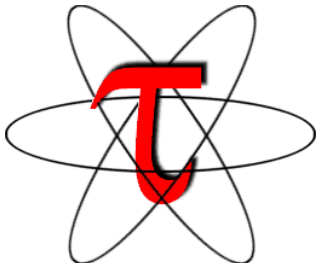
Performance Evaluation using TAU Performance System for Scientific Software

Sameer Shende and Allen D. Malony
University of Oregon
WSSSPE4
http://tau.uoregon.edu

TAU Performance System®



- **Tuning and Analysis Utilities (22+ year project)**
- **Comprehensive performance profiling and tracing**
  - Integrated, scalable, flexible, portable
  - Targets all parallel programming/execution paradigms
- **Integrated performance toolkit**
  - Instrumentation, measurement, analysis, visualization
  - Widely-ported performance profiling / tracing system
  - Performance data management and data mining
  - Open source (BSD-style license)
  - Uses performance and control variables to interface with MVAPICH2
- **Integrates with application frameworks**
- **http://tau.uoregon.edu**

Understanding Application Performance using TAU

- **How much time** is spent in each application routine and outer *loops*? Within loops, what is the contribution of each *statement*?
- **How many instructions** are executed in these code regions? Floating point, Level 1 and 2 *data cache misses*, hits, branches taken?
- **What is the memory usage** of the code? When and where is memory allocated/de-allocated? Are there any memory leaks?
- **What are the I/O characteristics** of the code? What is the peak read and write *bandwidth* of individual calls, total volume?
- **What is the contribution of each *phase*** of the program? What is the time wasted/spent waiting for collectives, and I/O operations in Initialization, Computation, I/O phases?
- **How does the application *scale***? What is the efficiency, runtime breakdown of performance across different core counts?
- **How can I tune MPI for better performance?** What performance and control does MVAPICH2 export to observe and control its performance?

Examples

Simplifying the use of TAU!

Uninstrumented code:

- % mpif90 –g –O3 matmult.f90
- % mpirun –np 16 ./a.out

With TAU:

- % mpirun –np 16 tau_exec ./a.out
- % paraprof
- For more Information at the statement level:
- % mpirun –np 16 tau_exec –ebs ./a.out (or use TAU_SAMPLING=1)
- To rewrite the binary to instrument individual functions (using MAQAO):
- % tau_rewrite a.out a.inst; mpirun –np 16 ./a.inst (beta)
- % pprof -a | more
- % paraprof (GUI)

TAU for Heterogeneous Measurement

Multiple performance perspectives

Integrate Host-GPU support in TAU measurement framework

- Enable use of each measurement approach
- Include use of PAPI and CUPTI
- Provide profiling and tracing support

Tutorial

- Use TAU library wrapping of libraries
- Use `tau_exec` to work with binaries

  % ./a.out (uninstrumented)

  % tau_exec –T <configuration tags> –cupti ./a.out

  % paraprof

TAU Execution Command (tau_exec)

Uninstrumented execution

- % mpirun -np 256 ./a.out

Track GPU operations

- % mpirun –np 256 tau_exec –cupti ./a.out
- % mpirun –np 256 tau_exec –cupti -um ./a.out (for Unified Memory)
- % 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 OpenMP, I/O, and MPI performance (MPI enabled by default)

- % mpirun -np 256 tau_exec –ompt -io ./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 –ebs_source=<PAPI_COUNTER> -ebs_period=<overflow_count>

Using TAU

TAU supports several measurement and thread options

Phase profiling, profiling with hardware counters (papi), MPI library, CUDA, Beacon (backplane for event notification – online monitoring), PDT (automatic source instrumentation) …

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:

% export TAU_MAKEFILE=$TAU/Makefile.tau-papi-mpi-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)

Choosing TAU_MAKEFILE

```
% ls $TAU/Makefile.*
```

Makefile.tau-mpi-pdt
Makefile.tau-papi-mpi-pdt
Makefile.tau-icpc-papi-mpi-pdt
Makefile.tau-icpc-papi-ompt-mpi-pdt-openmp
Makefile.tau-icpc-papi-ompt-pdt-openmp
Makefile.tau-mpi-pdt-openmp-opari
Makefile.tau-mpi-pthread-python-pdt
Makefile.tau-papi-mpi-pdt-openmp-opari-scorep
Makefile.tau-papi-mpi-pdt-scorep
Makefile.tau-papi-mpi-pthread-pdt
Makefile.tau-papi-pthread-pdt

For an MPI+F90 application with MPI, you may choose

Makefile.tau-papi-mpi-pdt

- Supports MPI instrumentation, papi, and PDT for automatic source instrumentation

