TAU Performance System®

Sameer Shende, John Linford, Srinath Vadlamani, Sam Khuvis
University of Oregon and ParaTools, Inc.

sameer@cs.uoregon.edu, info@paratools.com

http://tau.uoregon.edu

NCAR Center Green Campus, Boulder, CO

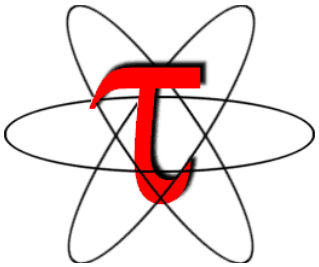
Monday, April 4, 2016, 2:30pm

Download slides from:

http://paratools.com/SEA16

TAU Performance System®

http://tau.uoregon.edu



- **Tuning and Analysis Utilities (20+ 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)
- **Integrates with application frameworks**

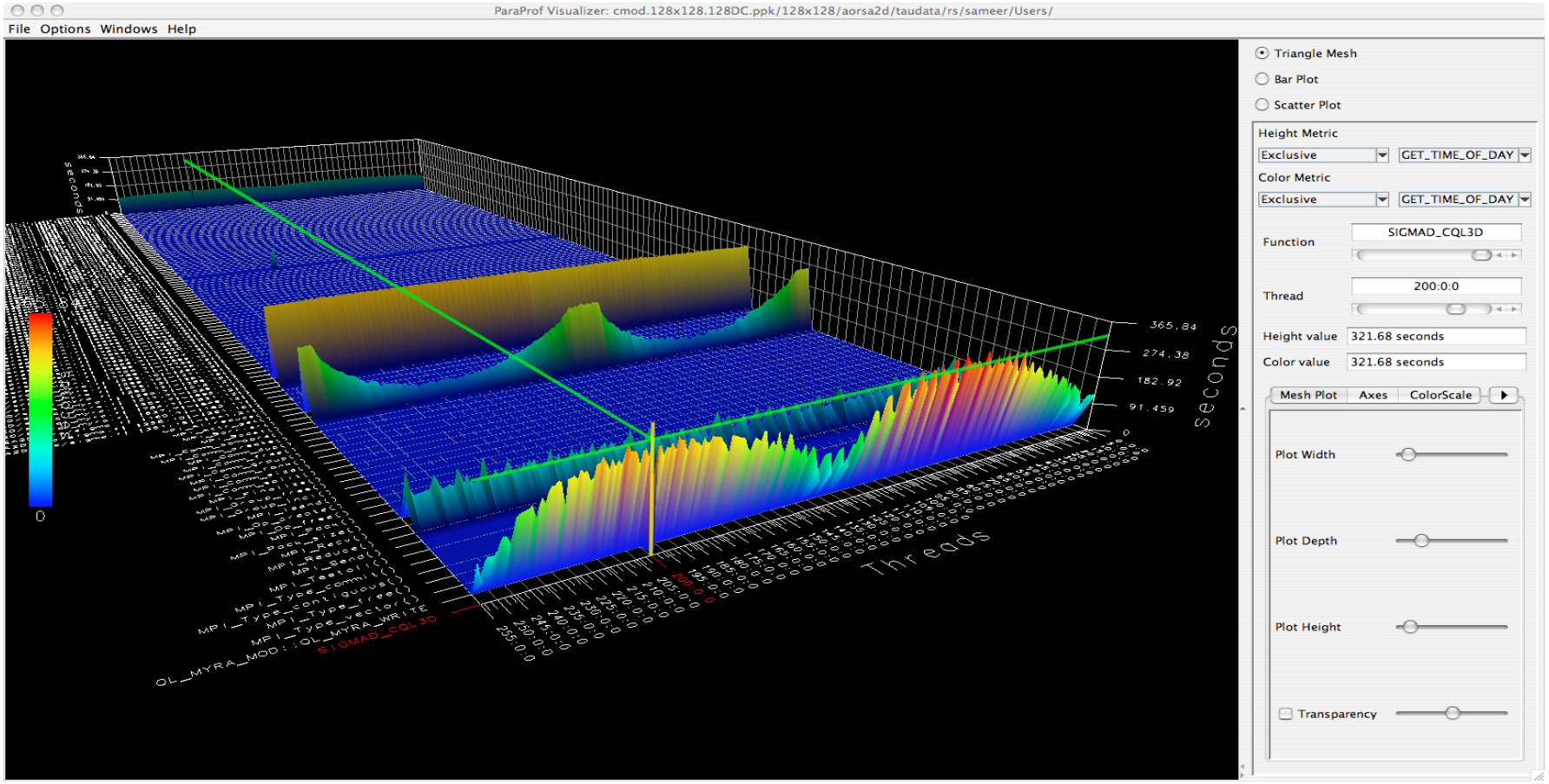
Understanding Application Performance using TAU

- **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?

Setting up workshop materials

```
On yellowstone.ucar.edu:
% module use /glade/apps/opt/ParaTools/modulefiles
% module load workshop/sea16
% tar xvzf $WORKSHOP_MATERIALS
% cd tau-sea16; cat README


mpirun.lsf ./matmult
Changes to:
mpirun.lsf tau_exec -ebs ./matmult

NOTE: Workshop materials also available at:
http://www.paratools.com/SEA16
```

ParaProf 3D Profile Browser

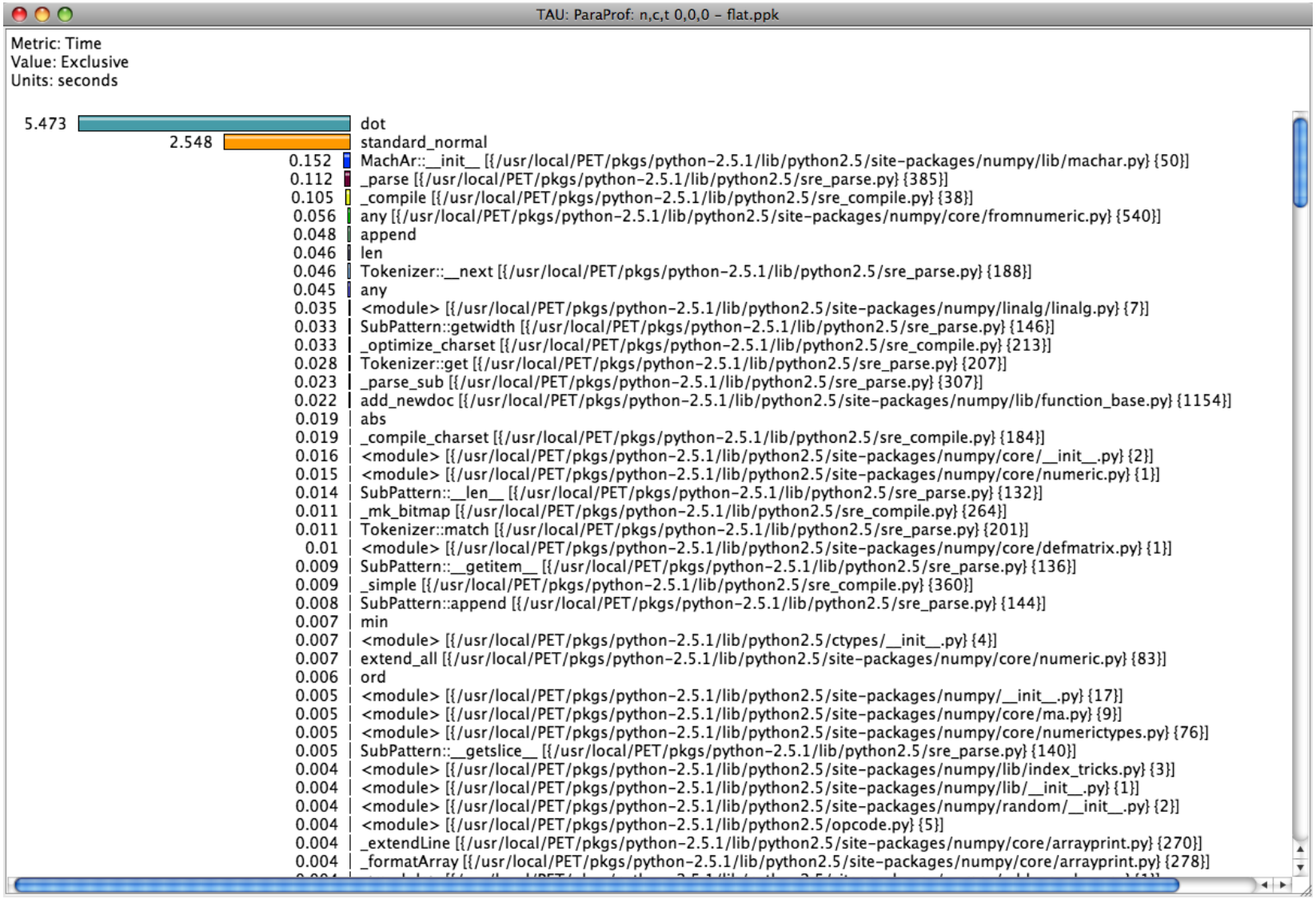


Profiling Python applications

- Create a top-level Python wrapper
- Launch tau_python …

```
% mpirun.lsf python ./app.py
% mpirun.lsf tau_python ./app.py

% paraprof
```

Profiling Python applications



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)

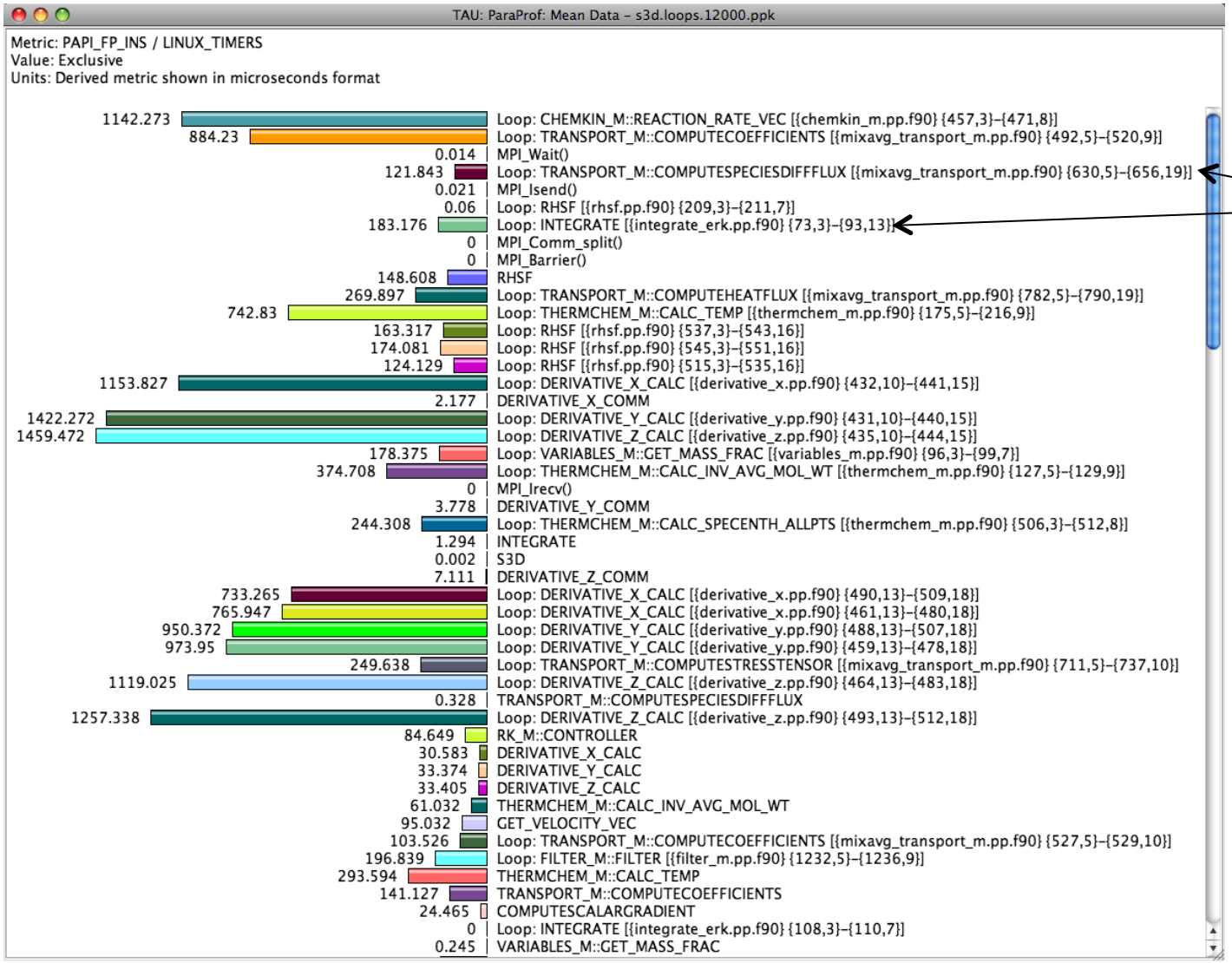
- % mpirun –np 256 tau_exec –ebs ./a.out
- Also –ebs_source=<PAPI_COUNTER> -ebs_period=<overflow_count>

Advanced TAU

New Features in TAU

- *tau_exec*: A tool to simplify TAU's usage
- PDT: New parsers from ROSE Compiler, LLNL [Dan Quinlan, CASC]
- Power profiling in TAU
- Support for accelerators: CUDA, OpenCL, Intel Xeon Phi Co-processor
- Event based sampling
- OpenMP instrumentation using OpenMP Tools Interface with Intel compilers (OMPT)
- Support for Intel TBB, CILK, and MPC [paratools.com/mpc]
- Binary rewriting with MAQAO (Beta)
- ParaProf 3D topology displays
- TAUdb using H2 database

Identifying Potential Bottlenecks



low mflops in loops?

OpenACC with PGI compilers

TAU: ParaProf: Statistics for: node 0, thread 0 - benchADM_acc2.ppk

Name	Exclusive...	Inclusive... ▽	Calls	Child...
.TAU application	4.982	9.443	1	5,168
openacc_enqueue_upload bench_staggeredleapfrog2 [{/storage	0.694	3.35	3,700	29,867
cuMemcpyHtoDAsync_v2	2.47	2.47	3,700	0
cuEventRecord	0.06	0.06	7,400	0
cuDeviceGetCount	0.032	0.032	7,401	0
cuEventElapsedTime	0.031	0.031	3,700	0
cuCtxSynchronize	0.031	0.031	3,700	0
cuEventSynchronize	0.028	0.028	3,700	0
cuDeviceGetAttribute	0.002	0.002	249	0
cuDeviceGetName	0	0	3	0
cuDeviceTotalMem_v2	0	0	3	0
cuCtxGetDevice	0	0	1	0
cuEventCreate	0	0	2	0
cuDeviceGet	0	0	3	0
cuDriverGetVersion	0	0	1	0
cuCtxGetCurrent	0	0	2	0
cuCtxGetApiVersion	0	0	1	0
cuInit	0	0	1	0
openacc_enqueue_download bench_staggeredleapfrog2 [{/stora	0.116	0.556	600	5,407
cuMemcpyDtoHAsync_v2	0.405	0.405	600	0
cuEventRecord	0.013	0.013	1,800	0
cuDeviceGetCount	0.005	0.005	1,200	0

% configure –c++=pgCC –cc=pgcc –fortran=pgi …
% tau_exec –T pgi -openacc -cupti ./a.out

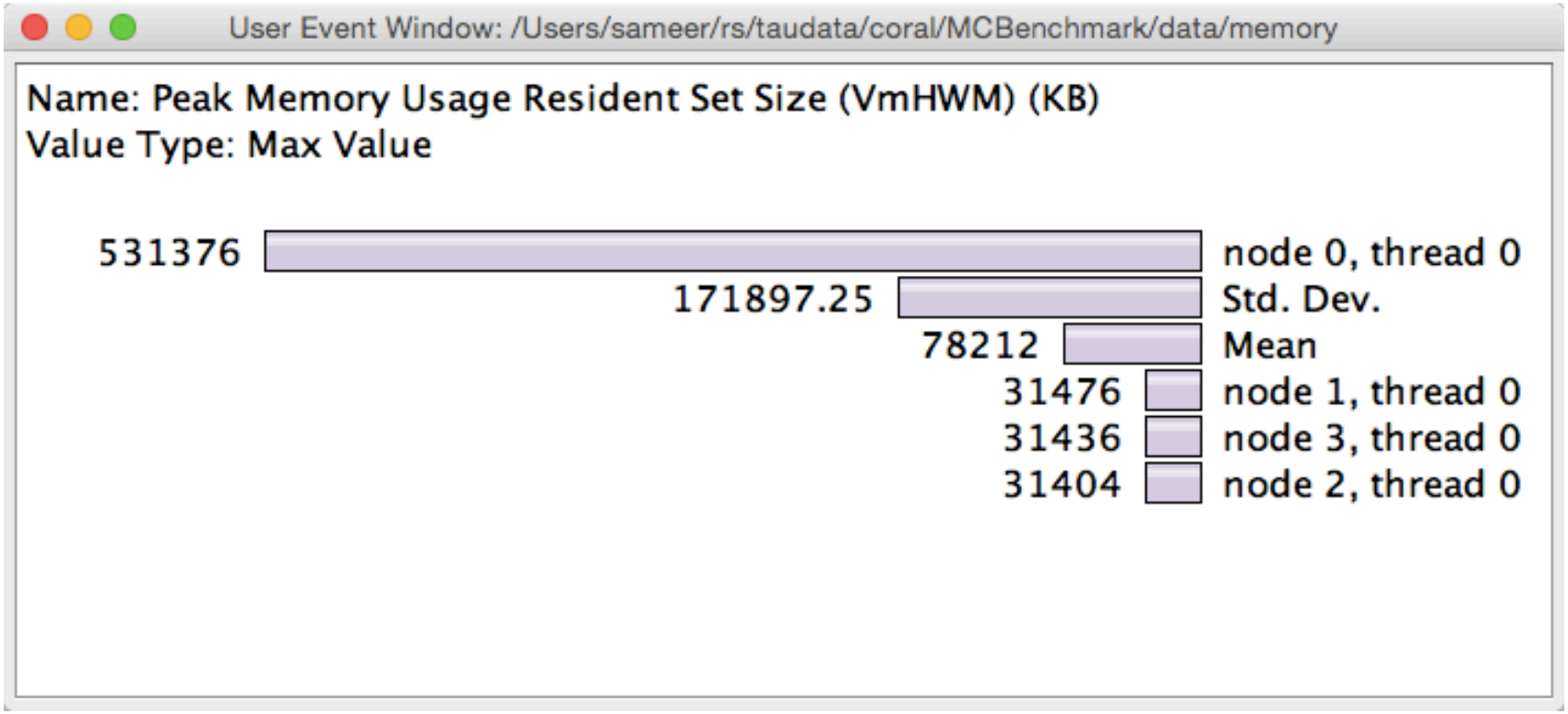
Tracking OpenACC Data Transfers

TAU: ParaProf: Context Events for: node 0, thread 0 - benchADM_acc2.ppk

Name	Total	NumSamples	MaxValue	MinValue	MeanValue	Std. Dev.
.TAU application						
openacc_enqueue_upload bench_staggeredleapfrog2 [{/st						
cuMemcpyHtoDAsync_v2						
[GROUP=MAX_MARKER] Bytes copied from Host to De	512,000	1	512,000	512,000	512,000	0
Bytes copied from Host to Device	973,016,000	3,700	512,000	120	262,977.297	255,846.506
openacc_enqueue_download bench_staggeredleapfrog2 [{						
cuMemcpyDtoHAsync_v2						
Bytes copied from Device to Host	307,200,000	600	512,000	512,000	512,000	0
Bytes copied from Device to Host	307,200,000	600	512,000	512,000	512,000	0
Bytes copied from Host to Device	973,016,000	3,700	512,000	120	262,977.297	255,846.506
[GROUP=MAX_MARKER] Bytes copied from Host to Device	512,000	1	512,000	512,000	512,000	0

% configure –c++=pgCC –cc=pgcc –fortran=pgi …
% tau_exec –T pgi -openacc -cupti ./a.out
Context events show extent of variation

Measuring Memory Footprint



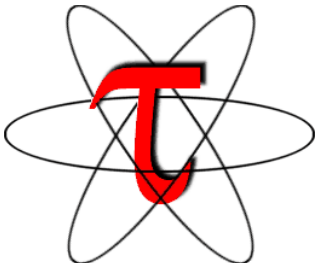
% export TAU_TRACK_MEMORY_FOOTPRINT=1

Paraprof:

Right click on a node -> Show Context Event Window -> see memory events

TAU Performance System®

http://tau.uoregon.edu



- **Tuning and Analysis Utilities (20+ 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)
- **Integrates with application frameworks**

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 is the high water mark of the memory footprint?
- **What is the energy and power** usage of the code?
- **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?

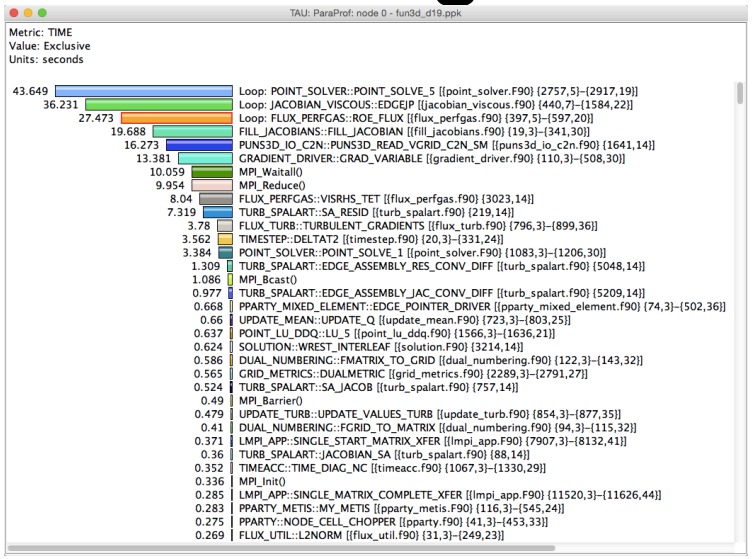
What does TAU support?

C/C++
Fortran
pthreads
Intel
GNU
MPC
CUDA
UPC
OpenACC
Intel MIC
LLVM
PGI
Linux
BlueGene
NVIDIA
OpenCL
Python
GPI
Java
MPI
OpenMP
Cray
Sun
AIX
Windows
Fujitsu
ARM64
Power 8
OS X

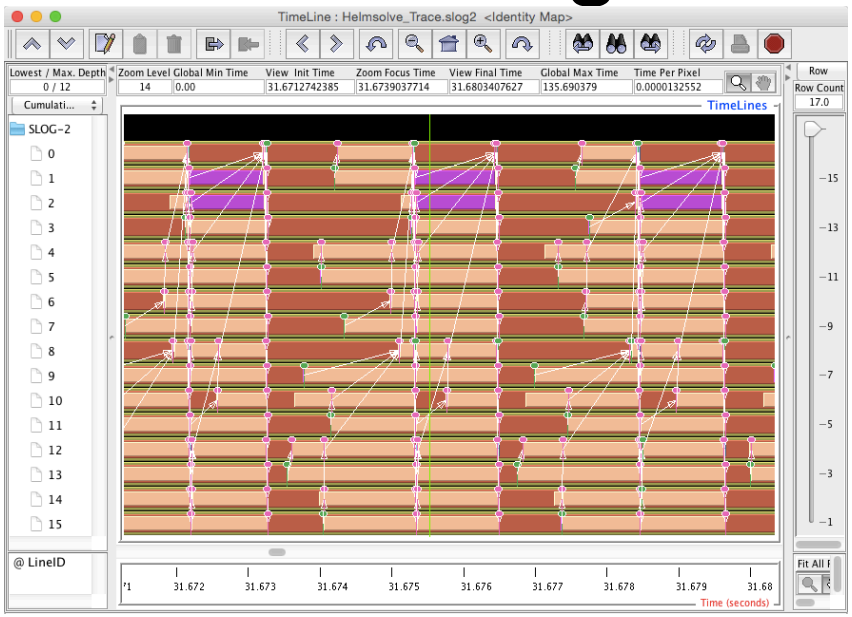
Insert yours here

Profiling and Tracing

Profiling



Tracing



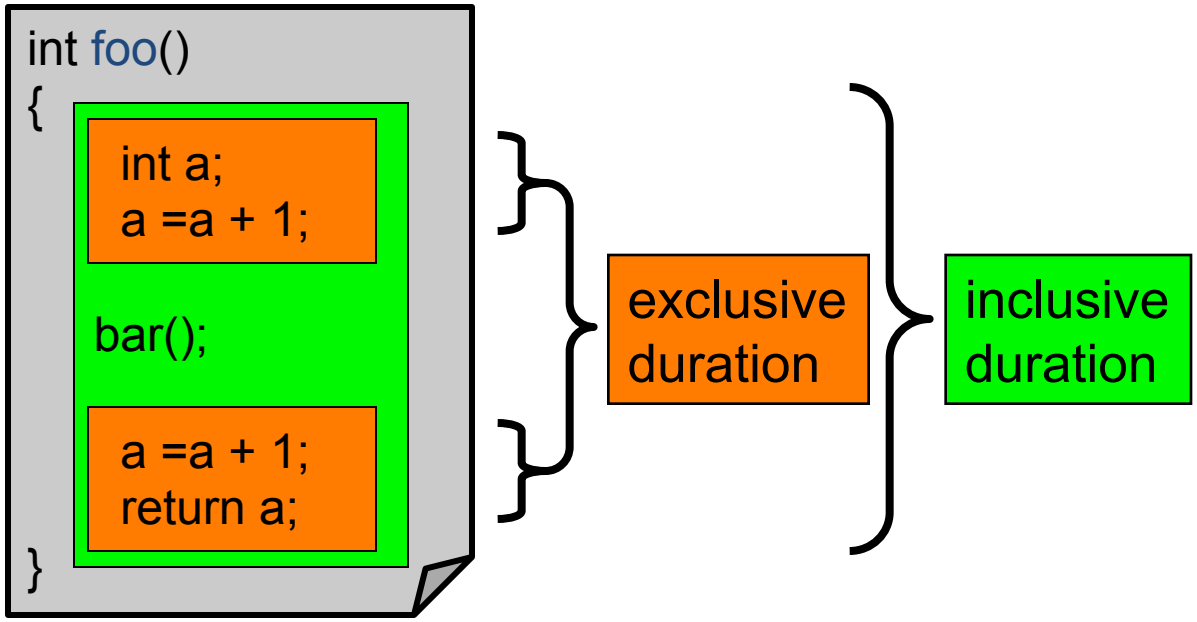
- Profiling and tracing

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

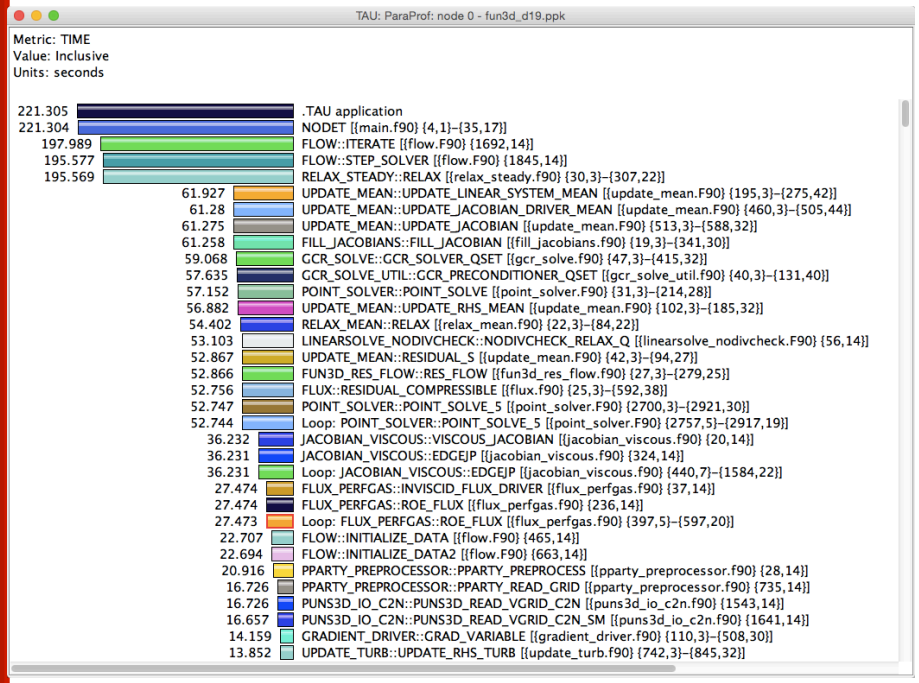
  Tracing shows you **when** the events take place on a timeline

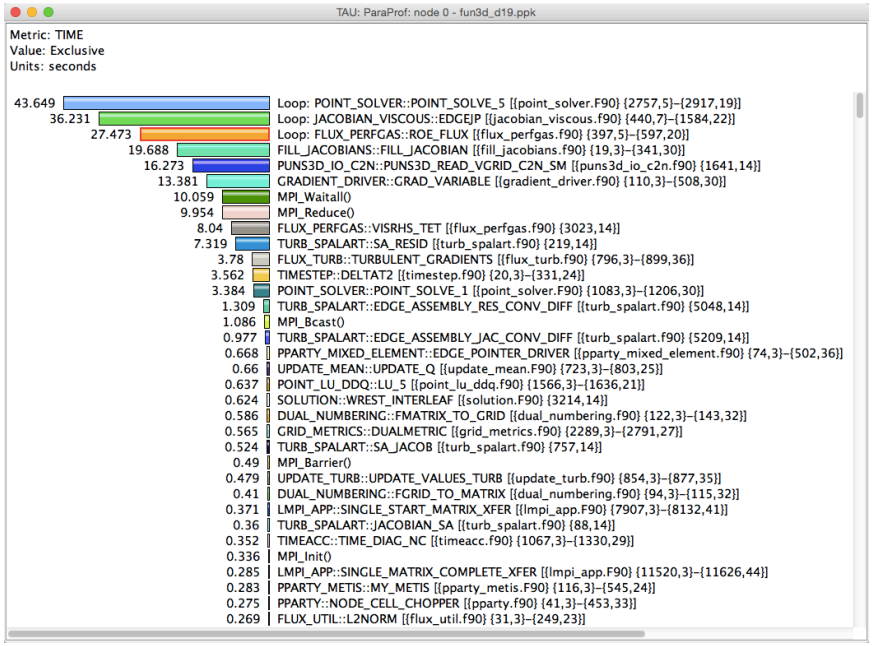
Inclusive vs. Exclusive Measurements

- Performance with respect to code regions
- Exclusive measurements for region only
- Inclusive measurements includes child regions



Inclusive vs. Exclusive Measurements

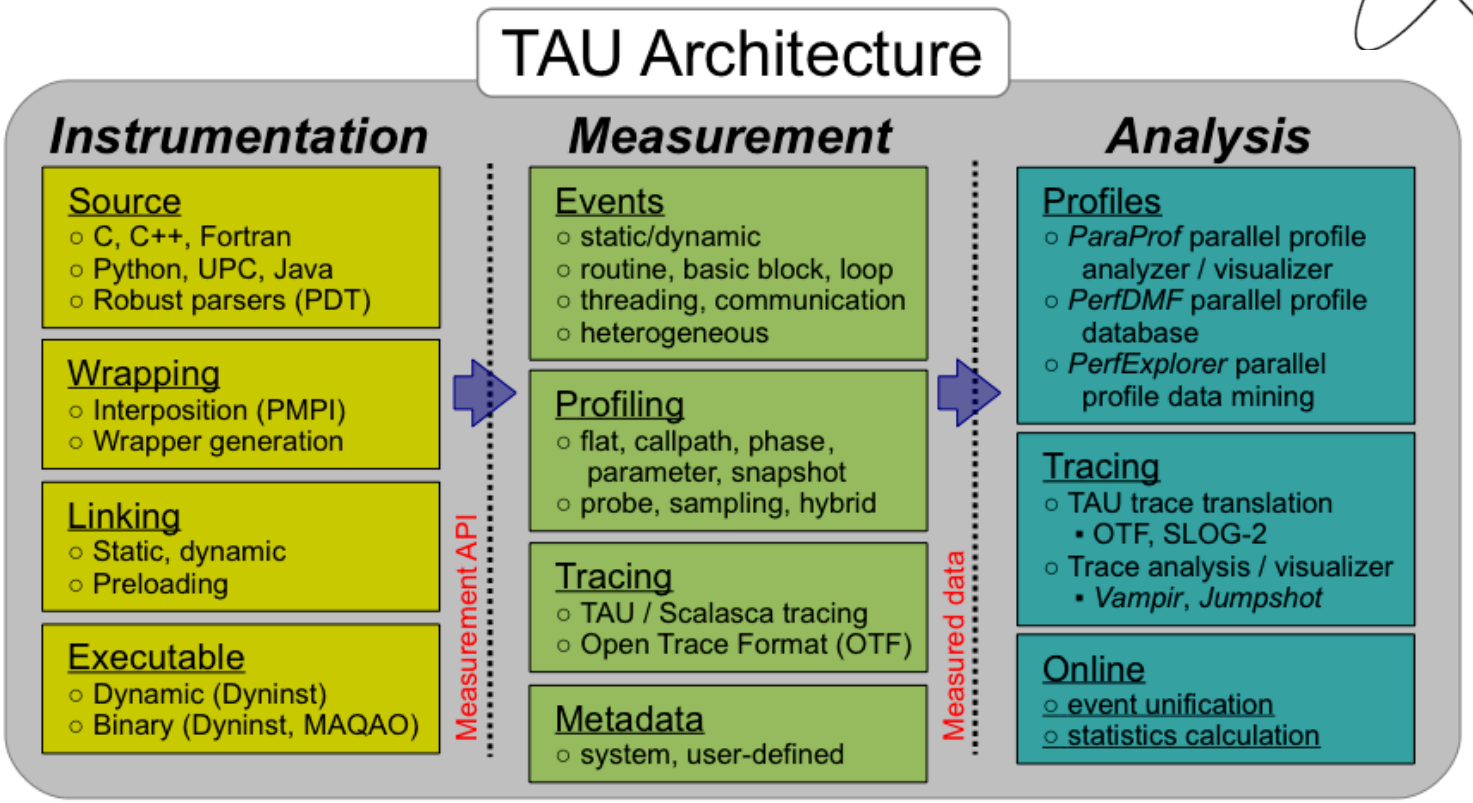




Inclusive time

Exclusive time

TAU Architecture and Workflow



TAU Architecture and Workflow

Instrumentation: **Add probes to perform measurements**

- Source code instrumentation using pre-processors and compiler scripts
- Wrapping external libraries (I/O, MPI, Memory, CUDA, OpenCL, pthread)
- Rewriting the binary executable

Measurement: **Profiling or tracing using various metrics**

- Direct instrumentation (Interval events measure exclusive or inclusive duration)
- Indirect instrumentation (Sampling measures statement level contribution)
- Throttling and runtime control of low-level events that execute frequently
- Per-thread storage of performance data
- Interface with external packages (e.g. PAPI hw performance counter library)

Analysis: **Visualization of profiles and traces**

- 3D visualization of profile data in paraprof or perfexplorer tools
- Trace conversion & display in external visualizers (Vampir, Jumpshot, ParaVer)

Instrumentation

Direct and indirect performance observation

- Instrumentation invokes performance measurement
- Direct measurement with *probes*
- Indirect measurement with periodic sampling or hardware performance counter overflow interrupts
- Events measure performance data, metadata, context, etc.

