TAU

High Performance Software Tools to Fast-Track Development of Scalable and Sustainable Applications

The 11th DOE ACTS Workshop
Berkeley, CA – Aug 20, 2010

Sameer Shende
Performance Research Lab
University of Oregon
http://tau.uoregon.edu

ACTS COLLECTION

Office of Science
U.S. DEPARTMENT OF ENERGY

UNIVERSITY OF OREGON

Acknowledgments: PRL, UO

- Prof. Allen D. Malony, Comp. & Info Sci., Dir. NIC
- Wyatt Spear
- Dr. Chee Wai Lee
- Scott Biersdorff
- Alan Morris
- Suzanne Millstein
- Dr. Rob Yelle
- Carl Woeck
- William Voorhess

TAU Performance System®

- *Tuning* and *Analysis* *Utilities* (15+ year project)
- Performance problem solving framework for HPC
  - Integrated, scalable, flexible, portable
  - Target all parallel programming / execution paradigms
- Integrated performance toolkit (open source)
  - Instrumentation, measurement, analysis, visualization
  - Widely-ported performance profiling / tracing system
  - Performance data management and data mining
- Broad application use (NSF, DOE, DOD, …)

TAU Performance System

- http://tau.uoregon.edu/
- Multi-level performance instrumentation
  - Multi-language automatic source instrumentation
- Flexible and configurable performance measurement
- Widely-ported parallel performance profiling system
  - Computer system architectures and operating systems
  - Different programming languages and compilers
- Support for multiple parallel programming paradigms
  - Multi-threading, message passing, mixed-mode, hybrid
- Integration in complex software, systems, applications

Tuning and Analysis Utilities

What is TAU?

- TAU is a performance evaluation tool
- It supports parallel profiling and tracing
- Profiling shows you how much (total) time was spent in each routine
- Tracing shows you *when* the events take place in each process along a timeline
- TAU uses a package called PDT for automatic instrumentation of the source code
- Profiling and tracing can measure time as well as hardware performance counters from your CPU
- TAU can automatically instrument your source code (routines, loops, I/O, memory, phases, etc.)
- TAU runs on all HPC platforms and it is free (BSD style license)
- TAU has instrumentation, measurement and analysis tools
  - paraprof is TAU's 3D profile browser
- To use TAU's automatic source instrumentation, you need to set a couple of environment variables and substitute the name of your compiler with a TAU shell script

Using TAU: A brief Introduction

- TAU supports several measurement options (profiling, tracing, profiling with hardware counters, etc.)
- Each measurement configuration of TAU corresponds to a unique stub makefile and library that is generated when you configure it
- To instrument source code using PDT
  - Choose an appropriate TAU stub makefile in <arch>/lib:

  % setenv TAU_MAKEFILE $TAU/Makefile.tau-mpi-pdt

  % setenv TAU_OPTIONS '-optVerbose …' (see tau_compiler.sh -help)

  And use tau_f90.sh, tau_cxx.sh or tau_cc.sh as Fortran, C++ or C compilers:

  % mpif90 foo.f90

  changes to

  % tau_f90.sh foo.f90

- Execute application and analyze performance data:

  % pprof (for text based profile display)

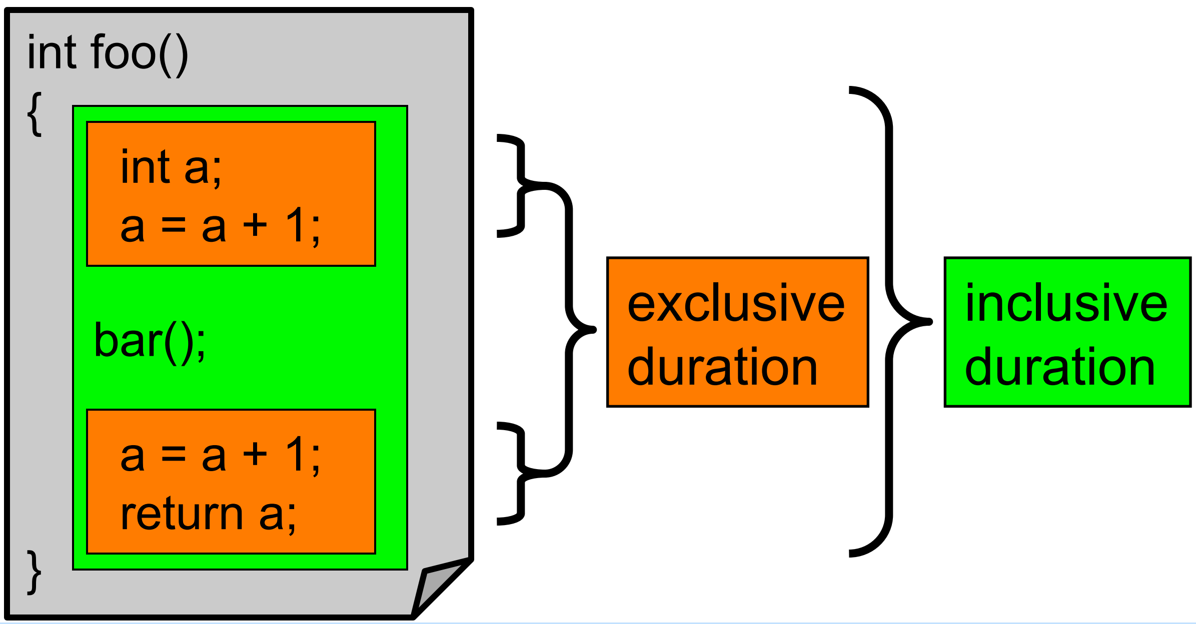
  % paraprof (for GUI)

Direct Observation: Events

- Event types
  - Interval events (begin/end events)
    - measures performance between begin and end
    - metrics monotonically increase
  - Atomic events
    - used to capture performance data state
- Code events
  - Routines, classes, templates
  - Statement-level blocks, loops
- User-defined events
  - Specified by the user
- Abstract mapping events

Inclusive and Exclusive Profiles

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



Interval Events, Atomic Events in TAU

```
xterm
NODE 0;CONTEXT 0;THREAD 0:
---------------------------------------------------------------------------------------
%Time    Exclusive    Inclusive       #Call      #Subrs  Inclusive Name
              msec   total msec                          usec/call
---------------------------------------------------------------------------------------
100.0        0.187        1,105           1          44    1105659 int main(int, char **) C
 93.2        1,030        1,030           1           0    1030654 MPI_Init()
  5.9        0.879           65          40         320       1637 void func(int, int) C
  4.6           51           51          40           0       1277 MPI_Barrier()
  1.2           13           13         120           0        111 MPI_Recv()
  0.8            9            9           1           0       9328 MPI_Finalize()
  0.0        0.137        0.137         120           0          1 MPI_Send()
  0.0        0.086        0.086          40           0          2 MPI_Bcast()
  0.0        0.002        0.002           1           0          2 MPI_Comm_size()
  0.0        0.001        0.001           1           0          1 MPI_Comm_rank()
---------------------------------------------------------------------------------------

USER EVENTS Profile :NODE 0, CONTEXT 0, THREAD 0
---------------------------------------------------------------------------------------
NumSamples   MaxValue   MinValue  MeanValue  Std. Dev.  Event Name
---------------------------------------------------------------------------------------
       365  5.138E+04      44.39   3.09E+04  1.234E+04  Heap Memory Used (KB) : Entry
       365  5.138E+04       2064  3.115E+04   1.21E+04  Heap Memory Used (KB) : Exit
        40         40         40         40          0  Message size for broadcast
---------------------------------------------------------------------------------------
                                                                  27,1          1%
```

Interval event e.g., routines (start/stop)

Atomic events (trigger with value)

% setenv TAU_CALLPATH_DEPTH 0

% setenv TAU_TRACK_HEAP 1

Atomic Events, Context Events

```
---------------------------------------------------------------------------------------
%Time    Exclusive    Inclusive       #Call      #Subrs  Inclusive Name
              msec   total msec                          usec/call
---------------------------------------------------------------------------------------
100.0        0.253        1,106           1          44    1106701 int main(int, char **) C
 93.2        1,031        1,031           1           0    1031311 MPI_Init()
  6.0            1           66          40         320       1650 void func(int, int) C
  5.7           63           63          40           0       1588 MPI_Barrier()
  0.8            9            9           1           0       9119 MPI_Finalize()
  0.1            1            1         120           0         10 MPI_Recv()
  0.0        0.141        0.141         120           0          1 MPI_Send()
  0.0        0.085        0.085          40           0          2 MPI_Bcast()
  0.0        0.001        0.001           1           0          1 MPI_Comm_size()
  0.0            0            0           1           0          0 MPI_Comm_rank()
---------------------------------------------------------------------------------------

USER EVENTS Profile :NODE 0, CONTEXT 0, THREAD 0
---------------------------------------------------------------------------------------
NumSamples   MaxValue   MinValue  MeanValue  Std. Dev.  Event Name
---------------------------------------------------------------------------------------
        40         40         40         40          0  Message size for broadcast
       365  5.139E+04      44.39  3.091E+04  1.234E+04  Heap Memory Used (KB) : Entry
        40  5.139E+04       3097  3.114E+04  1.227E+04  Heap Memory Used (KB) : Entry : MPI_Barrier()
        40  5.139E+04   1.13E+04  3.134E+04  1.187E+04  Heap Memory Used (KB) : Entry : MPI_Bcast()
         1       2067       2067       2067          0  Heap Memory Used (KB) : Entry : MPI_Comm_rank()
         1       2066       2066       2066          0  Heap Memory Used (KB) : Entry : MPI_Comm_size()
         1  5.139E+04  5.139E+04  5.139E+04  0.0006905  Heap Memory Used (KB) : Entry : MPI_Finalize()
         1      57.56      57.56      57.56          0  Heap Memory Used (KB) : Entry : MPI_Init()
       120  5.139E+04   1.13E+04  3.134E+04  1.187E+04  Heap Memory Used (KB) : Entry : MPI_Recv()
       120  5.139E+04  1.129E+04  3.134E+04  1.187E+04  Heap Memory Used (KB) : Entry : MPI_Send()
         1      44.39      44.39      44.39          0  Heap Memory Used (KB) : Entry : int main(int, char **) C
        40  5.036E+04       2068  3.011E+04  1.227E+04  Heap Memory Used (KB) : Entry : void func(int, int) C
                                                                                   4,9           1%
```