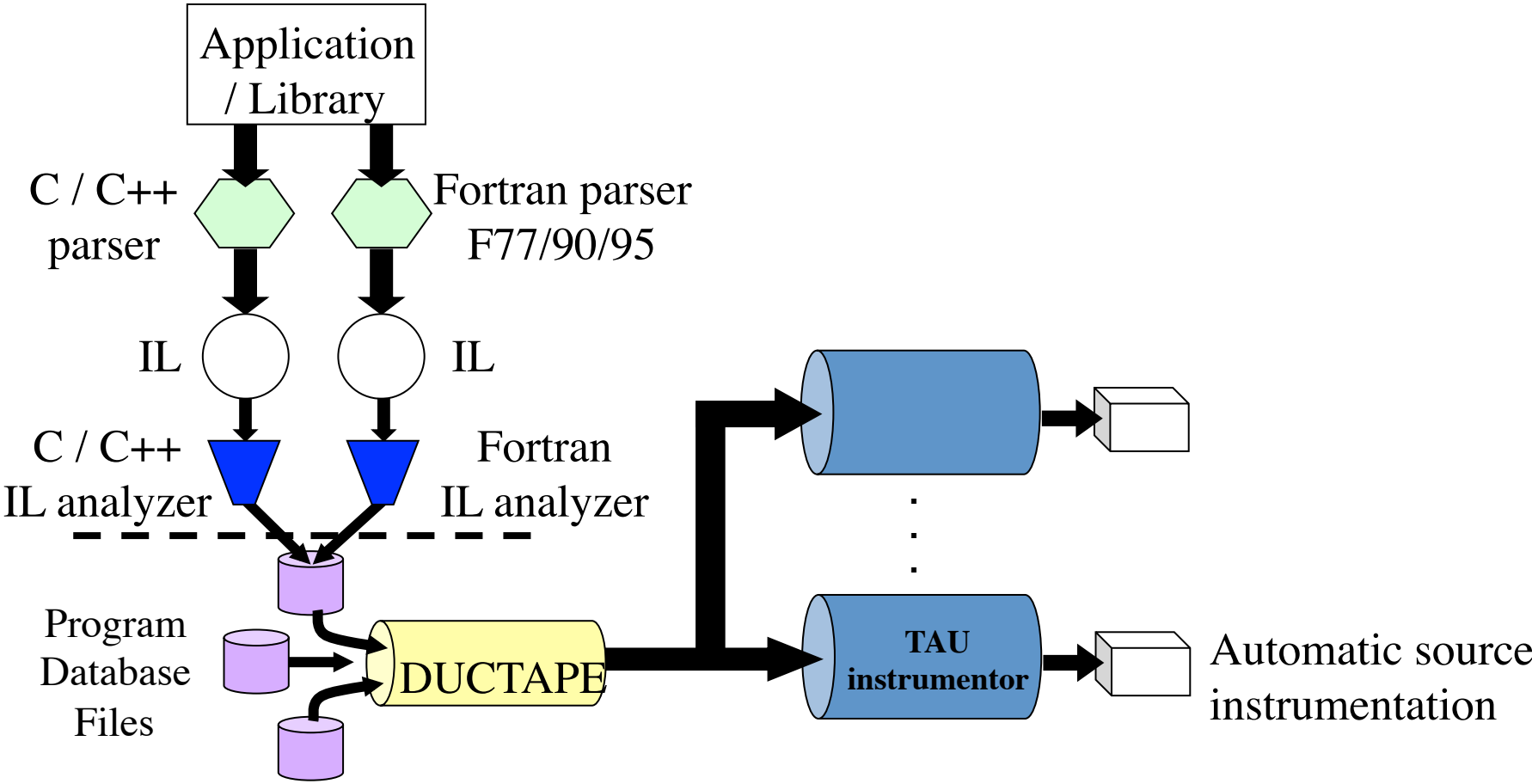
```
% export TAU_MAKEFILE=$TAU/Makefile.tau-papi-mpi-pdt
% tau_f90.sh matrix.f90 -o matrix
OR with build systems:
% make CC=tau_cc.sh CXX=tau_cxx.sh F90=tau_f90.sh
% cmake -DCMAKE_Fortran_COMPILER=tau_f90.sh
       -DCMAKE_C_COMPILER=tau_cc.sh -DCMAKE_CXX_COMPILER=tau_cxx.sh
% mpirun -np 1024 ./matrix
% paraprof
```