User-defined events

- ***Interval*** (start/stop) events to measure exclusive & inclusive duration
- ***Atomic events*** take measurements at a single point
  - Measures total, samples, min/max/mean/std. deviation statistics
- ***Context events*** are atomic events with executing context
  - Measures above statistics for a given calling path

Direct Observation Events

Interval events (begin/end events)

- Measures exclusive & inclusive durations between events
- Metrics monotonically increase
- Example: Wall-clock timer

Atomic events (trigger with data value)

- Used to capture performance data state
- Shows extent of variation of triggered values (min/max/mean)
- Example: heap memory consumed at a particular point

Code events

- Routines, classes, templates
- Statement-level blocks, loops
- Example: for-loop begin/end

Direct Instrumentation Options in TAU

Source Code Instrumentation

- Automatic instrumentation using pre-processor based on static analysis of source code (PDT), creating an instrumented copy
- Compiler generates instrumented object code
- Manual instrumentation

Library Level Instrumentation

- Statically or dynamically linked wrapper libraries
  - MPI, I/O, memory, etc.
- Wrapping external libraries where source is not available

Runtime pre-loading and interception of library calls

Binary Code instrumentation

- Rewrite the binary, runtime instrumentation

Virtual Machine, Interpreter, OS level instrumentation

Examples

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:

% source tau.bashrc (or module load tau…)

% export TAU_MAKEFILE=$TAU/Makefile.tau-icpc-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

Makefile.tau-intel12-mpich2-icpc-mpi-pdt

Makefile.tau-intel12-mpich2-icpc-mpi-python-pdt

Makefile.tau-intel16-mpich2-icpc-mpi-pdt

Makefile.tau-intel16-mpich2-icpc-mpi-python-pdt

Makefile.tau-intel16-mpich2-icpc-ompt-mpi-pdt-openmp

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

Makefile.tau-intel16-mpich2-icpc-mpi-pdt

- Supports MPI instrumentation & PDT for automatic source instrumentation

```
% export TAU_MAKEFILE=$TAU/Makefile.tau-intel16-mpich2-icpc-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.lsf ./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.

Binary Rewriting Instrumentation

- Support for both **static and dynamic** executables
- Specify a list of routines to instrument
- Specify the TAU measurement library to be injected
- **Dyninst [U. Wisconsin, U. Maryland]**:

```
% tau_run -T [tags] a.out -o a.inst
```

- **MAQAO [Intel Exascale Labs, UVSQ]**:

```
% tau_rewrite -T [tags] a.out -o a.inst
```

- **Pebil [SDSC]**:

```
% tau_pebil_rewrite -T [tags] a.out \
  -o a.inst
```

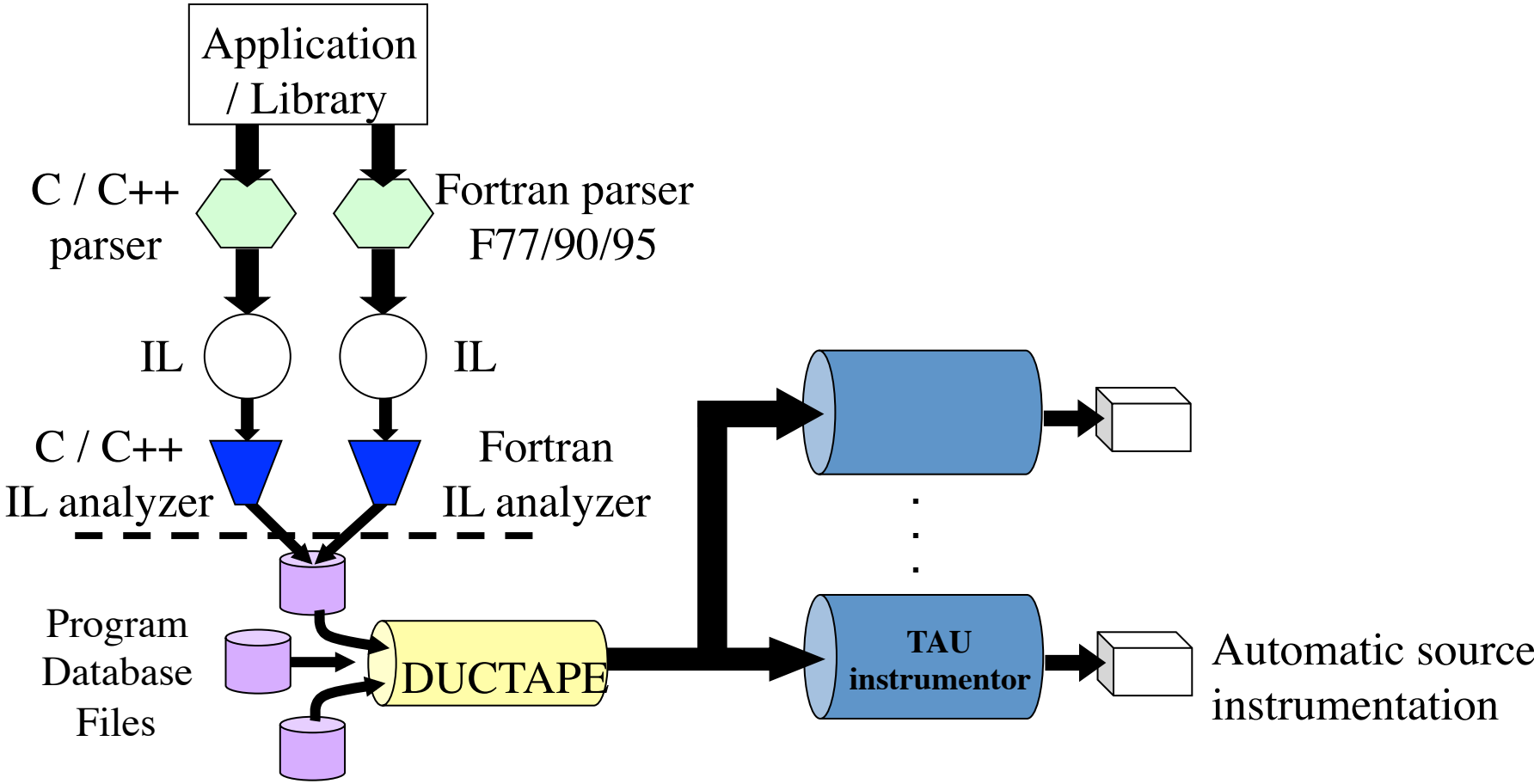
- Execute the application to get measurement data:

```
% mpirun -np 4  ./a.inst
```

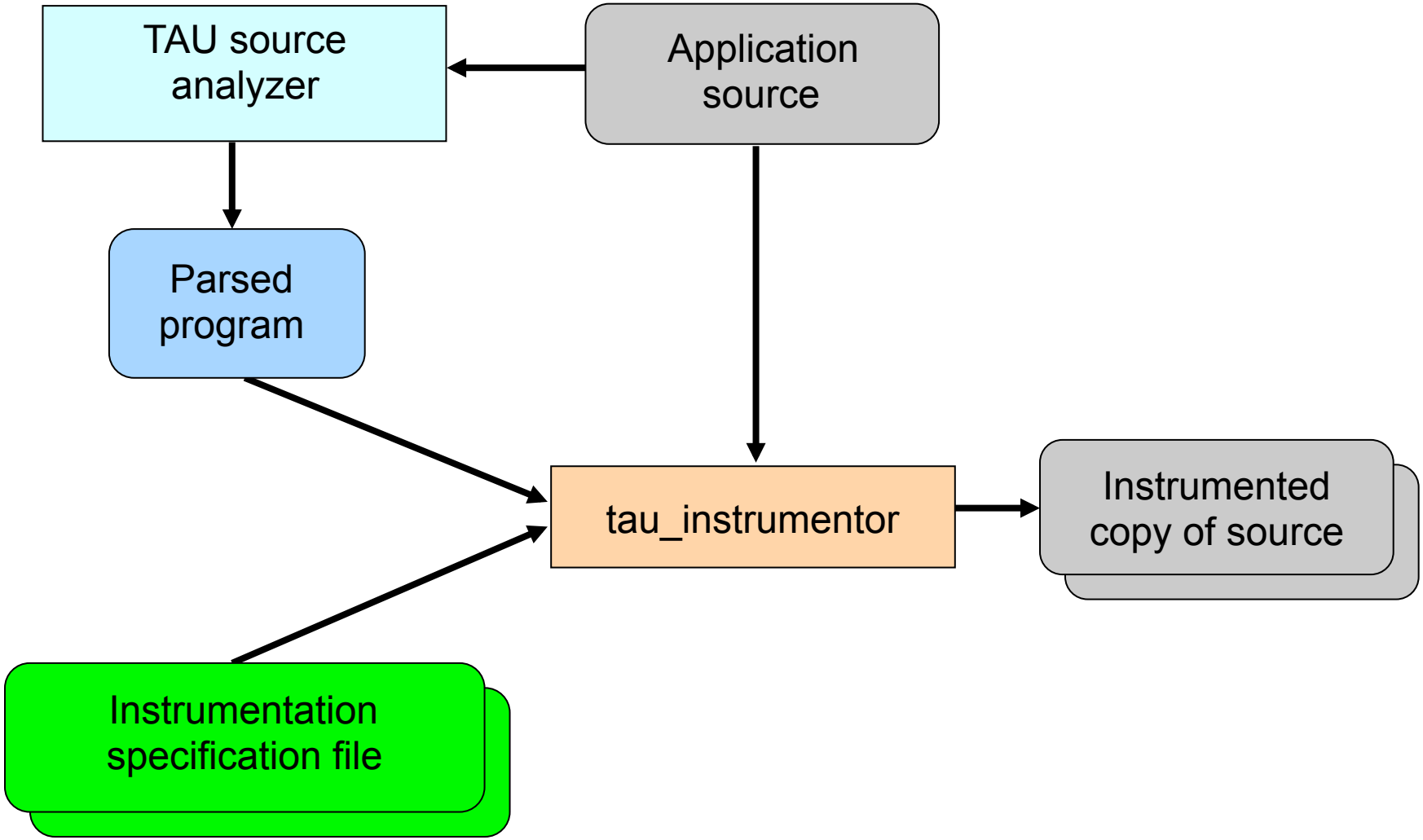
Binary Rewriting Instrumentation

```
% mpif90 -g matmult.f90 -o matmult
% tau_rewrite matmult matmult.i

Or use a selective instrumentation file (include/exclude lists)
% tau_rewrite -f select.tau -T icpc,papi ./matmult -o matmult.i
% mpirun -np 256  ./matmult.i
% paraprof
```

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



Automatic Source Instrumentation using PDT



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
```

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

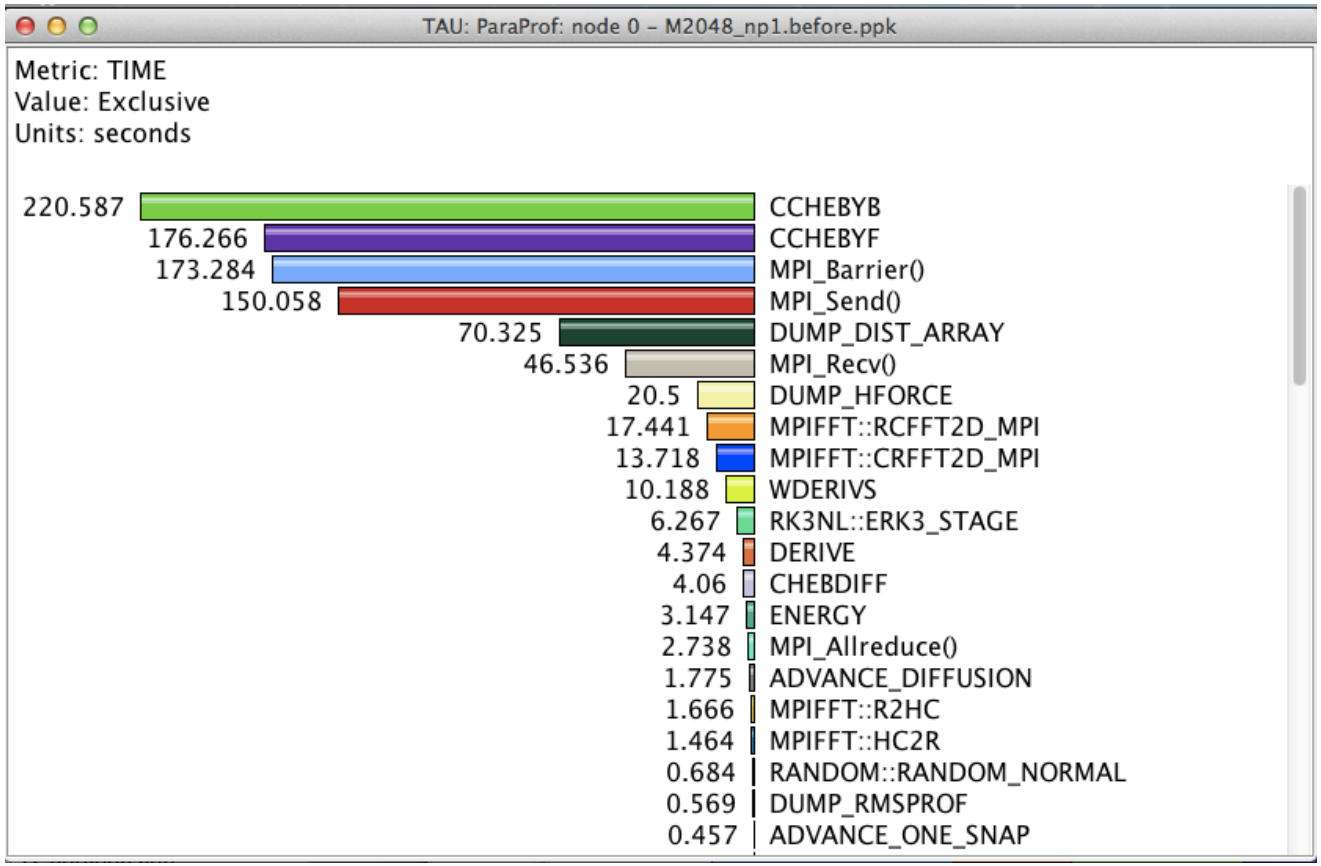
```
CXX = mpicxx
F90 = mpif90
CXXFLAGS =
LIBS = -lm
OBJS = f1.o f2.o f3.o … fn.o

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

After

```
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 $<
```

Routine Level Profile

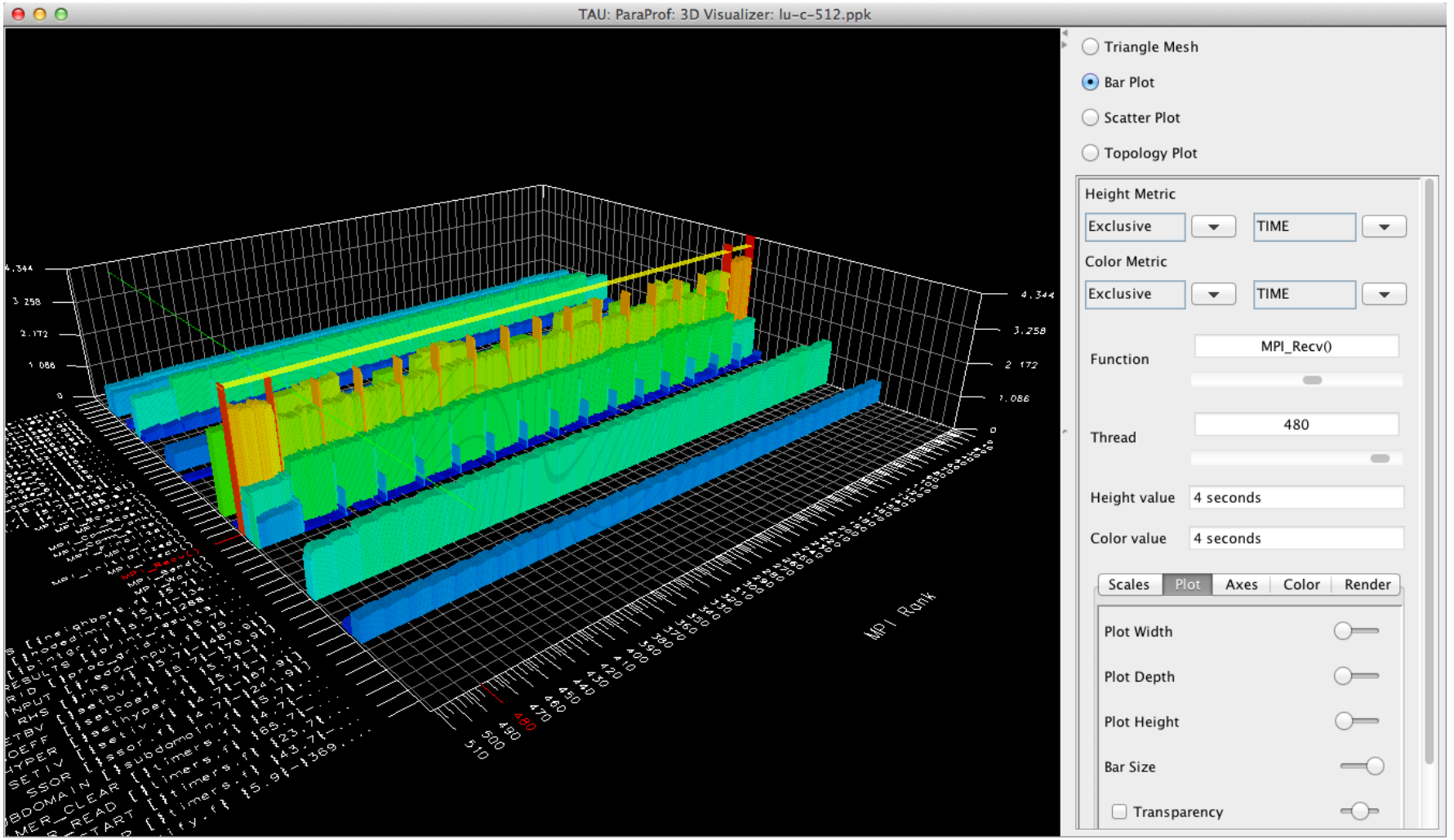
How much time is spent in each application routine?



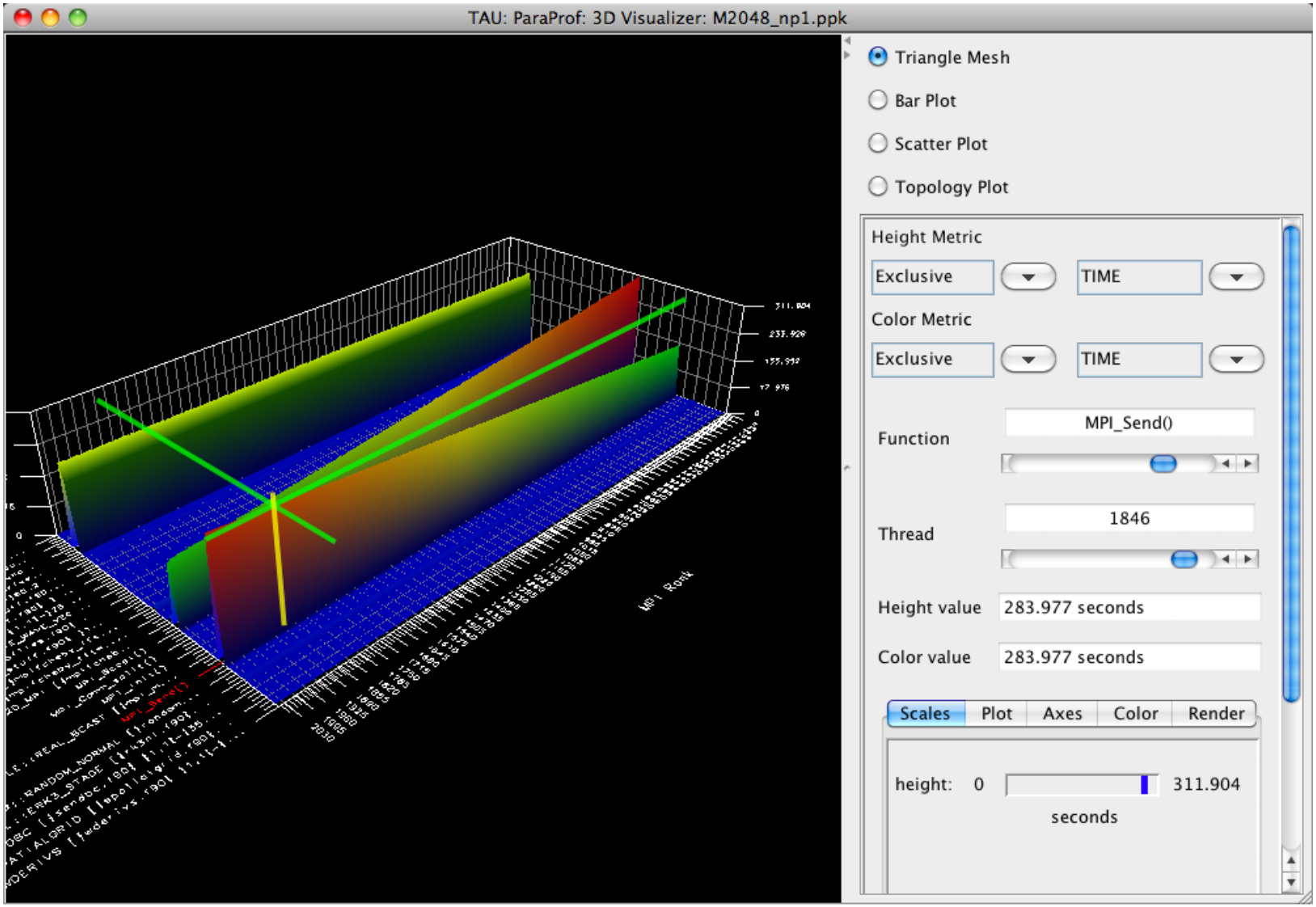
Generating a flat profile with MPI

```
% source /usr/global/tools/tau/training/tau.bashrc
% export TAU_MAKEFILE=$TAU/Makefile.tau-icpc-papi-mpi-pdt
% make F90=tau_f90.sh
Or
% tau_f90.sh matmult.f90
% mxterm 1 16 40; mpirun -np 4  ./a.out
% paraprof
To view. To view the data locally on the workstation,
% paraprof --pack app.ppk
  Move the app.ppk file to your desktop.
% paraprof app.ppk

Click on the "node 0" label to see profile for that node. Right
click to see other options. Windows -> 3D Visualization for 3D
window.
```

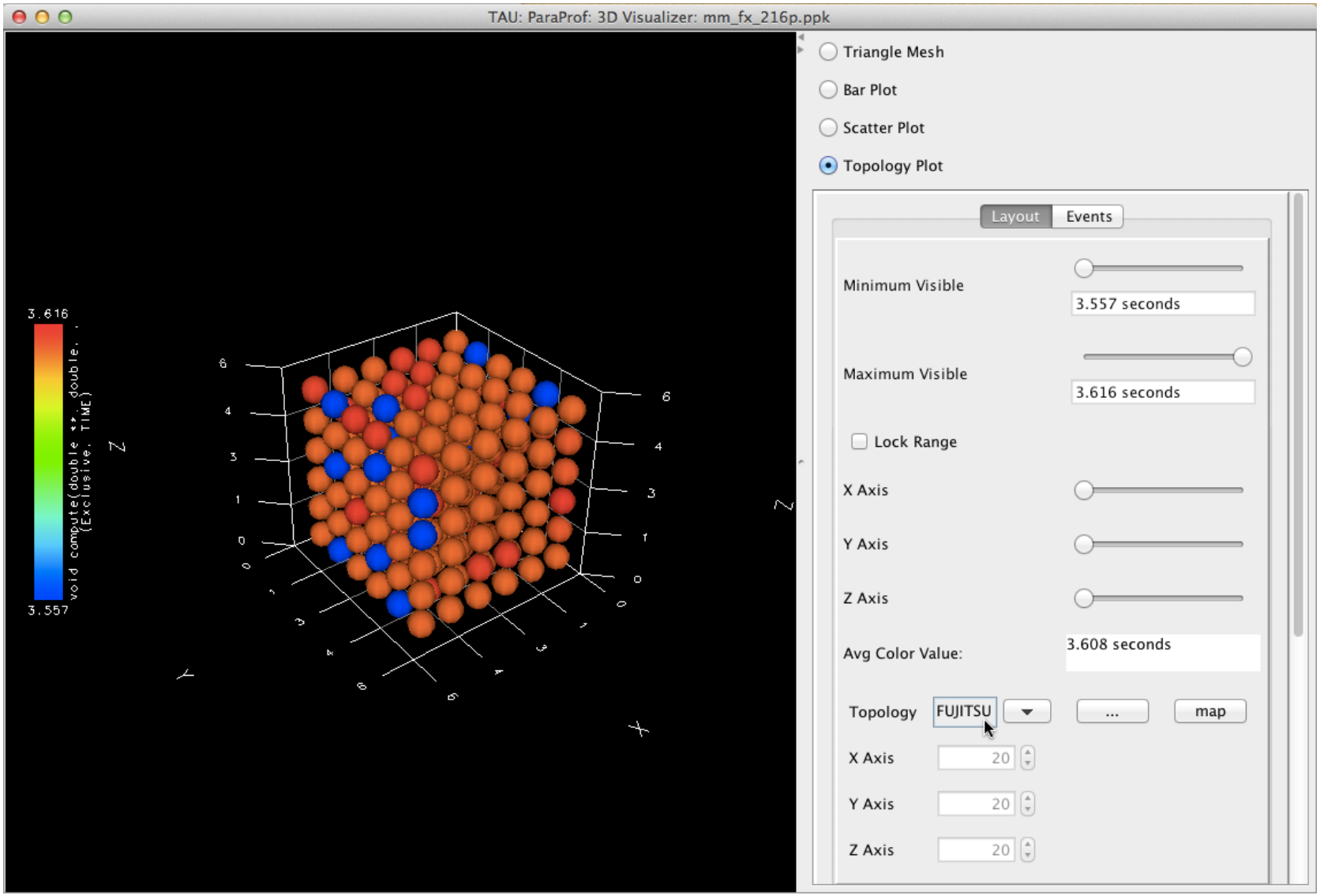
ParaProf 3D Profile Browser



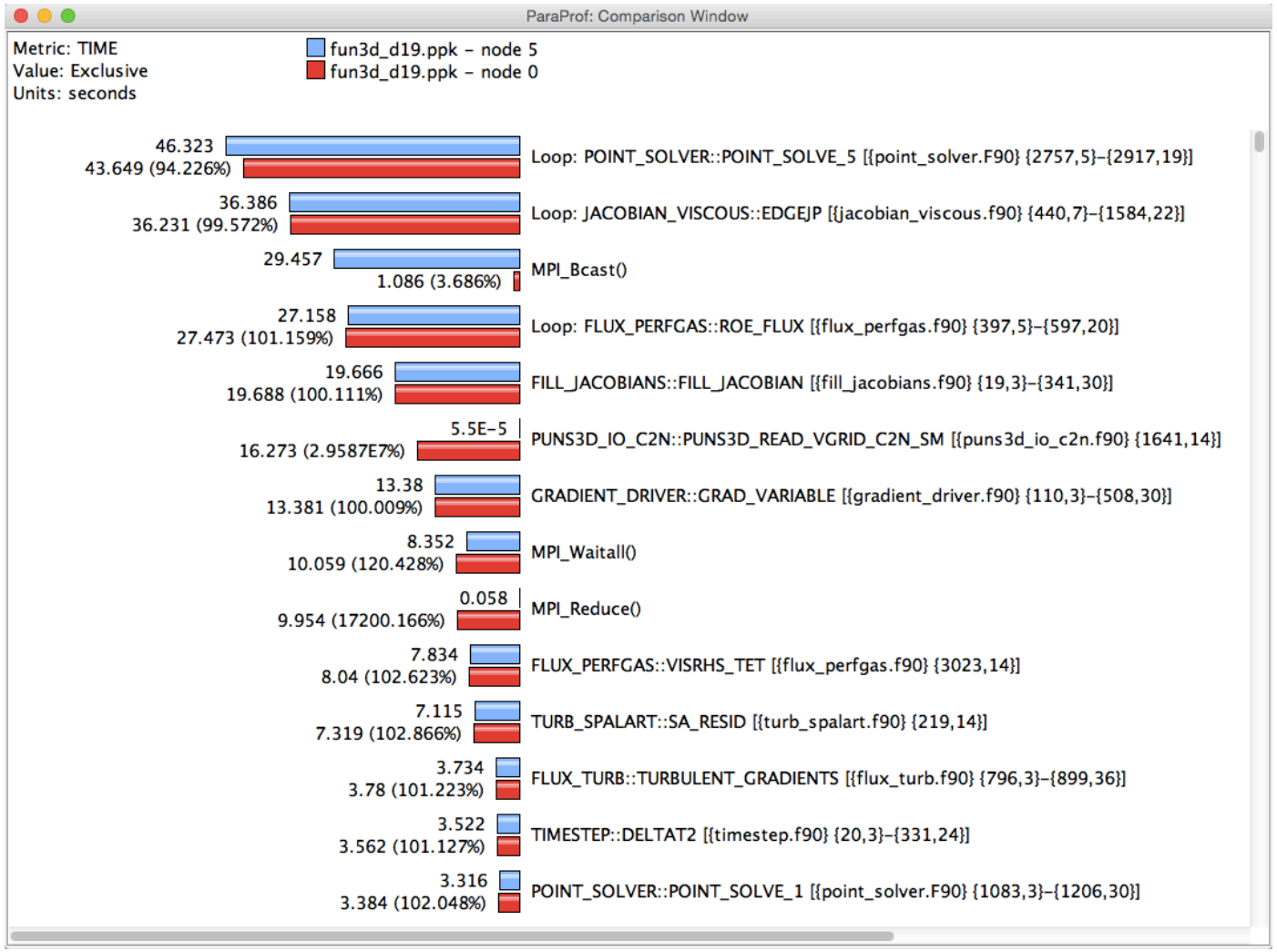
ParaProf


TAU: ParaProf: 3D Visualizer: M2048_np1.ppk
Triangle Mesh
Bar Plot
Scatter Plot
Topology Plot
Height Metric
Exclusive
TIME
Color Metric
Exclusive
TIME
Function
MPI_Send()
Thread
1846
Height value
283.977 seconds
Color value
283.977 seconds
Scales
Plot
Axes
Color
Render
height: 0
311.904
seconds
MPI Rank

ParaProf 3D Topology Display



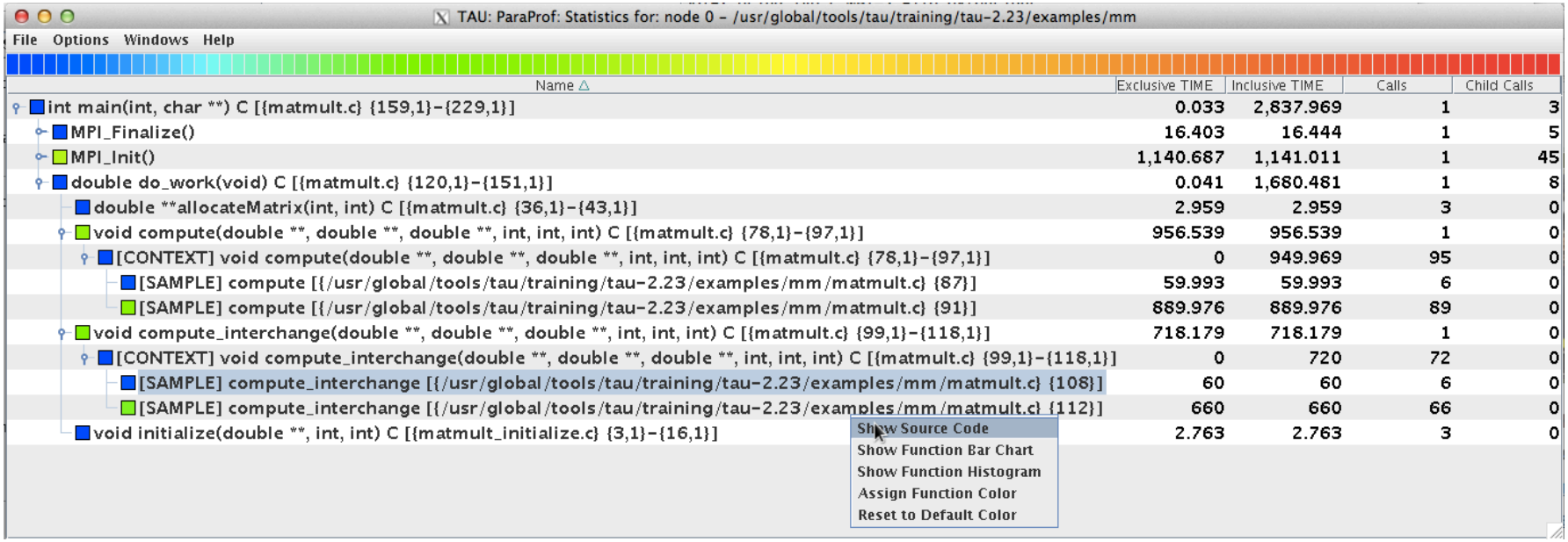
ParaProf Comparison Window



Comparing Rank 0 with 5.

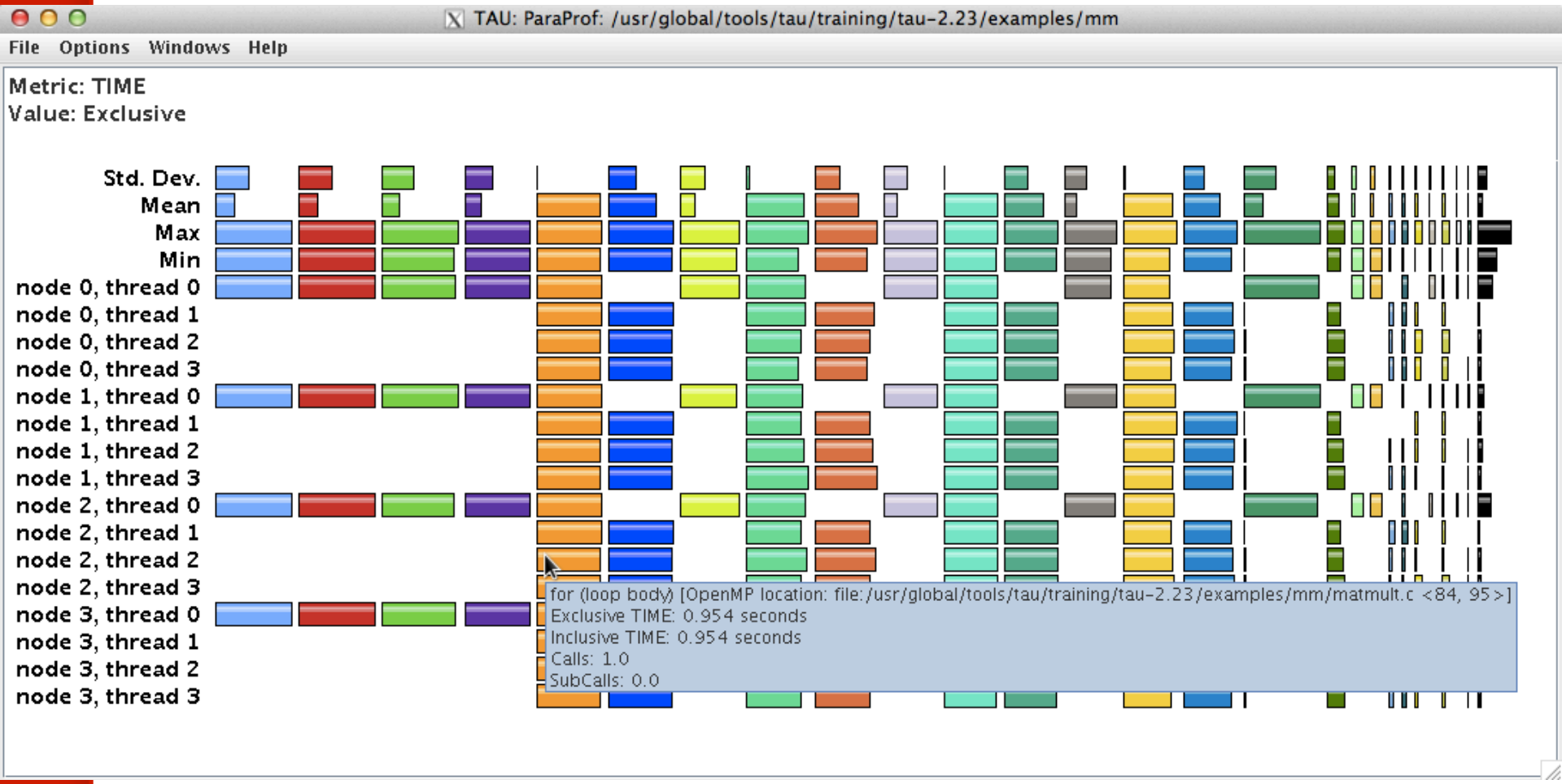
Right click on “node 5” -> Add node to comparison window

Event Based Sampling in TAU



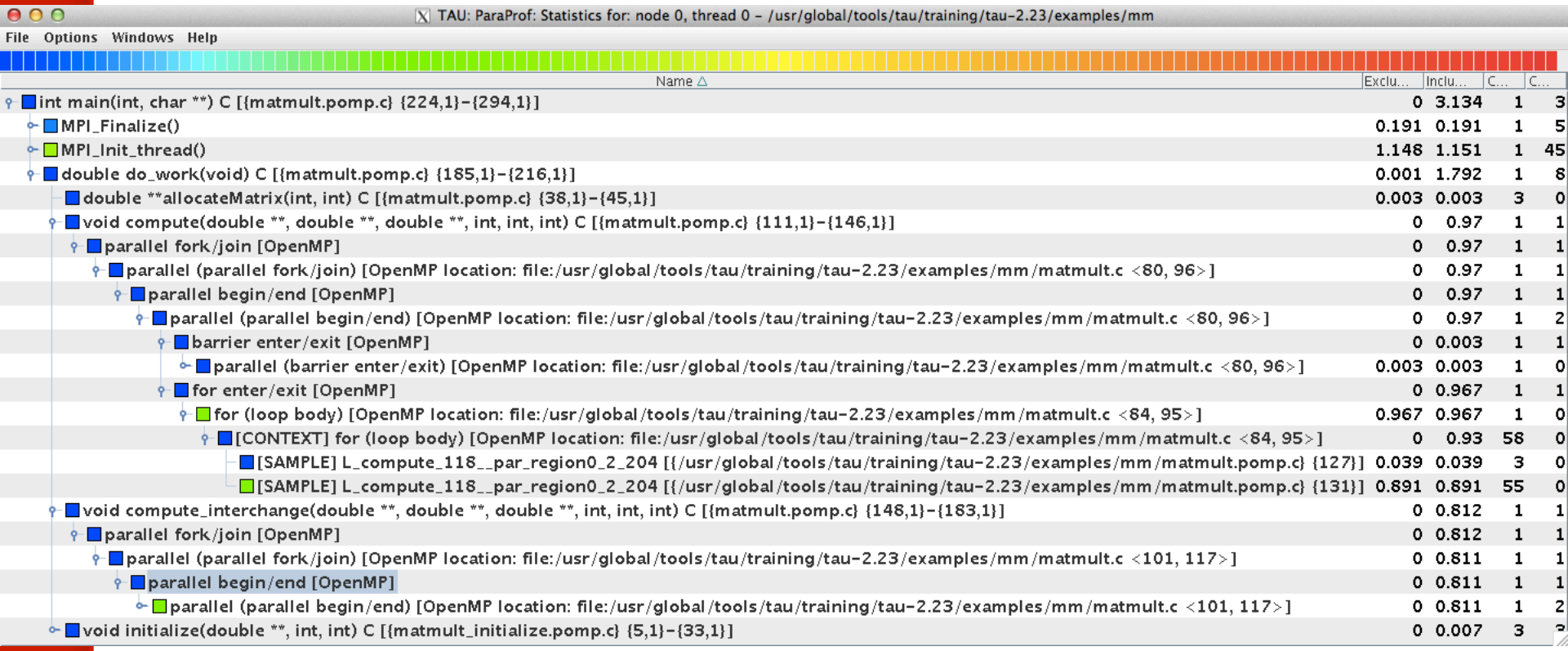