Atomic event

Context event = atomic event + executing context

% setenv TAU_CALLPATH_DEPTH 1

% setenv TAU_TRACK_HEAP 1

Context Events (default)

```
xterm
NODE 0;CONTEXT 0;THREAD 0:
---------------------------------------------------------------------------------------
%Time    Exclusive    Inclusive       #Call      #Subrs  Inclusive Name
              msec   total msec                          usec/call
---------------------------------------------------------------------------------------
100.0        0.357        1,114           1          44    1114040 int main(int, char **) C
 92.6        1,031        1,031           1           0    1031066 MPI_Init()
  6.7           72           74          40         320       1865 void func(int, int) C
  0.7            8            8           1           0       8002 MPI_Finalize()
  0.1            1            1         120           0         12 MPI_Recv()
  0.1        0.608        0.608          40           0         15 MPI_Barrier()
  0.0        0.136        0.136         120           0          1 MPI_Send()
  0.0        0.095        0.095          40           0          2 MPI_Bcast()
  0.0        0.001        0.001           1           0          1 MPI_Comm_size()
  0.0            0            0           1           0          0 MPI_Comm_rank()
---------------------------------------------------------------------------------------

USER EVENTS Profile :NODE 0, CONTEXT 0, THREAD 0
---------------------------------------------------------------------------------------
NumSamples   MaxValue   MinValue  MeanValue  Std. Dev.  Event Name
---------------------------------------------------------------------------------------
       365  5.139E+04      44.39  3.091E+04  1.234E+04  Heap Memory Used (KB) : Entry
         1      44.39      44.39      44.39          0  Heap Memory Used (KB) : Entry : int main(int, char **) C
         1       2068       2068       2068          0  Heap Memory Used (KB) : Entry : int main(int, char **) C => MPI_Comm_rank()
         1       2066       2066       2066          0  Heap Memory Used (KB) : Entry : int main(int, char **) C => MPI_Comm_size()
         1  5.139E+04  5.139E+04  5.139E+04          0  Heap Memory Used (KB) : Entry : int main(int, char **) C => MPI_Finalize()
         1      57.58      57.58      57.58          0  Heap Memory Used (KB) : Entry : int main(int, char **) C => MPI_Init()
        40  5.036E+04       2069  3.011E+04  1.228E+04  Heap Memory Used (KB) : Entry : int main(int, char **) C => void func(int, int) C
        40  5.139E+04       3098  3.114E+04  1.227E+04  Heap Memory Used (KB) : Entry : void func(int, int) C => MPI_Barrier()
        40  5.139E+04   1.13E+04  3.134E+04  1.187E+04  Heap Memory Used (KB) : Entry : void func(int, int) C => MPI_Bcast()
       120  5.139E+04   1.13E+04  3.134E+04  1.187E+04  Heap Memory Used (KB) : Entry : void func(int, int) C => MPI_Recv()
       120  5.139E+04   1.13E+04  3.134E+04  1.187E+04  Heap Memory Used (KB) : Entry : void func(int, int) C => MPI_Send()
       365  5.139E+04       2065  3.116E+04   1.21E+04  Heap Memory Used (KB) : Exit
                                                                                               3,7
```

Context event = atomic event + executing context

% setenv TAU_CALLPATH_DEPTH 2

% setenv TAU_TRACK_HEAP 1

A New Approach: tau_exec

- Runtime instrumentation by pre-loading the measurement library
- Works on dynamic executables (default under Linux)
- Substitutes I/O, MPI and memory allocation/ deallocation routines with instrumented calls
- Track interval events (e.g., time spent in write()) as well as atomic events (e.g., how much memory was allocated) in wrappers
- Accurately measure I/O and memory usage

Issues

- Heap memory usage reported by the mallinfo() call is not 64-bit clean.
  - 32 bit counters in Linux roll over when > 4GB memory is used
  - We keep track of heap memory usage in 64 bit counters inside TAU
- Compensation of perturbation introduced by tool
  - Only show what application uses
  - Create guards for TAU calls to not track I/O and memory allocations/ de-allocations performed inside TAU
- Provide broad POSIX I/O and memory coverage

tau_exec: Usage

sameer on l2: /usr/cta/pet/pkgs/tau2

File Edit View Terminal Tabs Help

```
 > tau_exec

Usage: tau_exec [options] [--] <exe> <exe options>

Options:
        -v        verbose mode
        -qsub     Use qsub mode (see below)
        -io       track I/O
        -memory   track memory
        -T <DISABLE,ICPC,MPI,PDT,PROFILE,PTHREAD,PYTHON,SERIAL> : specify TAU option
        -XrunTAUsh-<options> : specify TAU library directly

Notes:
        Defaults if unspecified: -T MPI
        MPI is assumed unless SERIAL is specified

Example:
    mpirun -np 2 tau_exec -io ./ring

qsub mode:
    Original:
      qsub -n 1 --mode smp -t 10 ./a.out
    With TAU:
      tau_exec -qsub -io -memory -- qsub -n 1 --mode smp -t 10 ./a.out

 > 
```

tau_exec

- Uninstrumented execution
  - % mpirun –np 256 ./a.out
- Track MPI Performance
  - % mpirun –np 256 tau_exec ./a.out
- Track I/O Performance (MPI enabled by default)
  - % mpirun –np 256 tau_exec –io ./a.out
- Track Memory
  - % setenv TAU_TRACK_MEMORY_LEAKS 1
  - % mpirun –np 256 tau_exec –memory ./a.out
- Track I/O and Memory
  - % mpirun –np 256 tau_exec –io –memory ./a.out

tau_exec: A tool to simplify Memory, I/O evaluation

xterm

```
> cd ~/workshop-point/matmult
> mpif90 matmult.f90 -o matmult
> mpirun -np 4 ./matmult
>
> # To use tau_exec to measure the I/O and memory usage:
> mpirun -np 4 tau_exec -io -memory ./matmult
>
> # To measure memory leaks and get complete callpaths
> setenv TAU_TRACK_MEMORY_LEAKS 1
> setenv TAU_CALLPATH_DEPTH 100
> mpirun -np 4 tau_exec -io -memory ./matmult
> paraprof
> # Right click on a given rank (e.g, "node 2") and choose "Show Context Event
> # Window" and expand the ".TAU Application" node to see the callpath
> # To use a different configuration (e.g., Makefile.tau-papi-mpi-pdt)
> setenv TAU_METRICS TIME:PAPI_FP_INS:PAPI_L1_DCM
> mpirun -np 4 tau_exec -io -memory -T papi,mpi,pdt ./matmult
> # Using tau_exec with DyninstAPI:
> tau_run matmult -o matmult.i
> mpirun -np 4 tau_exec -io -memory ./matmult.i
>
> tau_run -XrunTAUsh-papi-mpi-pdt matmult -o matmult.i
> mpirun -np 4 tau_exec -io -memory -T papi,mpi,pdt ./matmult.i
> paraprof
```

Environment Variables in TAU

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_LEAKS	0	Setting to 1 turns on leak detection
TAU_TRACK_HEAP or TAU_TRACK_HEADROOM	0	Setting to 1 turns on tracking heap memory/headroom at routine entry & exit using context events (e.g., Heap at Entry: main=>foo=>bar)
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_SYNCHRONIZE_CLOCKS	1	Synchronize clocks across nodes to correct timestamps in traces
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 separted list generates other metrics. (e.g., TIME:linuxtimers:PAPI_FP_OPS:PAPI_NATIVE_<event>)

Memory Leaks in MPI

TAU: ParaProf: Context Events for thread: n,c,t, 0,0,0 – samarc_obe_4p_iomem_cp.ppk

Name △	Total	MeanValue	NumSamples	MaxValue	MinValue	Std. Dev.
▼ .TAU application						
▼ MPI_Finalize()						
free size	23,901,253	22,719.822	1,052	2,099,200	2	186,920.948
malloc size	5,013,902	65,972.395	76	5,000,000	2	569,732.815
MEMORY LEAK!	5,000,264	500,026.4	10	5,000,000	3	1,499,991.2
▼ read()						
Bytes Read	4	4	1	4	4	0
READ Bandwidth (MB/s) <file="pipe">		0.308	1	0.308	0.308	0
Bytes Read <file="pipe">	4	4	1	4	4	0
READ Bandwidth (MB/s)		0.308	1	0.308	0.308	0
▼ write()						
WRITE Bandwidth (MB/s)		0.635	102	12	0	1.472
Bytes Written <file="/dev/infiniband/rdma_cm">	24	24	1	24	24	0
Bytes Written	1,456	14.275	102	28	4	5.149
WRITE Bandwidth (MB/s) <file="/dev/infiniband/uverbs0">		0.528	97	12	0.089	1.32
Bytes Written <file="pipe">	64	16	4	28	4	12
WRITE Bandwidth (MB/s) <file="/dev/infiniband/rdma_cm">		1.714	1	1.714	1.714	0
Bytes Written <file="/dev/infiniband/uverbs0">	1,368	14.103	97	24	12	4.562
WRITE Bandwidth (MB/s) <file="pipe">		2.967	4	5.6	0	2.644
▼ writev()						
WRITE Bandwidth (MB/s)		4.108	2	7.667	0.549	3.559
Bytes Written	297	148.5	2	230	67	81.5
WRITE Bandwidth (MB/s) <file="socket">		4.108	2	7.667	0.549	3.559
Bytes Written <file="socket">	297	148.5	2	230	67	81.5
▼ readv()						
Bytes Read	112	28	4	36	20	8
READ Bandwidth (MB/s) <file="socket">		25.5	4	36	10	11.079
Bytes Read <file="socket">	112	28	4	36	20	8
READ Bandwidth (MB/s)		25.5	4	36	10	11.079
▼ MPI_Comm_free()						
free size	10,952	195.571	56	1,024	48	255.353
▶ read()						
▶ MPI_Type_free()						
▶ MPI_Init()						
▼ fopen64()						
free size	231,314	263.456	878	568	35	221.272
MEMORY LEAK!	1,105,956	1,868.169	592	7,200	32	3,078.574
malloc size	1,358,286	901.318	1,507	7,200	32	2,087.737
▶ OurMain()						
▶ fclose()						

Instrumentation Issues

- Dynamic Instrumentation using DyninstAPI [U. Wisconsin, Madison, and U. Maryland]
- Pre-execution instrumentation
- Shell script spawned the task on the node and instrumented it
- As the number of processors increased, more time was wasted:
  - transferring un-instrumented executables to the compute nodes,
  - Instrumenting the application binary
- Solution: Binary re-writing!