Configuration tags for tau_exec

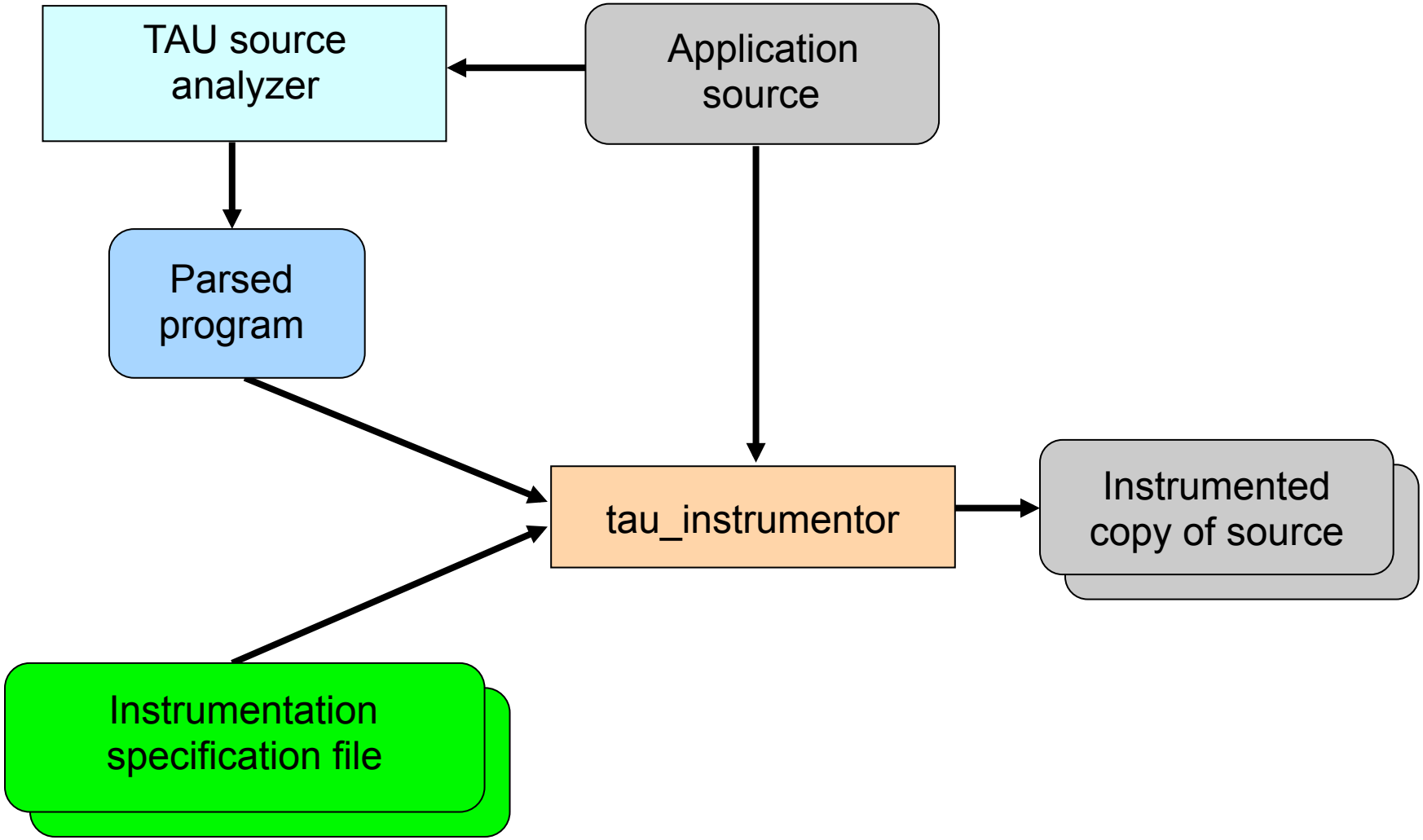
```
% ./configure -pdt=<dir> -mpi -papi=<dir>; make install
```
Creates in $TAU:
```
Makefile.tau-papi-mpi-pdt(Configuration parameters in stub makefile)
shared-papi-mpi-pdt/libTAU.so

% ./configure -pdt=<dir> -mpi; make install  creates
Makefile.tau-mpi-pdt
shared-mpi-pdt/libTAU.so
```
To explicitly choose preloading of shared-<options>/libTAU.so change:
```
% mpirun -np 256 ./a.out     to
% mpirun -np 256  tau_exec -T <comma_separated_options> ./a.out

% mpirun -np 256  tau_exec -T papi,mpi,pdt ./a.out
```
Preloads `$TAU/shared-papi-mpi-pdt/libTAU.so`
```
% mpirun -np 256  tau_exec -T papi ./a.out
```
Preloads `$TAU/shared-papi-mpi-pdt/libTAU.so` by matching.
```
% mpirun -np 256 tau_exec -T papi,mpi,pdt -s ./a.out
```
Does not execute the program. Just displays the library that it will preload if executed without the **–s** option.

NOTE: -mpi configuration is selected by default. Use –T serial for Sequential programs.

TAU's Static Analysis System: Program Database Toolkit (PDT)



Automatic Source Instrumentation using PDT



Automatic Instrumentation

- **Use TAU's compiler wrappers**
  - Simply replace `CXX` with `tau_cxx.sh, etc.`
  - Automatically instruments source code, links with TAU libraries.
- **Use `tau_cc.sh` for C, `tau_f90.sh` for Fortran, `tau_upc.sh` for UPC, etc.**

Before

```
% cat Makefile
CXX = mpicxx
F90 = mpif90
CXXFLAGS =
LIBS = -lm
OBJS = f1.o f2.o f3.o … fn.o

app: $(OBJS)
      $(CXX) $(LDFLAGS) $(OBJS) -o $@
      $(LIBS)
.cpp.o:
      $(CXX) $(CXXFLAGS) -c $<

% make
```

After

```
% cat Makefile
CXX = tau_cxx.sh
F90 = tau_f90.sh
CXXFLAGS =
LIBS = -lm
OBJS = f1.o f2.o f3.o … fn.o

app: $(OBJS)
      $(CXX) $(LDFLAGS) $(OBJS) -o $@
      $(LIBS)
.cpp.o:
      $(CXX) $(CXXFLAGS) -c $<

% export TAU_MAKEFILE=
      $TAU/Makefile.tau-papi-mpi-pdt
% make
```

Selective Instrumentation File

```
% export TAU_OPTIONS='-optTauSelectFile=select.tau …'
% cat select.tau
BEGIN_INCLUDE_LIST
int main#
int dgemm#
END_INCLUDE_LIST
BEGIN_FILE_INCLUDE_LIST
Main.c
Blas/*.f77
END_FILE_INCLUDE_LIST
# replace include with exclude list

BEGIN_INSTRUMENT_SECTION
loops routine="foo"
loops routine="int main#"
END_INSTRUMENT_SECTION
```

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
  - ./configure –mpi -bfd=download -pdt=<dir> -papi=<dir> ...
  - make install
- **Using TAU:**
  - export TAU_MAKEFILE=<taudir>/x86_64/ lib/Makefile.tau-<TAGS>
  - % export TAU_OPTIONS='-optTauSelectFile=select.tau'
  - make CC=tau_cc.sh CXX=tau_cxx.sh F90=tau_f90.sh

Compile-Time Options

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

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

Compile-Time Options (contd.)