```
% export TAU_MAKEFILE=$TAU/Makefile.tau-icpc-papi-mpi-pdt
% make CC=tau_cc.sh CXX=tau_cxx.sh
% export TAU_SAMPLING=1
% mpirun –np 256 ./a.out
% paraprof
```

Mixed MPI and OpenMP Instrumentation



Options -> Uncheck “Stack Bars Together”

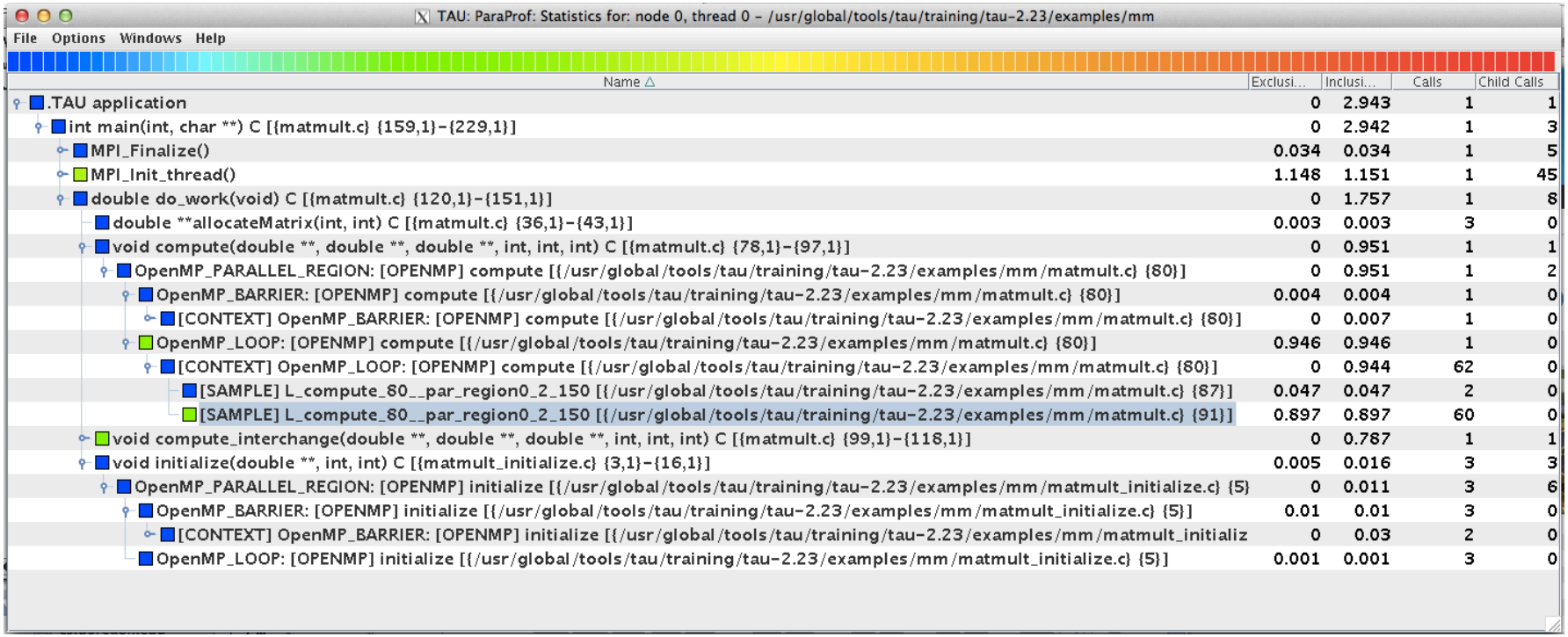
Opari OpenMP Instrumentation, Sampling



Name	Exclu...	Inclu...	C...	C...
int main(int, char **) C [{matmult.pomp.c} {224,1}-{294,1}]	0	3.134	1	3
MPI_Finalize()	0.191	0.191	1	5
MPI_Init_thread()	1.148	1.151	1	45
double do_work(void) C [{matmult.pomp.c} {185,1}-{216,1}]	0.001	1.792	1	8
double **allocateMatrix(int, int) C [{matmult.pomp.c} {38,1}-{45,1}]	0.003	0.003	3	0
void compute(double **, double **, double **, int, int, int) C [{matmult.pomp.c} {111,1}-{146,1}]	0	0.97	1	1
parallel fork/join [OpenMP]	0	0.97	1	1
parallel (parallel fork/join) [OpenMP location: file:/usr/global/tools/tau/training/tau-2.23/examples/mm/matmult.c <80, 96>]	0	0.97	1	1
parallel begin/end [OpenMP]	0	0.97	1	1
parallel (parallel begin/end) [OpenMP location: file:/usr/global/tools/tau/training/tau-2.23/examples/mm/matmult.c <80, 96>]	0	0.97	1	2
barrier enter/exit [OpenMP]	0	0.003	1	1
parallel (barrier enter/exit) [OpenMP location: file:/usr/global/tools/tau/training/tau-2.23/examples/mm/matmult.c <80, 96>]	0.003	0.003	1	0
for enter/exit [OpenMP]	0	0.967	1	1
for (loop body) [OpenMP location: file:/usr/global/tools/tau/training/tau-2.23/examples/mm/matmult.c <84, 95>]	0.967	0.967	1	0
[CONTEXT] for (loop body) [OpenMP location: file:/usr/global/tools/tau/training/tau-2.23/examples/mm/matmult.c <84, 95>]	0	0.93	58	0
[SAMPLE] L_compute_118__par_region0_2_204 [{/usr/global/tools/tau/training/tau-2.23/examples/mm/matmult.pomp.c} {127}]	0.039	0.039	3	0
[SAMPLE] L_compute_118__par_region0_2_204 [{/usr/global/tools/tau/training/tau-2.23/examples/mm/matmult.pomp.c} {131}]	0.891	0.891	55	0
void compute_interchange(double **, double **, double **, int, int, int) C [{matmult.pomp.c} {148,1}-{183,1}]	0	0.812	1	1
parallel fork/join [OpenMP]	0	0.812	1	1
parallel (parallel fork/join) [OpenMP location: file:/usr/global/tools/tau/training/tau-2.23/examples/mm/matmult.c <101, 117>]	0	0.811	1	1
parallel begin/end [OpenMP]	0	0.811	1	1
parallel (parallel begin/end) [OpenMP location: file:/usr/global/tools/tau/training/tau-2.23/examples/mm/matmult.c <101, 117>]	0	0.811	1	2
void initialize(double **, int, int) C [{matmult_initialize.pomp.c} {5,1}-{33,1}]	0	0.007	3	

```
% export TAU_MAKEFILE=$TAU/Makefile.tau-icpc-papi-mpi-pdt-opari-openmp
% make CC=tau_cc.sh CXX=tau_cxx.sh
% export TAU_SAMPLING=1; export OMP_NUM_THREADS=16
% mpirun –np 256 ./a.out
% paraprof
```

TAU's support for Intel and GNU OMPT



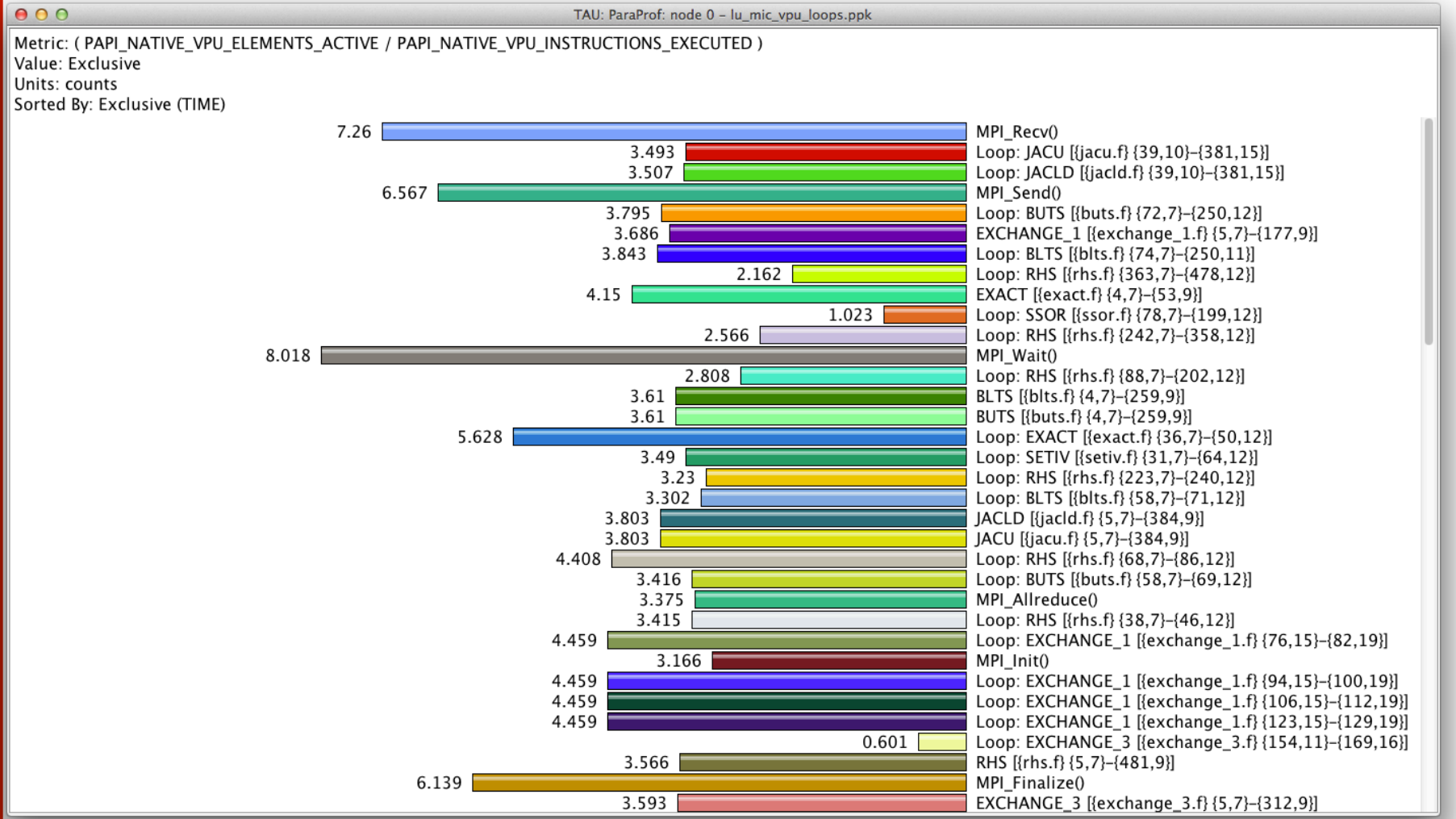
Name	Exclusi...	Inclusi...	Calls	Child Calls
.TAU application	0	2.943	1	1
int main(int, char **) C [{matmult.c} {159,1}-{229,1}]	0	2.942	1	3
MPI_Finalize()	0.034	0.034	1	5
MPI_Init_thread()	1.148	1.151	1	45
double do_work(void) C [{matmult.c} {120,1}-{151,1}]	0	1.757	1	8
double **allocateMatrix(int, int) C [{matmult.c} {36,1}-{43,1}]	0.003	0.003	3	0
void compute(double **, double **, double **, int, int, int) C [{matmult.c} {78,1}-{97,1}]	0	0.951	1	1
OpenMP_PARALLEL_REGION: [OPENMP] compute [{/usr/global/tools/tau/training/tau-2.23/examples/mm/matmult.c} {80}]	0	0.951	1	2
OpenMP_BARRIER: [OPENMP] compute [{/usr/global/tools/tau/training/tau-2.23/examples/mm/matmult.c} {80}]	0.004	0.004	1	0
[CONTEXT] OpenMP_BARRIER: [OPENMP] compute [{/usr/global/tools/tau/training/tau-2.23/examples/mm/matmult.c} {80}]	0	0.007	1	0
OpenMP_LOOP: [OPENMP] compute [{/usr/global/tools/tau/training/tau-2.23/examples/mm/matmult.c} {80}]	0.946	0.946	1	0
[CONTEXT] OpenMP_LOOP: [OPENMP] compute [{/usr/global/tools/tau/training/tau-2.23/examples/mm/matmult.c} {80}]	0	0.944	62	0
[SAMPLE] L_compute_80__par_region0_2_150 [{/usr/global/tools/tau/training/tau-2.23/examples/mm/matmult.c} {87}]	0.047	0.047	2	0
[SAMPLE] L_compute_80__par_region0_2_150 [{/usr/global/tools/tau/training/tau-2.23/examples/mm/matmult.c} {91}]	0.897	0.897	60	0
void compute_interchange(double **, double **, double **, int, int, int) C [{matmult.c} {99,1}-{118,1}]	0	0.787	1	1
void initialize(double **, int, int) C [{matmult_initialize.c} {3,1}-{16,1}]	0.005	0.016	3	3
OpenMP_PARALLEL_REGION: [OPENMP] initialize [{/usr/global/tools/tau/training/tau-2.23/examples/mm/matmult_initialize.c} {5}	0	0.011	3	6
OpenMP_BARRIER: [OPENMP] initialize [{/usr/global/tools/tau/training/tau-2.23/examples/mm/matmult_initialize.c} {5}]	0.01	0.01	3	0
[CONTEXT] OpenMP_BARRIER: [OPENMP] initialize [{/usr/global/tools/tau/training/tau-2.23/examples/mm/matmult_initializ	0	0.03	2	0
OpenMP_LOOP: [OPENMP] initialize [{/usr/global/tools/tau/training/tau-2.23/examples/mm/matmult_initialize.c} {5}]	0.001	0.001	3	0

Configure TAU with –ompt=download (without –opari)

```
% export TAU_MAKEFILE=$TAU/Makefile.tau-papi-mpi-pdt-ompt-openmp
% make CC=tau_cc.sh CXX=tau_cxx.sh
% export TAU_SAMPLING=1; export OMP_NUM_THREADS=16
% mpirun –np 256 tau_exec –T ompt,papi,pdt  –ompt ./a.out
% paraprof
```

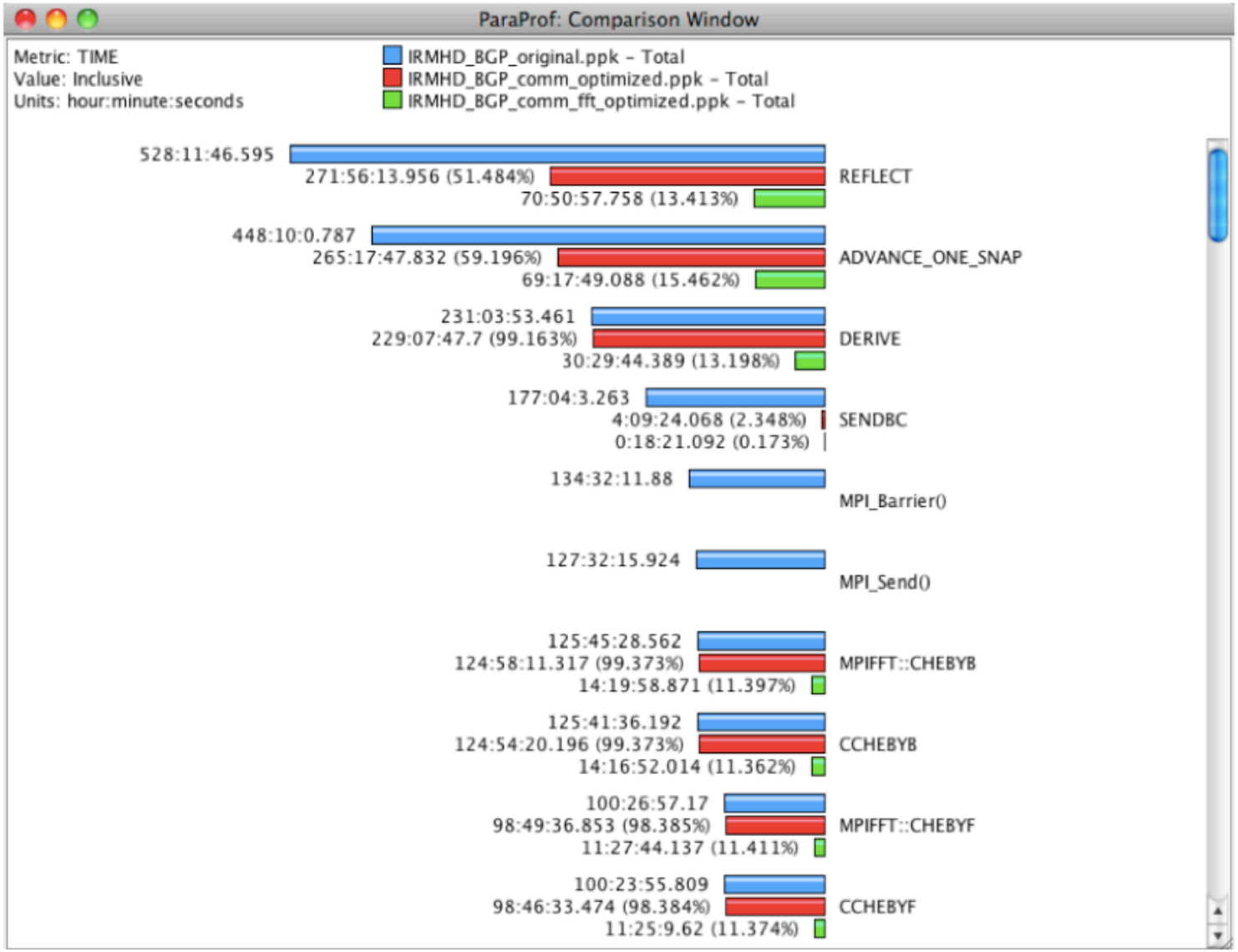
NOTE: Instrumentation is at the source, MPI, and OpenMP levels with sampling

Evaluating Extent of Vectorization on MIC

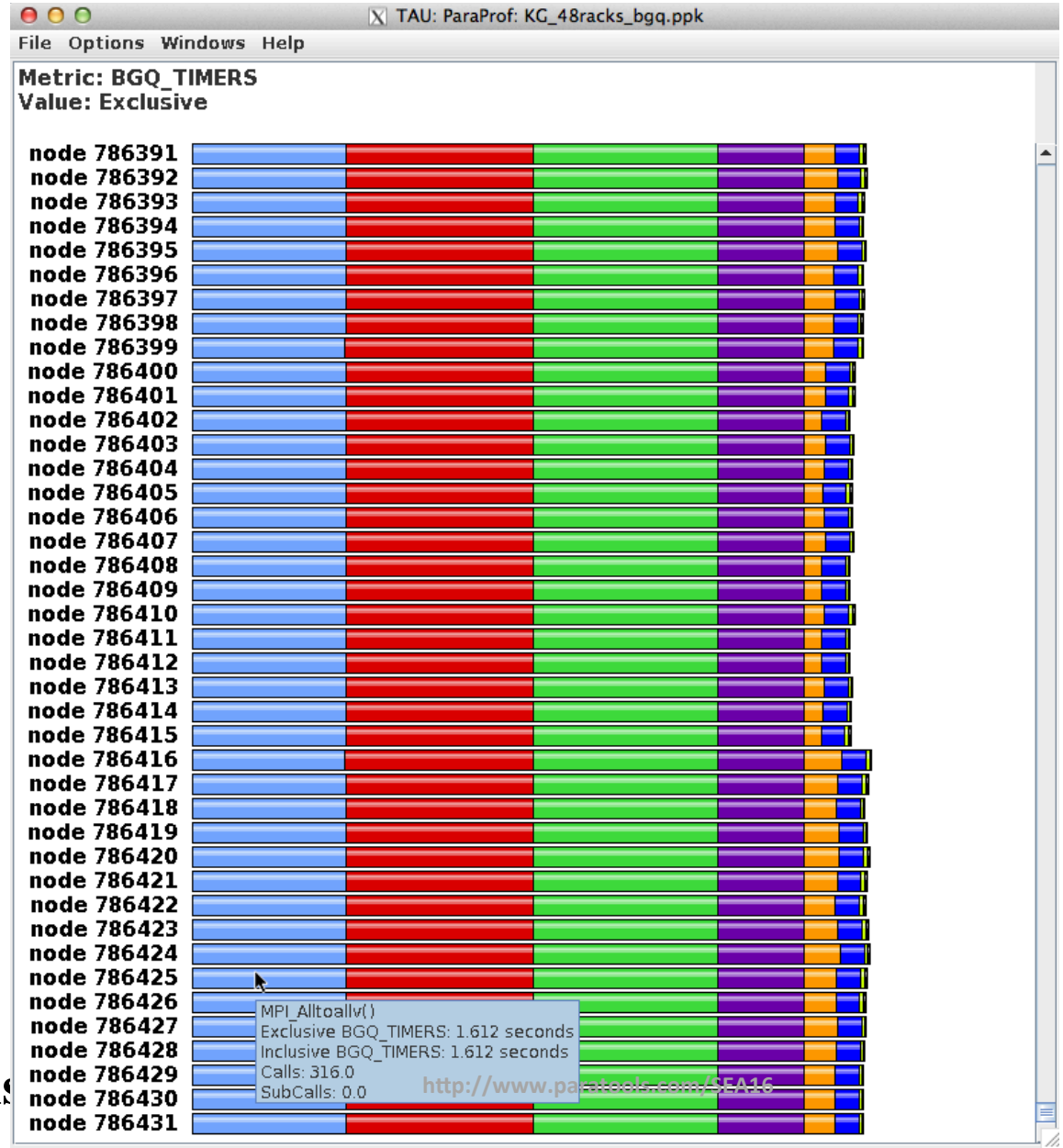



```
% export TAU_MAKEFILE=$TAUROOT/mic_linux/lib/Makefile.tau-papi-mpi-pdt
% export TAU_METRICS=TIME,
        PAPI_NATIVE_VPU_ELEMENTS_ACTIVE,PAPI_NATIVE_VPU_INSTRUCTIONS_EXECUTED
```

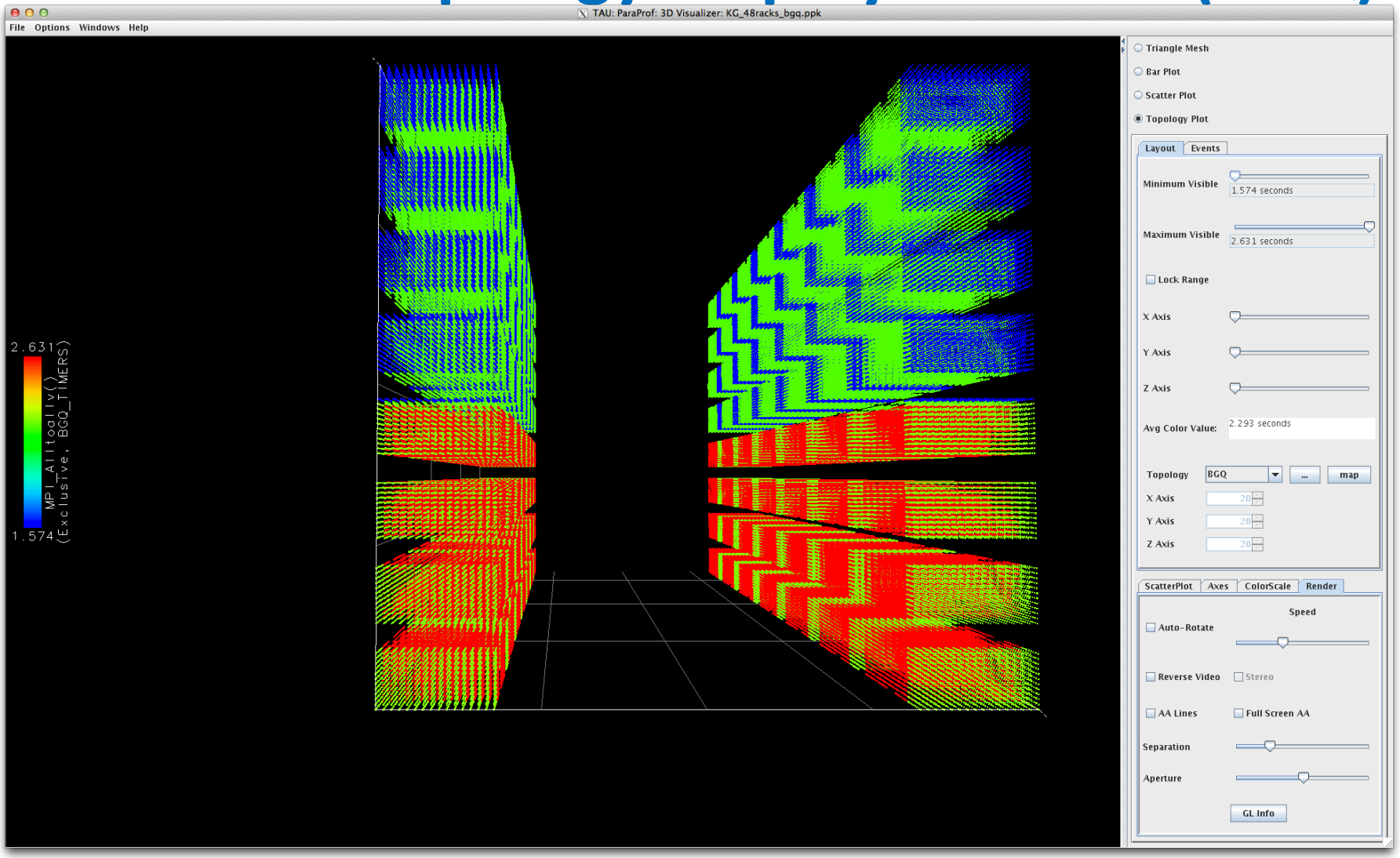
ParaProf Comparison Window



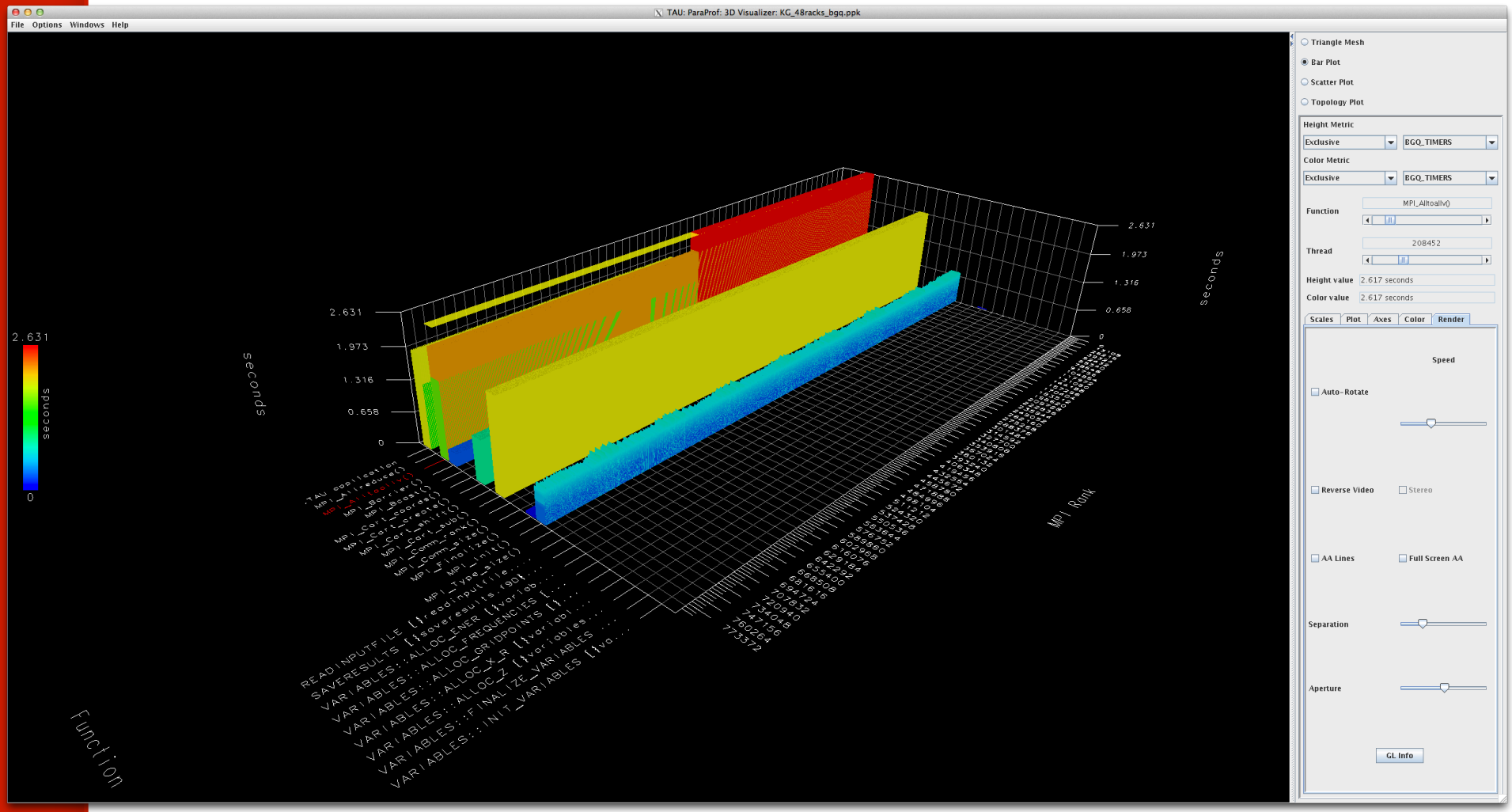
TAU's ParaProf Profile Browser



ParaProf's Topology Display Window (BGQ)

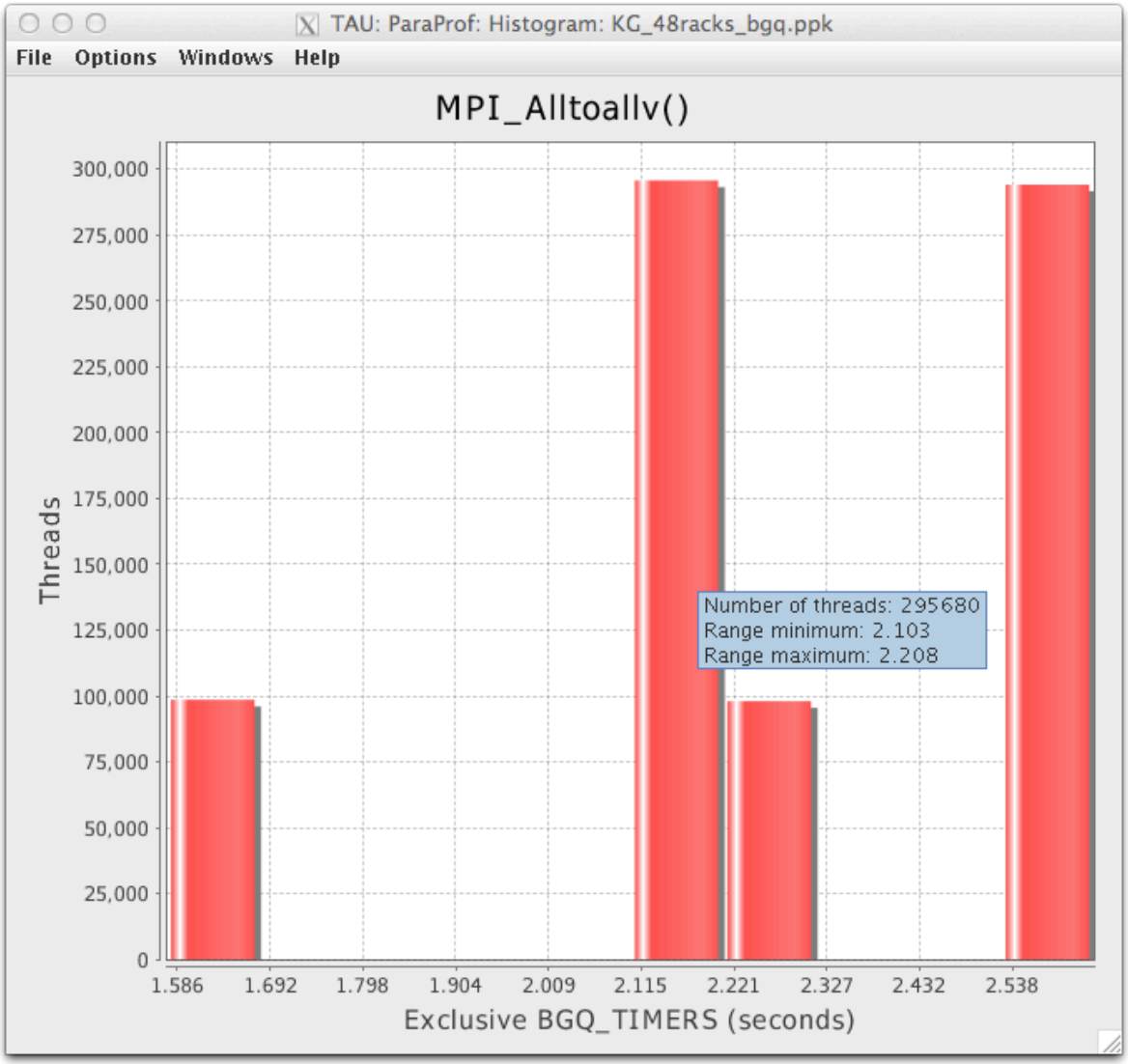


ParaProf's Scalable 3D Visualization (BGQ)

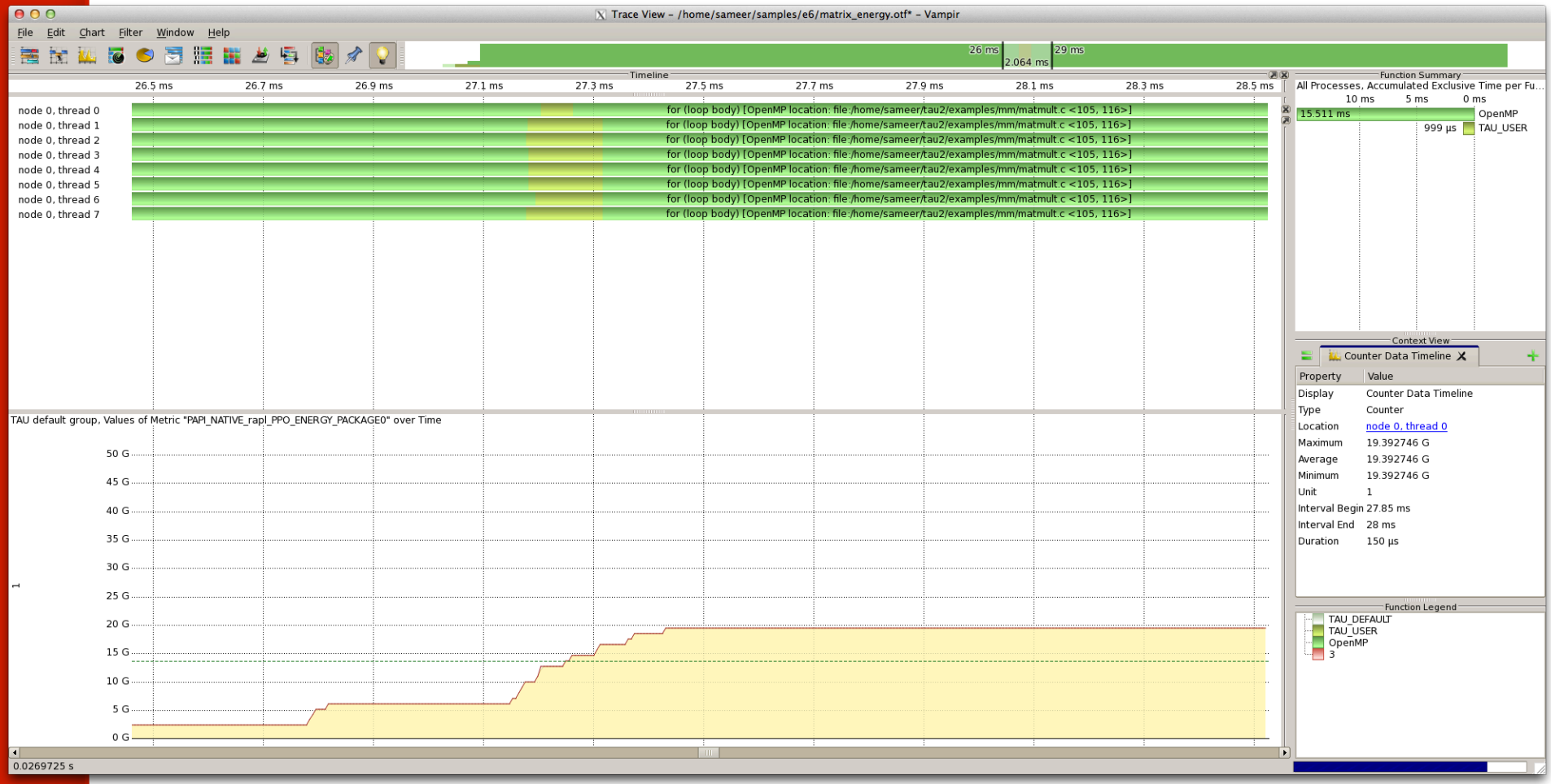


786,432 ranks

ParaProf Histogram Display



Tracing Energy Usage with TAU and Vampir



TAU's Runtime Merging of Profile Data

TAU: ParaProf Manager

File Options Help

- Applications
  - Standard Applications
    - Default App
      - Default Exp
        - KG_48racks_bgq.ppk
          - BGQ_TIMERS
  - Default (jdbc:h2:/home3/sameer/
  - perfexplorer_working (jdbc:h2:/h

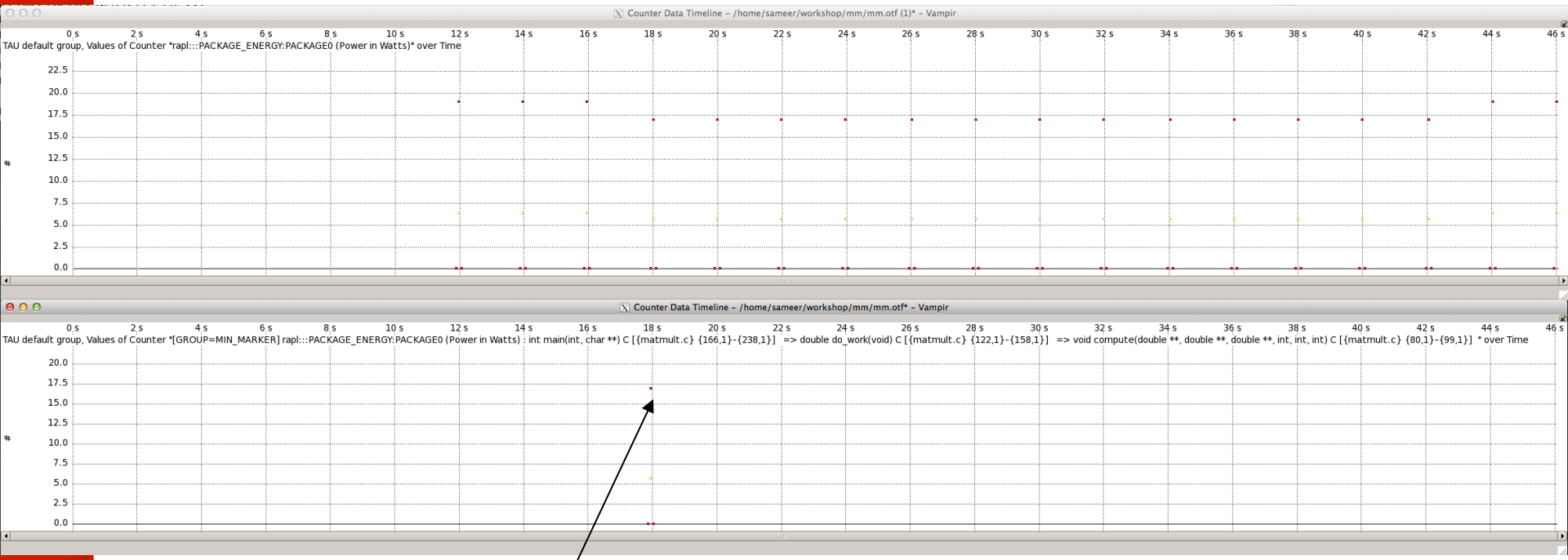
TrialField	Value
Name	KG_48racks_bgq.ppk
Application ID	0
Experiment ID	0
Trial ID	0
BGQ Block Thread ID	3342328
BGQ Coords	(188,108,272)
BGQ DDR Size (MB)	16384
BGQ Job ID	221109
BGQ Node ID	77280
BGQ Node Name	R2b-M0-N0f-J00 <8,12,16,16,2>
BGQ Period	(0,0,0)
BGQ Physical HW Thread ID	0
BGQ Physical Processor ID	15
BGQ Process Count	16
BGQ Processor Core ID	15
BGQ Processor Count	4
BGQ Processor ID	60
BGQ Processor Thread ID	0
BGQ Rank	786431
BGQ Size	(8,12,16,16,2,64)
BGQ tCoord	15
CPU MHz	1600.000000MHz
CPU Type	A2 (Blue Gene/Q)
CWD	/gpfs/mira-fs0/projects/MiraBootCamp2013/KG/tau/49152
Command Line	/gpfs/mira-fs0/projects/MiraBootCamp2013/KG/tau/49152/./Kg
Executable	/gpfs/mira-fs0/projects/MiraBootCamp2013/KG/tau/49152/./Kg
File Type Index	0
File Type Name	ParaProf Packed Profile
Hostname	Q2H-I3-J03.mira.i2b
Local Time	2013-05-24T19:20:06+00:00
MPI Processor Name	Task 786431 of 786432 (7,11,15,15,1,15) R2B-M0-N15-J00
Memory Size	16718464 kB
Node Name	Q2H-I3-J03.mira.i2b
OS Machine	BGQ
OS Name	CNK
OS Release	2.6.32-279.14.1.bgq.el6_V1R2M0_26.ppc64
OS Version	1
Starting Timestamp	1369423205614897
TAU Architecture	bgq
TAU Config	-BGQTIMERS -arch=bgq -pdt=/home/projects/tau/pdt_latest -pdt_c++=xlC -mpi -papi=/soft/perftools/tau/papi_latest -bfd=/home/projects/tau/tau2/bgq/binutils-2.20 -iowrapper
TAU Makefile	/soft/perftools/tau/tau-2.22.2p1/bgq/lib/Makefile.tau-bgqtimers-papi-mpi-pdt
TAU MetaData Merge Time	0.000545 seconds
TAU Profile Merge Time	47.34 seconds
TAU Unification Time	0.01323 seconds
TAU Version	2.22.2
TAU_CALLPATH	off
TAU_CALLPATH_DEPTH	2
TAU_CALLSITE_LIMIT	1

47.34 seconds

% export TAU_PROFILE_FORMAT=merged

It took ~48 seconds to merge and write profiles from 786,432 ranks

Marker Events: Tracing



When an atomic event exceeds the max or min value by a threshold (say 20%), a marker context event is triggered to record the callstack.

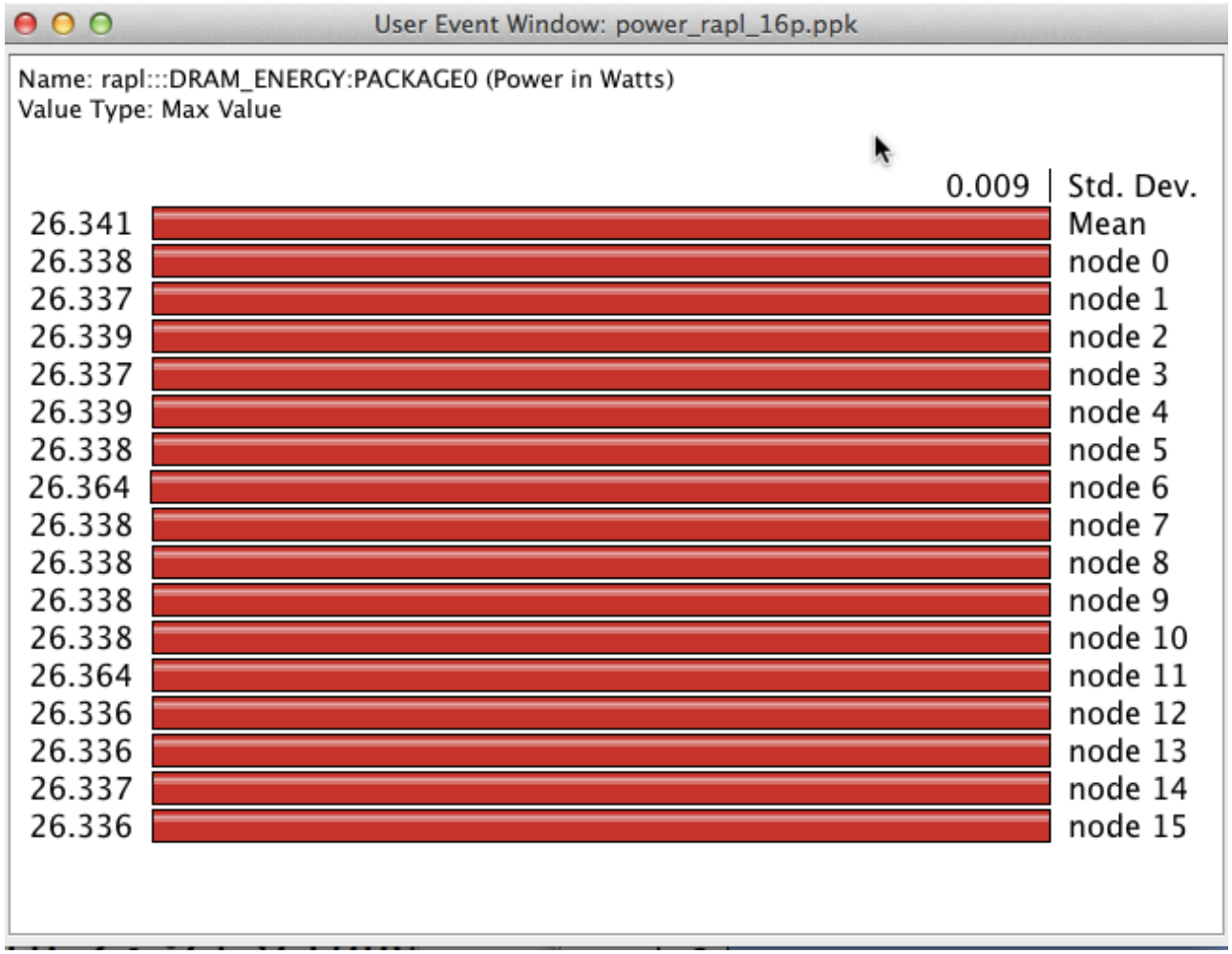
Marker events show sudden spikes

TAU: ParaProf: Context Events for: node 2 - rapl_marker_16p.ppk

Name	MaxValue	MinValue	NumSamples	MeanValue	Std. Dev.
int main(int, char **) C [{matmult.c}{165,1}-{237,1}]					
double do_work(void) C [{matmult.c}{126,1}-{157,1}]					
void compute(double **, double **, double **, int, int, int) C [{matmult.c}{84,1}-{103,1}]					
[GROUP=MAX_MARKER] rapl:::DRAM_ENERGY:PACKAGE0 (Power in Watts)	17.585	17.469	5	17.521	0.037
[GROUP=MAX_MARKER] rapl:::DRAM_ENERGY:PACKAGE1 (Power in Watts)	15.261	15.218	4	15.237	0.016
[GROUP=MAX_MARKER] rapl:::PACKAGE_ENERGY:PACKAGE0 (Power in Watts)	118.903	114.923	22	116.98	1.201
[GROUP=MAX_MARKER] rapl:::PACKAGE_ENERGY:PACKAGE1 (Power in Watts)	113.466	110.207	22	111.778	0.996
[GROUP=MAX_MARKER] rapl:::PP0_ENERGY:PACKAGE0 (Power in Watts)	100.138	96.266	24	98.206	1.13
[GROUP=MAX_MARKER] rapl:::PP0_ENERGY:PACKAGE1 (Power in Watts)	95.846	92.758	24	94.319	0.937
[GROUP=MIN_MARKER] rapl:::DRAM_ENERGY:PACKAGE0 (Power in Watts)	17.397	17.303	4	17.358	0.035
[GROUP=MIN_MARKER] rapl:::DRAM_ENERGY:PACKAGE1 (Power in Watts)	15.048	15.042	2	15.045	0.003
int mysleep(int) C [{matmult.c}{46,1}-{49,1}]					
[GROUP=MIN_MARKER] rapl:::DRAM_ENERGY:PACKAGE0 (Power in Watts)	15.84	15.84	1	15.84	0
[GROUP=MIN_MARKER] rapl:::DRAM_ENERGY:PACKAGE1 (Power in Watts)	14.275	14.275	1	14.275	0
[GROUP=MIN_MARKER] rapl:::PACKAGE_ENERGY:PACKAGE1 (Power in Watts)	96.853	96.853	1	96.853	0
[GROUP=MIN_MARKER] rapl:::PACKAGE_ENERGY:PACKAGE0 (Power in Watts)	93.125	93.125	1	93.125	0
[GROUP=MIN_MARKER] rapl:::PP0_ENERGY:PACKAGE0 (Power in Watts)	75.096	75.096	1	75.096	0
[GROUP=MIN_MARKER] rapl:::PP0_ENERGY:PACKAGE1 (Power in Watts)	79.646	79.646	1	79.646	0
void compute_interchange(double **, double **, double **, int, int, int) C [{matmult.c}{105,1}-{124,1}]					
[GROUP=MAX_MARKER] rapl:::DRAM_ENERGY:PACKAGE0 (Power in Watts)	26.064	25.711	2	25.887	0.176
[GROUP=MAX_MARKER] rapl:::DRAM_ENERGY:PACKAGE1 (Power in Watts)	24.373	23.965	4	24.232	0.159
[GROUP=MAX_MARKER] rapl:::PACKAGE_ENERGY:PACKAGE0 (Power in Watts)	126.872	125.182	6	125.732	0.557
[GROUP=MAX_MARKER] rapl:::PACKAGE_ENERGY:PACKAGE1 (Power in Watts)	124.377	116.689	5	122.428	2.885
[GROUP=MAX_MARKER] rapl:::PP0_ENERGY:PACKAGE0 (Power in Watts)	103.981	102.21	6	102.769	0.584
[GROUP=MAX_MARKER] rapl:::PP0_ENERGY:PACKAGE1 (Power in Watts)	102.615	101.693	4	102.115	0.33
rapl:::DRAM_ENERGY:PACKAGE0 (Power in Watts)	26.064	15.84	36	19.053	3.39
rapl:::DRAM_ENERGY:PACKAGE1 (Power in Watts)	24.373	14.275	36	16.435	3.155
rapl:::PACKAGE_ENERGY:PACKAGE0 (Power in Watts)	126.872	93.125	36	117.729	5.403
rapl:::PACKAGE_ENERGY:PACKAGE1 (Power in Watts)	124.377	96.853	36	112.961	4.776
rapl:::PP0_ENERGY:PACKAGE0 (Power in Watts)	103.981	75.096	36	98.208	4.466
rapl:::PP0_ENERGY:PACKAGE1 (Power in Watts)	102.615	79.646	36	94.872	3.662