Re-writing Binaries

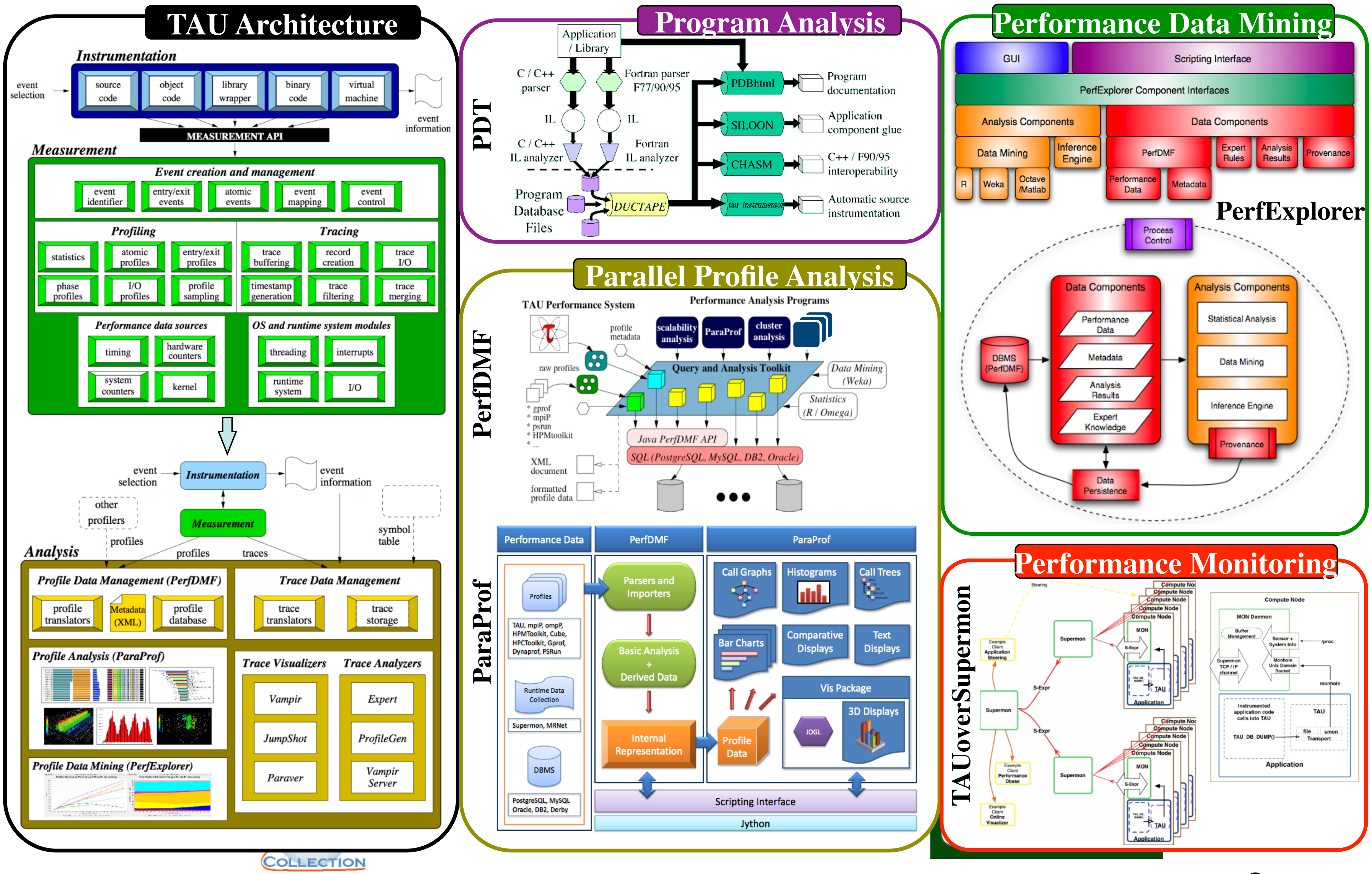
- Support for both static and dynamic executables
- Specify the list of routines to instrument/exclude from instrumentation
- Specify the TAU measurement library to be injected
- Simplify the usage of TAU:
  - To instrument:
    - % tau_run a.out –o a.inst
  - To perform measurements, execute the application:
    - % mpirun –np 4 ./a.inst
  - To analyze the data:
    - % paraprof

Using TAU with DyninstAPI : tau_run

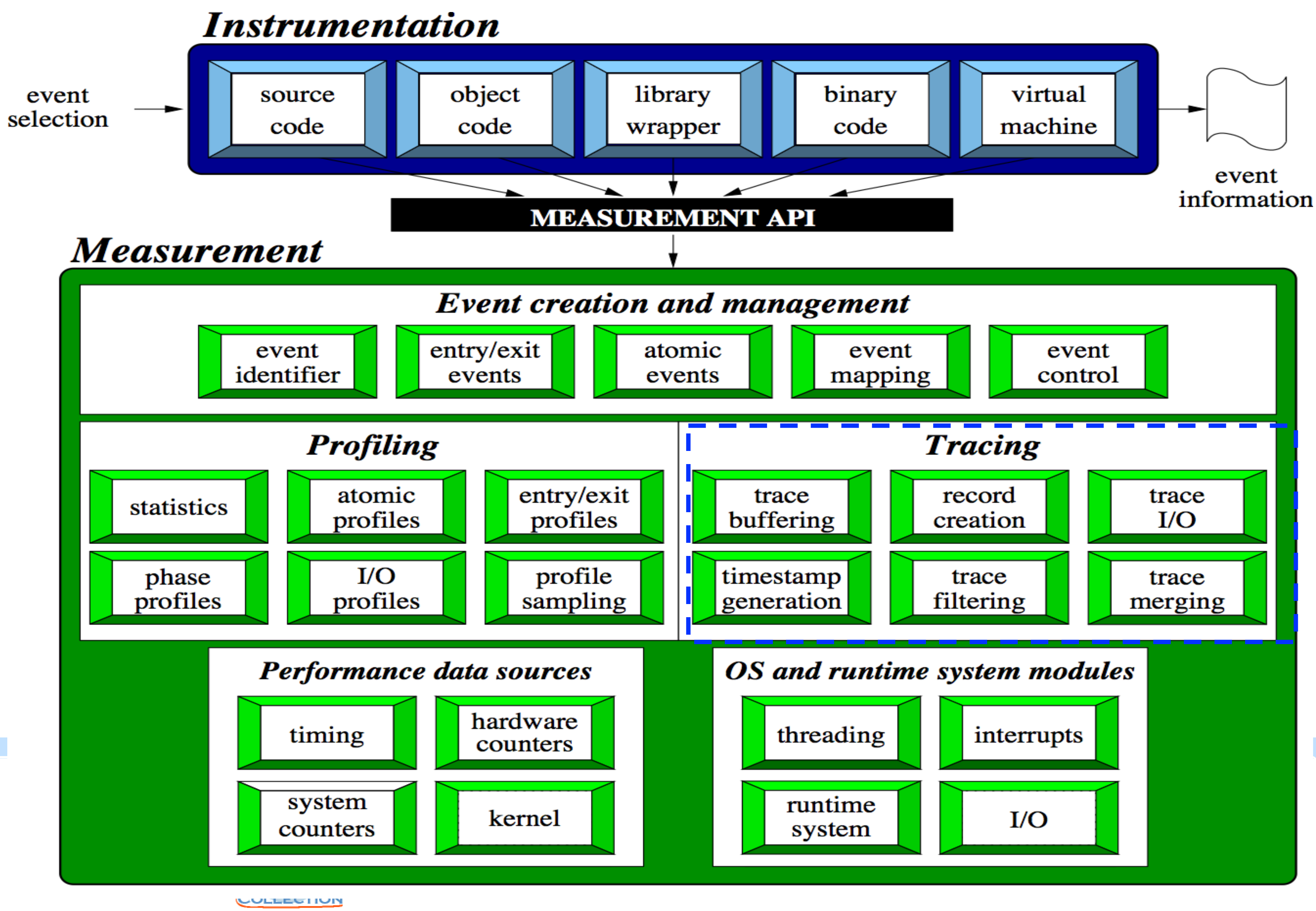
livetau@paratools01:~

```
/home/livetau% cd ~/tutorial
/home/livetau/tutorial% # Build an uninstrumented bt NAS Parallel Benchmark
/home/livetau/tutorial% make bt CLASS=W NPROCS=4
/home/livetau/tutorial% cd bin
/home/livetau/tutorial/bin% # Run the instrumented code
/home/livetau/tutorial/bin% mpirun -np 4 ./bt_W.4
/home/livetau/tutorial/bin%
/home/livetau/tutorial/bin% # Instrument the executable using TAU with DyninstAPI
/home/livetau/tutorial/bin%
/home/livetau/tutorial/bin% tau_run ./bt_W.4 -o ./bt.i
/home/livetau/tutorial/bin% rm -rf profile.* MULT*
/home/livetau/tutorial/bin% mpirun -np 4 ./bt.i
/home/livetau/tutorial/bin% paraprof
/home/livetau/tutorial/bin%
/home/livetau/tutorial/bin% # Choose a different TAU configuration
/home/livetau/tutorial/bin% ls $TAU/libTAUsh
libTAUsh-depthlimit-mpi-pdt.so*              libTAUsh-papi-pdt.so*
libTAUsh-mpi-pdt.so*                         libTAUsh-papi-pthread-pdt.so*
libTAUsh-mpi-pdt-upc.so*                     libTAUsh-param-mpi-pdt.so*
libTAUsh-mpi-python-pdt.so*                  libTAUsh-pdt.so*
libTAUsh-papi-mpi-pdt.so*                    libTAUsh-pdt-trace.so*
libTAUsh-papi-mpi-pdt-upc.so*                libTAUsh-phase-papi-mpi-pdt.so*
libTAUsh-papi-mpi-pdt-upc-udp.so*            libTAUsh-pthread-pdt.so*
libTAUsh-papi-mpi-pdt-vampirtrace-trace.so* libTAUsh-python-pdt.so*
libTAUsh-papi-mpi-python-pdt.so*
/home/livetau/tutorial/bin% ls $TAU/libTAUsh-
/home/livetau/tutorial/bin%
/home/livetau/tutorial/bin% tau_run -XrunTAUsh-papi-mpi-pdt-vampirtrace-trace bt_W.4 -o bt.vpt
/home/livetau/tutorial/bin% setenv VT_METRICS PAPI_FP_INS:PAPI_L1_DCM
/home/livetau/tutorial/bin% mpirun -np 4 ./bt.vpt
/home/livetau/tutorial/bin% vampir bt.vpt.otf &
/home/livetau/tutorial/bin%
```