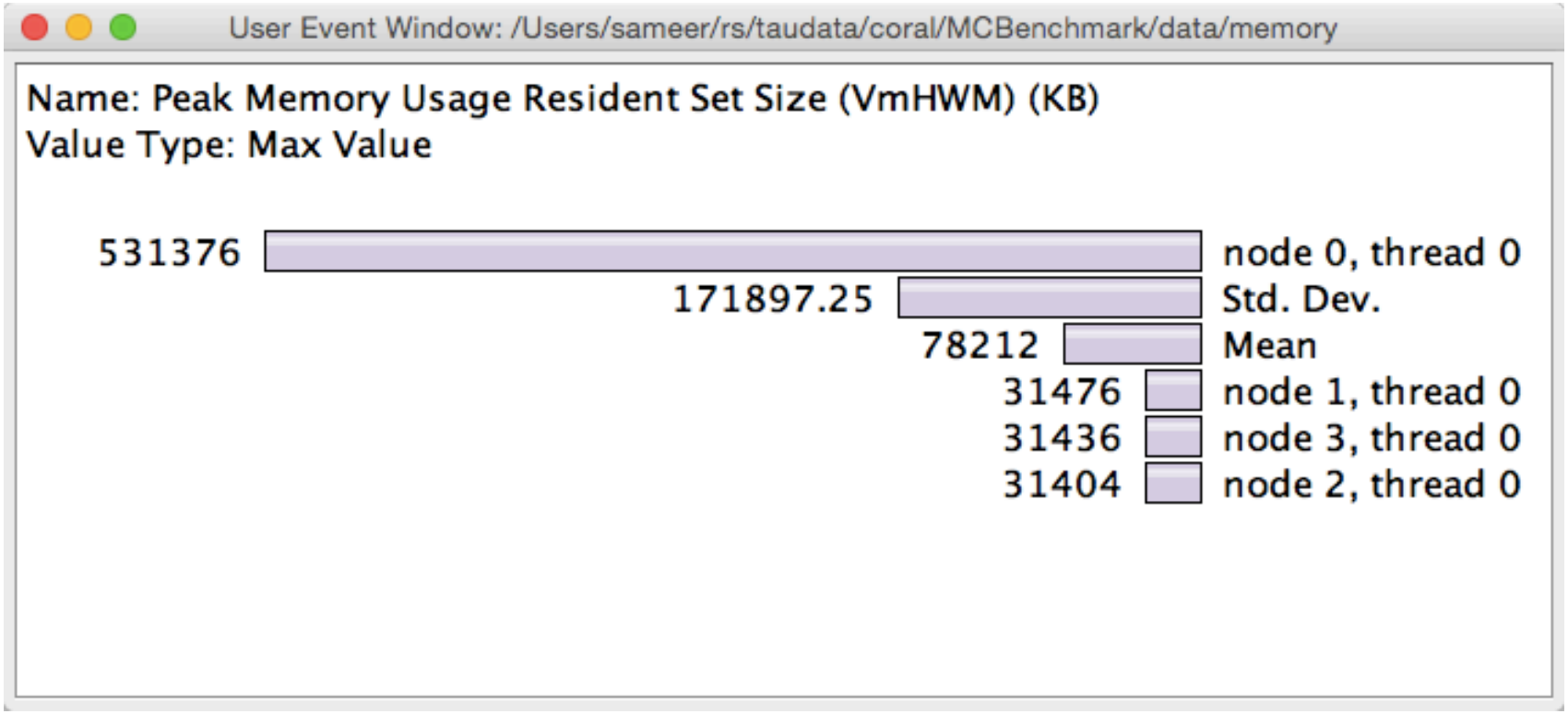
Optional parameters for the TAU_OPTIONS environment variable:
% tau_compiler.sh

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

See tau_compiler.sh for a full list of TAU_OPTIONS.

…

Measuring Memory Footprint



% export TAU_TRACK_MEMORY_FOOTPRINT=1
Paraprof:
Right click on a node -> Show Context Event Window -> see memory events