```
% export TAU_EVENT_THRESHOLD 0.5
```

Profiling in TAU



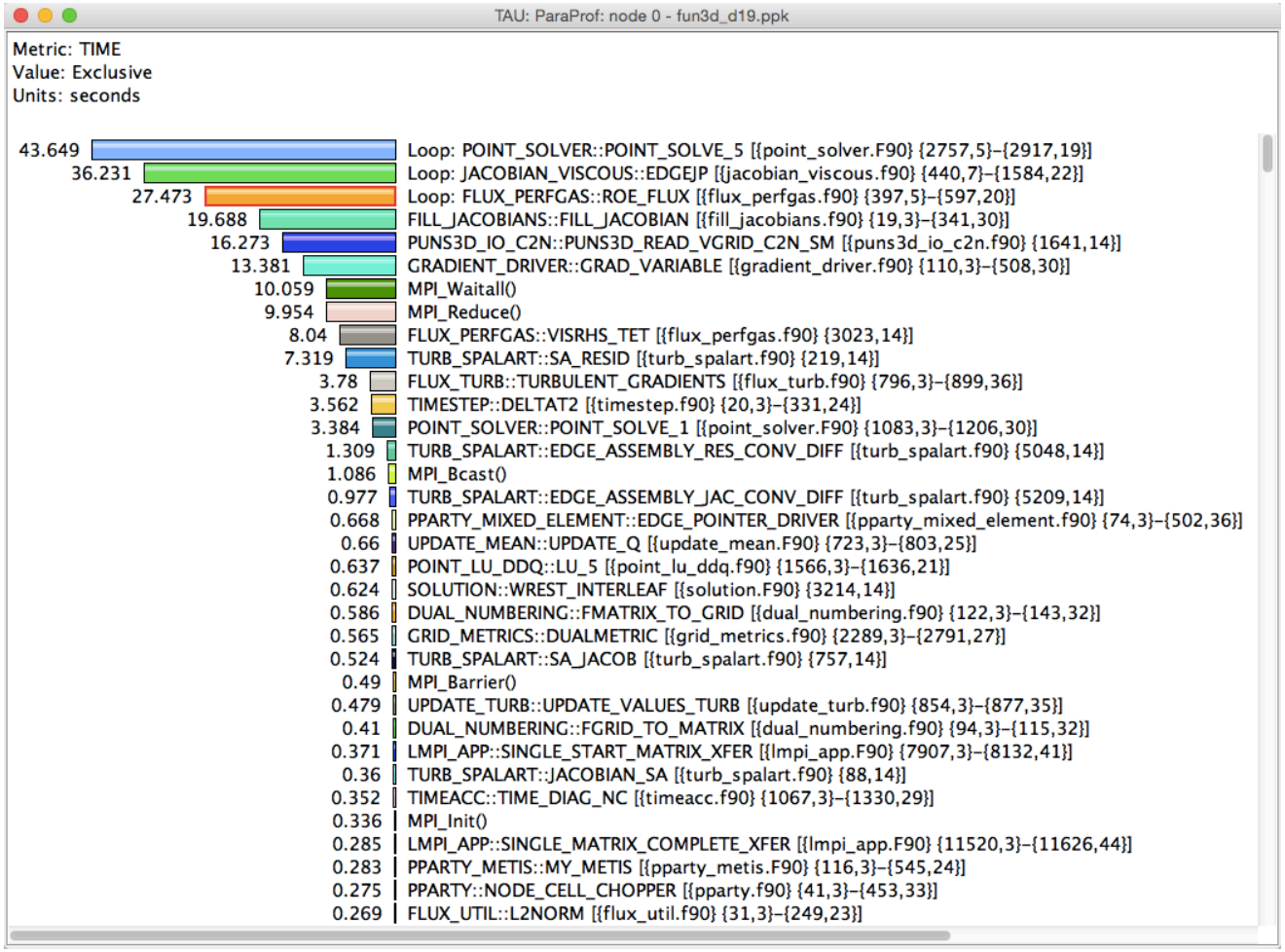
Generating a loop level profile

```
% export TAU_MAKEFILE=$TAU/Makefile.tau-icpc-papi-mpi-pdt
% export TAU_OPTIONS='-optTauSelectFile=select.tau -optVerbose'
% cat select.tau
  BEGIN_INSTRUMENT_SECTION
  loops routine="#"
  END_INSTRUMENT_SECTION

% module load tau
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)

% paraprof --pack app.ppk
  Move the app.ppk file to your desktop.

% paraprof app.ppk
```

Loop Level Instrumentation

Goal: What loops account for the most time? How much?

Flat profile with wallclock time with loop instrumentation:



Profiling with multiple counters

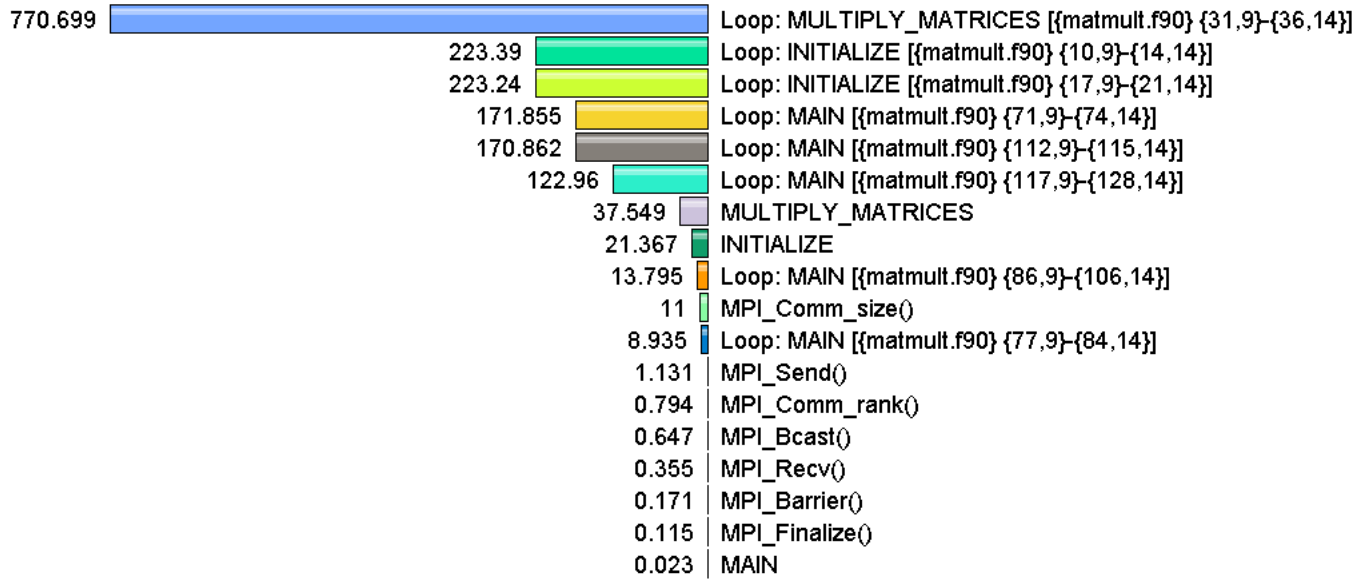
```
% source tau.bashrc
% export TAU_MAKEFILE=$TAU/Makefile.tau-icpc-papi-mpi-pdt
% export TAU_OPTIONS='-optTauSelectFile=select.tau -optVerbose'
% cat select.tau
  BEGIN_INSTRUMENT_SECTION
  loops routine="#"
  END_INSTRUMENT_SECTION
% make F90=tau_f90.sh
% mxterm 1 16 40
% export TAU_METRICS=TIME,PAPI_FP_INS,PAPI_L1_DCM
% mpirun -np 4  ./matmult
% paraprof --pack app.ppk
  Move the app.ppk file to your desktop.
% paraprof app.ppk
  Choose Options -> Show Derived Panel -> Click PAPI_FP_INS,
 Click "/",  Click TIME, Apply, Choose new metric by double
clicking.
```

Computing FLOPS per loop

Goal: What is the execution rate of my loops in MFLOPS?

Flat profile with PAPI_FP_INS and time with loop instrumentation:

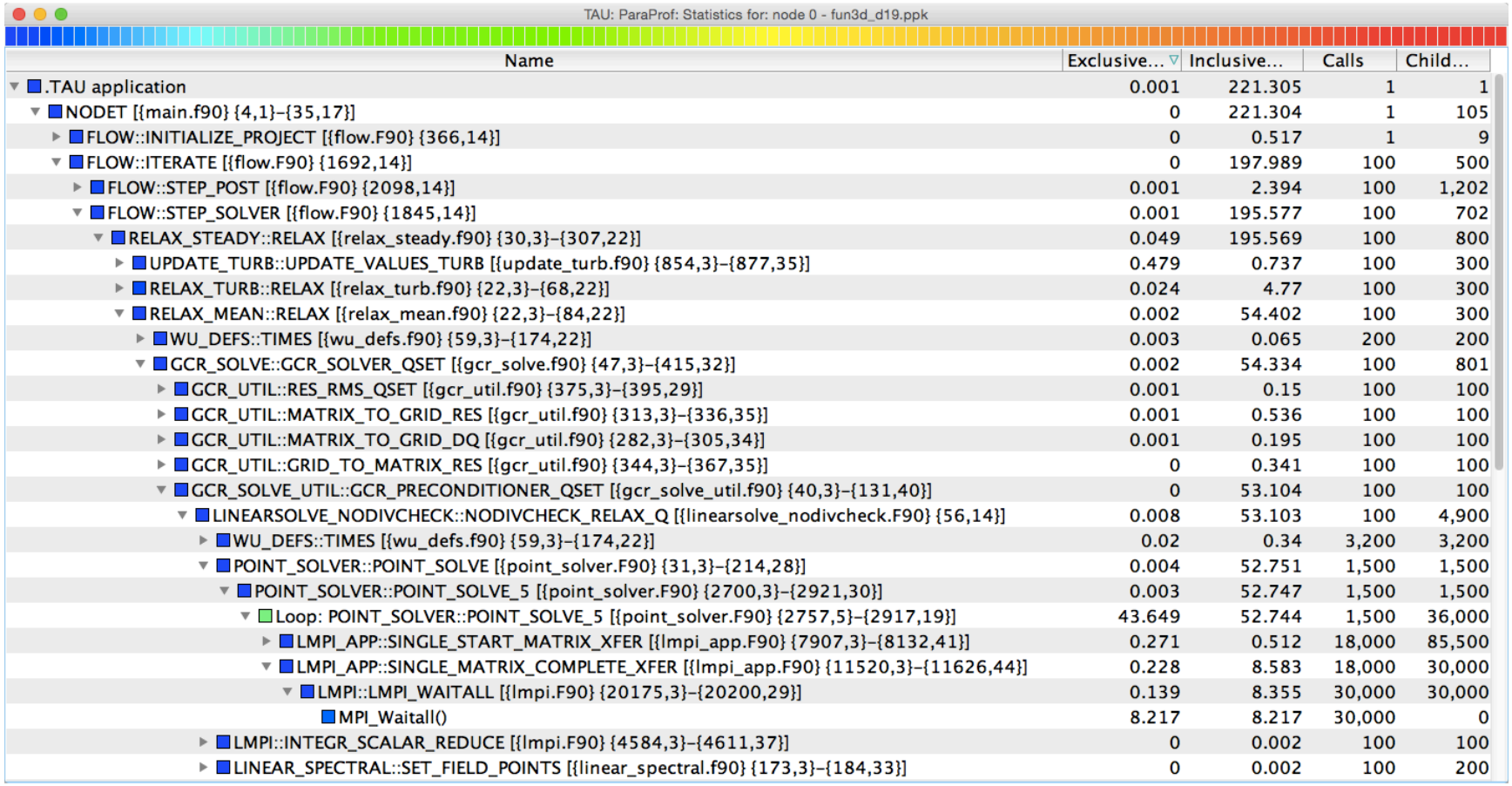
Metric: PAPI_FP_INS / GET_TIME_OF_DAY
Value: Exclusive
Units: Derived metric shown in microseconds format



Generate a Callpath Profile

```
% source tau.bashrc
% export TAU_MAKEFILE=$TAU/Makefile.tau-icpc-papi-mpi-pdt
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)

% mxterm 1 16 40
% export TAU_CALLPATH=1
% export TAU_CALLPATH_DEPTH=100
(truncates all calling paths to a specified depth)
% mpirun -np 4    ./a.out
% paraprof --pack app.ppk
  Move the app.ppk file to your desktop.
% paraprof app.ppk
(Windows -> Thread -> Call Graph)
```

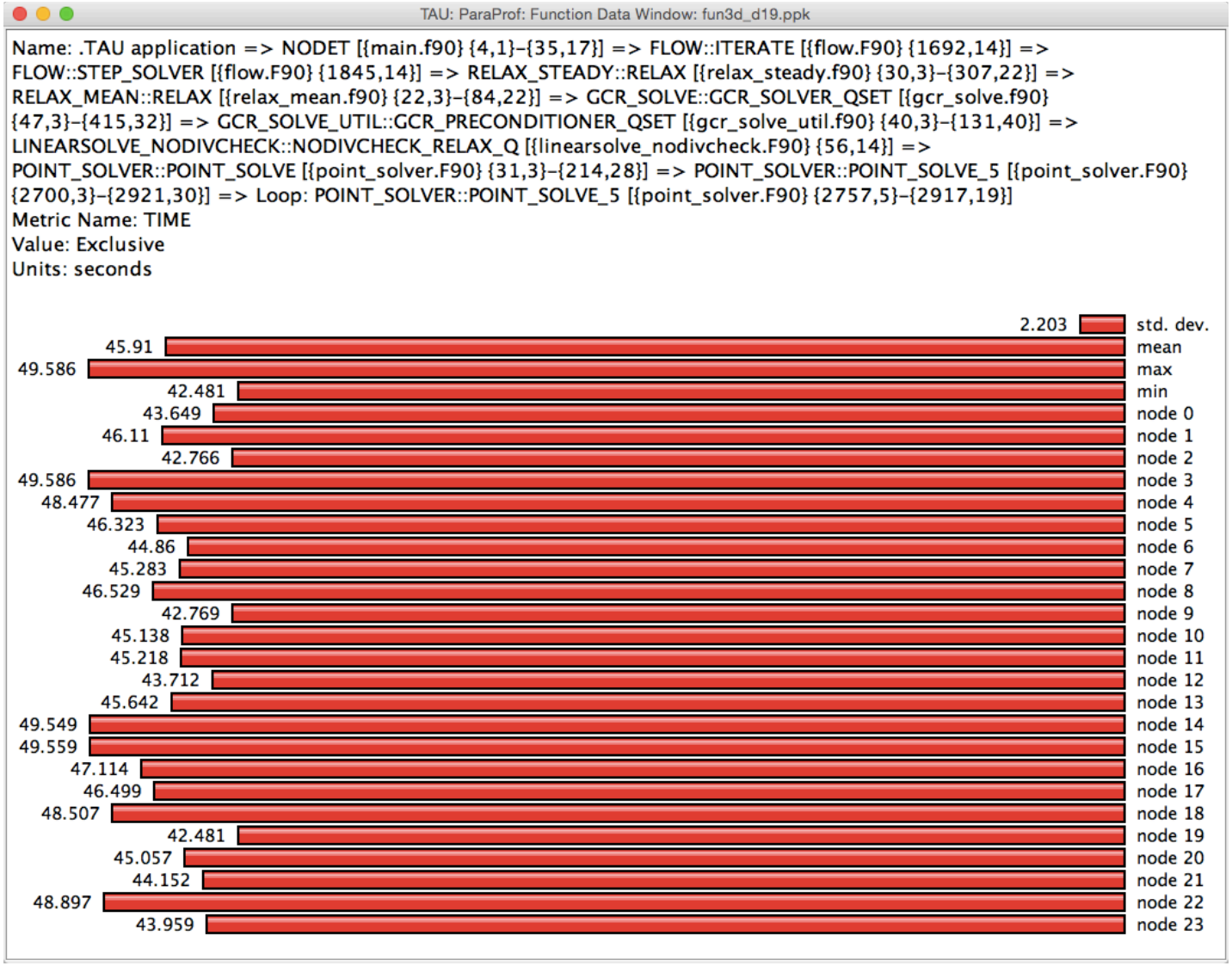
Callpath Profiling: FUN3D



Name	Exclusive...	Inclusive...	Calls	Child...
.TAU application	0.001	221.305	1	1
NODET [{main.f90} {4,1}-{35,17}]	0	221.304	1	105
FLOW::INITIALIZE_PROJECT [{flow.F90} {366,14}]	0	0.517	1	9
FLOW::ITERATE [{flow.F90} {1692,14}]	0	197.989	100	500
FLOW::STEP_POST [{flow.F90} {2098,14}]	0.001	2.394	100	1,202
FLOW::STEP_SOLVER [{flow.F90} {1845,14}]	0.001	195.577	100	702
RELAX_STEADY::RELAX [{relax_steady.f90} {30,3}-{307,22}]	0.049	195.569	100	800
UPDATE_TURB::UPDATE_VALUES_TURB [{update_turb.f90} {854,3}-{877,35}]	0.479	0.737	100	300
RELAX_TURB::RELAX [{relax_turb.f90} {22,3}-{68,22}]	0.024	4.77	100	300
RELAX_MEAN::RELAX [{relax_mean.f90} {22,3}-{84,22}]	0.002	54.402	100	300
WU_DEFS::TIMES [{wu_defs.f90} {59,3}-{174,22}]	0.003	0.065	200	200
GCR_SOLVE::GCR_SOLVER_QSET [{gcr_solve.f90} {47,3}-{415,32}]	0.002	54.334	100	801
GCR_UTIL::RES_RMS_QSET [{gcr_util.f90} {375,3}-{395,29}]	0.001	0.15	100	100
GCR_UTIL::MATRIX_TO_GRID_RES [{gcr_util.f90} {313,3}-{336,35}]	0.001	0.536	100	100
GCR_UTIL::MATRIX_TO_GRID_DQ [{gcr_util.f90} {282,3}-{305,34}]	0.001	0.195	100	100
GCR_UTIL::GRID_TO_MATRIX_RES [{gcr_util.f90} {344,3}-{367,35}]	0	0.341	100	100
GCR_SOLVE_UTIL::GCR_PRECONDITIONER_QSET [{gcr_solve_util.f90} {40,3}-{131,40}]	0	53.104	100	100
LINEARSOLVE_NODIVCHECK::NODIVCHECK_RELAX_Q [{linearsolve_nodivcheck.F90} {56,14}]	0.008	53.103	100	4,900
WU_DEFS::TIMES [{wu_defs.f90} {59,3}-{174,22}]	0.02	0.34	3,200	3,200
POINT_SOLVER::POINT_SOLVE [{point_solver.F90} {31,3}-{214,28}]	0.004	52.751	1,500	1,500
POINT_SOLVER::POINT_SOLVE_5 [{point_solver.F90} {2700,3}-{2921,30}]	0.003	52.747	1,500	1,500
Loop: POINT_SOLVER::POINT_SOLVE_5 [{point_solver.F90} {2757,5}-{2917,19}]	43.649	52.744	1,500	36,000
LMPI_APP::SINGLE_START_MATRIX_XFER [{lmpi_app.F90} {7907,3}-{8132,41}]	0.271	0.512	18,000	85,500
LMPI_APP::SINGLE_MATRIX_COMPLETE_XFER [{lmpi_app.F90} {11520,3}-{11626,44}]	0.228	8.583	18,000	30,000
LMPI::LMPI_WAITALL [{lmpi.F90} {20175,3}-{20200,29}]	0.139	8.355	30,000	30,000
MPI_Waitall()	8.217	8.217	30,000	0
LMPI::INTEGR_SCALAR_REDUCE [{lmpi.F90} {4584,3}-{4611,37}]	0	0.002	100	100
LINEAR_SPECTRAL::SET_FIELD_POINTS [{linear_spectral.f90} {173,3}-{184,33}]	0	0.002	100	200

