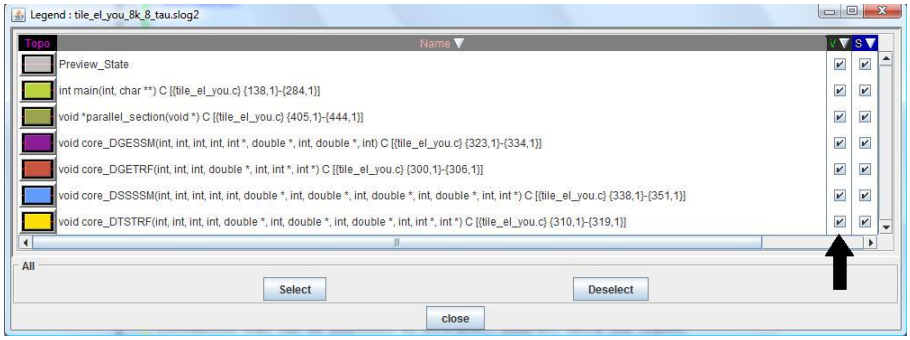

Figure 4: The legend window. The arrow points to the checkboxes which can be used to turn on and off displaying of individual functions.

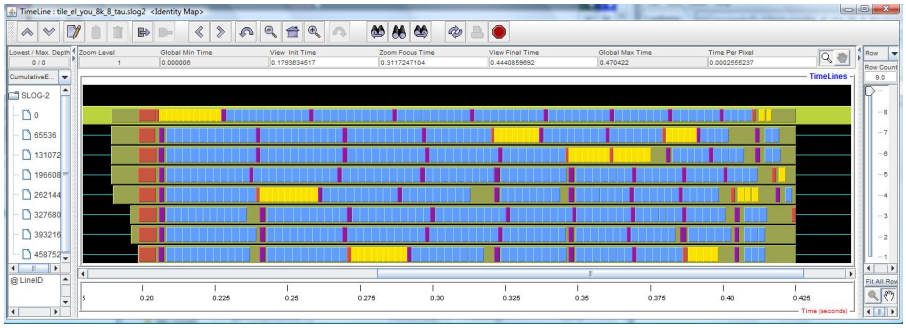

Figure 5: Sample TAU trace of PLASMA tile LU factorization on 8 cores.

After unchecking the boxes in the legend window, the trace will still look the same. At the top of the trace window, click the  button to refresh the trace. The trace window should now resemble the two traces in the following examples section.

To produce a trace showing the work PLASMA is doing, only the functions and files in the (PLASMA_DIR)/core_blas folder need to be traced.

6 Examples

Figures 5 and 6 show what a trace of PLASMA should look like. Both traces only include the functions found in the (PLASMA_DIR)/core_blas directory, the main function and the parallel_section function.

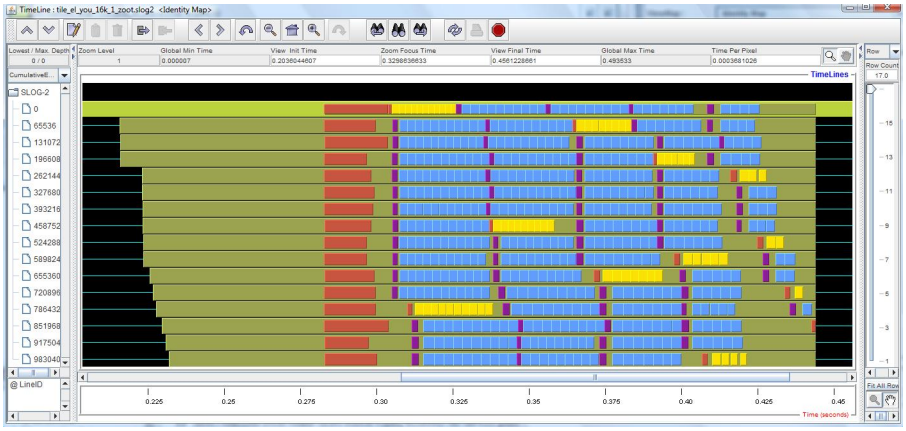

Figure 6: Sample TAU trace of PLASMA tile LU factorization on 16 cores.