Other Runtime Environment Variables

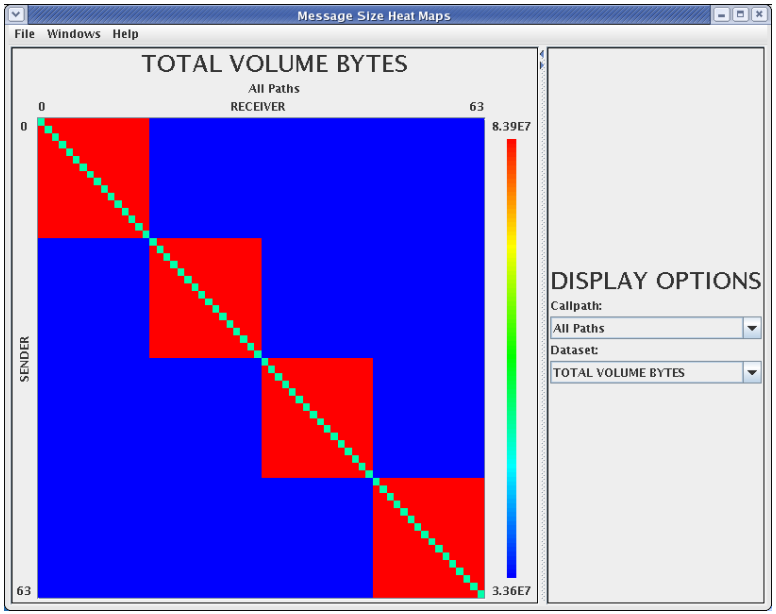
Environment Variable	Default	Description
TAU_TRACE	0	Setting to 1 turns on tracing
TAU_CALLPATH	0	Setting to 1 turns on callpath profiling
TAU_TRACK_MEMORY_FOOTPRINT	0	Setting to 1 turns on tracking memory usage by sampling periodically the resident set size and high water mark of memory usage
TAU_TRACK_POWER	0	Tracks instantaneous 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	0	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. Enabled by default to remove 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_COMPENSATE	0	Setting to 1 enables runtime compensation of instrumentation overhead
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., TIME,ENERGY,PAPI_FP_INS,PAPI_NATIVE_<event>:<subevent>)

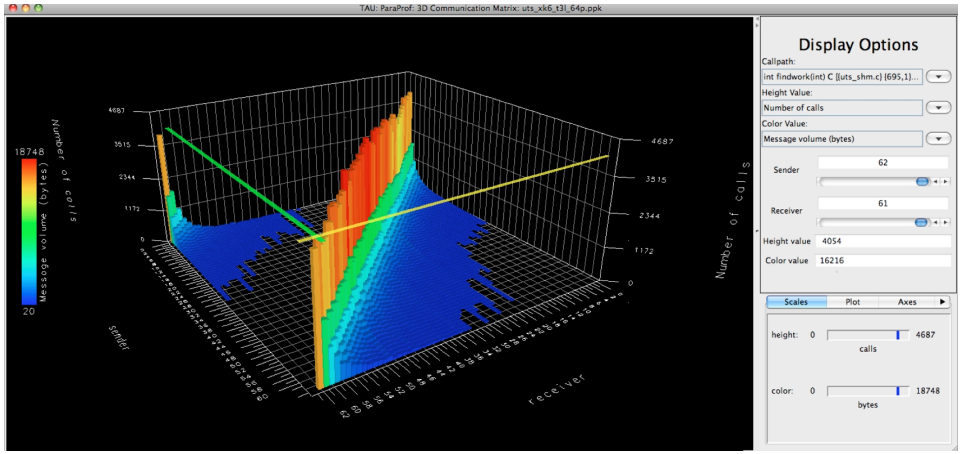
Runtime Environment Variables (contd.