```
% export TAU_CALLPATH=1
% export TAU_CALLPATH_DEPTH=100
```

ParaProf Function Window



ParaProf Callpath Thread Relations Window

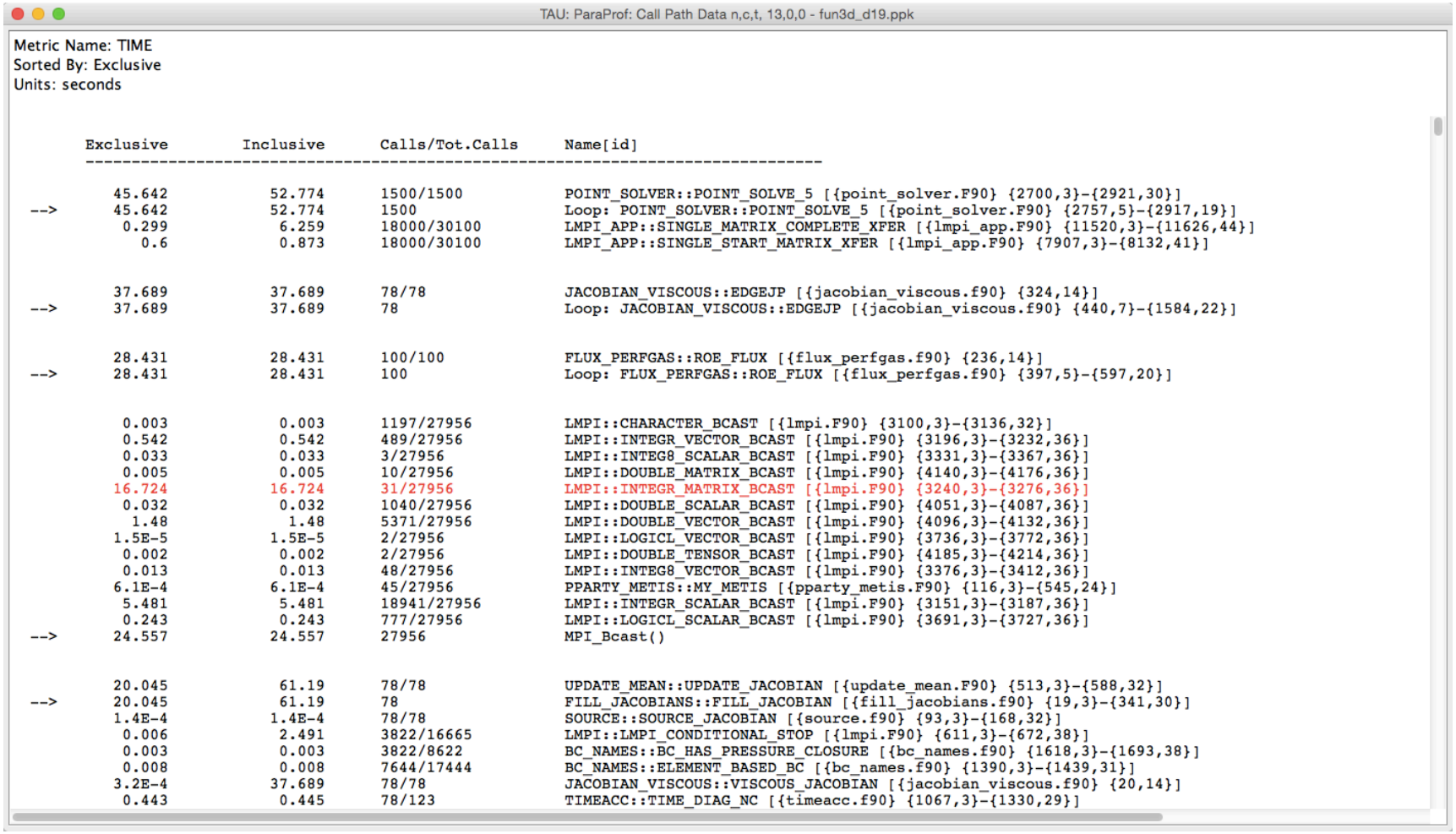
TAU: ParaProf: Call Path Data n,c,t, 5,0,0 - fun3d_d19.ppk

Metric Name: TIME
Sorted By: Exclusive
Units: seconds

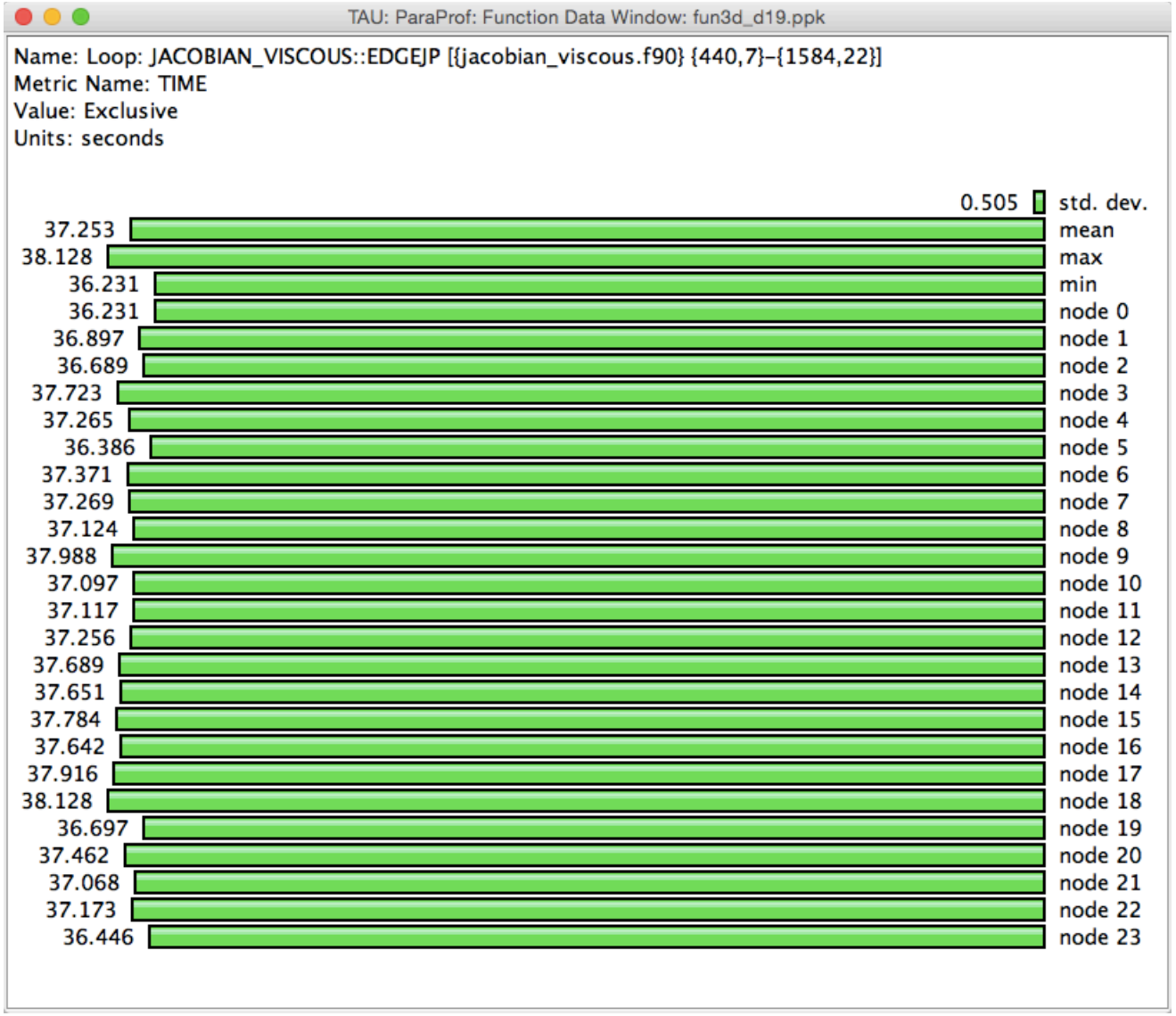
	Exclusive	Inclusive	Calls/Tot.Calls	Name[id]
	46.323	53.014	1500/1500	POINT_SOLVER::POINT_SOLVE_5 [{point_solver.F90} {2700,3}-{2921,30}]
-->	46.323	53.014	1500	Loop: POINT_SOLVER::POINT_SOLVE_5 [{point_solver.F90} {2757,5}-{2917,19}]
	0.16	6.379	18000/30100	LMPI_APP::SINGLE_MATRIX_COMPLETE_XFER [{lmpi_app.F90} {11520,3}-{11626,44}]
	0.166	0.311	18000/30100	LMPI_APP::SINGLE_START_MATRIX_XFER [{lmpi_app.F90} {7907,3}-{8132,41}]
	36.386	36.386	78/78	JACOBIAN_VISCOUS::EDGEJP [{jacobian_viscous.f90} {324,14}]
-->	36.386	36.386	78	Loop: JACOBIAN_VISCOUS::EDGEJP [{jacobian_viscous.f90} {440,7}-{1584,22}]
	4.7E-4	4.7E-4	45/27956	PPARTY_METIS::MY_METIS [{pparty_metis.F90} {116,3}-{545,24}]
	0.242	0.242	777/27956	LMPI::LOGICL_SCALAR_BCAST [{lmpi.F90} {3691,3}-{3727,36}]
	0.002	0.002	2/27956	LMPI::DOUBLE_TENSOR_BCAST [{lmpi.F90} {4185,3}-{4214,36}]
	16.689	16.689	31/27956	LMPI::INTEGR_MATRIX_BCAST [{lmpi.F90} {3240,3}-{3276,36}]
	0.013	0.013	48/27956	LMPI::INTEG8_VECTOR_BCAST [{lmpi.F90} {3376,3}-{3412,36}]
	0.587	0.587	489/27956	LMPI::INTEGR_VECTOR_BCAST [{lmpi.F90} {3196,3}-{3232,36}]
	0.003	0.003	1197/27956	LMPI::CHARACTER_BCAST [{lmpi.F90} {3100,3}-{3136,32}]
	1.521	1.521	5371/27956	LMPI::DOUBLE_VECTOR_BCAST [{lmpi.F90} {4096,3}-{4132,36}]
	1.2E-5	1.2E-5	2/27956	LMPI::LOGICL_VECTOR_BCAST [{lmpi.F90} {3736,3}-{3772,36}]
	0.012	0.012	1040/27956	LMPI::DOUBLE_SCALAR_BCAST [{lmpi.F90} {4051,3}-{4087,36}]
	0.055	0.055	3/27956	LMPI::INTEG8_SCALAR_BCAST [{lmpi.F90} {3331,3}-{3367,36}]
	10.328	10.328	18941/27956	LMPI::INTEGR_SCALAR_BCAST [{lmpi.F90} {3151,3}-{3187,36}]
	0.005	0.005	10/27956	LMPI::DOUBLE_MATRIX_BCAST [{lmpi.F90} {4140,3}-{4176,36}]
-->	29.457	29.457	27956	MPI_Bcast()
	27.158	27.158	100/100	FLUX_PERFGAS::ROE_FLUX [{flux_perfgas.f90} {236,14}]
-->	27.158	27.158	100	Loop: FLUX_PERFGAS::ROE_FLUX [{flux_perfgas.f90} {397,5}-{597,20}]

Shows the contribution of parents and children for each routine (marked by an arrow)

ParaProf Callpath Thread Relations Window



ParaProf Function Window

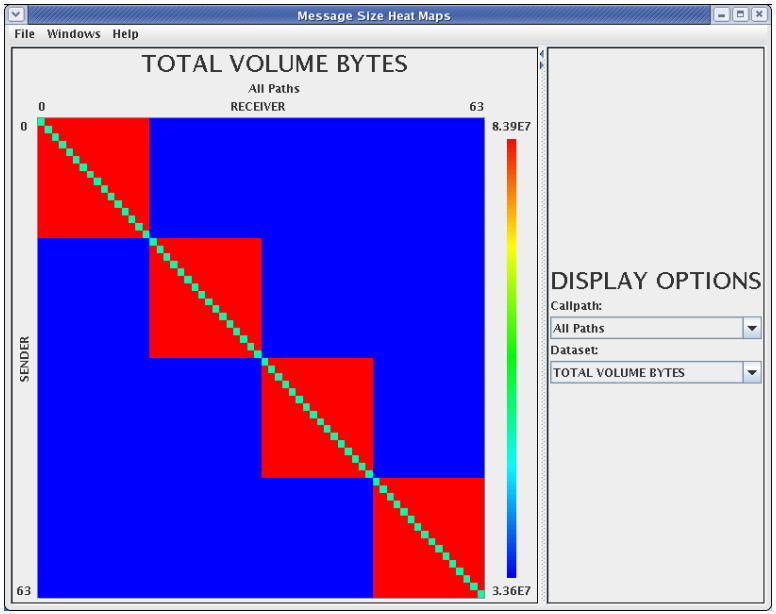


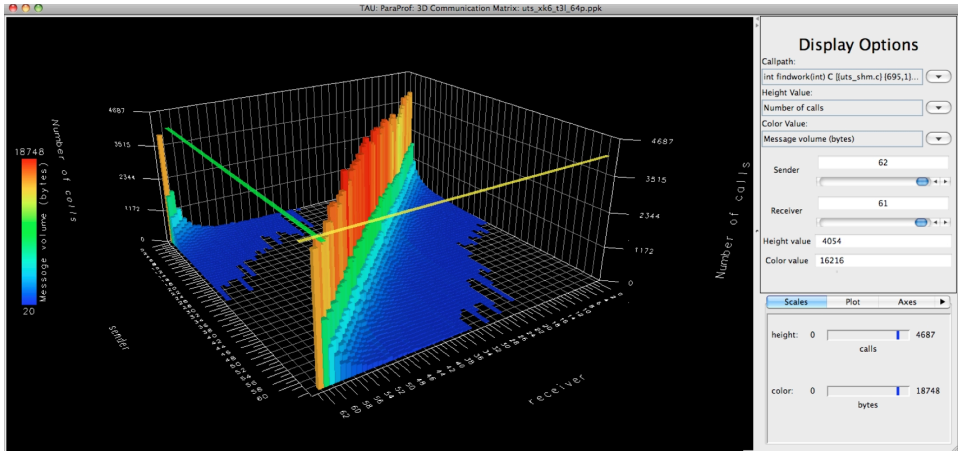
Generating Communication Matrix

```
% source tau.bashrc
% export TAU_MAKEFILE=$TAU/Makefile.tau-icpc-papi-mpi-pdt
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)

% mxterm 1 16 40
% export TAU_COMM_MATRIX=1
% mpirun -np 4    ./a.out

% paraprof
(Windows -> Communication Matrix)
(Windows -> 3D Communication Matrix)
```

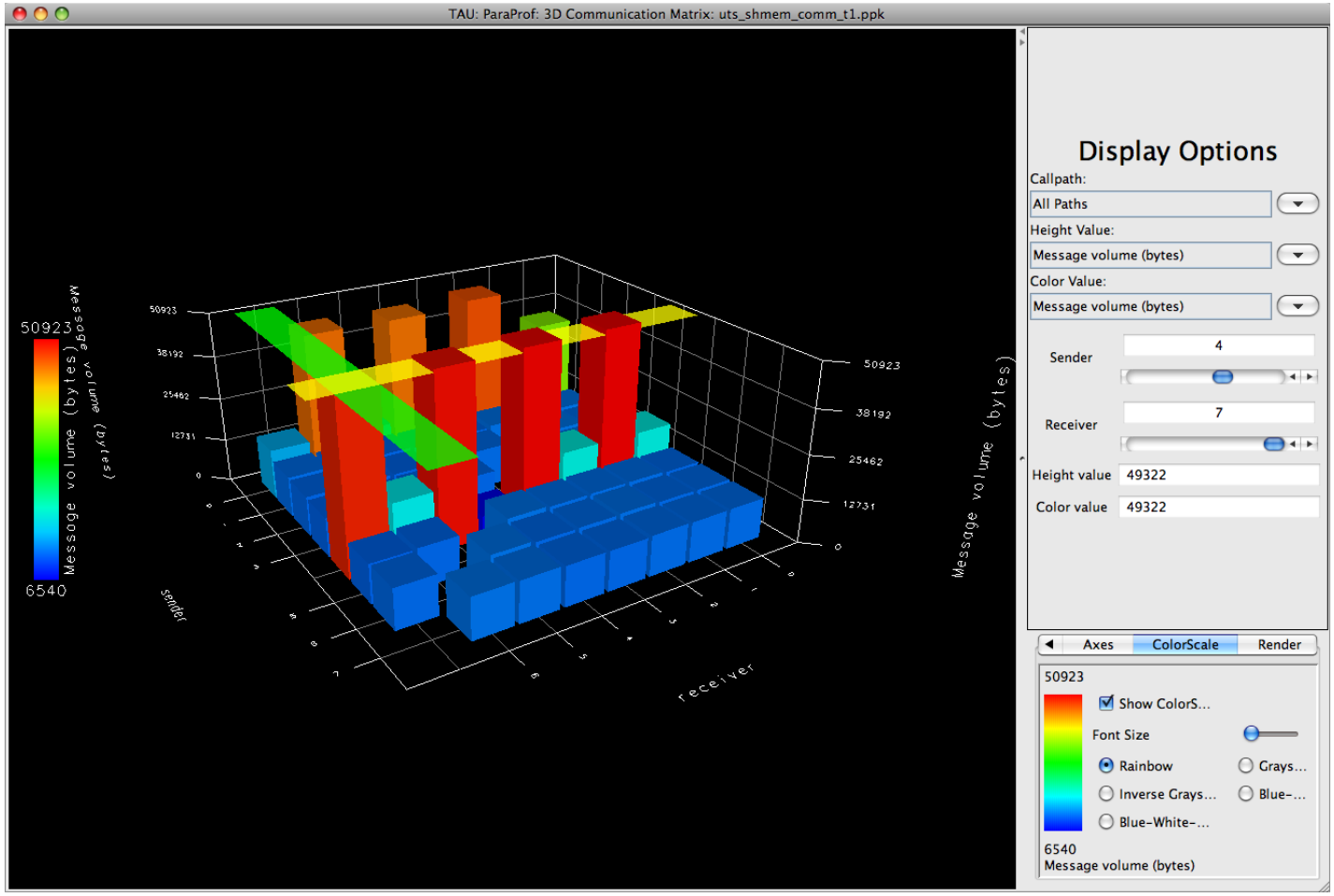
Communication Matrix Display

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





SHMEM Communication Matrix



Compiler-based Instrumentation

- Compiler automatically **emits instrumentation calls** in the object code instead of parsing the source code using PDT
- To enable: export TAU_OPTIONS="-optCompInst"
- Configure TAU with "-bfd=download" for best results

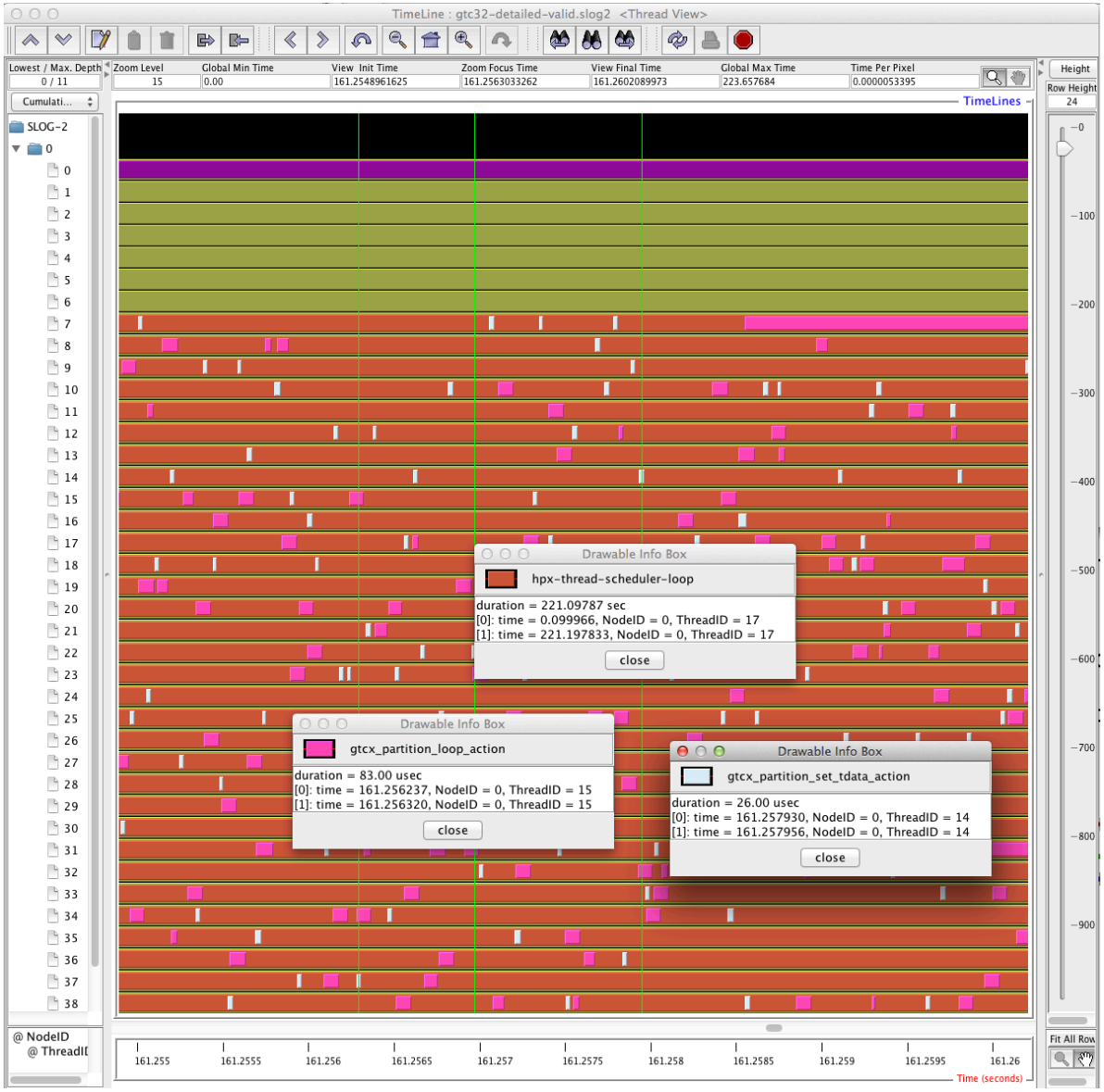
Use Compiler-Based Instrumentation

```
% source tau.bashrc
% export TAU_MAKEFILE=$TAU/Makefile.tau-icpc-papi-mpi-pdt
% export TAU_OPTIONS='-optCompInst -optQuiet'

% make CC=tau_cc.sh CXX=tau_cxx.sh F90=tau_f90.sh

NOTE: You may also use the short-hand scripts taucc, tauf90,
taucxx instead of specifying TAU_OPTIONS and using the
traditional tau_<cc,cxx,f90>.sh scripts. These scripts use
compiler-based instrumentation by default.

% make CC=taucc CXX=taucxx F90=tauf90

% mpirun -np 4 ./a.out
% paraprof --pack app.ppk
  Move the app.ppk file to your desktop.
% paraprof app.ppk
```

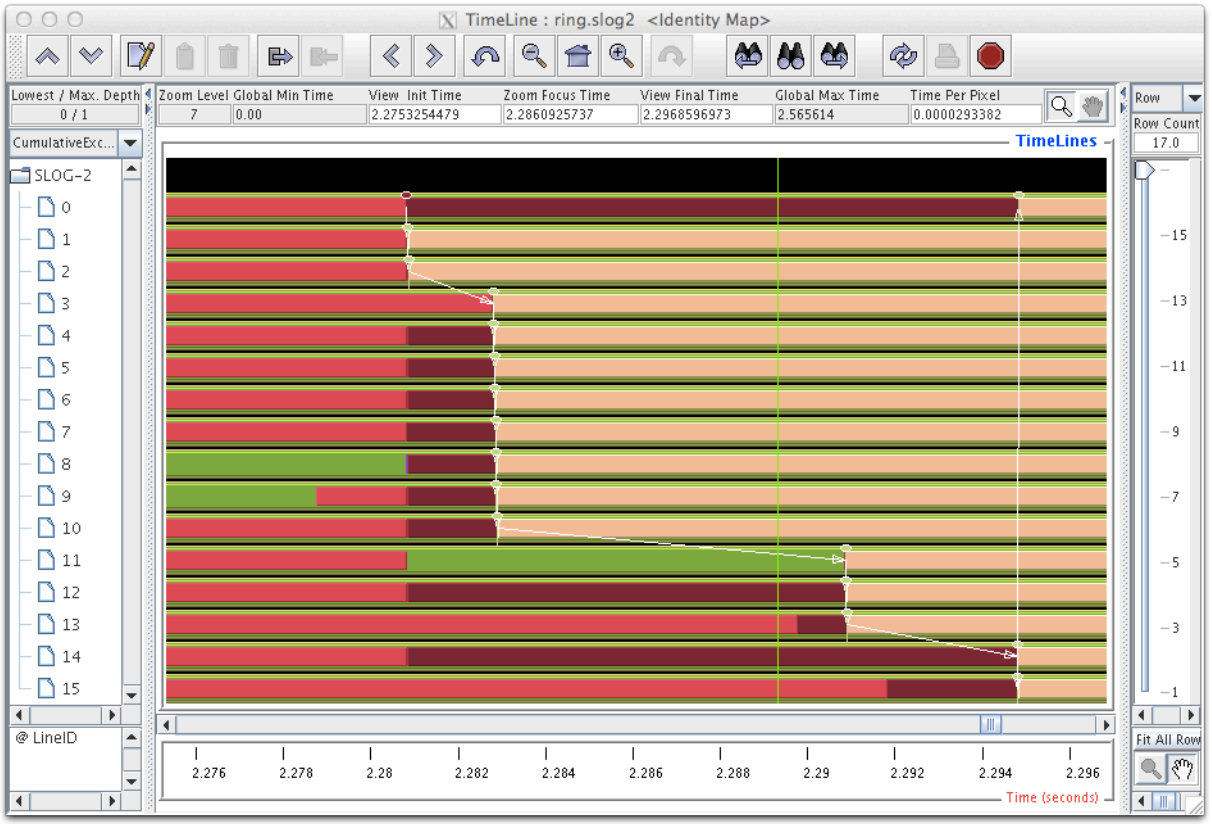
Compiler-based Instrumentation

```
% export TAU_MAKEFILE=$TAU/Makefile.tau-icpc-papi-mpi-pdt
% export TAU_OPTIONS='-optCompInst -optQuiet'

% make CC=tau_cc.sh CXX=tau_cxx.sh F90=tau_f90.sh

NOTE: You may also use the short-hand scripts taucc, tauf90,
taucxx instead of specifying TAU_OPTIONS and using the
traditional tau_<cc,cxx,f90>.sh scripts. These scripts use
compiler-based instrumentation by default.

% make CC=taucc CXX=taucxx F90=tauf90

% mpirun -np 4 ./a.out
% paraprof --pack app.ppk
  Move the app.ppk file to your desktop.
% paraprof app.ppk
```

Jumpshot Trace Visualizer in TAU

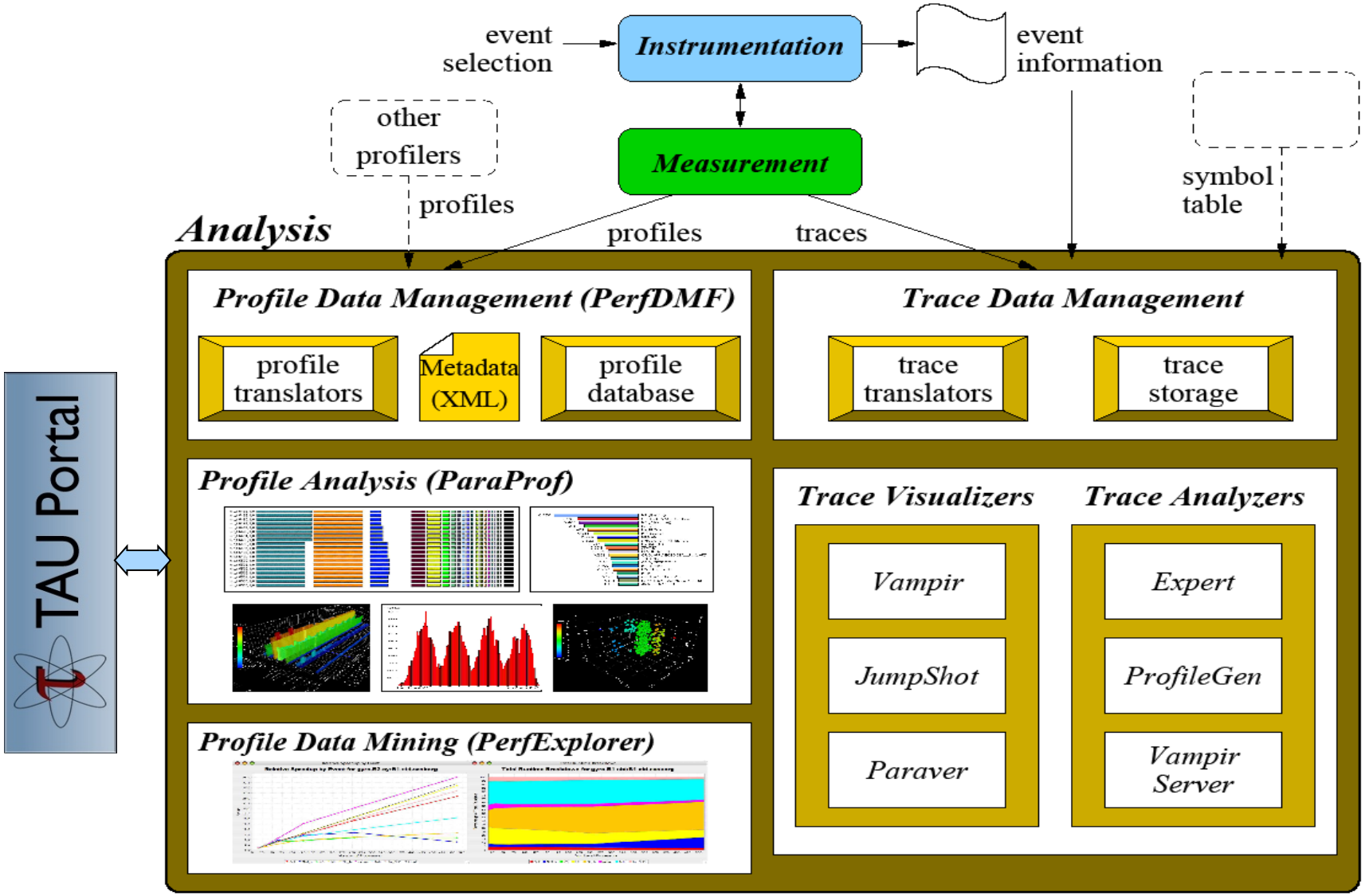


Tracing Communication in Jumpshot




```
% export TAU_MAKEFILE=$TAU/Makefile.tau-icpc-papi-mpi-pdt
% cmake –DCMAKE_CXX_COMPILER=tau_cxx.sh; make –j 8
% export TAU_TRACE=1
% mpirun –np 16 ./a.out ; tau_treemerge.pl; tau2slog2 tau.trc tau.edf –o a.slog2
% jumpshot a.slog2 &
```

Performance Analysis



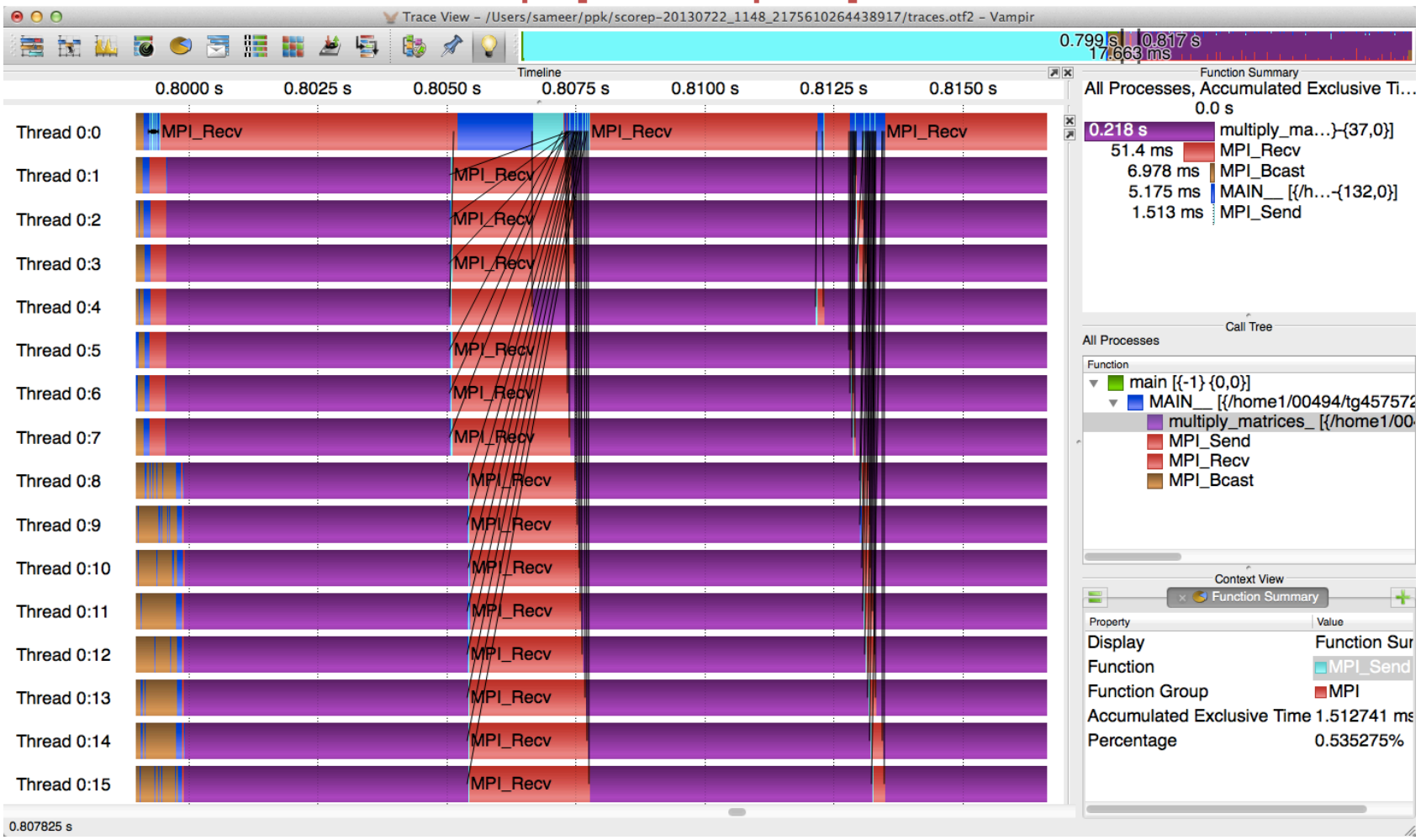
Generating Event Traces

```
% source tau.bashrc
% export TAU_MAKEFILE=$TAU/Makefile.tau-icpc-papi-mpi-pdt
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)

% mxterm 1 16 40
% export TAU_TRACE=1
% mpirun -np 4    ./a.out

Merge and convert the tracefiles:
% tau_treemerge.pl
For Vampir (OTF):
% tau2otf tau.trc tau.edf app.otf; vampir app.otf

For Jumpshot (SLOG2):
% tau.trc tau.edf -o app.slog2; jumpshot app.slog2

For ParaVer:
% tau_convert -paraver tau.trc tau.edf app.prv; paraver app.prv
```

Generating a Trace File Using Score-P

Goal: Identify the temporal aspect of performance. What happens in my code at a given time? When?

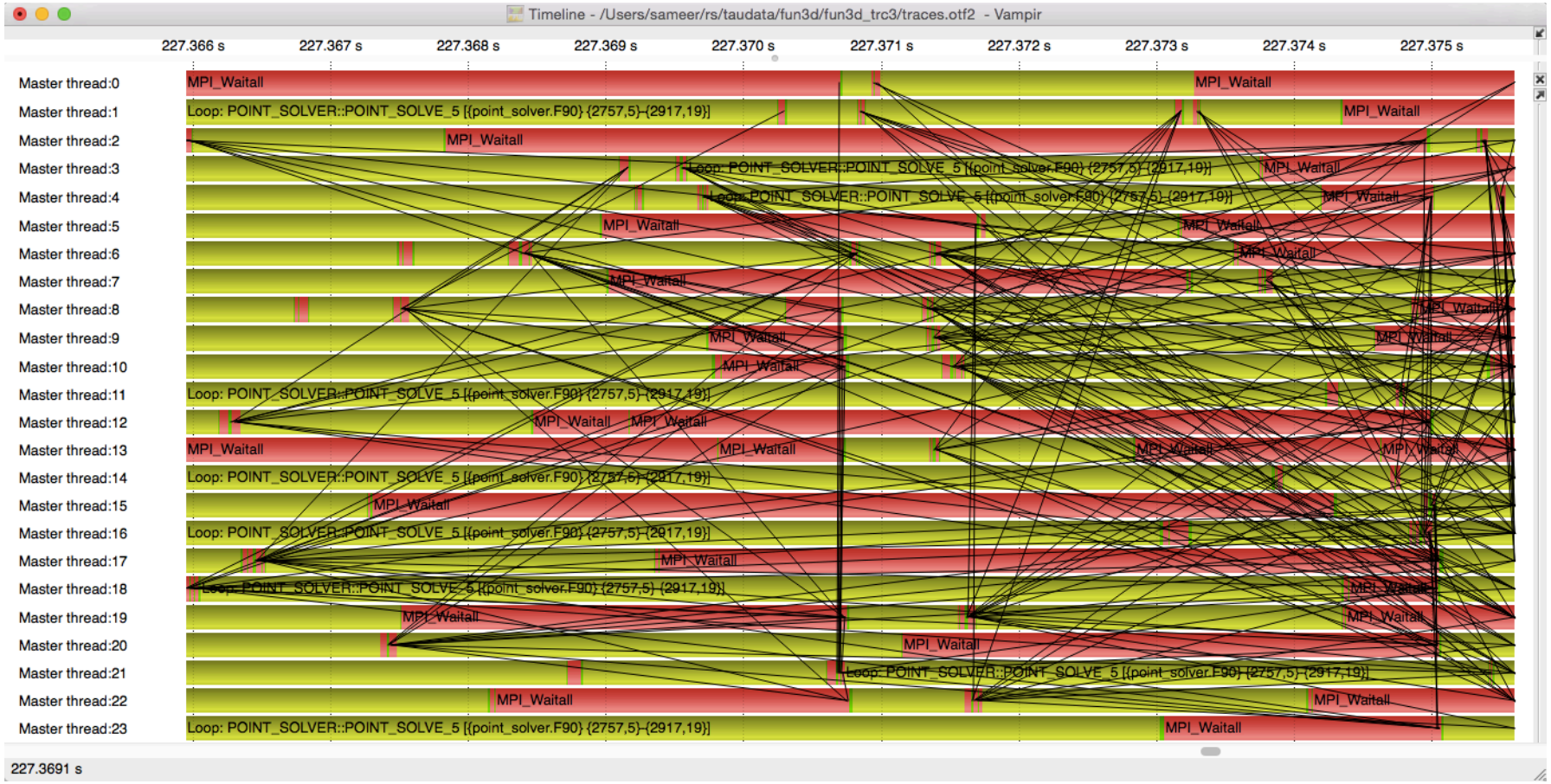
Event trace visualized in Vampir [www.vampir.eu]



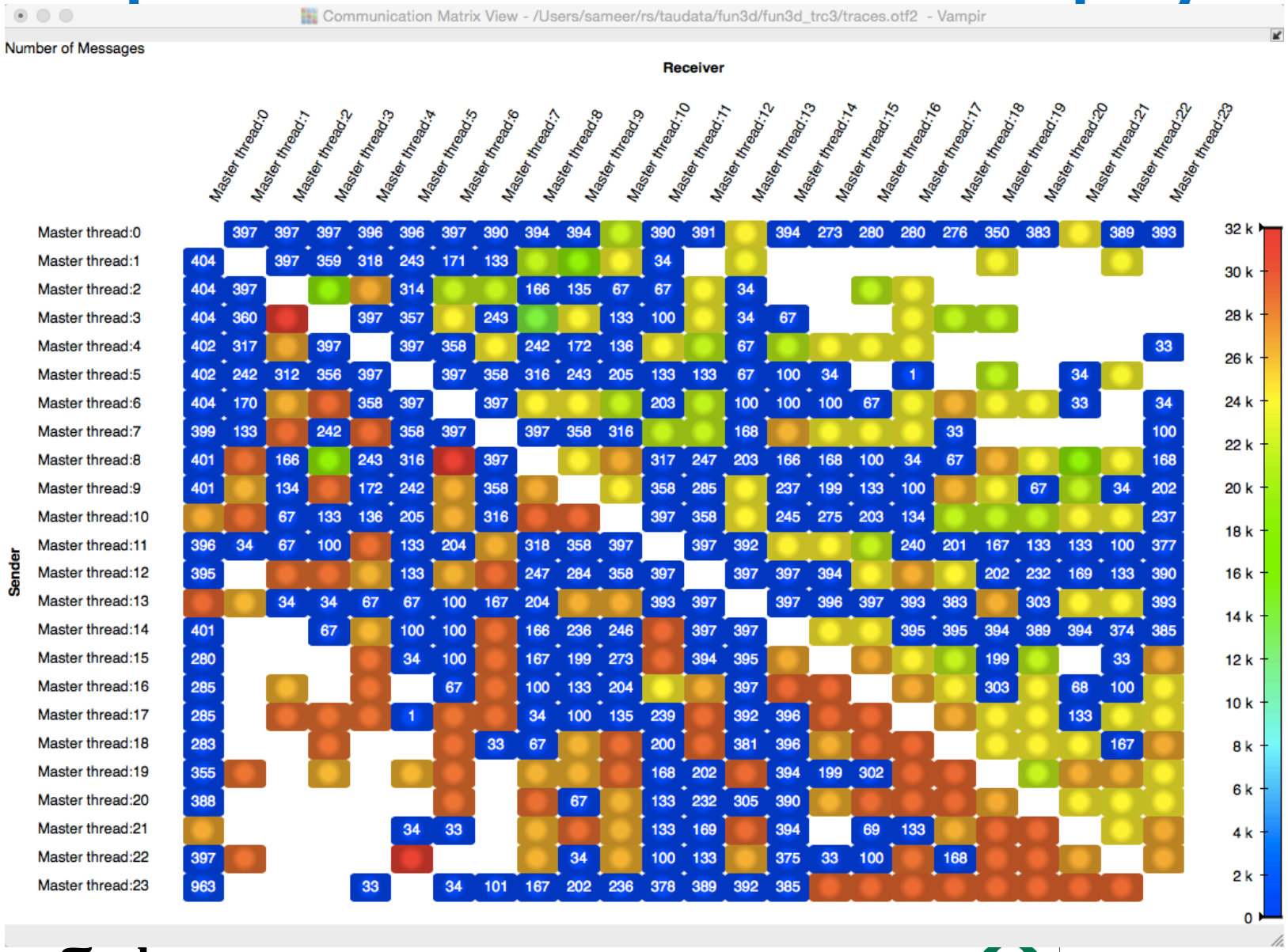
Vampir [T.U. Dresden] Timeline Display

Rank 0 performs I/O while all other ranks wait in MPI_Bcast()

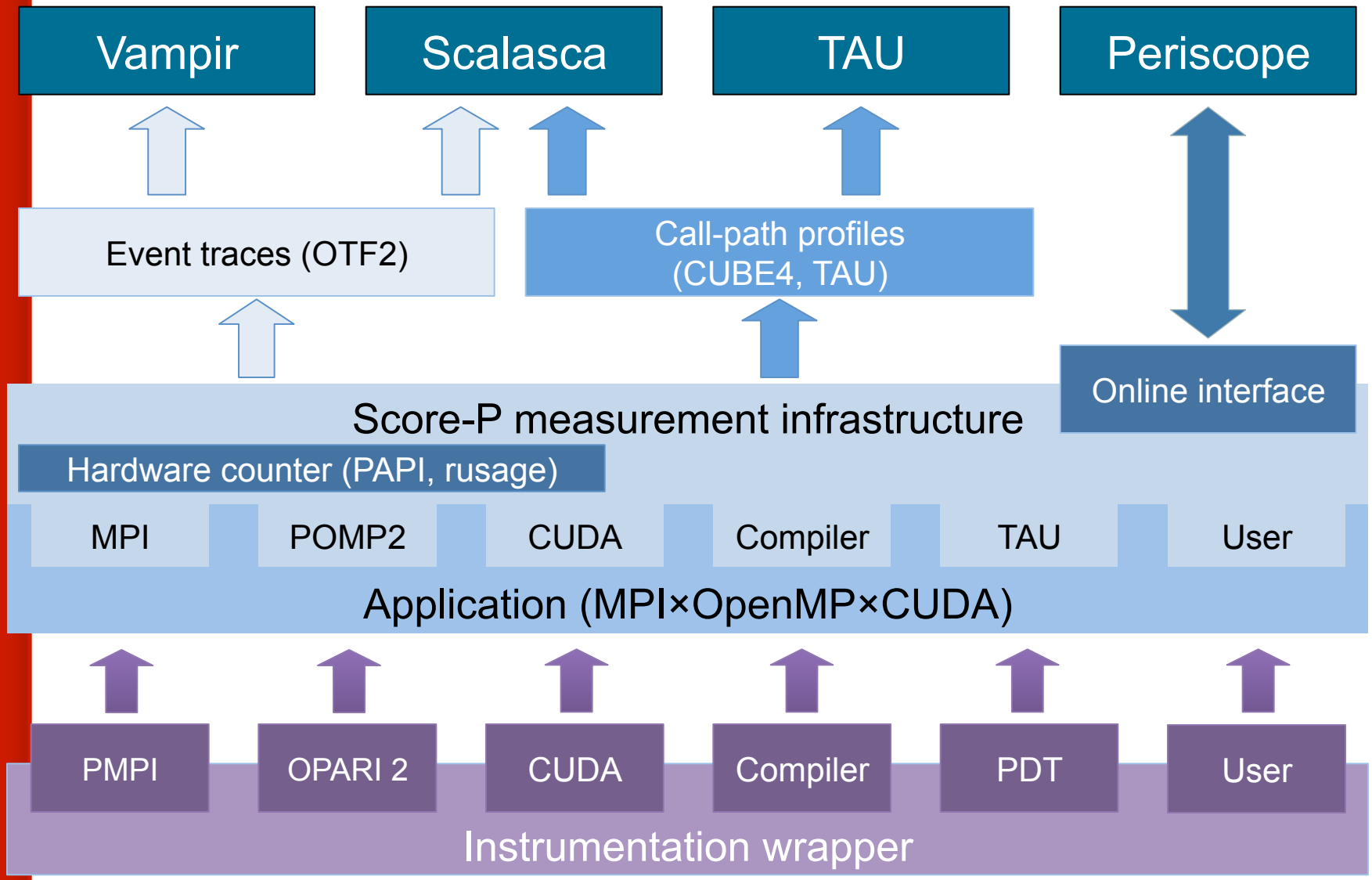
Vampir Timeline Display



Vampir Communication Matrix Display



Score-P Architecture: Common Layer



Score-P Partners

Forschungszentrum Jülich, Germany

German Research School for Simulation Sciences, Aachen, Germany

Gesellschaft für numerische Simulation mbH Braunschweig, Germany

RWTH Aachen, Germany

Technische Universität Dresden, Germany

Technische Universität München, Germany

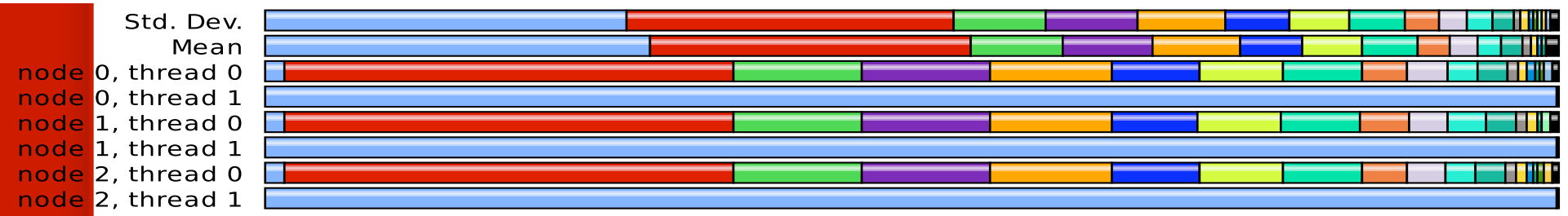
University of Oregon, Eugene, USA

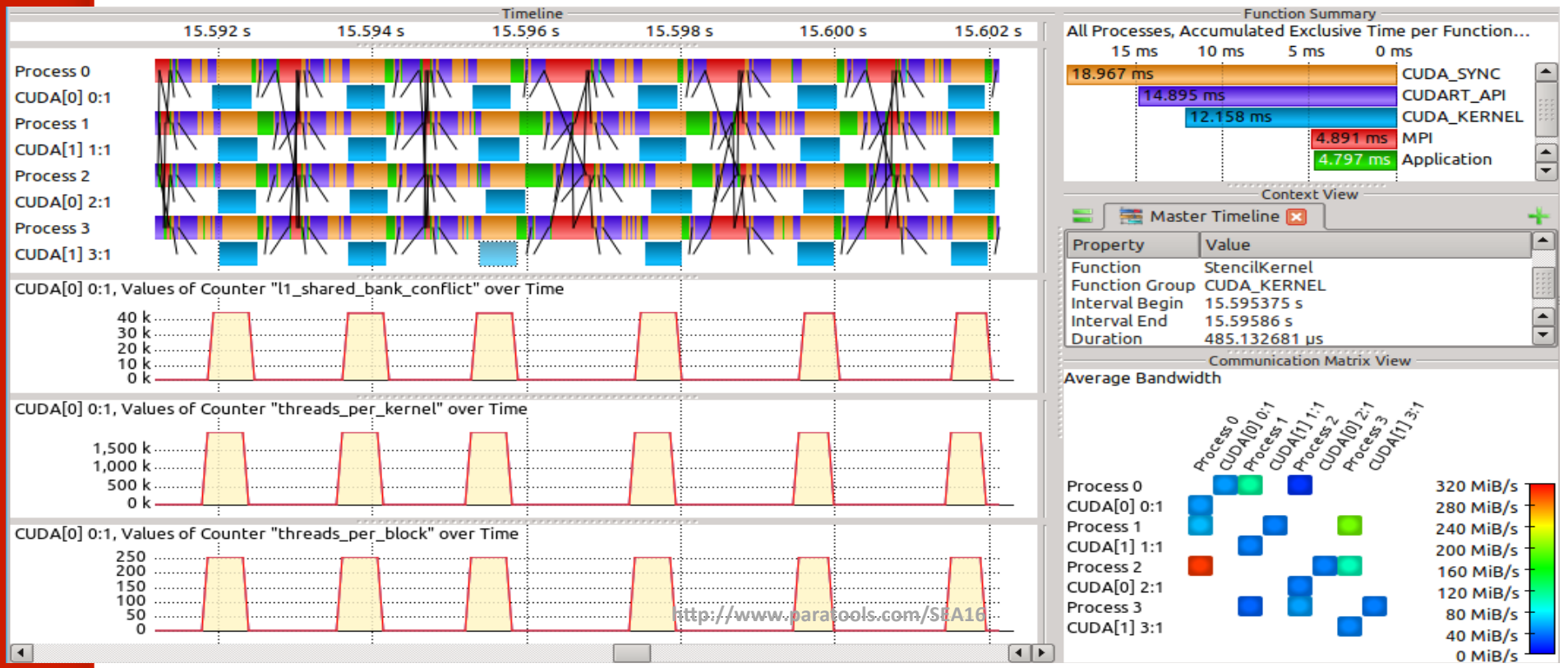
Generating OTF 2 Event Traces