TAU Performance System Components



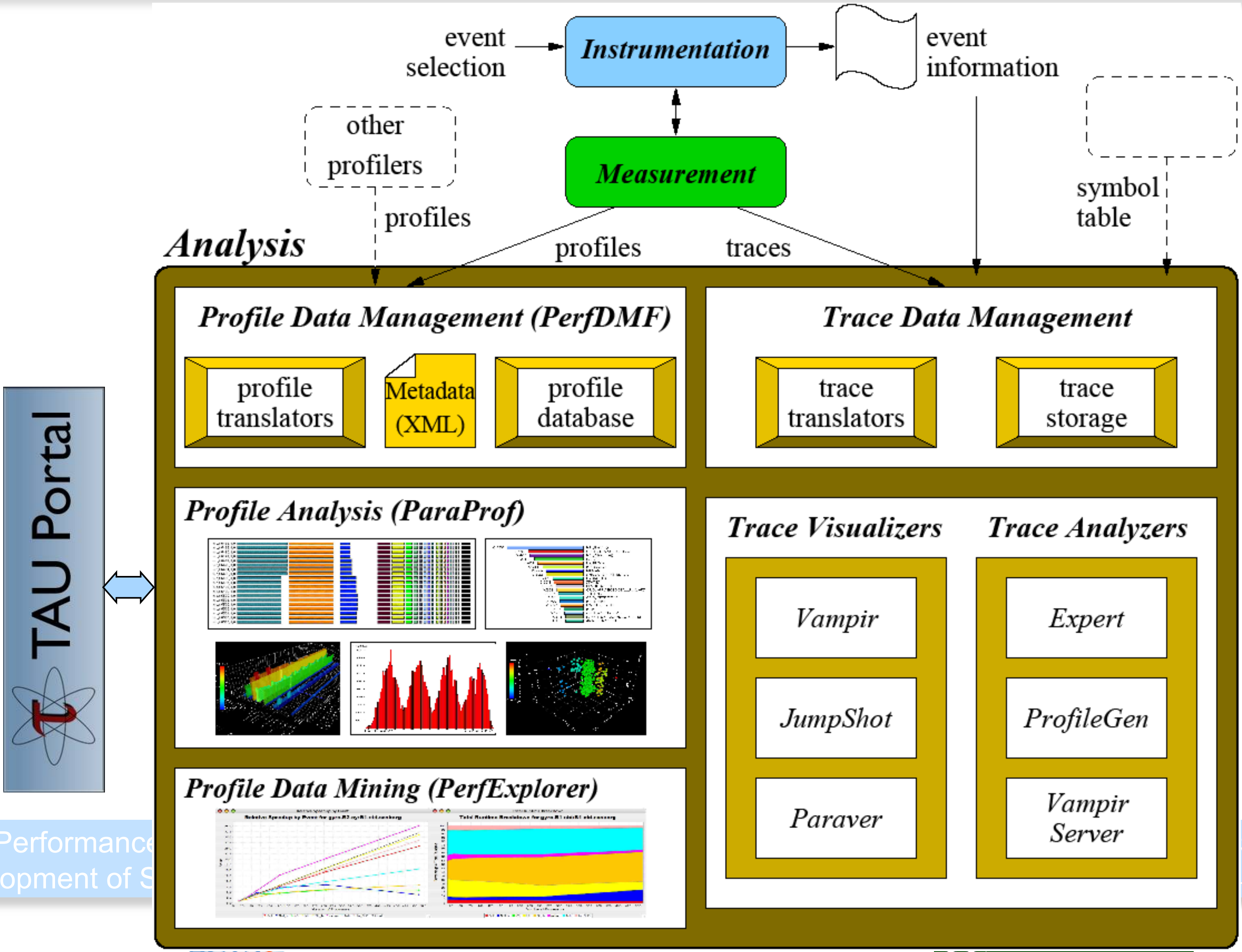
TAU Instrumentation / Measurement



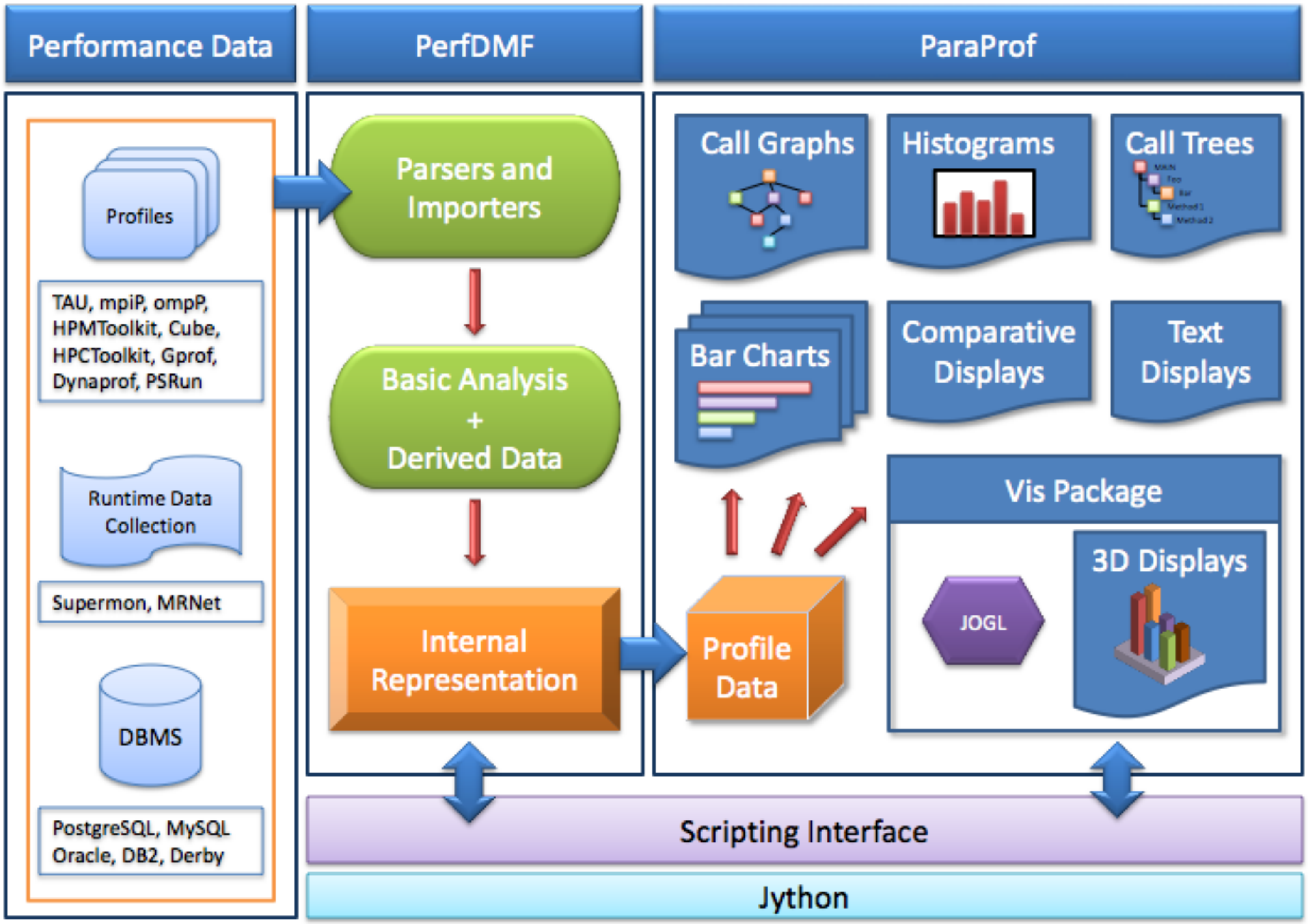
TAU Instrumentation

- Flexible instrumentation mechanisms at multiple levels
  - Source code
    - manual (TAU API, TAU Component API)
    - automatic
      - C, C++, F77/90/95 (Program Database Toolkit (*PDT*))
      - OpenMP (directive rewriting (*Opari*), *POMP spec*)
  - Object code
    - pre-instrumented libraries (e.g., MPI using *PMPI*)
    - statically-linked and dynamically-linked
  - Executable code
    - dynamic instrumentation (pre-execution) (*DynInstAPI*)
    - virtual machine instrumentation (e.g., Java using *JVMPI*)
    - Python interpreter based instrumentation at runtime
  - Proxy Components