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

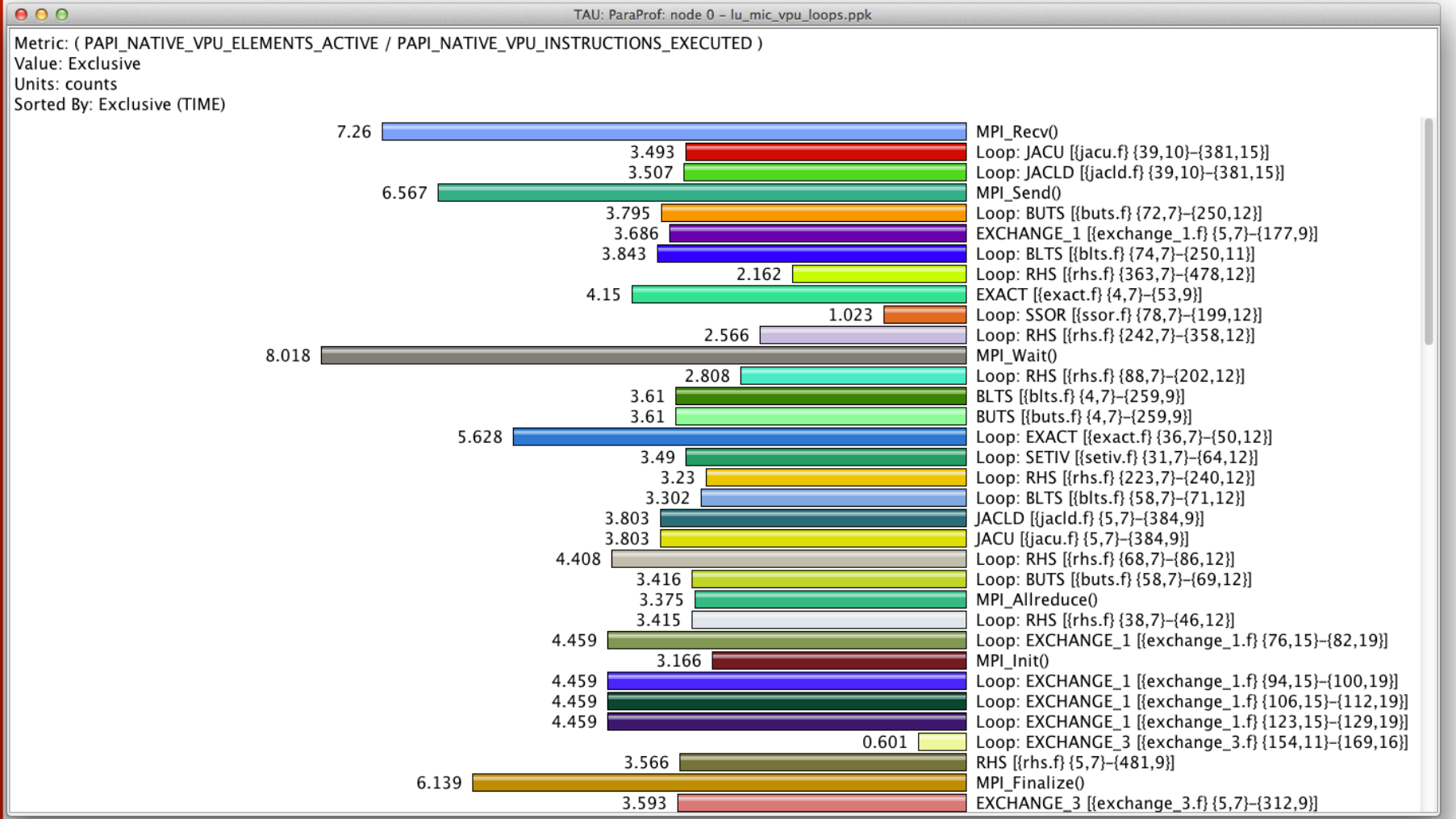
Communication Matrix Display

Goal: What is the volume of inter-process communication? Along which calling path?