```
% source tau.bashrc
% export TAU_MAKEFILE=$TAU/Makefile.tau-icpc-mpi-pdt-scorep
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)

% mxterm 1 16 40
% mpirun -np 256 ./a.out
% cd scorep-<id>/
% vampir traces.otf2
Or use Vampirserver and connect vampir running locally to a remote
    Vampirserver

NOTE: Vampir is a commercial product [www.vampir.eu]
There is no need to merge or convert traces (an expensive operation!)
```

Stencil2D Parallel Profile / Trace in Vampir

Metric: TAUGPU_TIME
Value: Exclusive





Tags in tau_exec and other tools

```
% cd $TAU; ls Makefile.*
Makefile.tau-icpc-papi-mpi-pdt

% mxterm 1 16 40; mpirun -np 4 ./matrix
% tau_exec -T icpc,mpi,pdt ./a.out

Chooses Makefile.tau-icpc-mpi,pdt and associated libraries.

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

-T <list_of_tags> is used in several TAU tools:

- tau_run
- tau_rewrite
- tau_exec
- tau_gen_wrapper

Three Instrumentation Techniques for Wrapping External Libraries

Pre-processor based substitution by re-defining a call (e.g., read)

- Tool defined header file with same name <*unistd.h*> takes precedence
- Header redefines a routine as a different routine using macros
- Substitution: *read()* substituted by preprocessor as *tau_read()* at callsite

Preloading a library at runtime

- Library preloaded (*LD_PRELOAD* env var in Linux) in the address space of executing application intercepts calls from a given library
- Tool's wrapper library defines *read(),* gets address of global *read()* symbol (dlsym), internally calls timing calls around call to global read

Linker based substitution

- Wrapper library defines *__wrap_read* which calls *__real_read* and linker is passed *-Wl,-wrap,read* to substitute all references to read from application's object code with the *__wrap_read* defined by the tool

Preprocessor based substitution

Pre-processor based substitution by re-defining a call

- Compiler replaces read() with tau_read() in the body of the source code

Advantages:

- Simple to instrument
  - Preprocessor based replacement
  - A header file redefines the calls
  - No special linker or runtime flags required

Disadvantages

- Only works for C & C++ for replacing calls in the body of the code.
- Incomplete instrumentation: fails to capture calls in uninstrumented libraries (e.g., libhdf5.a)

Linker based substitution

Linker based substitution

- Wrapper library defines __wrap_read which calls __real_read and linker is passed -Wl,-wrap, read

Advantages

- Tool can intercept all references to a given call
- Works with static as well as dynamic executables
- No need to recompile the application source code, just re-link the application objects and libraries with the tool wrapper library

Disadvantages

- Wrapping an entire library can lengthen the linker command line with multiple –Wl,-wrap,<func> arguments. It is better to store these arguments in a file and pass the file to the linker
- Approach does not work with un-instrumented binaries

tau_gen_wrapper

Automates creation of wrapper libraries using TAU

Input:

- header file (foo.h)
- library to be wrapped (/path/to/libfoo.a)
- technique for wrapping
  - Preprocessor based redefinition (-d)
  - Runtime preloading (-r)
  - Linker based substitution (-w: default)
- Optional selective instrumentation file (-f select)
  - Exclude list of routines, or
  - Include list of routines

Output:

- wrapper library
- optional *link_options.tau* file (-w), pass –optTauWrapFile=<file> in TAU_OPTIONS environment variable

Design of wrapper generator (tau_gen_wrapper)

***tau_gen_wrapper* shell script:**

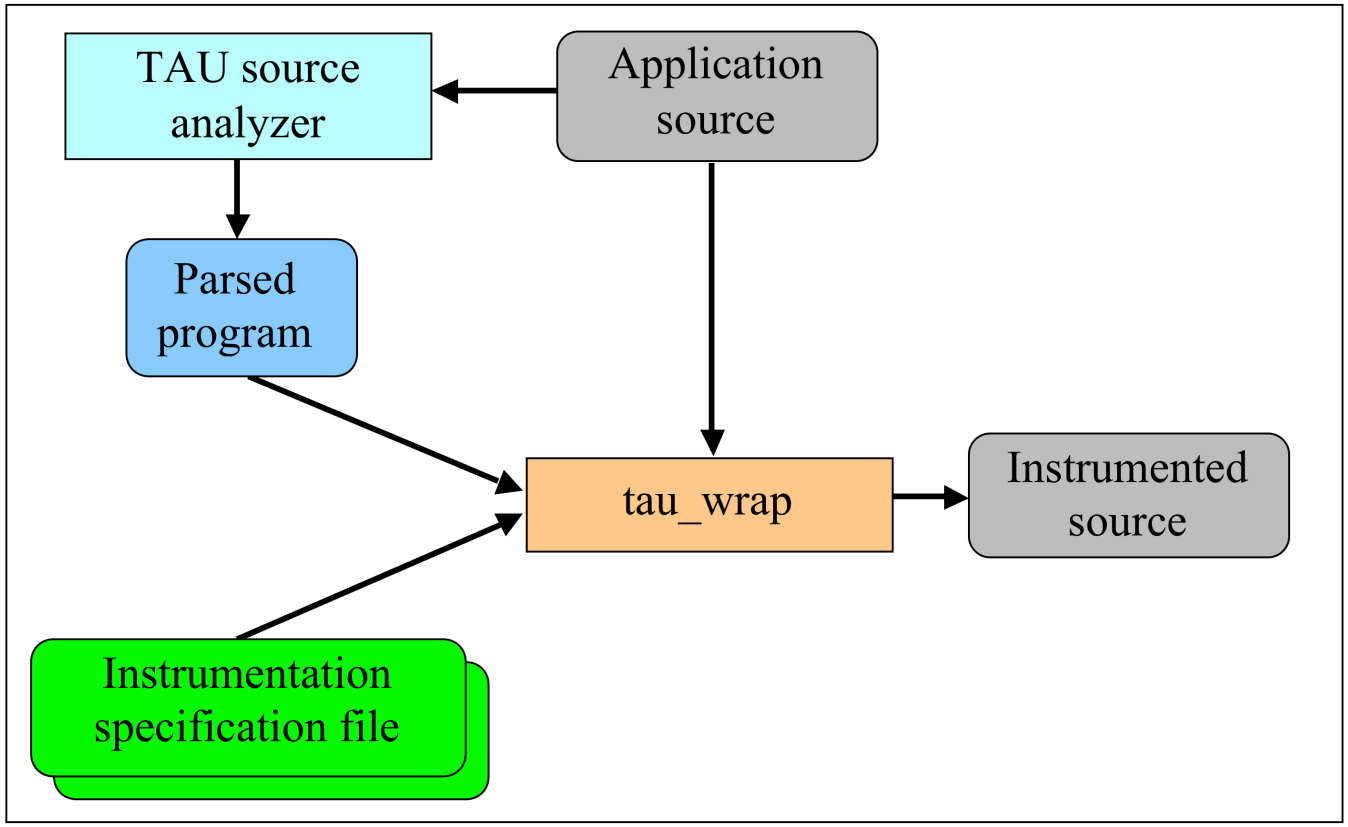
- parses source of header file using static analysis tool Program Database Toolkit (PDT)
- Invokes *tau_wrap*, a tool that generates
  - instrumented wrapper code,
  - an optional *link_options.tau* file (for linker-based substitution, -w)
  - Makefile for compiling the wrapper interposition library
- Builds the wrapper library using make

Use TAU_OPTIONS environment variable to pass location of link_options.tau file using

```
% export TAU_OPTIONS='-optTauWrapFile=<path/to/
link_options.tau> -optVerbose'
```

Use *tau_exec –loadlib=<wrapperlib.so>* to pass location of wrapper library for preloading based substitution

tau_wrap


TAU source analyzer
Application source
Parsed program
tau_wrap
Instrumented source
Instrumentation specification file

Using POSIX I/O wrapper library

Setting environment variable TAU_OPTIONS=-optTrackIO links in TAU's wrapper interposition library using linker-based substitution

Instrumented application generates bandwidth, volume data

Workflow:

- % export TAU_OPTIONS= '-optTrackIO –optVerbose'
- % export TAU_MAKEFILE=$TAU/Makefile.tau-icpc-papi-mpi-pdt
- % make CC=tau_cc.sh CXX=tau_cxx.sh F90=tau_f90.sh
- % mpirun –np 8 ./a.out
- % paraprof

Get additional data regarding individual arguments by setting environment variable TAU_TRACK_IO_PARAMS=1 prior to running

Preloading a wrapper library

Preloading a library at runtime

- Tool defines read(), gets address of global read() symbol (dlsym), internally calls timing calls around call to global read
- *tau_exec* tool uses this mechanism to intercept library calls

Advantages

- No need to re-compile or re-link the application source code
- Drop-in replacement library implemented using LD_PRELOAD environment variable under Linux, Cray CNL, IBM BG/P CNK, Solaris…

Disadvantages

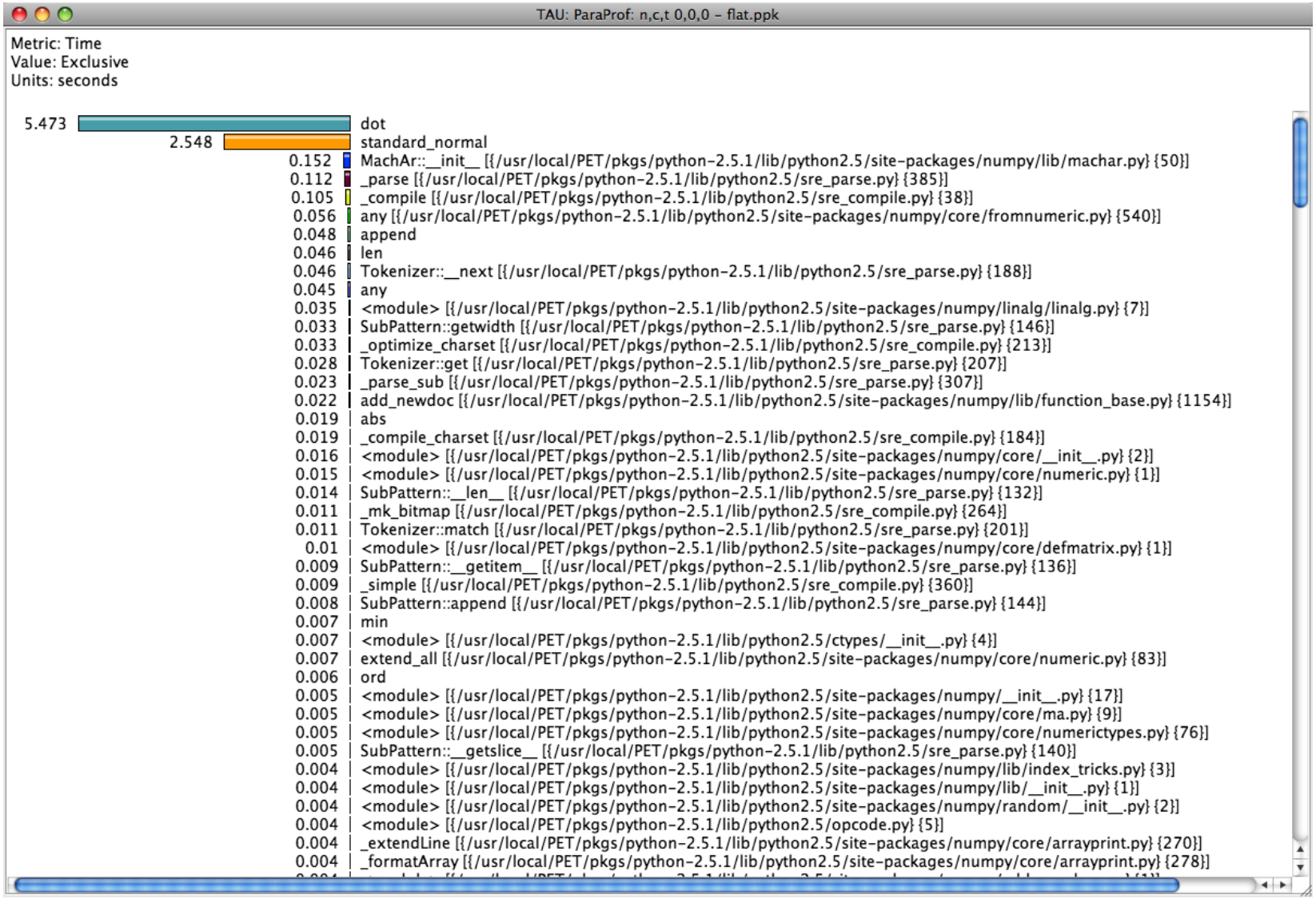
- Only works with dynamic executables. Default compilation mode under Cray XE6 and IBM BG/P is to use static executables
- Not all operating systems support preloading of dynamic shared objects (DSOs)

Profiling Python applications

- Create a top-level Python wrapper
- Launch tau_python …

```
% mpirun.lsf python ./app.py
% mpirun.lsf tau_python ./app.py

% paraprof
```

Profiling Python applications



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

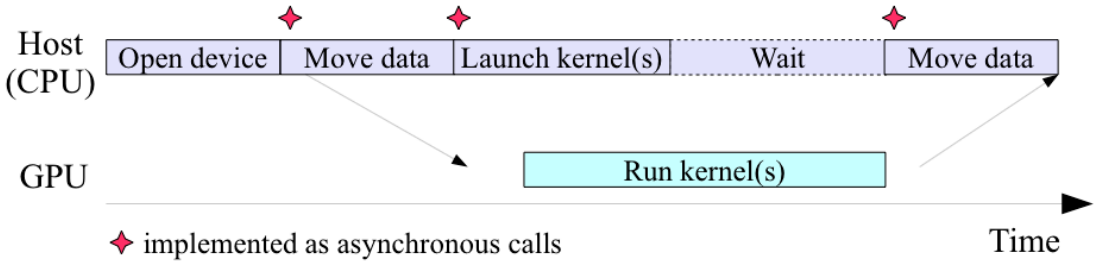
  % ./a.out (uninstrumented)

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

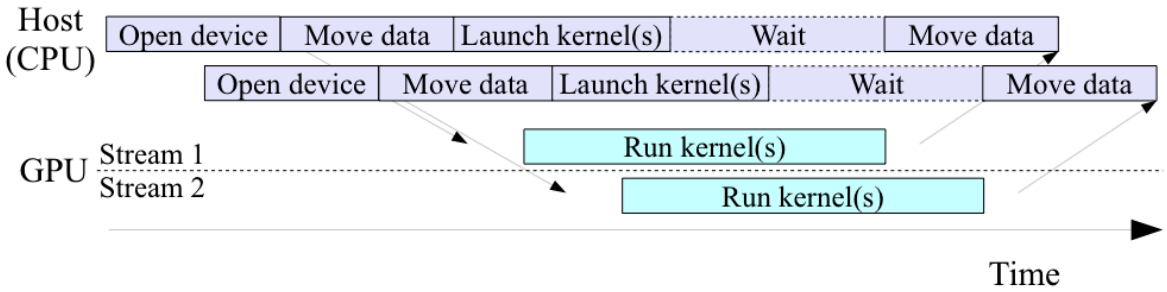
  % paraprof

Host (CPU) - GPU Scenarios

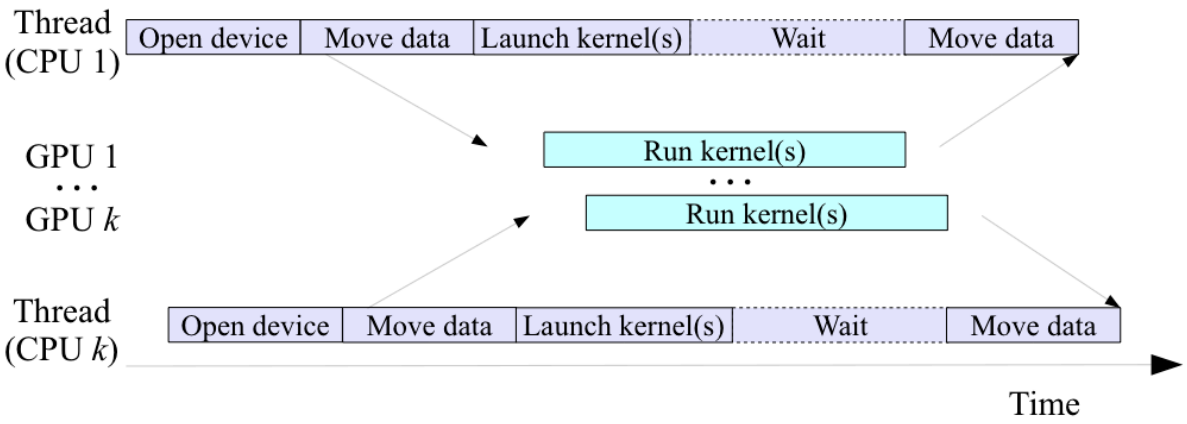
Single GPU



Multi-stream

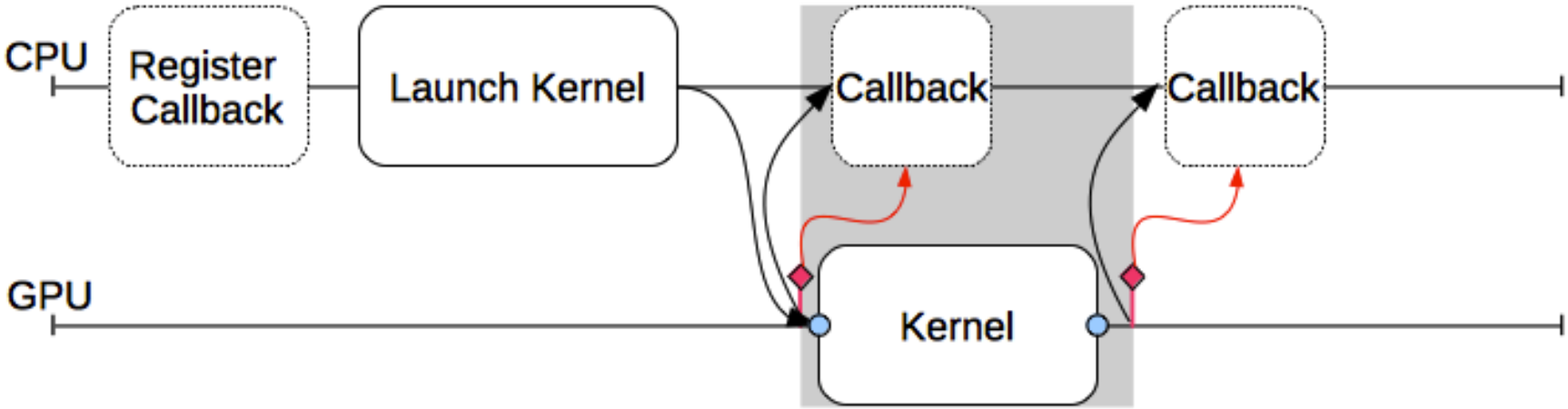


Multi-CPU, Multi-GPU



Host-GPU Measurement – Callback Method

- GPU driver libraries provide callbacks for certain routines and captures measurements
- Measurement tool registers the callbacks and processes performance data
- Application code is not modified



Method Support and Implementation

Synchronous method

- Place instrumentation appropriately around GPU calls (kernel launch, library routine, …)
- Wrap (synchronous) library with performance tool

Event queue method

- Utilize CUDA and OpenCL event support
- Again, need instrumentation to create and insert events in the streams with kernel launch and process events
- Can be implemented with driver library wrapping

Callback method

- Utilize language-level callback support in OpenCL
- Utilize NVIDIA CUDA Performance Tool Interface (CUPTI)
- Need to appropriately register callbacks

GPU Performance Measurement Tools

Support the Host-GPU performance perspective

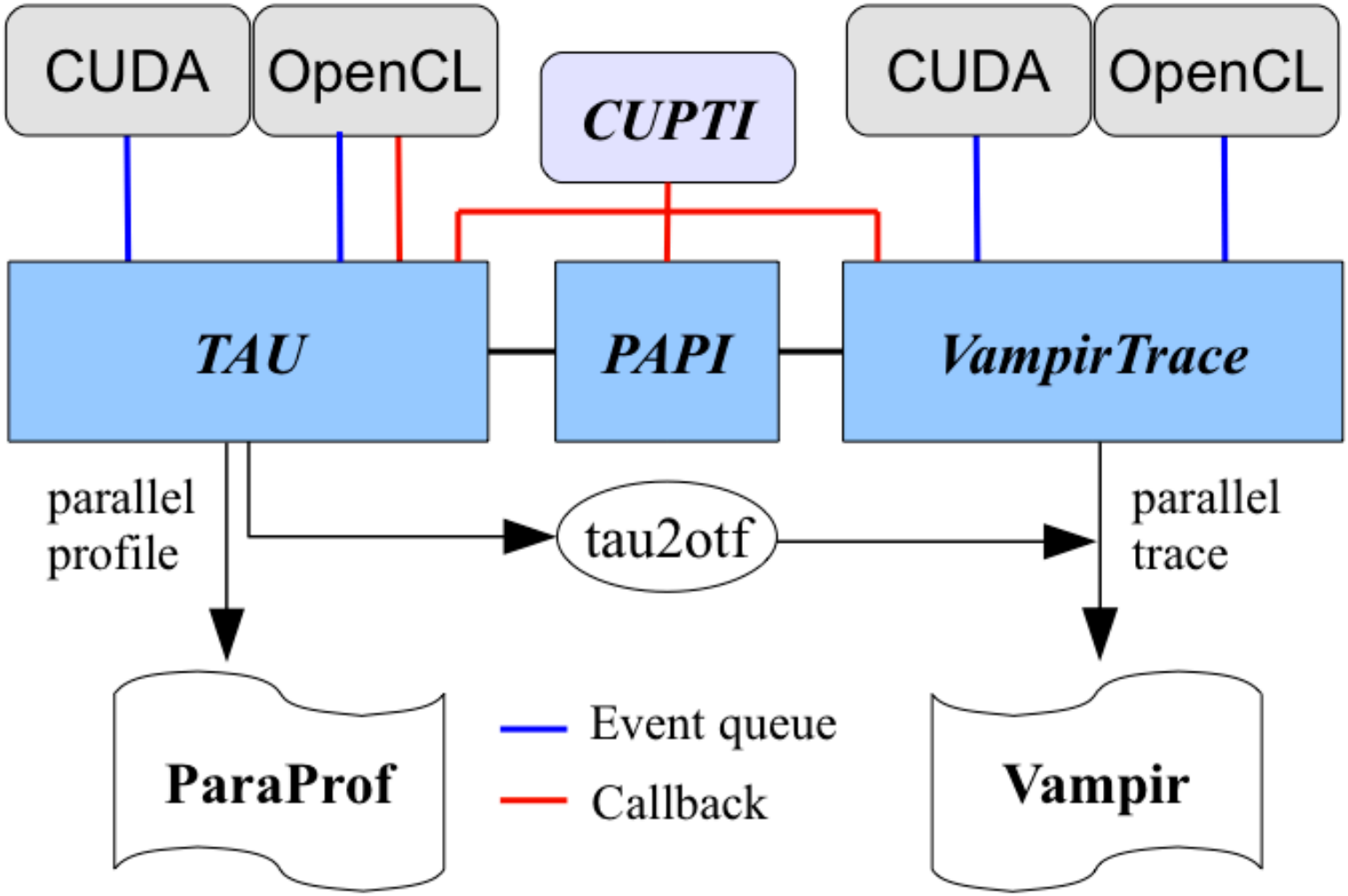
Provide integration with existing measurement system to facilitate tool use

Utilize support in GPU driver library and device

Tools

- TAU performance system
- Vampir
- PAPI
- NVIDIA CUPTI

GPU Performance Tool Interoperability



NVIDIA CUPTI

NVIDIA is developing CUPTI to enable the creation of profiling and tracing tools

Callback API

- Interject tool code at the entry and exist to each CUDA runtime and driver API call