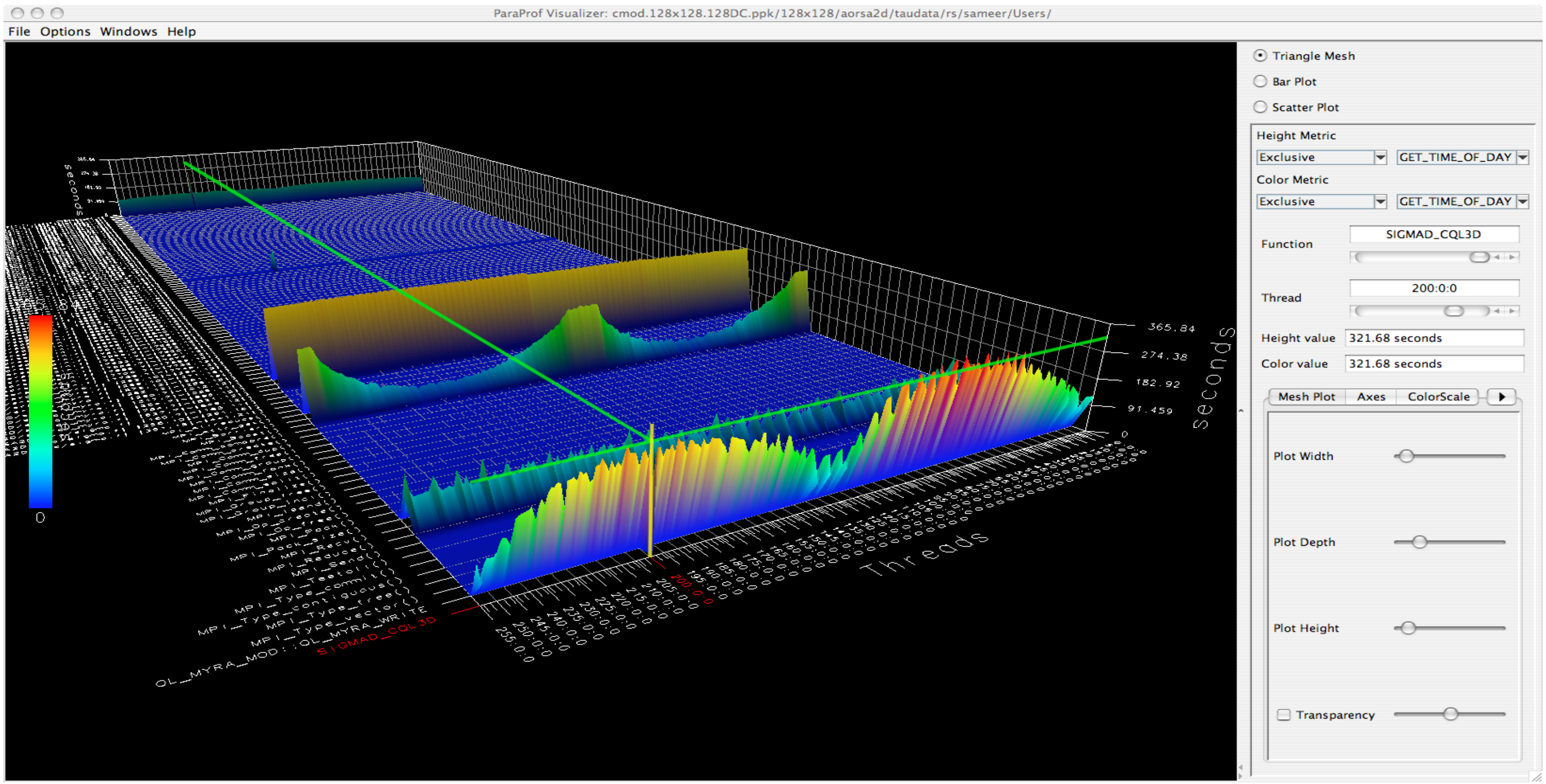
TAU Analysis



ParaProf Profile Analysis Framework



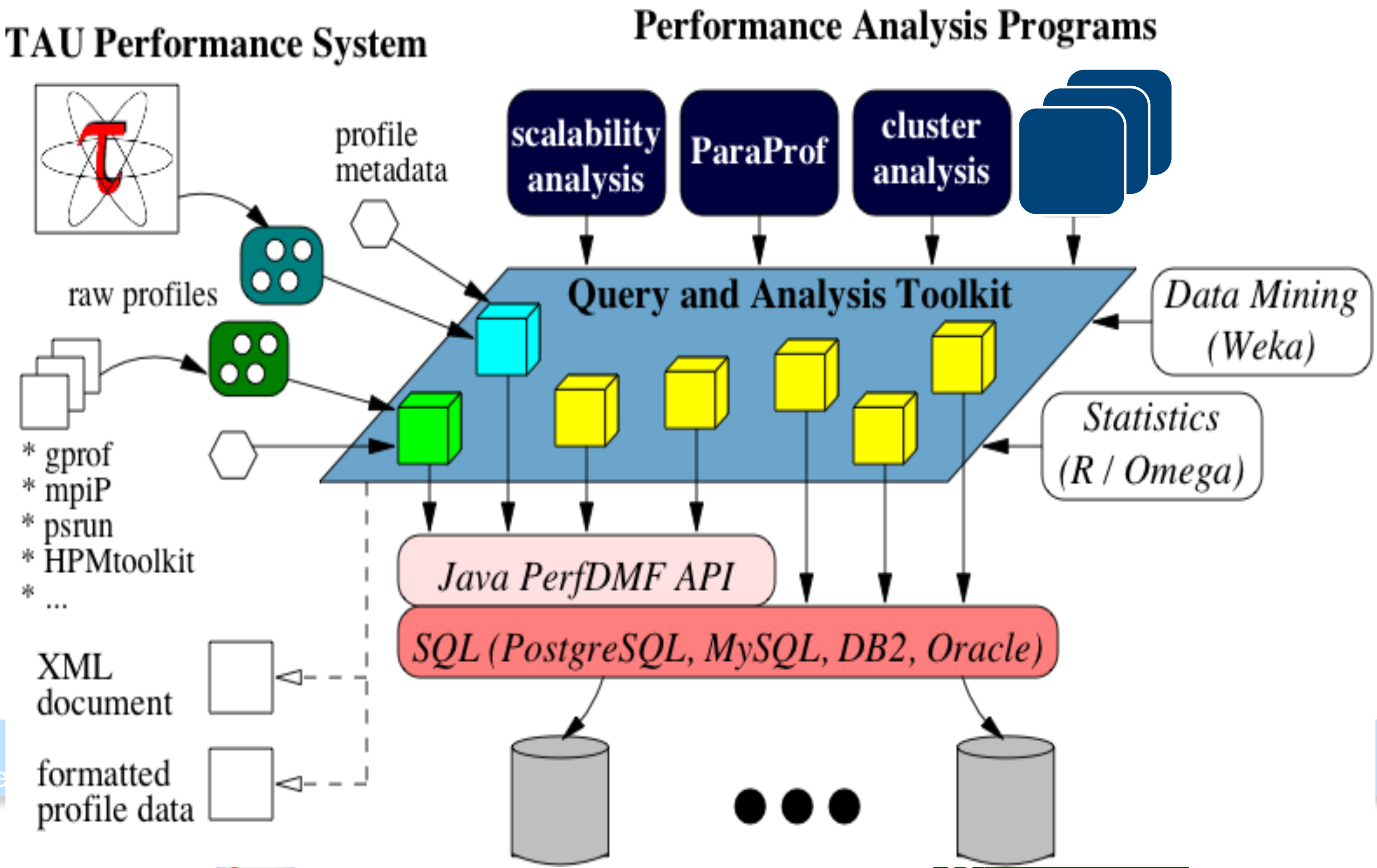
Parallel Profile Visualization: ParaProf