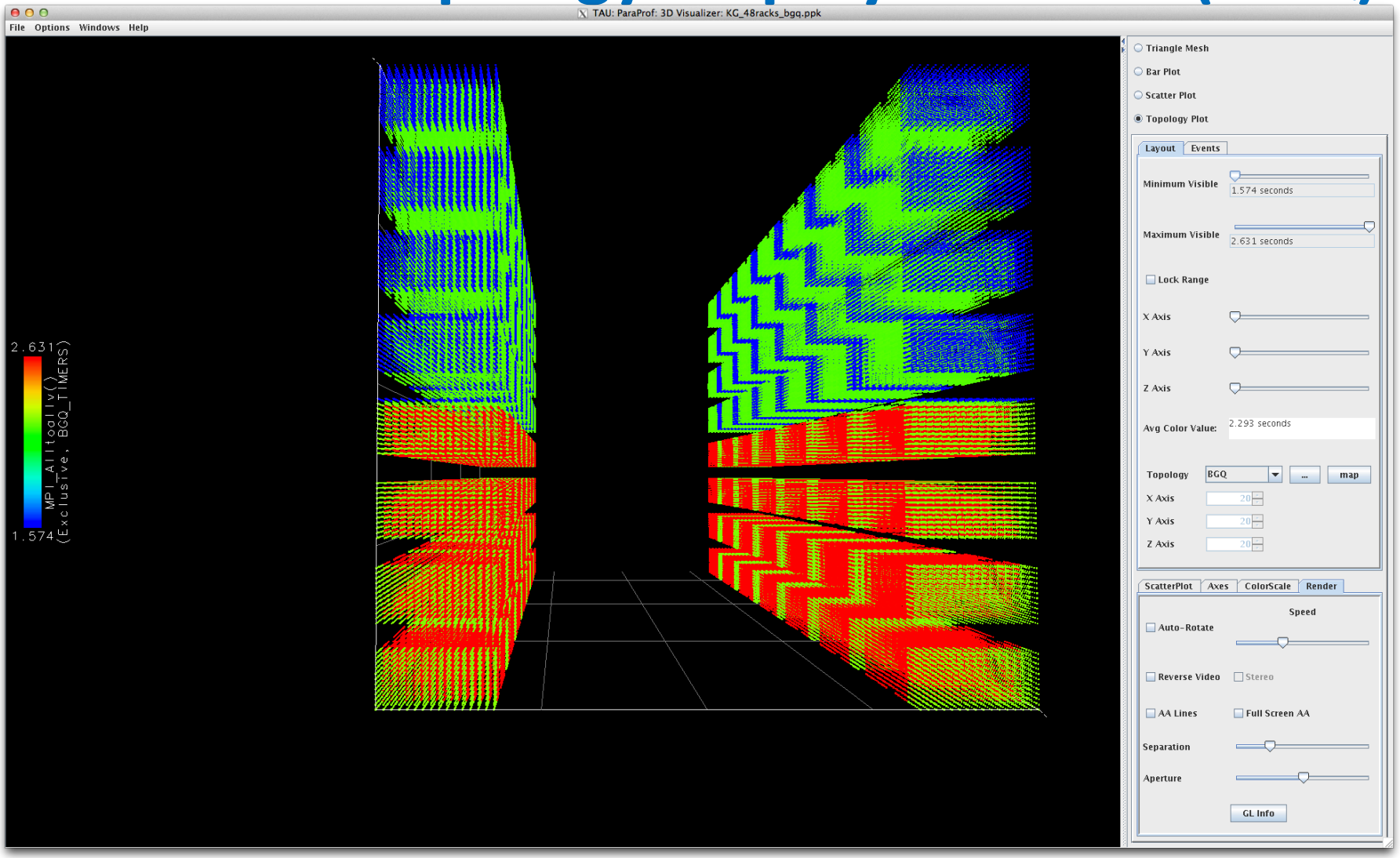


Evaluating Extent of Vectorization on MIC

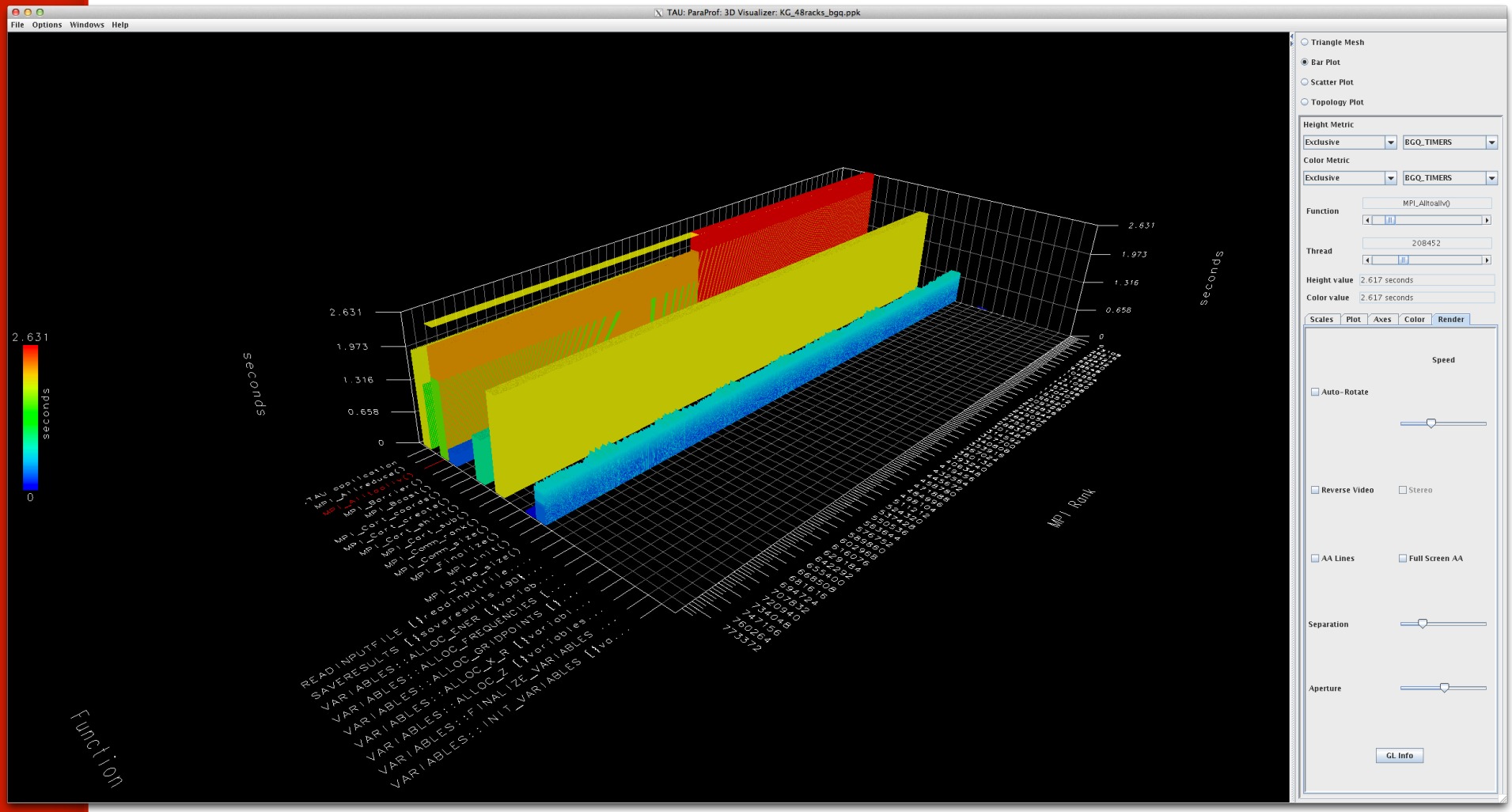



```
% export TAU_MAKEFILE=$TAUROOT/mic_linux/lib/Makefile.tau-papi-mpi-pdt
% export TAU_METRICS=TIME,
        PAPI_NATIVE_VPU_ELEMENTS_ACTIVE,PAPI_NATIVE_VPU_INSTRUCTIONS_EXECUTED
```

ParaProf's Topology Display Window (BGQ)