Counter API

- Query, configure, start, stop, and read the counters on CUDA-enabled devices

CUPTI is delivered as a dynamic library

CUPTI is released with CUDA 4.0+

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 serial,cupti –cupti ./a.out
% paraprof
```

Example: SDK simpleMultiGPU

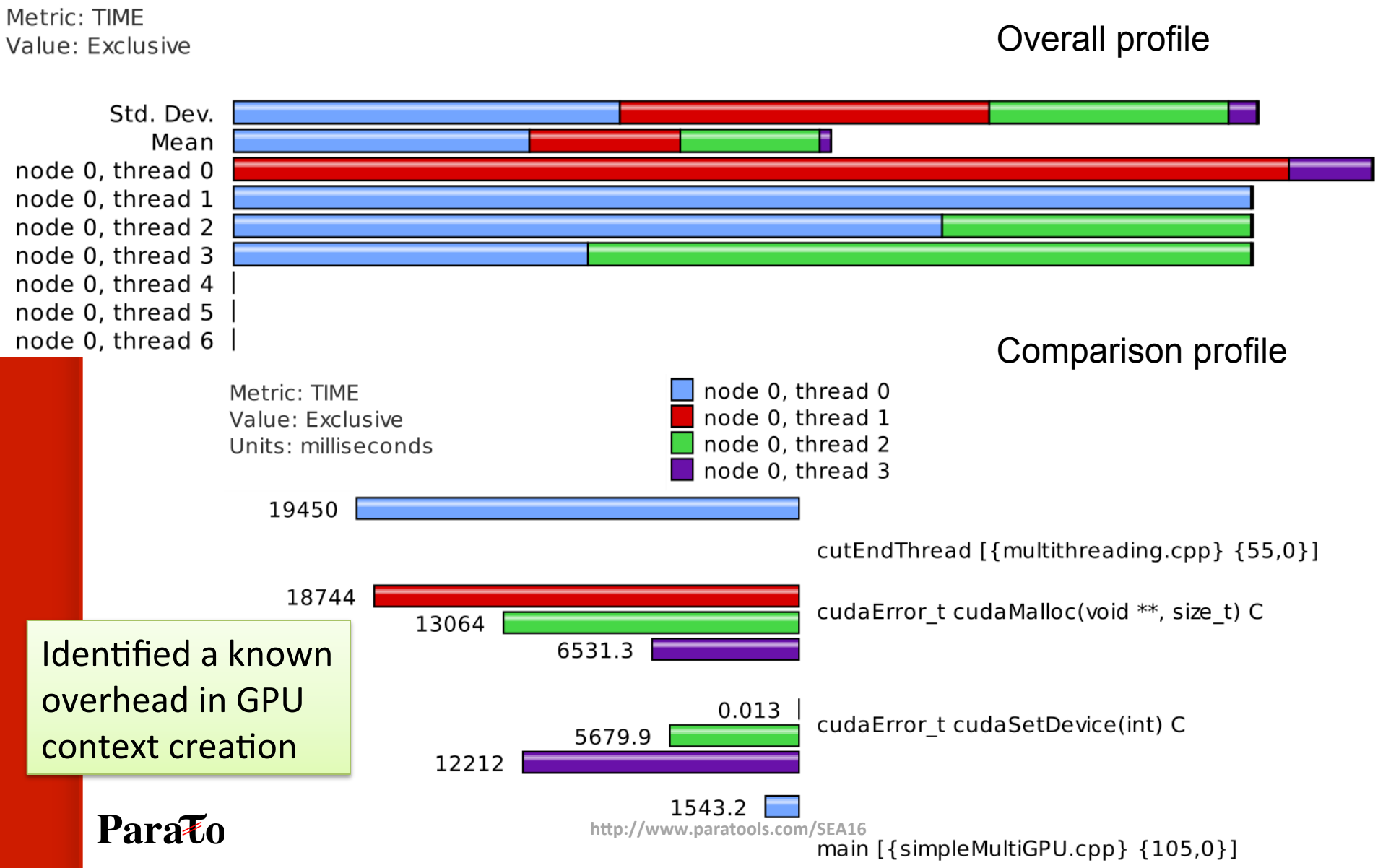
Demonstration of multiple GPU device use

main  ***solverThread***  ***reduceKernel***

Performance profile for:

- One *main* thread
- Three *solverThread* threads
- Three *reduceKernel* "threads"

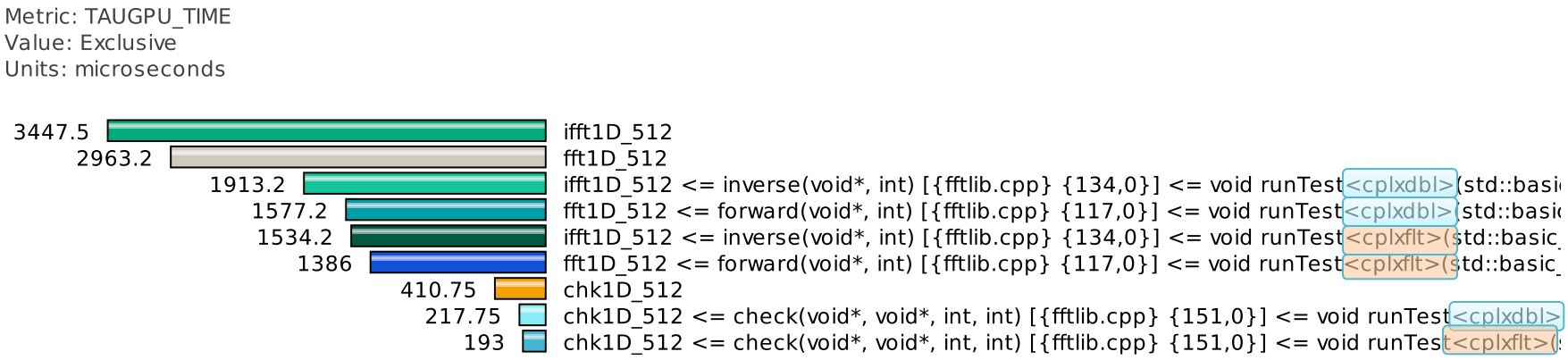
simpleMultiGPU Profile



Identified a known overhead in GPU context creation

SHOC FFT Profile with Callsite Info

TAU is able to associate callsite context information with kernel launch so that different kernel calls can be distinguished



Each kernel (ifft1D_512, fft1D_512 and chk1D_512) is broken down by call-site, either during the single precession or double precession step.

Example: SHOC Stencil2D

Compute 2D, 9-point stencil

- Multiple GPUs using MPI
- CUDA and OpenCL versions

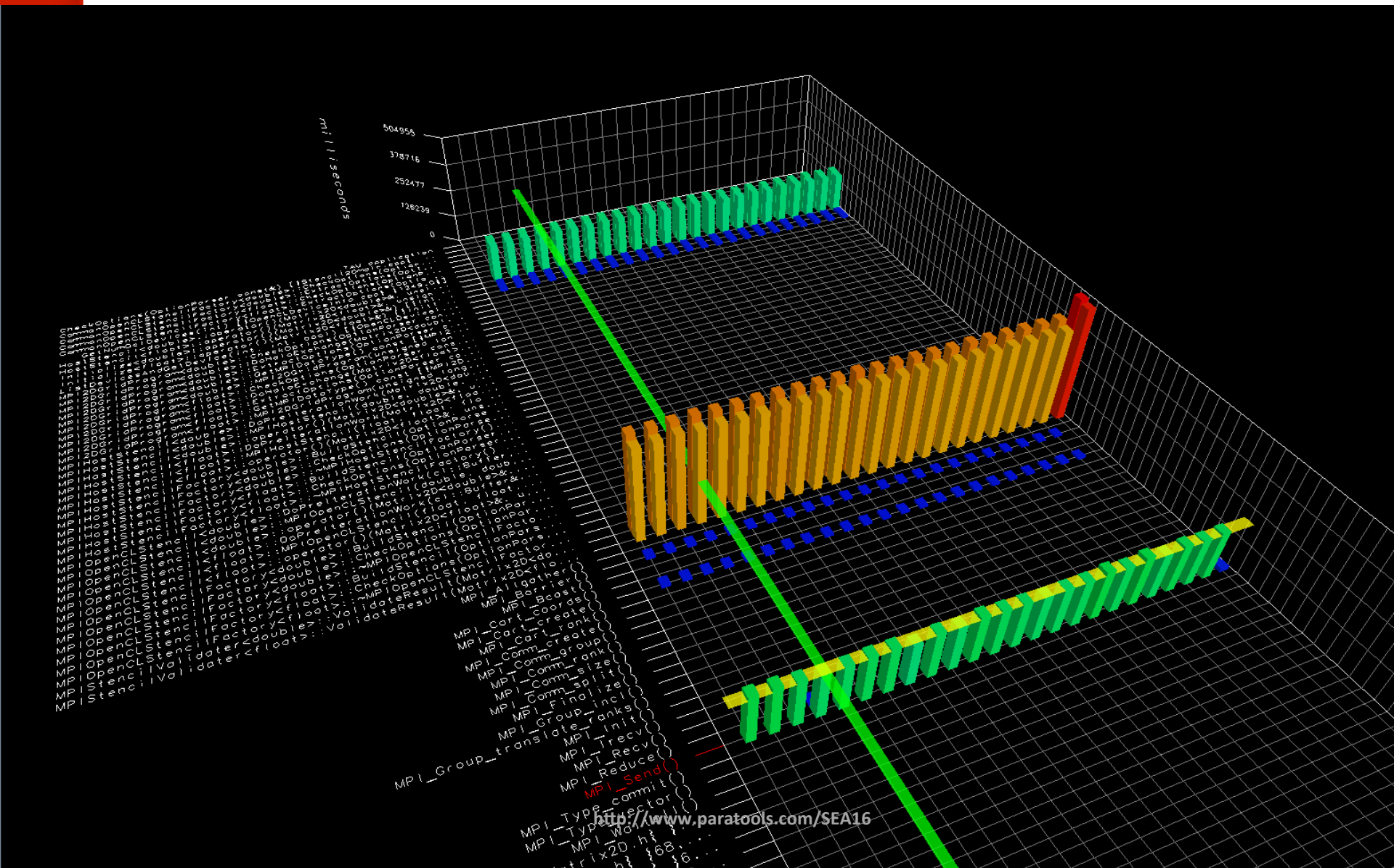
One Keeneland node with 3 GPUs

Eight Keeneland nodes with 24 GPUs

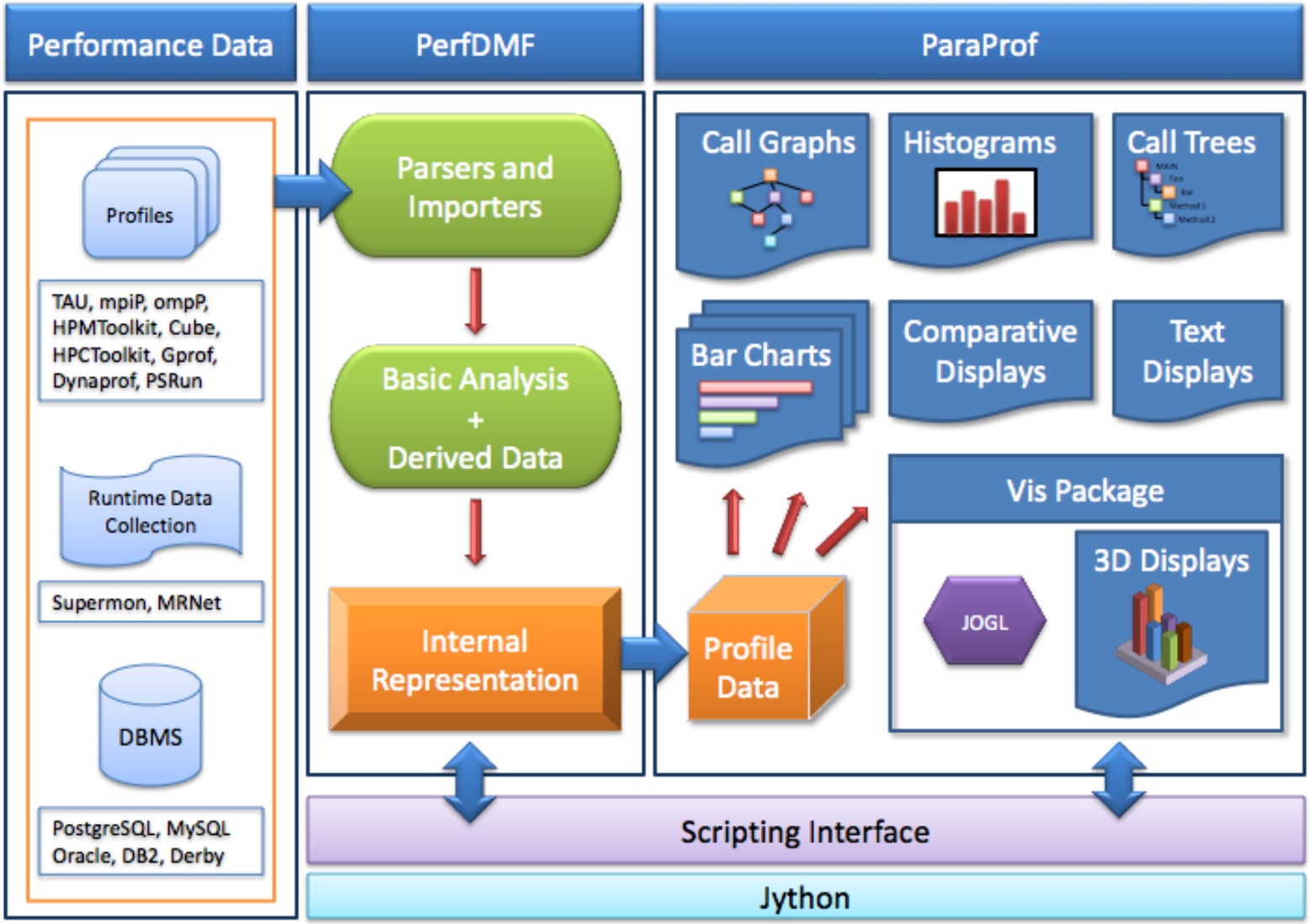
Performance profile and trace

- Application events
- Communication events
- Kernel execution

Stencil2D Parallel Profile

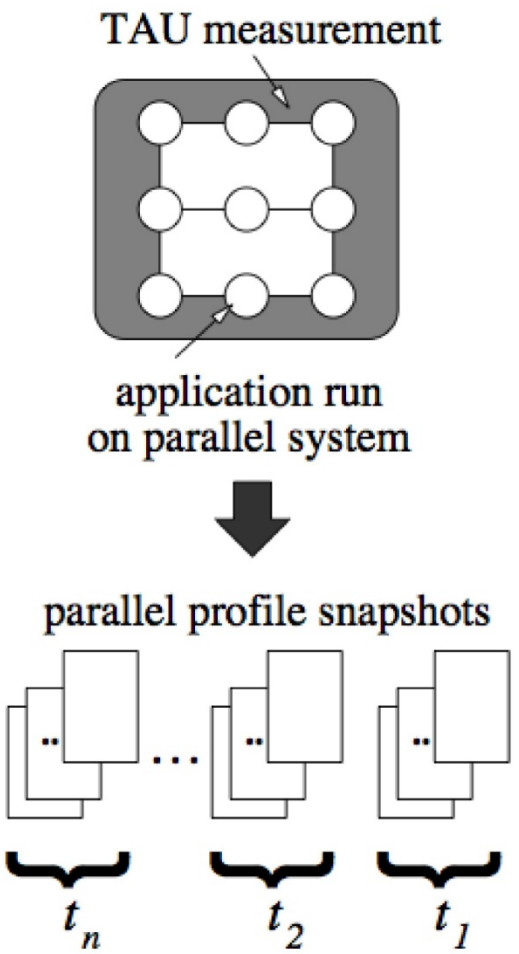


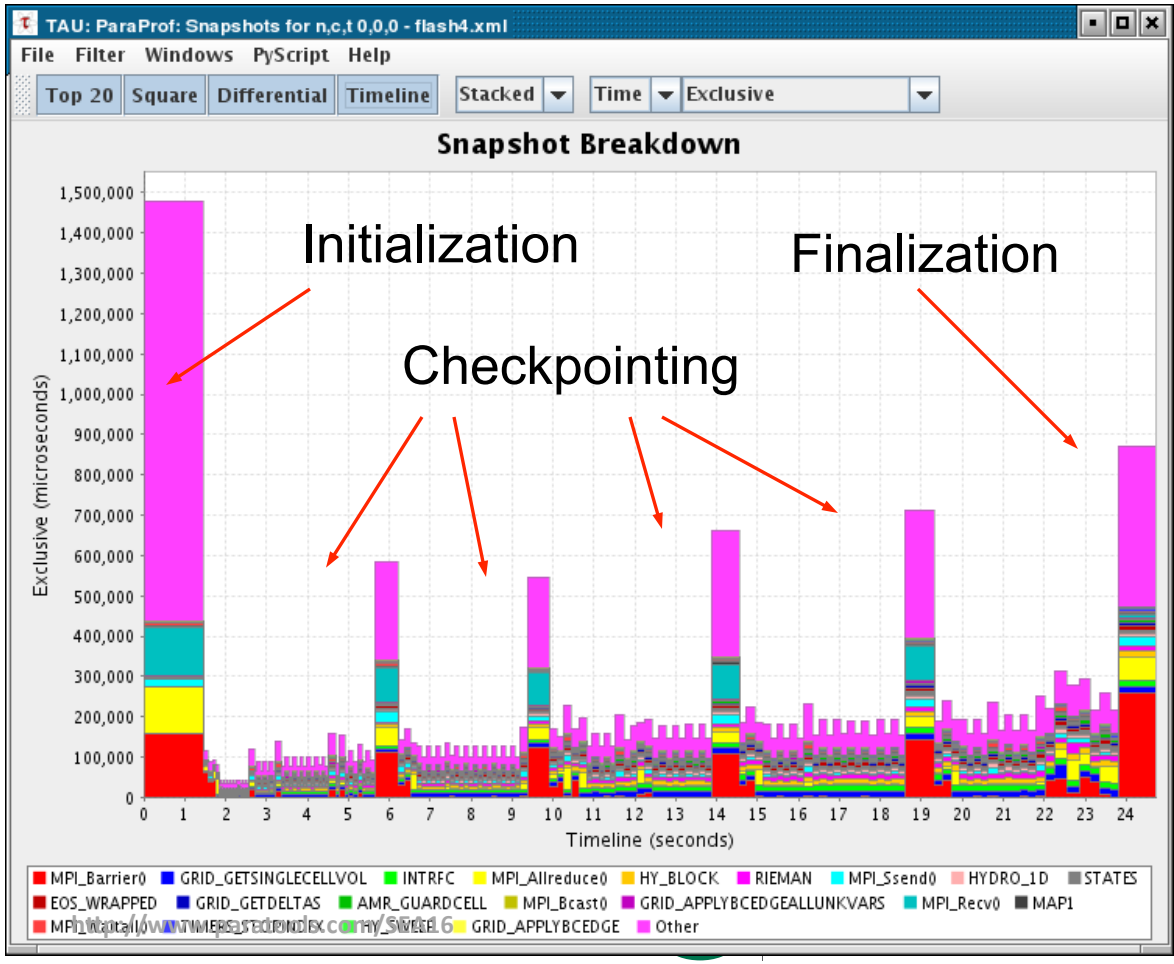
ParaProf Profile Analysis Framework



Profile Snapshots in ParaProf

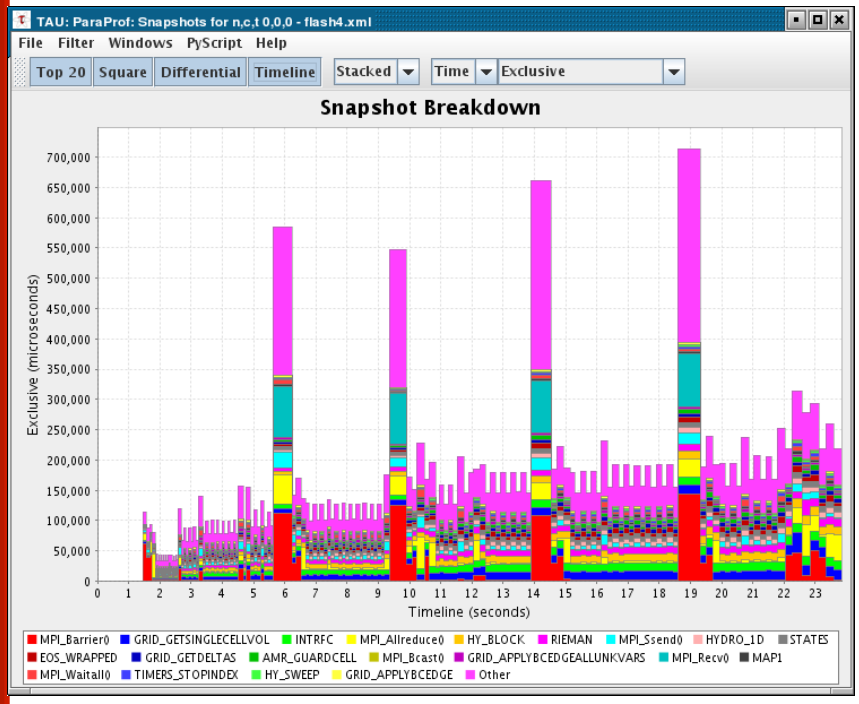
- Profile snapshots are profiles recorded at runtime
- Shows performance profile dynamics (all types allowed)



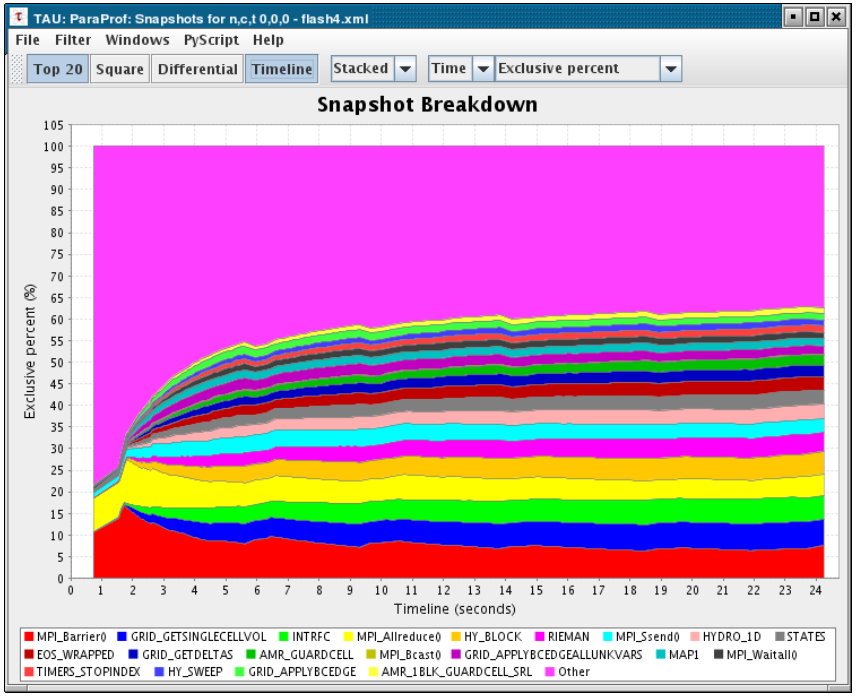


Profile Snapshot Views

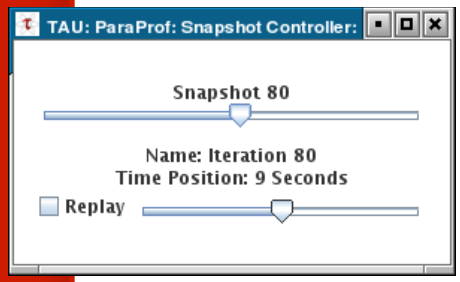
Percentage breakdown



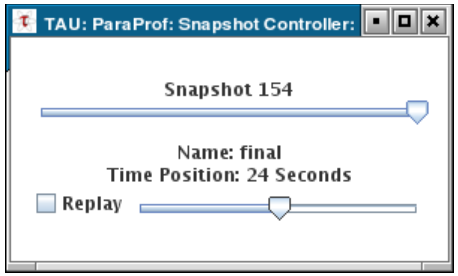
Only show main loop

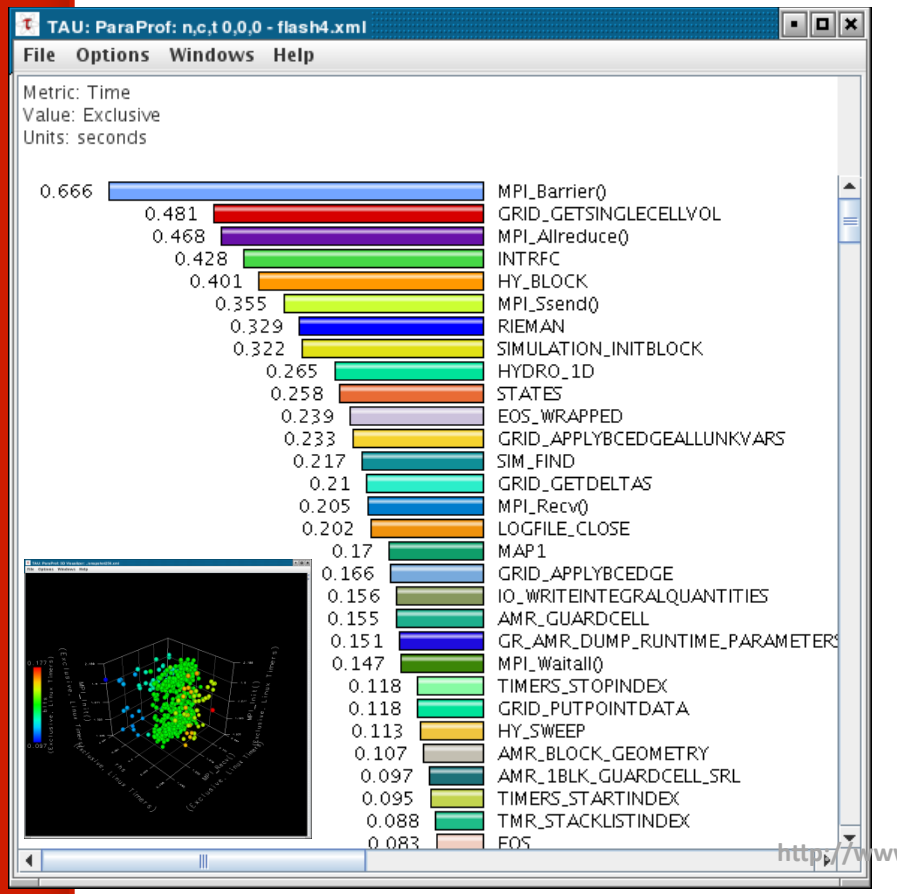


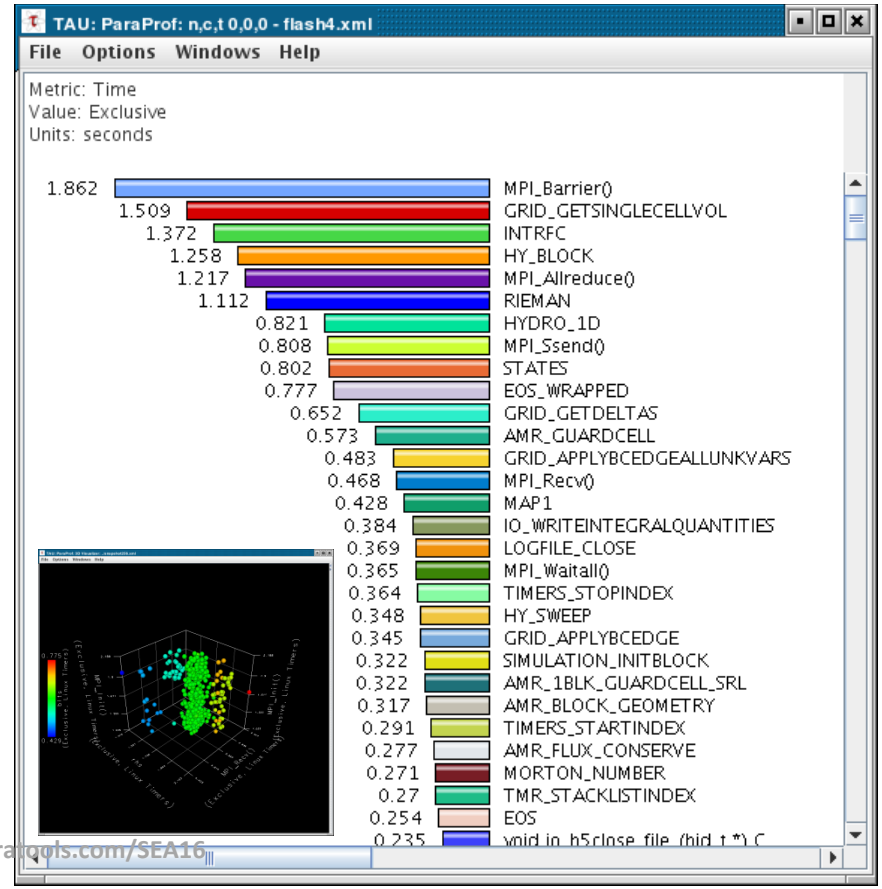
Snapshot Replay in ParaProf



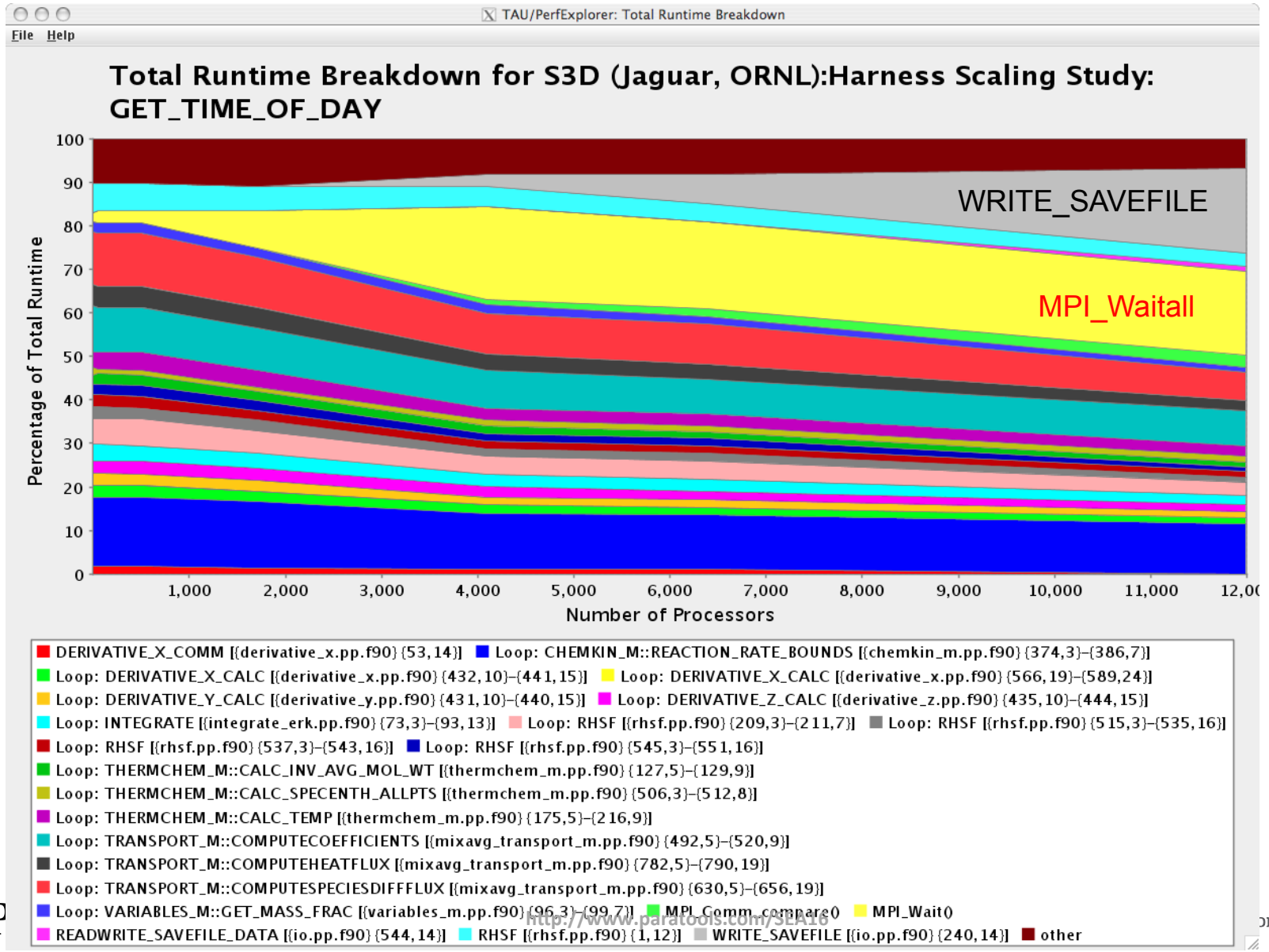
All windows dynamically update



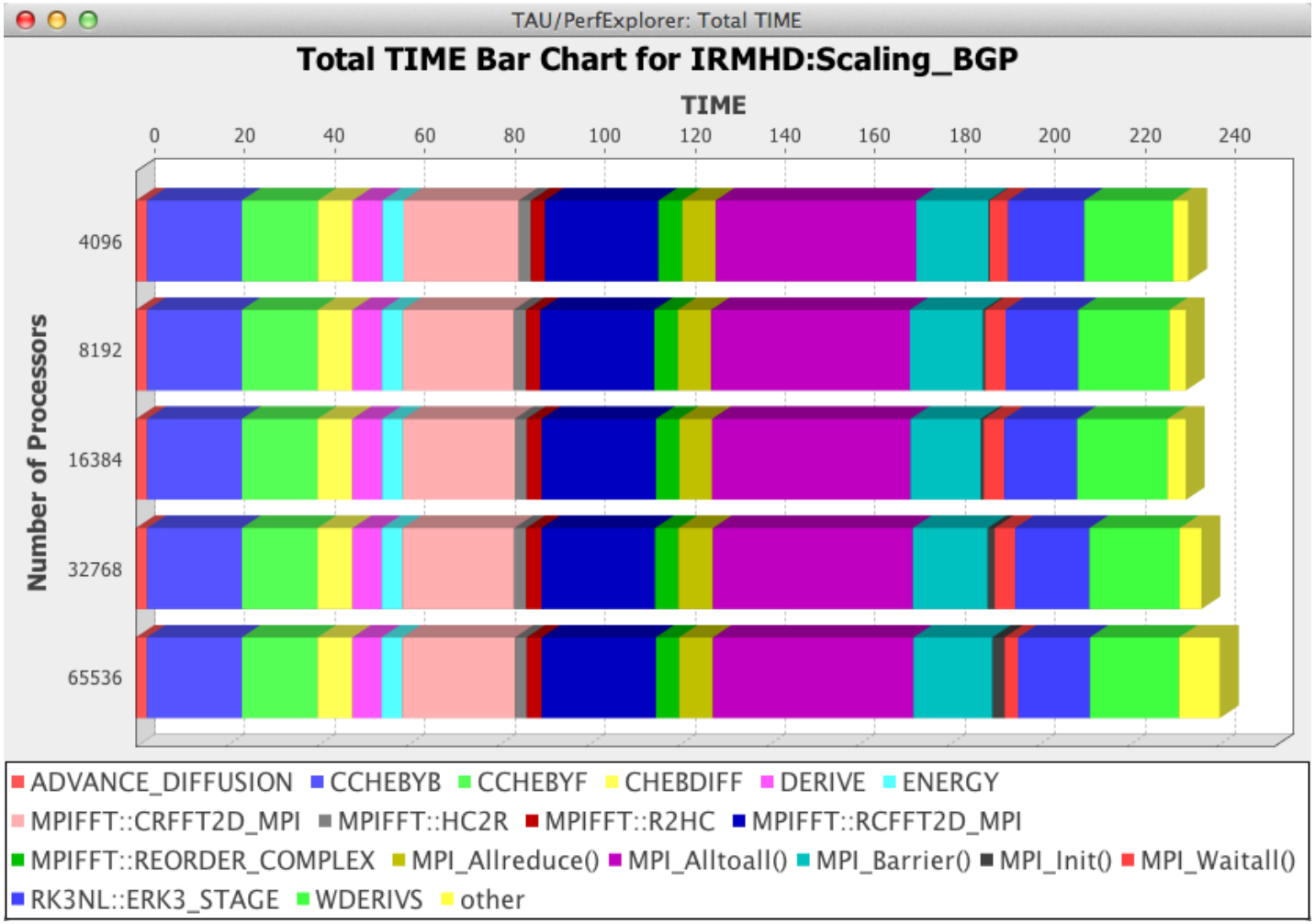




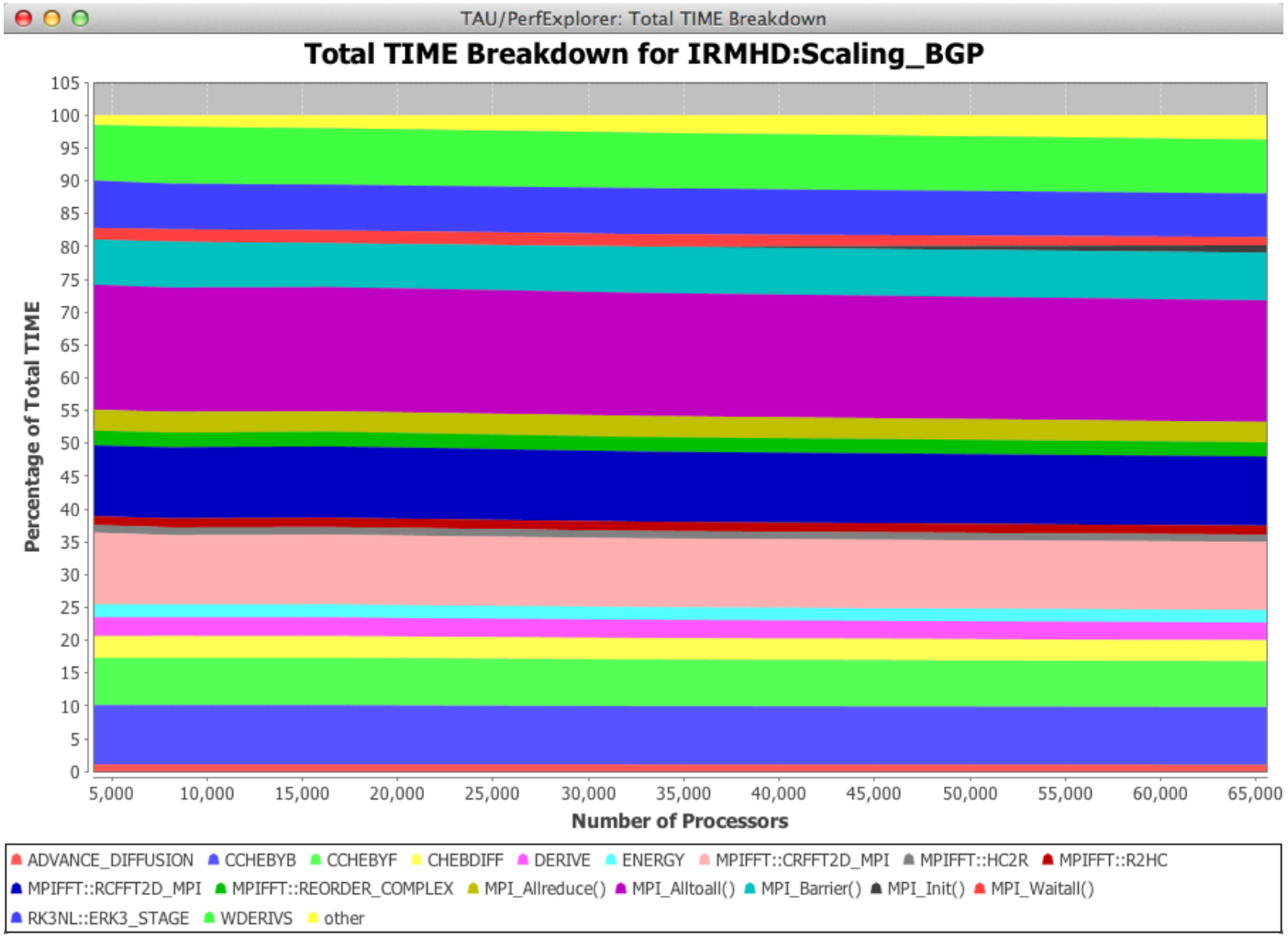
PerfExplorer – Runtime Breakdown



Evaluate Scalability

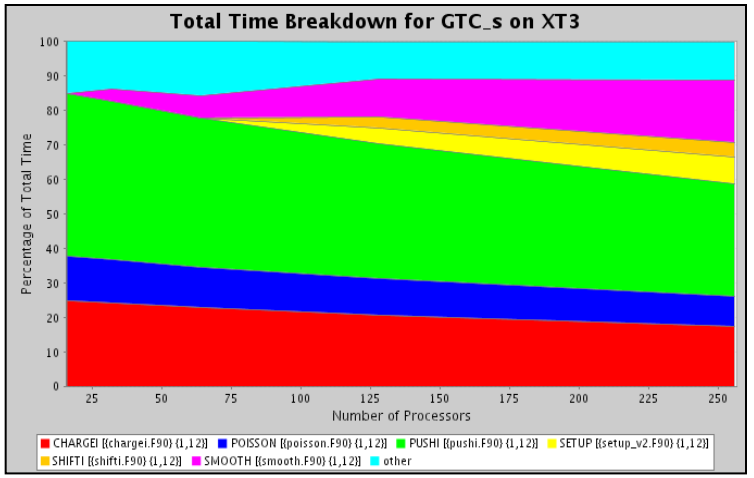


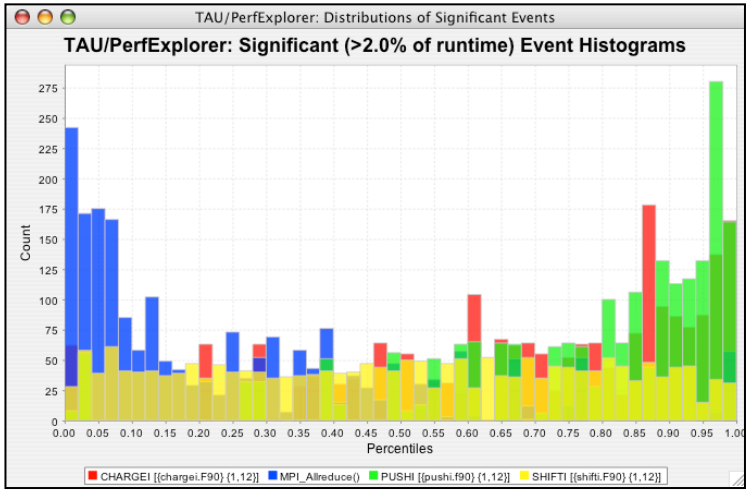
Runtime Breakdown



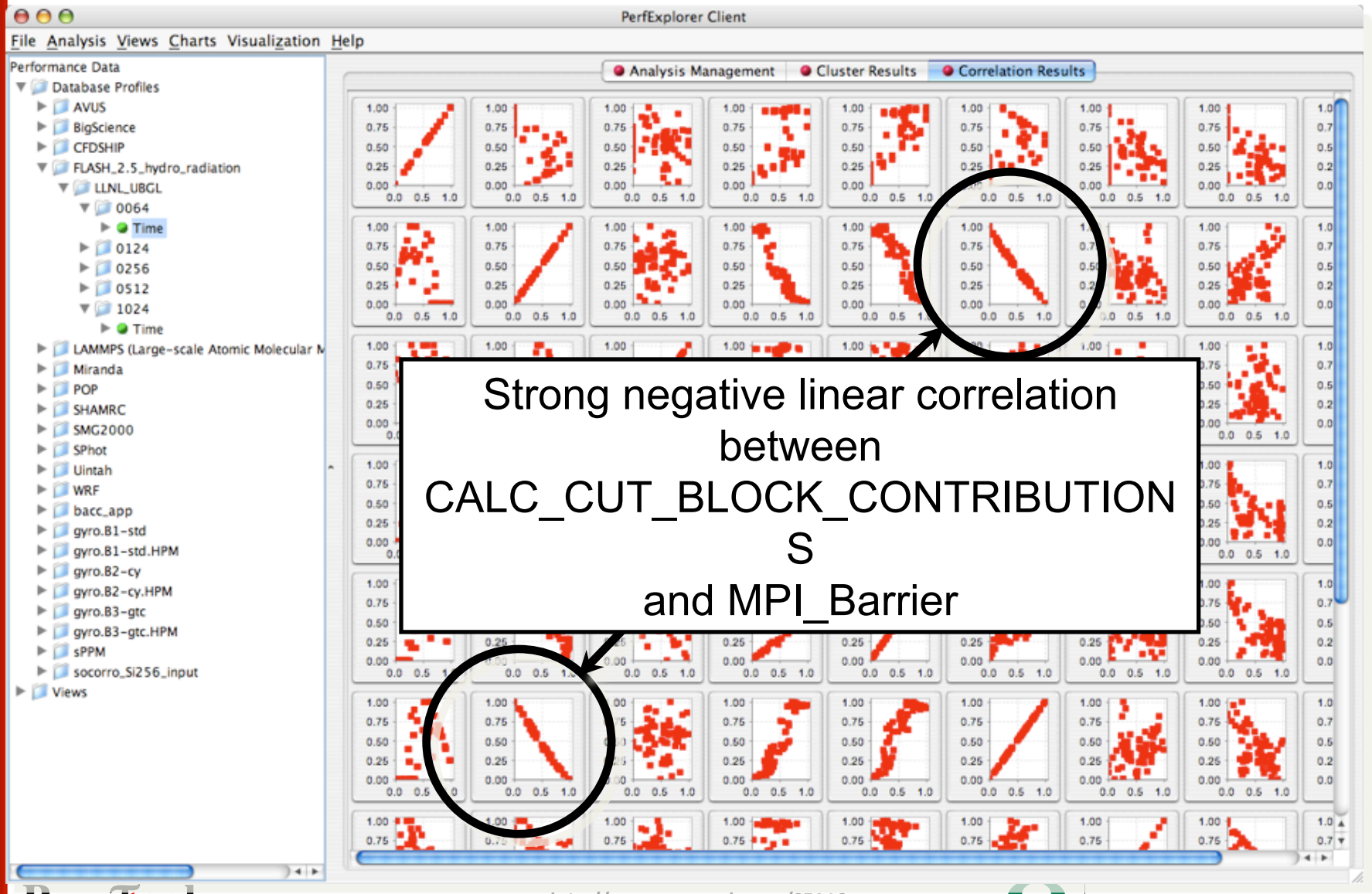
PerfExplorer – Relative Comparisons

- Total execution time
- Timesteps per second
- Relative efficiency
- Relative efficiency per event
- Relative speedup
- Relative speedup per event
- Group fraction of total
- Runtime breakdown
- Correlate events with total runtime
- Relative efficiency per phase
- Relative speedup per phase
- Distribution visualizations



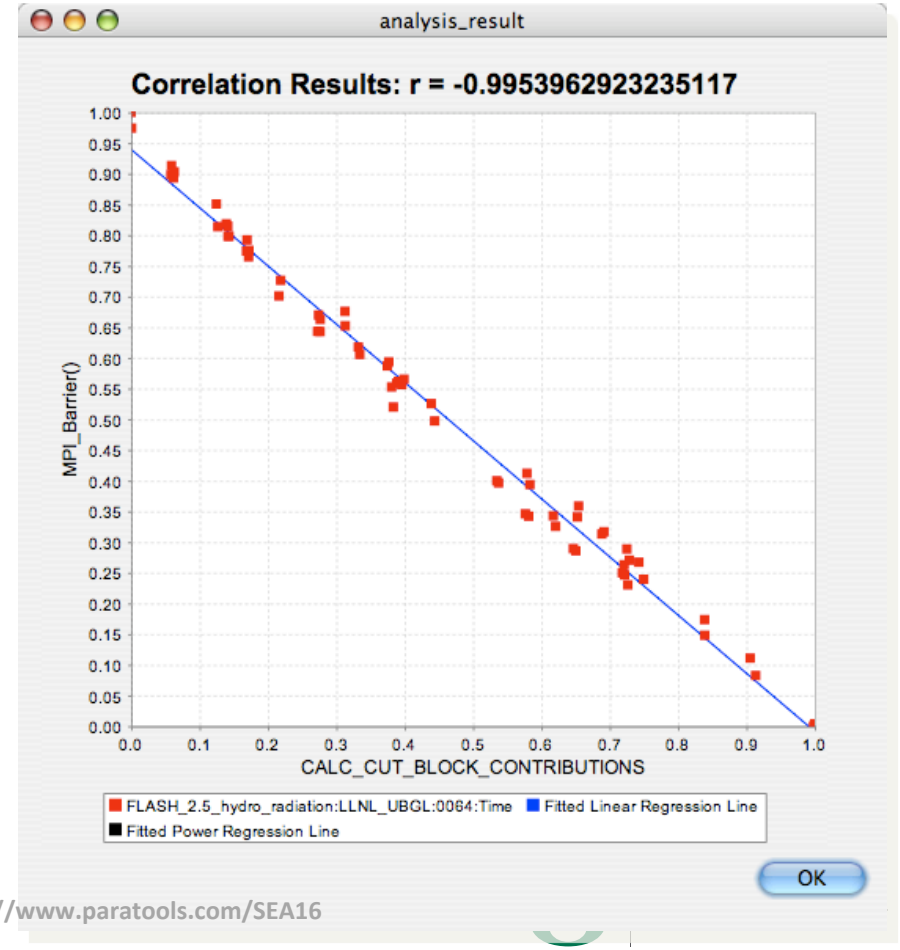


PerfExplorer – Correlation Analysis

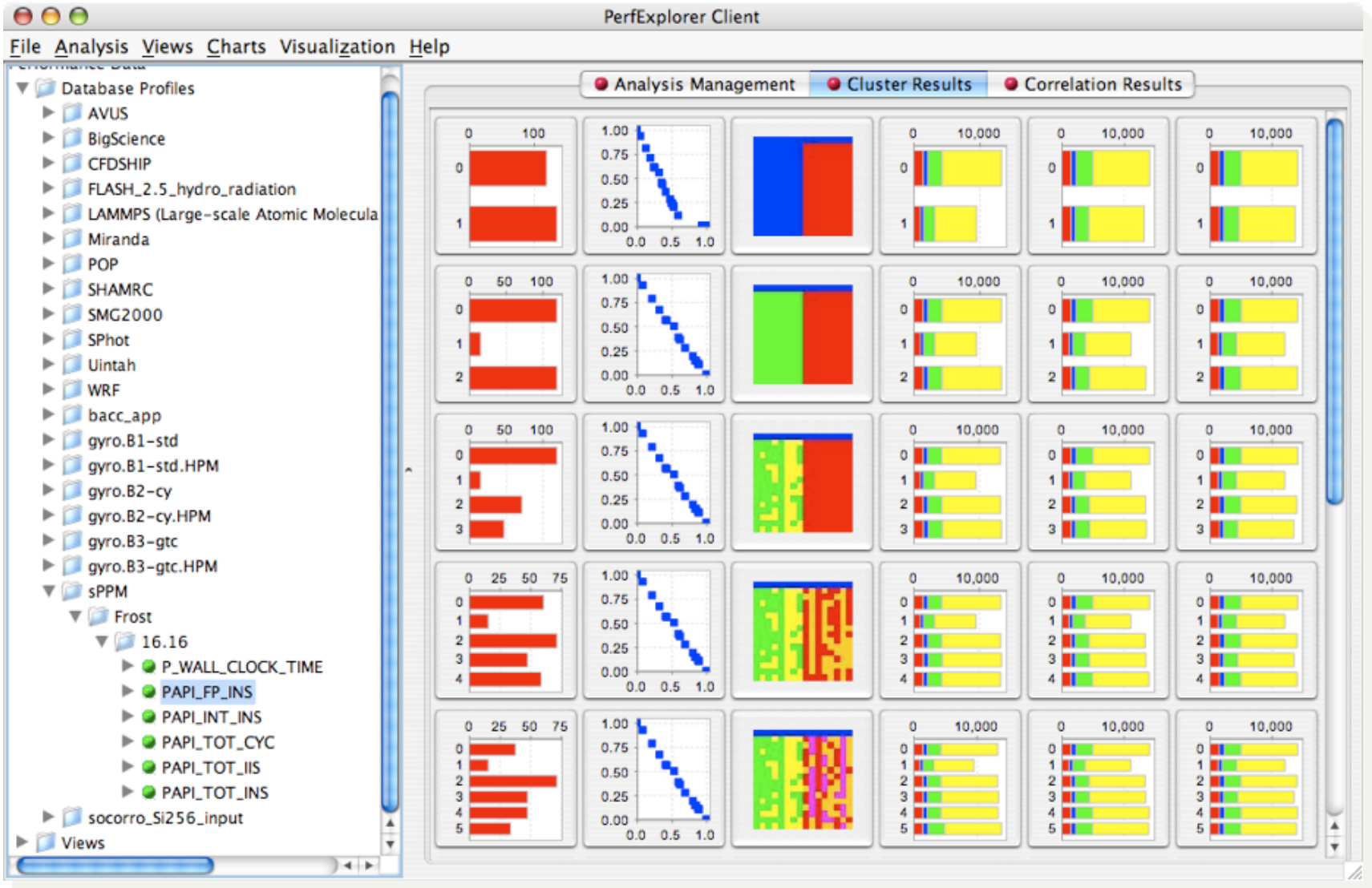


PerfExplorer – Correlation Analysis

-0.995 indicates strong, negative relationship. As CALC_CUT_BLOCK_CONTRIBUTIONS() increases in execution time, MPI_Barrier() decreases



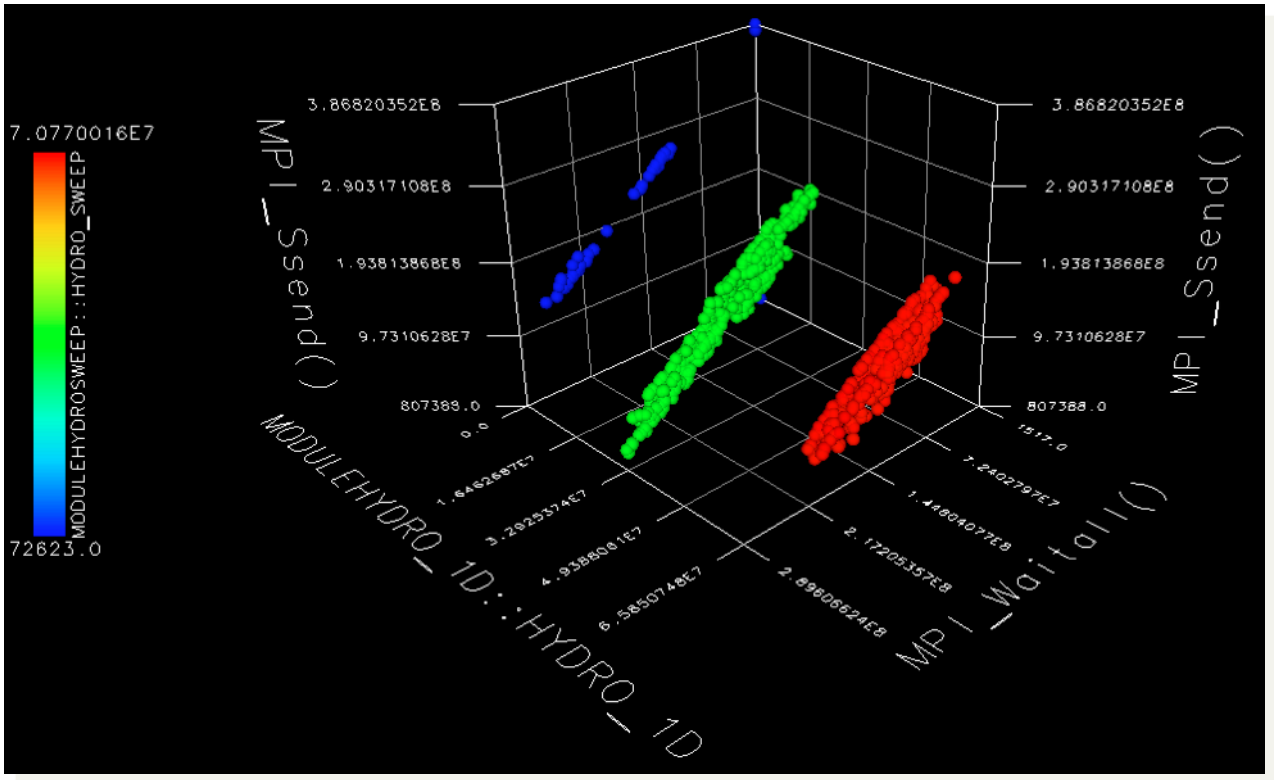
PerfExplorer – Cluster Analysis



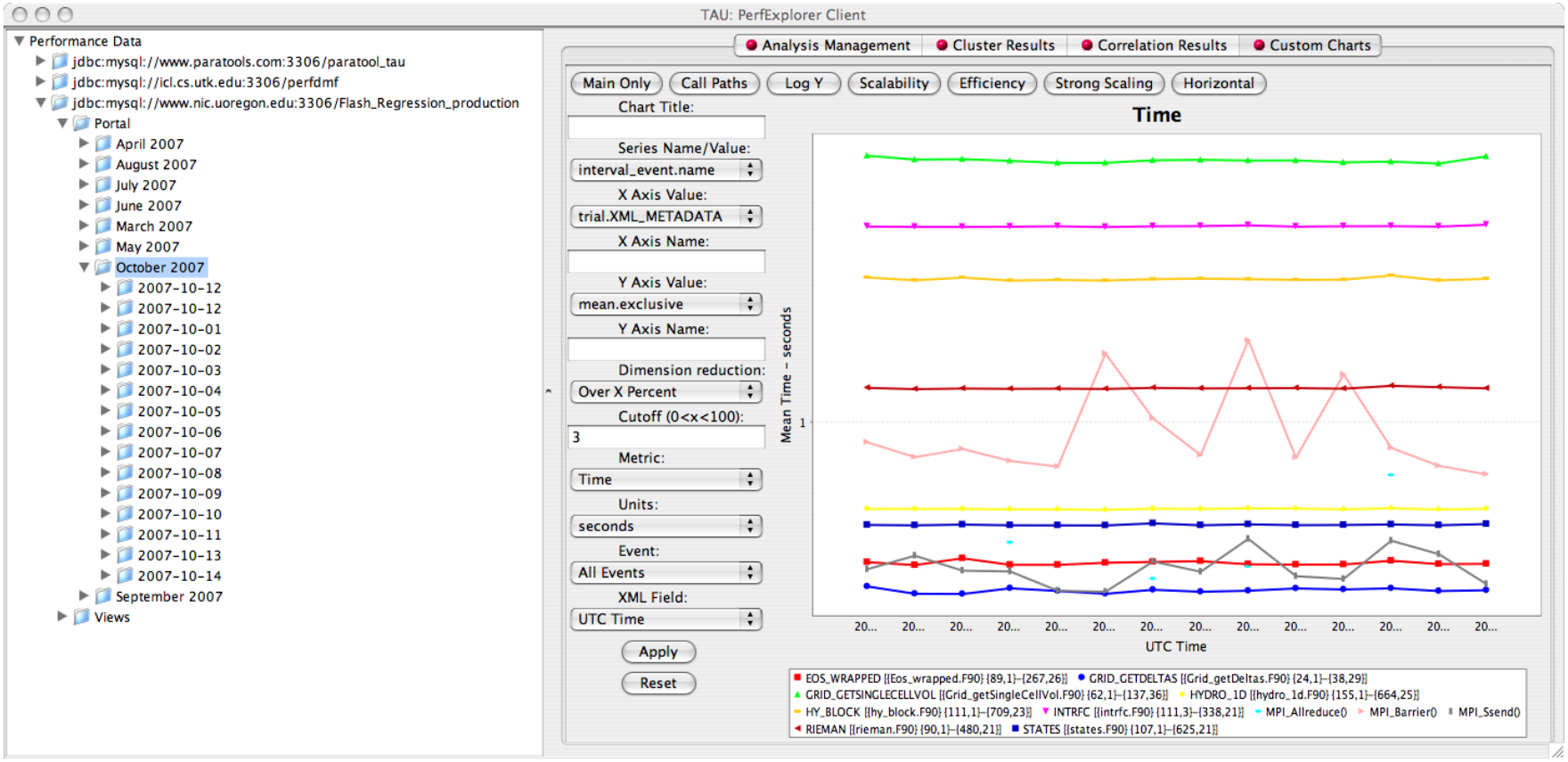
PerfExplorer – Cluster Analysis

Four significant events automatically selected

Clusters and correlations are visible



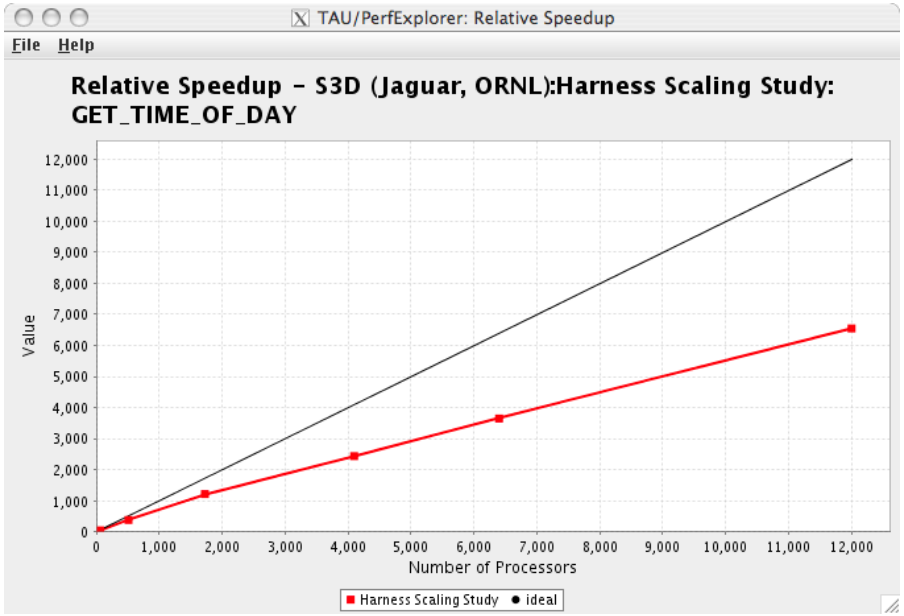
PerfExplorer – Performance Regression

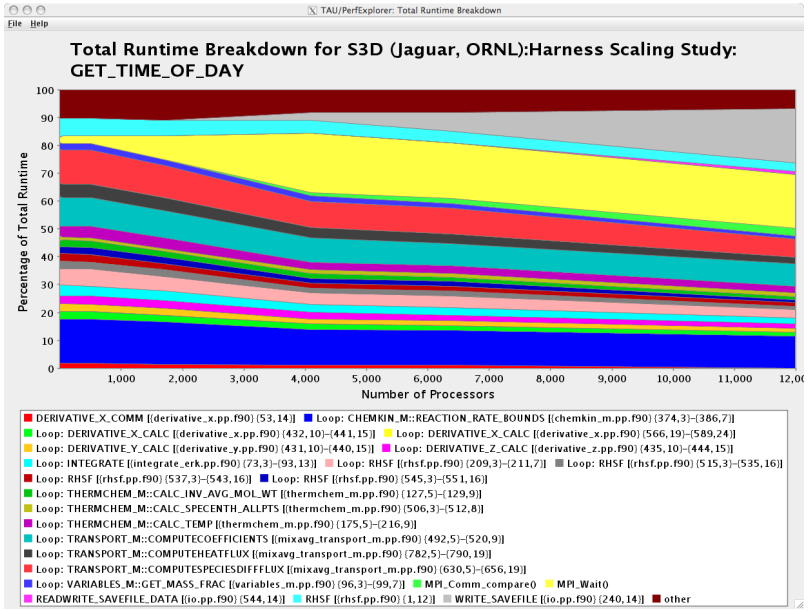


Evaluate Scalability

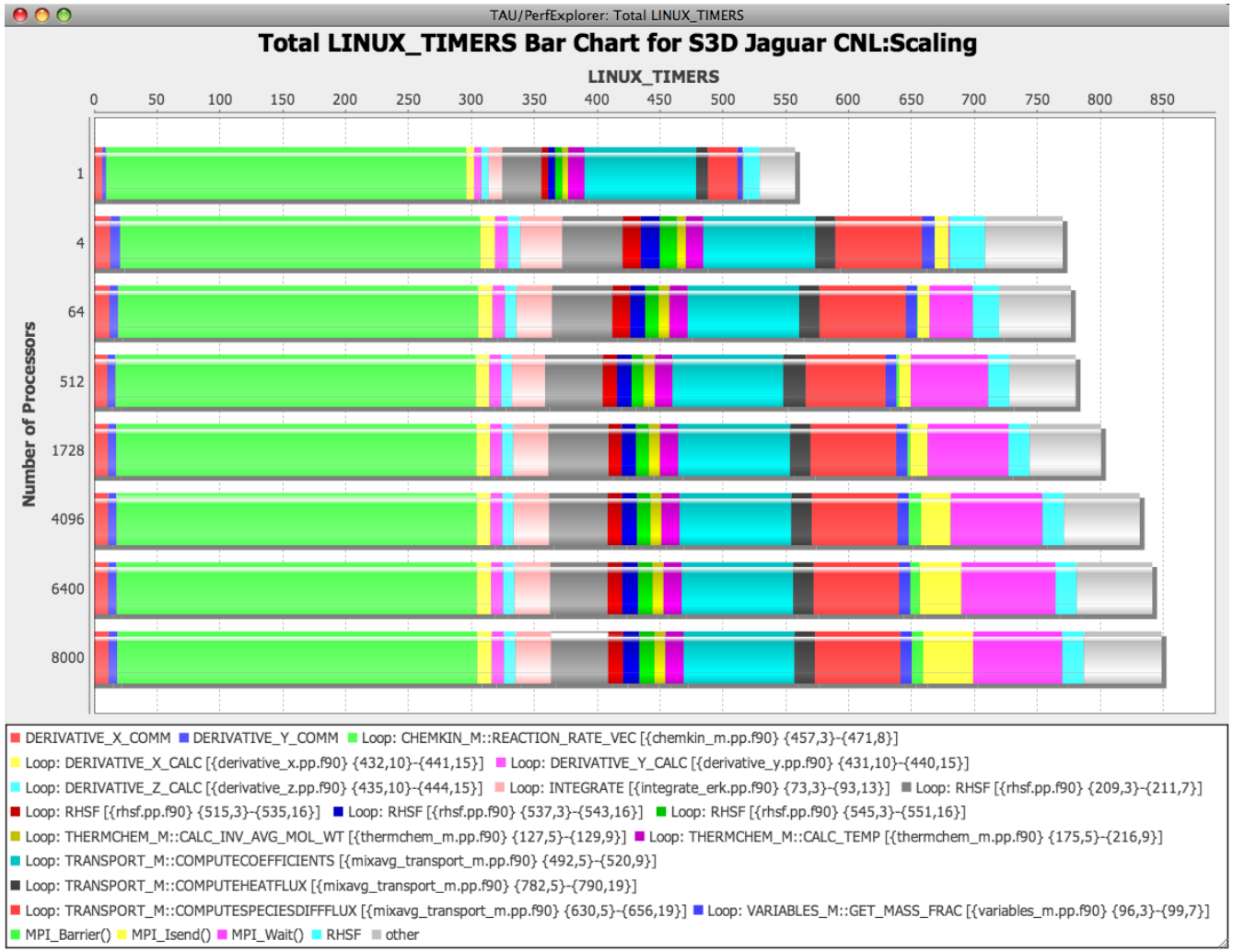
Goal: How does my application scale? What bottlenecks at what CPU counts?

Load profiles in PerfDMF database and examine with PerfExplorer

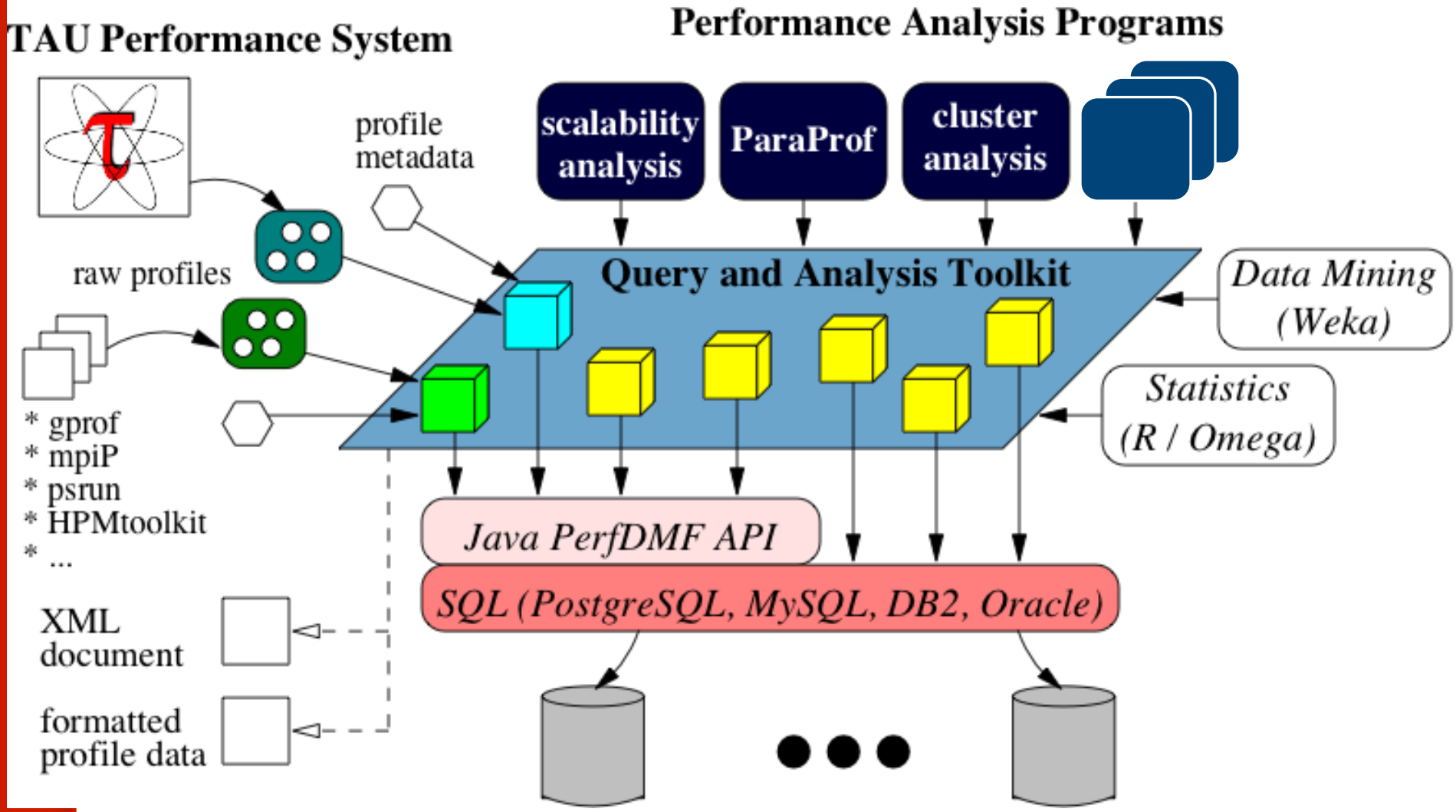




Usage Scenarios: Evaluate Scalability



TAUdb: Framework for Managing Performance Data



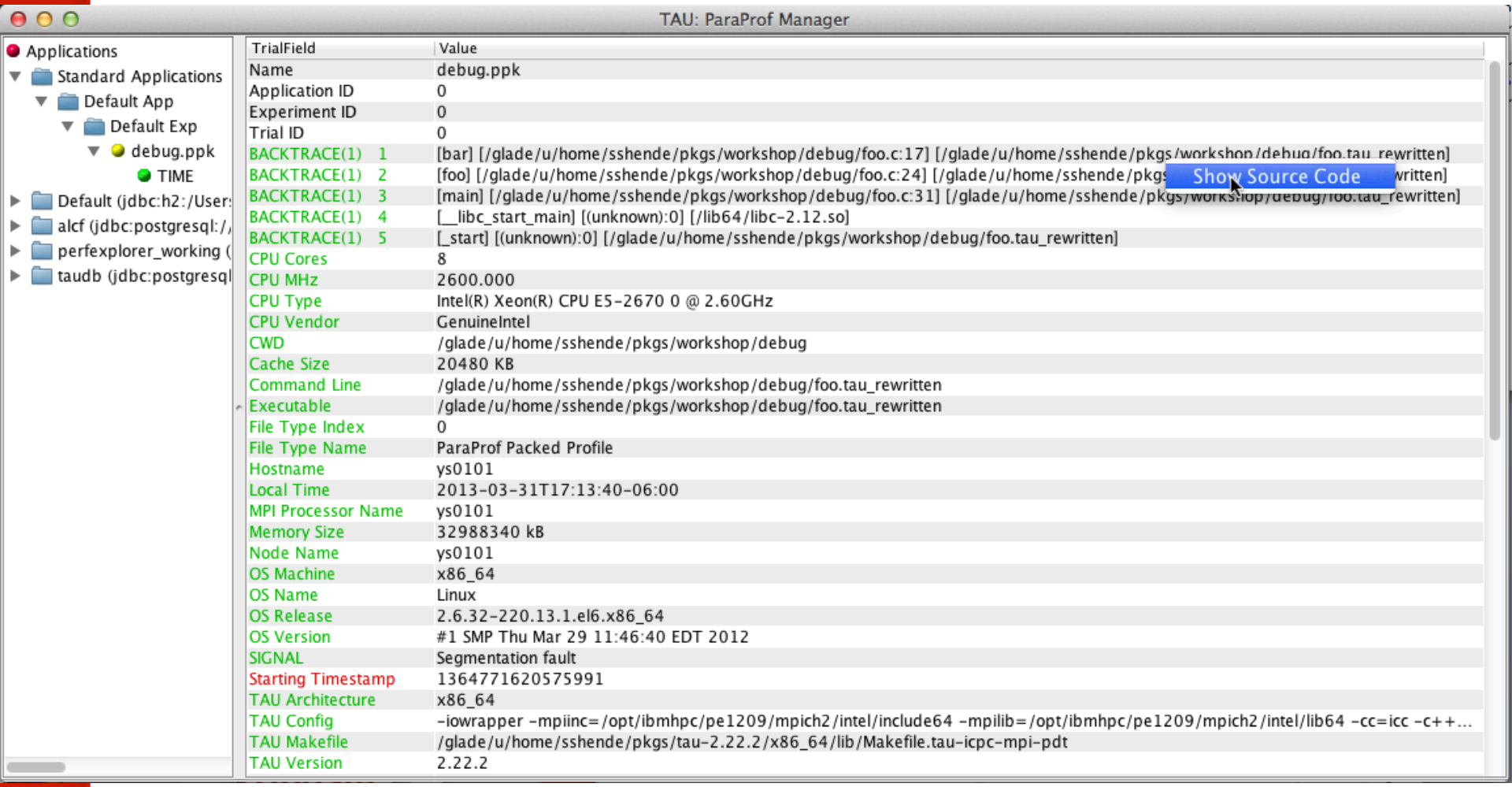
Evaluate Scalability using PerfExplorer Charts