ACTS COLLECTION

UNIVERSITY OF OREGON

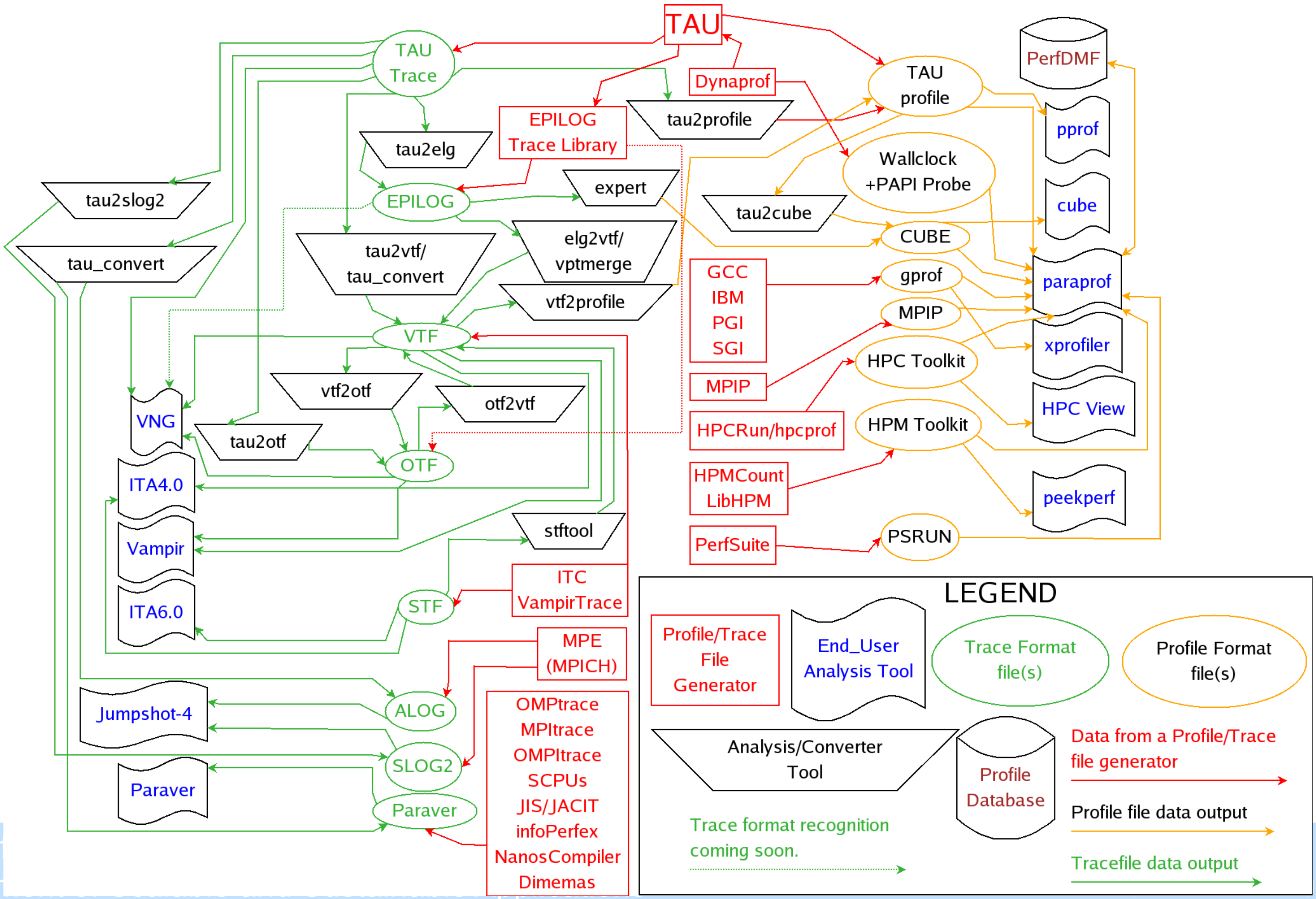
PerfDMF Architecture



High
Deve

COLLECTION

Building Bridges to Other Tools



ACTS COLLECTION

UNIVERSITY OF OREGON

Direct Performance Observation

- Execution actions of interest exposed as events
  - In general, actions reflect some execution state
    - presence at a code location or change in data
    - occurrence in parallelism context (thread of execution)
  - Events encode actions for performance system to observe
- Observation is direct
  - Direct instrumentation of program (system) code (probes)
  - Instrumentation invokes performance measurement
  - Event measurement: performance data, meta-data, context
- Performance experiment
  - Actual events + performance measurements
- Contrast with (indirect) event-based sampling

TAU Instrumentation Approach

- Support for standard program events
  - Routines, classes and templates
  - Statement-level blocks
  - Begin/End events (Interval events)
- Support for user-defined events
  - Begin/End events specified by user
  - Atomic events (e.g., size of memory allocated/freed)
  - Flexible selection of event statistics
- Provides static events and dynamic events
- Enables "semantic" mapping
- Specification of event groups (aggregation, selection)
- Instrumentation optimization

TAU Event Interface

- Events have a type, a group association, and a name
- TAU events names are character strings
  - Powerful way to encode event information
  - Inefficient way to communicate each event occurrence
- TAU maps a new event name to an event ID
  - Done when event is first encountered (get event handle)
  - Event ID is used for subsequent event occurrences
  - Assigning a uniform event ID a priori is problematic
- A new event is identified by a new event name in TAU
  - Can create new event names at runtime
  - Allows for dynamic events (TAU renames events)
  - Allows for context-based, parameter-based, phase events

Using TAU: A brief Introduction

- TAU supports several measurement options (profiling, tracing, profiling with hardware counters, etc.)
- Each measurement configuration of TAU corresponds to a unique stub makefile and library that is generated when you configure it
- To instrument source code using PDT
  - Choose an appropriate TAU stub makefile in <arch>/lib:

  % setenv TAU_MAKEFILE /usr/local/packages/tau/i386_linux/lib/Makefile.tau-mpi-pdt

  % setenv TAU_OPTIONS '-optVerbose …' (see tau_compiler.sh -help)

  And use tau_f90.sh, tau_cxx.sh or tau_cc.sh as Fortran, C++ or C compilers:

  % mpif90 foo.f90

  changes to

  % tau_f90.sh foo.f90

- Execute application and analyze performance data:

  % pprof (for text based profile display)

  % paraprof (for GUI)

TAU Measurement Configuration

```
% cd /usr/local/packages/tau/i386_linux/lib; ls Makefile.*
Makefile.tau-pdt
Makefile.tau-mpi-pdt
Makefile.tau-opari-openmp-mpi-pdt
Makefile.tau-mpi-scalasca-epilog-pdt
Makefile.tau-mpi-vampirtrace-pdt
Makefile.tau-multiplecounters-mpi-papi-pdt
Makefile.tau-multiplecounters-papi-mpi-openmp-opari-pdt
Makefile.tau-pthread-pdt…
```

- For an MPI+F90 application, you may want to start with:

Makefile.tau-mpi-pdt

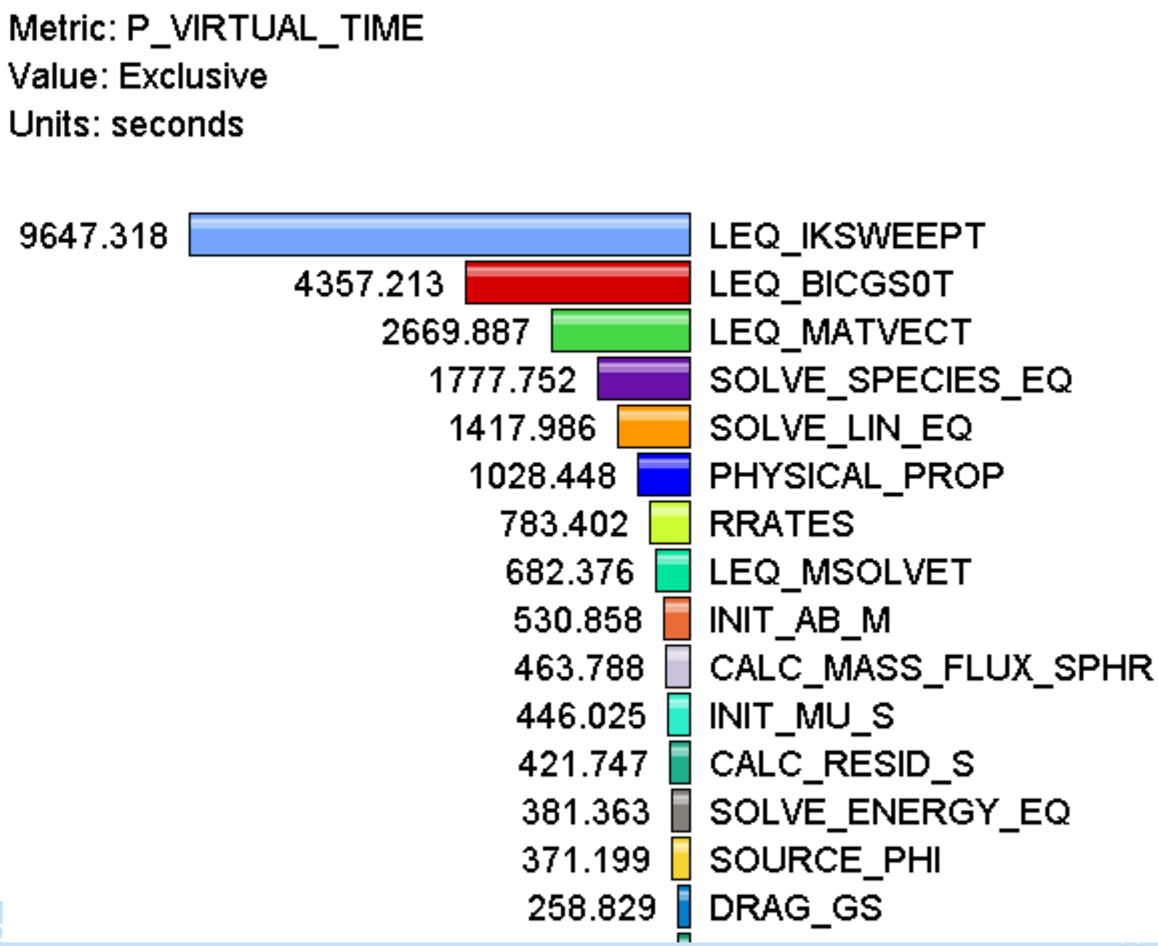
- Supports MPI instrumentation & PDT for automatic source instrumentation
- `% setenv TAU_MAKEFILE /usr/local/packages/tau/i386_linux/lib/Makefile.tau-mpi-pdt`
- `% tau_f90.sh matrix.f90 -o matrix`

ACTS COLLECTION

UNIVERSITY OF OREGON

Usage Scenarios: Routine Level Profile

- Goal: What routines account for the most time? How much?
- Flat profile with wallclock time:



Generating a flat profile with MPI

```
% setenv TAU_MAKEFILE /usr/local/packages/tau/i386_linux
                              /lib/Makefile.tau-mpi-pdt
% set path=(/usr/local/packages/tau/i386_linux/bin $path)
% tau_f90.sh matmult.f90 -o matmult
(Or edit Makefile and change F90=tau_f90.sh)

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

% paraprof app.ppk
```

TAU Measurement Configuration –

- ./configure –pdt=/usr/local/packages/pdtoolkit-3.16 -mpi Configure using PDT and MPI
- ./configure -papi=/usr/local/packages/papi-4.0.0 -pdt=<dir> -mpi ; make clean install
  - Use PAPI counters (one or more) with C/C++/F90 automatic instrumentation. Also instrument the MPI library.
- Typically configure multiple measurement libraries using installtau
- Past configurations are stored in TAU's .all_configs file and .installflags
- Each configuration creates a unique <arch>/lib/Makefile.tau<options> stub makefile. It corresponds to the configuration options used. e.g.,
  - /usr/local/packages/i386_linux/lib/Makefile.tau-mpi-pdt
  - /usr/local/packages/i386_linux/lib/Makefile.tau-mpi-papi-pdt

Compile-Time Environment Variables

- Optional parameters for TAU_OPTIONS: [tau_compiler.sh –help]

-optVerbose	Turn on verbose debugging messages
-optCompInst	Use compiler based instrumentation
-optNoCompInst	Do not revert to compiler instrumentation if source instrumentation fails.
-optDetectMemoryLeaks	Turn on debugging memory allocations/ de-allocations to track leaks
-optKeepFiles	Does not remove intermediate .pdb and .inst.* files
-optPreProcess	Preprocess Fortran sources before instrumentation
-optTauSelectFile=""	Specify selective instrumentation file for tau_instrumentor
-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)
-optPdtF95Reset=""	Reset options for Fortran parser in PDT (f95parse/gfparse)
-optPdtCOpts=""	Options for C parser in PDT (cparse). Typically $(TAU_MPI_INCLUDE) $(TAU_INCLUDE) $(TAU_DEFS)
-optPdtCxxOpts=""	Options for C++ parser in PDT (cxxparse). Typically $(TAU_MPI_INCLUDE) $(TAU_INCLUDE) $(TAU_DEFS)