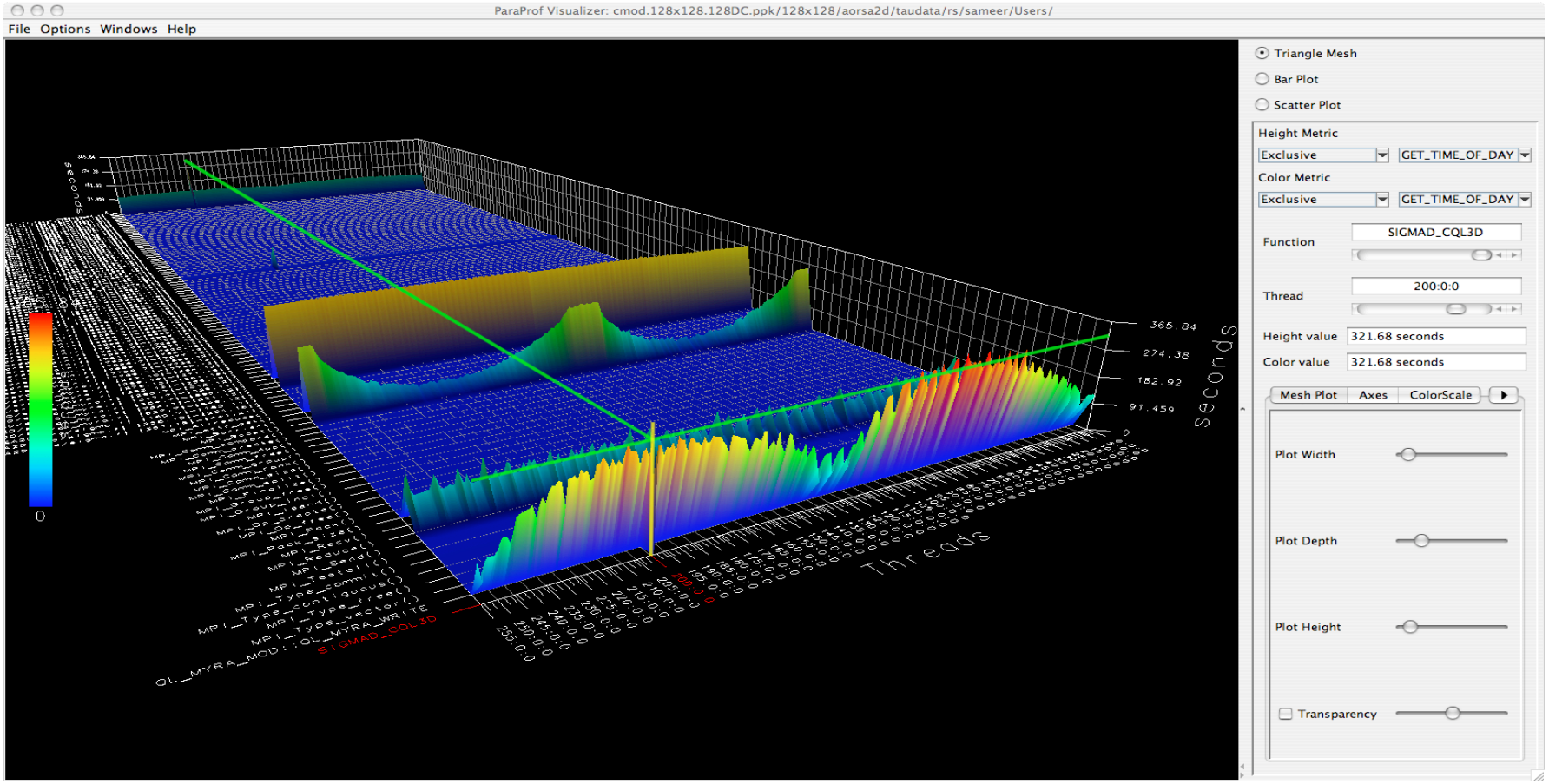


ParaProf's Scalable 3D Visualization (BGQ)

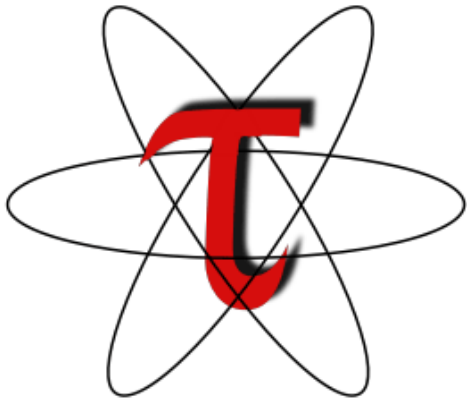


786,432 ranks

ParaProf 3D Profile Browser



Download TAU from U. Oregon



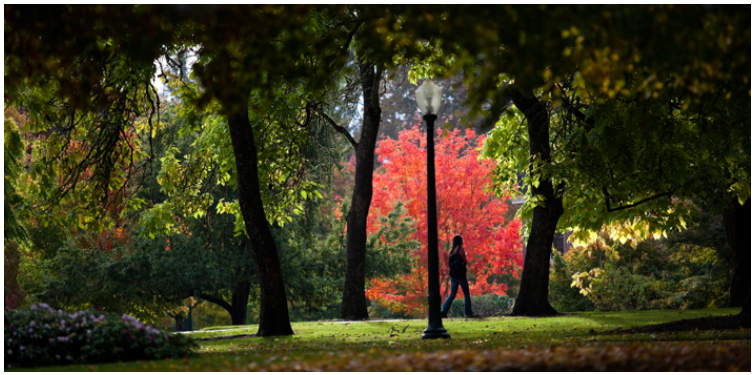
http://www.hpclinux.com [OVA file]

http://tau.uoregon.edu/tau.pptx

for more information

Free download, open source, BSD license

PRL, University of Oregon, Eugene



www.uoregon.edu

Support Acknowledgments

National Science Foundation (NSF)

- SI2-SSI, Glassbox

US Department of Energy (DOE)

- Office of Science contracts
- SciDAC, LBL contracts
- LLNL-LANL-SNL ASC/NNSA contract
- Battelle, PNNL contract
- ANL, ORNL contract

Department of Defense (DoD)

- PETTT, HPCMP

NASA

Partners:

University of Oregon

The Ohio State University

ParaTools, Inc.

University of Tennessee, Knoxville

T.U. Dresden, GWT

Juelich Supercomputing Center