```
% export TAU_MAKEFILE=$TAU/Makefile.tau-icpc-papi-mpi-pdt
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)
% qsub  run1p.job
% paraprof --pack 1p.ppk
% qsub  run2p.job …
% paraprof --pack 2p.ppk … and so on.
On your client:
% taudb_configure --create-default
% perfexplorer_configure
(Enter, y to load schema, defaults)
% paraprof
(load each trial: DB -> Add Trial -> Type (Paraprof Packed
Profile) -> OK, OR use taudb_loadtrial on the commandline)
% taudb_loadtrial -a App -x MyExp -n 4p 4p.ppk
% perfexplorer
(Charts -> Speedup)
OR:
wget http://tau.uoregon.edu/data.tgz; cat README in data
```

Multi-language Application Debugging

```
% export TAU_MAKEFILE=$TAU/Makefile.tau-icpc-papi-mpi-pdt
% export TAU_OPTIONS='-optMemDbg -optVerbose'
% make F90=tau_f90.sh CC=tau_cc.sh CXX=tau_cxx.sh
% mxterm 1 16 40
% export TAU_MEMDBG_PROTECT_ABOVE=1
% export TAU_MEMDBG_PROTECT_BELOW=1
% export TAU_MEMDBG_PROTECT_FREE=1
% mpirun -np 4   ./matmult
% paraprof
```

Multi-language Application Debugging



Location of segmentation violation

TAU: ParaProf: Source Browser: /glade/u/home/sshe...

File Help

```
1  #include <stdio.h>
2  #include <mpi.h>
3  #include <unistd.h>
4  #include <stdlib.h>
5
6  struct node {
7    int id;
8    struct node *next;
9  };
10
11 int bar(int x) {
12   int y;
13   struct node *t = (struct node *)malloc(sizeof(struct node));
14   t->next = NULL;
15   t->id = x;
16   printf("t -> id = %d\n", t->id);
17   y = t->next->id;
18   printf("y=%d\n",y);
19   return x;
20 }
21
22 int foo(int x) {
23   printf("foo: x = %d\n", x);
24   bar(x);
25   return x;
26 }
27
28 int main(int argc, char **argv) {
29   int ret;
30   MPI_Init(&argc, &argv);
31   ret = foo(29);
32   MPI_Finalize();
33   return ret;
34 }
```

Memory Leak Detection

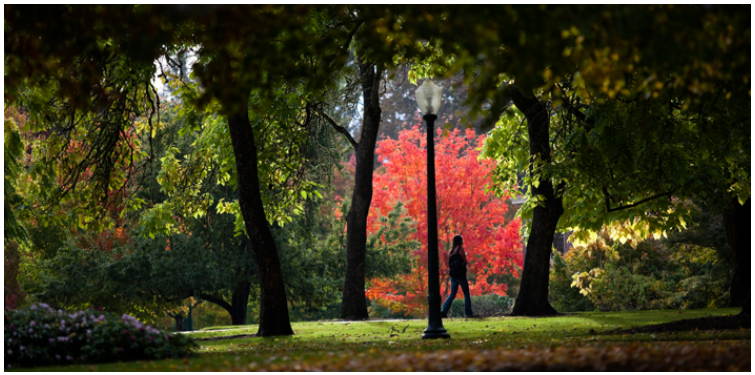
```
% export TAU_MAKEFILE=$TAU/Makefile.tau-icpc-papi-mpi-pdt
% export TAU_OPTIONS='-optMemDbg -optVerbose'
% make F90=tau_f90.sh CC=tau_cc.sh CXX=tau_cxx.sh
% mxterm 1 16 40

% export TAU_TRACK_MEMORY_LEAKS=1
% mpirun -np 4   ./matmult
% paraprof
```

Multi-language Memory Leak Detection

TAU: ParaProf: Context Events for: node 0 – memleak.ppk

Name	Total	NumSamples	MaxValue	MinValue	MeanValue	Std. Dev.
Heap Allocate	5,000,033	2	5,000,001	32	2,500,016.5	2,499,984.5
Heap Allocate <file=simple.c, line=15>	180	3	80	48	60	14.236
Heap Allocate <file=simple.c, line=23>	180	1	180	180	180	0
Heap Free <file=simple.c, line=18>	80	1	80	80	80	0
Heap Free <file=simple.c, line=25>	180	1	180	180	180	0
Heap Memory Used (KB)	4,884.829	8	4,883.196	0.047	610.604	1,614.888
▼ int foo(int) C [{simple.c} {36,1}-{44,1}]						
▼ int bar(int) C [{simple.c} {7,1}-{28,1}]						
Heap Allocate <file=simple.c, line=23>	180	1	180	180	180	0
Heap Free <file=simple.c, line=25>	180	1	180	180	180	0
▼ int g(int) C [{simple.c} {30,1}-{34,1}]						
▼ int bar(int) C [{simple.c} {7,1}-{28,1}]						
Heap Allocate <file=simple.c, line=15>	180	3	80	48	60	14.236
Heap Free <file=simple.c, line=18>	80	1	80	80	80	0
MEMORY LEAK! Heap Allocate <file=simple.c, line=15>	100	2	52	48	50	2
▼ int main(int, char **) C [{simple.c} {45,1}-{55,1}]						
▼ MPI_Finalize()						
Heap Allocate	5,000,033	2	5,000,001	32	2,500,016.5	2,499,984.5
MEMORY LEAK! Heap Allocate	5,000,033	2	5,000,001	32	2,500,016.5	2,499,984.5

PRL, University of Oregon, Eugene



www.uoregon.edu

Support Acknowledgments

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

National Science Foundation (NSF)

- Glassbox, SI-2

NASA

Partners:

University of Oregon

ParaTools, Inc.

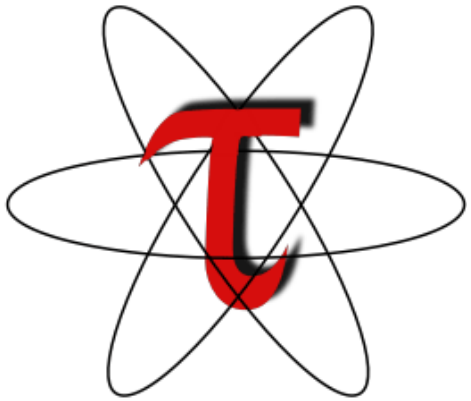
University of Tennessee, Knoxville

T.U. Dresden, GWT

Juelich Supercomputing Center




Download TAU from U. Oregon



http://www.hpclinux.com [LiveDVD]

Free download, open source, BSD license

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

Compile-Time Options

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

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

Compile-Time Options (contd.)

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

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

See tau_compiler.sh for a full list of TAU_OPTIONS.

…

Compiling Fortran Codes with TAU

If your Fortran code uses free format in .f files (fixed is default for .f), you may use:

% export TAU_OPTIONS= '-optPdtF95Opts="-R free" -optVerbose '

To use the compiler based instrumentation instead of PDT (source-based):

% export TAU_OPTIONS= '-optCompInst -optVerbose'

If your Fortran code uses C preprocessor directives (#include, #ifdef, #endif):

% export TAU_OPTIONS= '-optPreProcess -optVerbose'

To use an instrumentation specification file:

% export TAU_OPTIONS= '-optTauSelectFile=select.tau -optVerbose -optPreProcess'

% cat select.tau

```
BEGIN_EXCLUDE_LIST
FOO
END_EXCLUDE_LIST

BEGIN_INSTRUMENT_SECTION
loops  routine="#"
# this statement instruments all outer loops in all routines. # is wildcard as well as comment in first column.
END_INSTRUMENT_SECTION
```

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