…

Compiling Fortran Codes with TAU

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

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

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

% setenv TAU_OPTIONS '-optCompInst -optVerbose'

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

% setenv TAU_OPTIONS '-optPreProcess -optVerbose -optDetectMemoryLeaks'

- To use an instrumentation specification file:

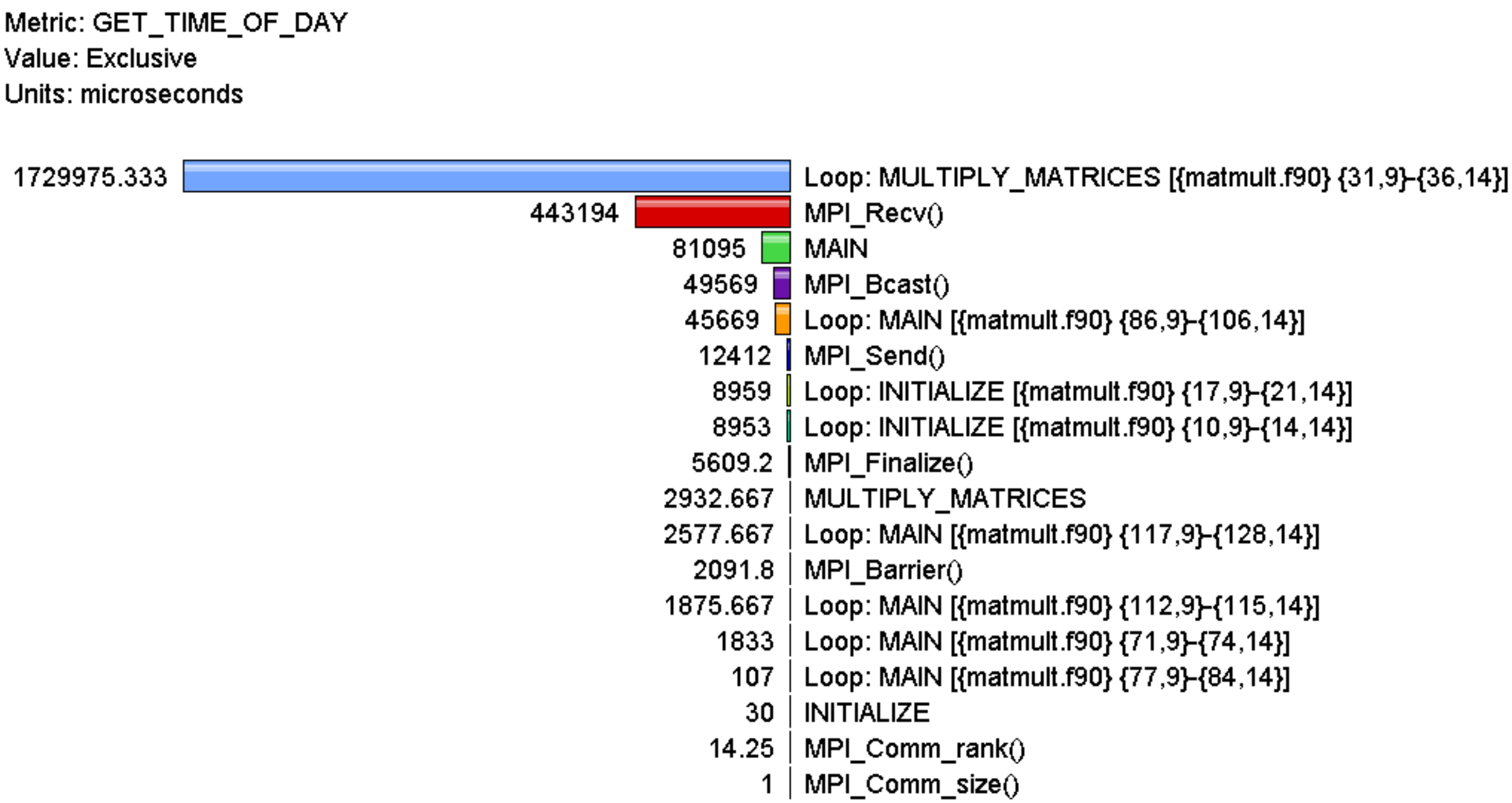
% setenv TAU_OPTIONS '-optTauSelectFile=mycmd.tau -optVerbose -optPreProcess'

% cat mycmd.tau

```
BEGIN_INSTRUMENT_SECTION
memory file="foo.f90" routine="#"
# instruments all allocate/deallocate statements in all routines in foo.f90
loops file="*" routine="#"
io file="abc.f90" routine="FOO"
END_INSTRUMENT_SECTION
```

Usage Scenarios: Loop Level Instrumentation

- Goal: What loops account for the most time? How much?
- Flat profile with wallclock time with loop instrumentation:

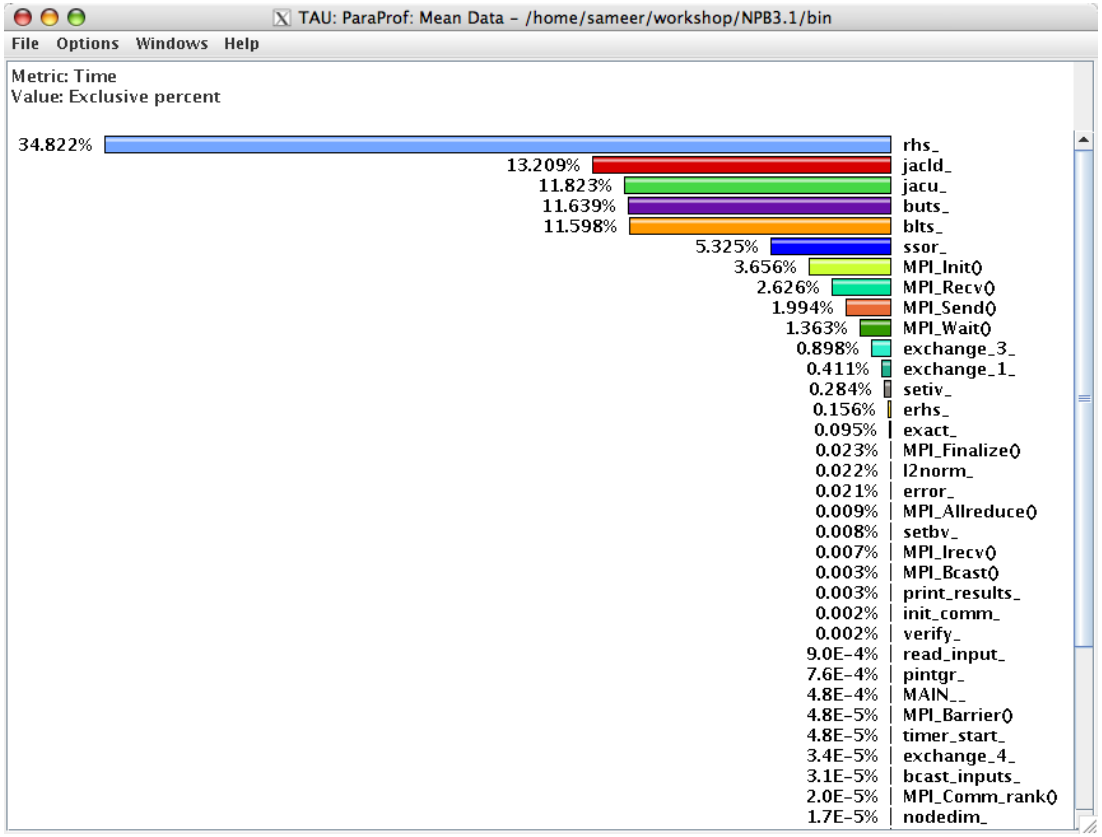


Generating a loop level profile

```
% setenv TAU_MAKEFILE /usr/local/packages/tau/i386_linux
                              /lib/Makefile.tau-mpi-pdt
% setenv TAU_OPTIONS '-optTauSelectFile=select.tau -optVerbose'
% cat select.tau
  BEGIN_INSTRUMENT_SECTION
  loops routine="#"
  END_INSTRUMENT_SECTION

% set path=(/usr/local/packages/tau/i386_linux/bin $path)
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)
% mpirun -np 4 ./a.out
% paraprof --pack app.ppk
  Move the app.ppk file to your desktop.

% paraprof app.ppk
```

Usage Scenarios: Compiler-based Instrumentation

- Goal: Easily generate routine level performance data using the compiler instead of PDT for parsing the source code

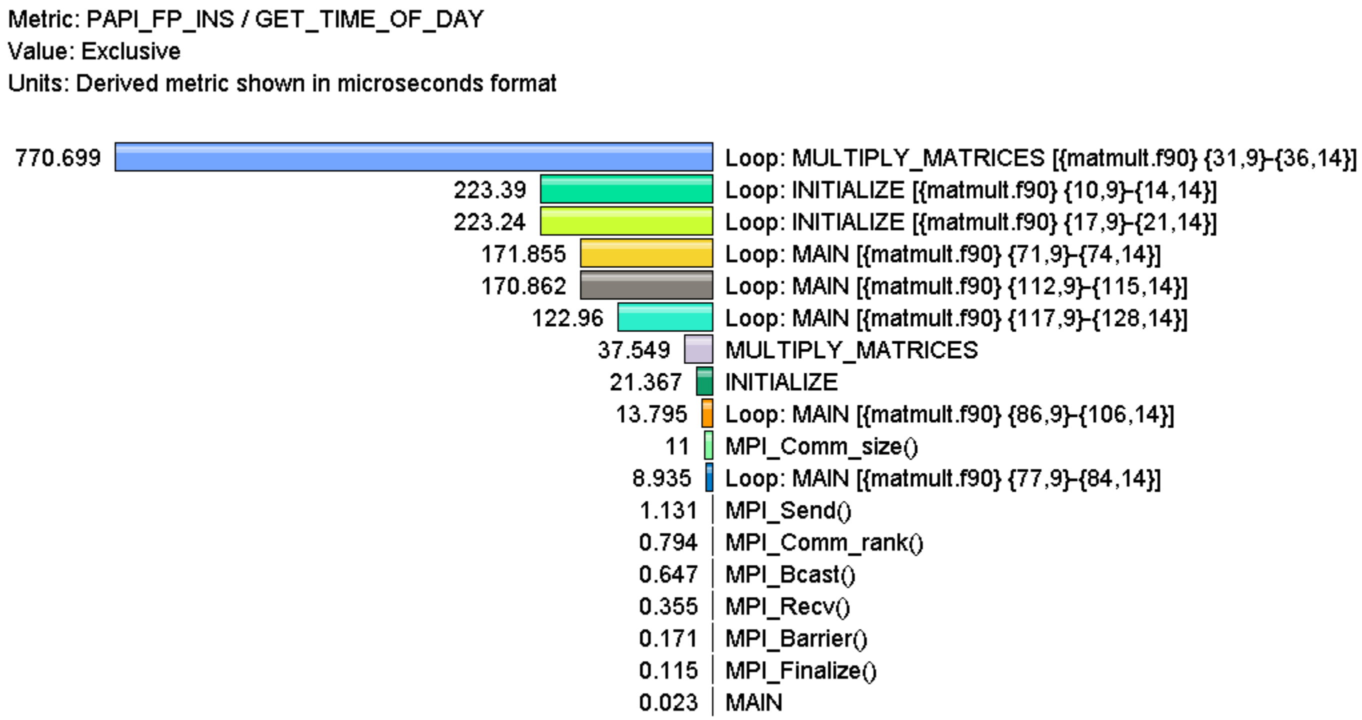


Use Compiler-Based Instrumentation

```
% setenv TAU_MAKEFILE /usr/local/packages/tau/i386_linux
                 /lib/Makefile.tau-mpi
% setenv TAU_OPTIONS '-optCompInst -optVerbose'
% % set path=(/usr/local/packages/tau/i386_linux/bin $path)
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)

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

Usage Scenarios: Calculate mflops in

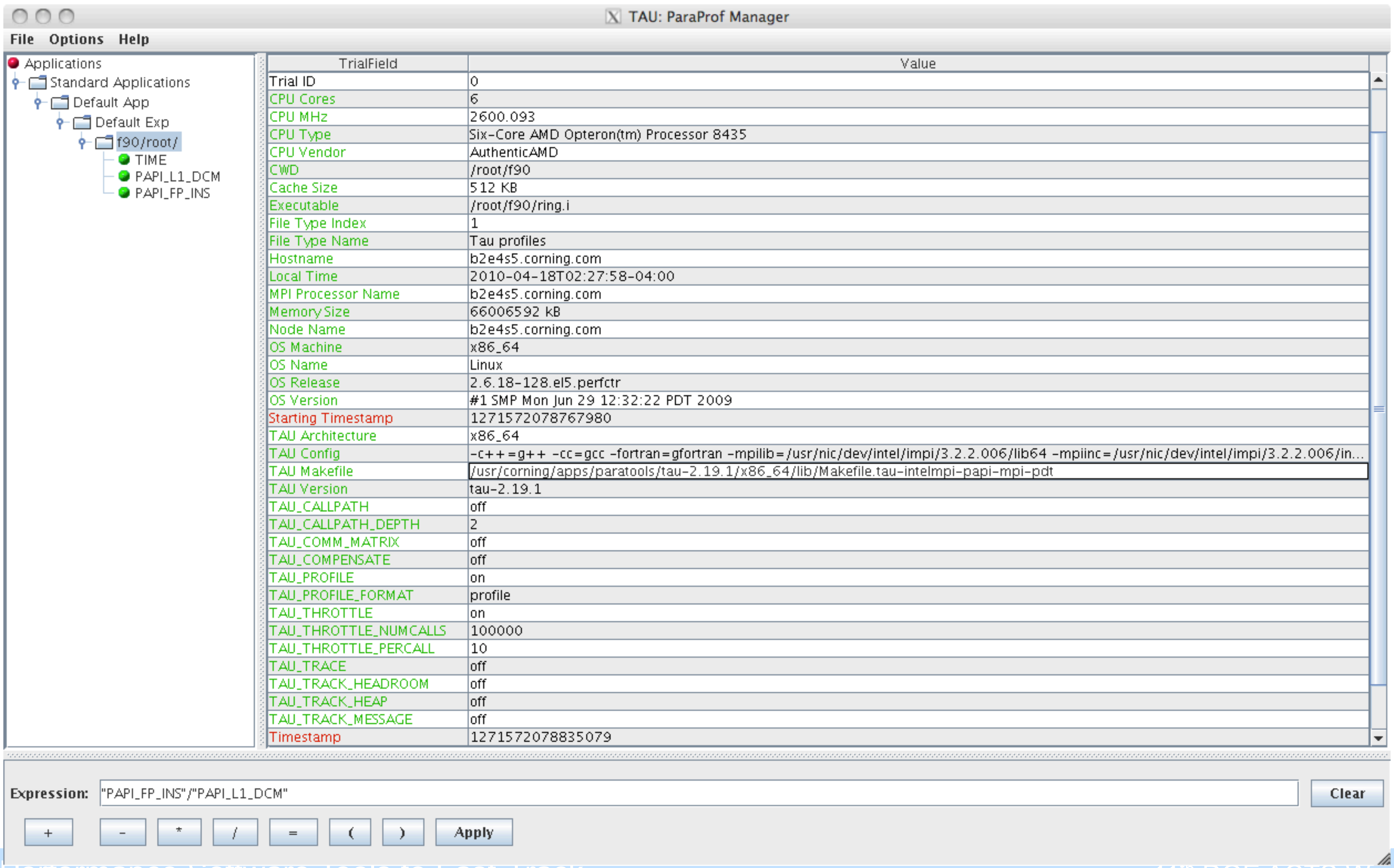
- Goal: What MFlops am I getting in all loops?
- Flat profile with PAPI_FP_INS/OPS and time with loop instrumentation:



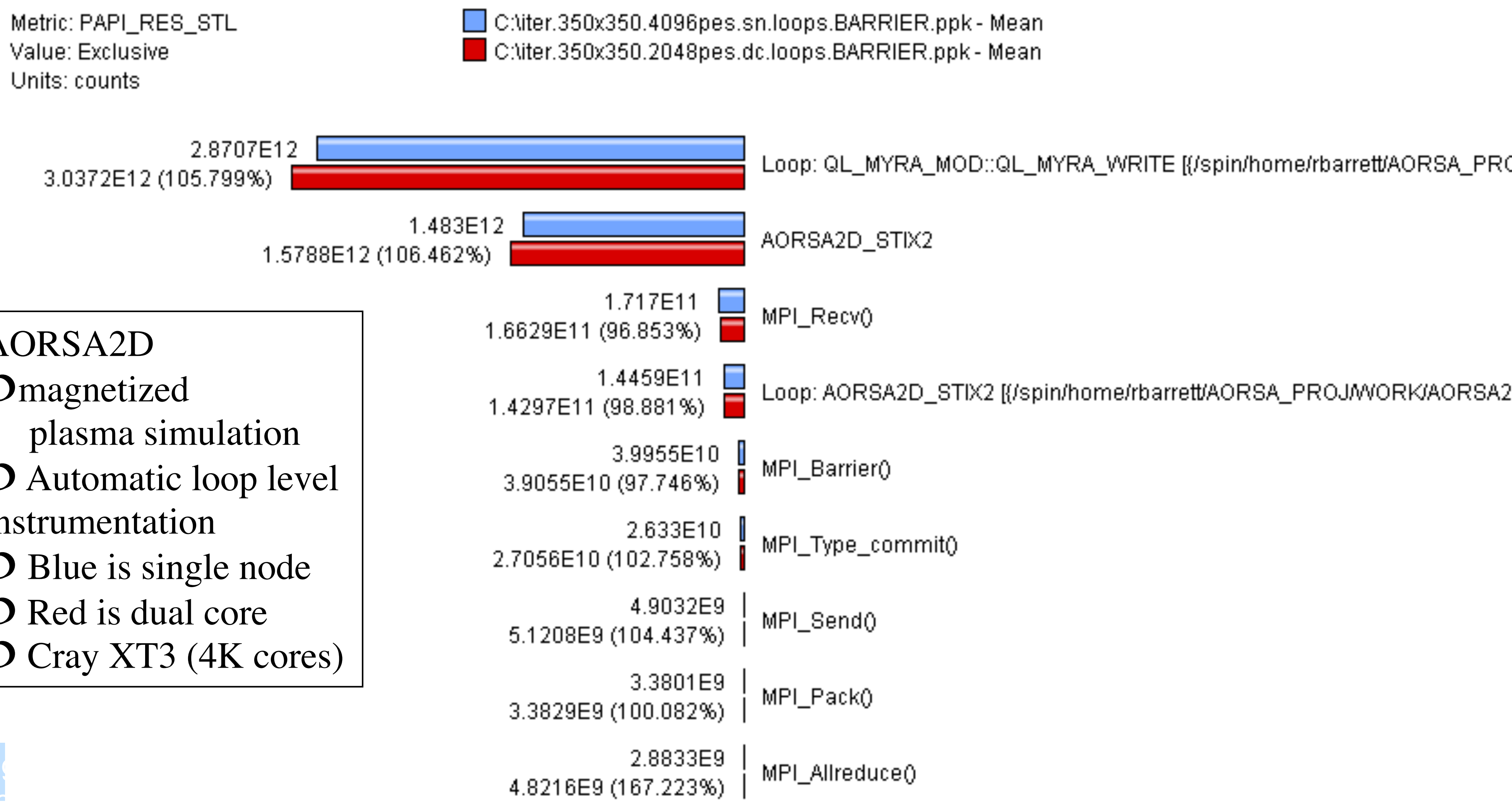
Generate a PAPI profile with 2 or more counters

```
% setenv TAU_MAKEFILE /usr/local/packages/tau/i386_linux
                            /lib/Makefile.tau-papi-mpi-pdt
% setenv TAU_OPTIONS '-optTauSelectFile=select.tau -optVerbose'
% cat select.tau
  BEGIN_INSTRUMENT_SECTION
  loops routine="#"
  END_INSTRUMENT_SECTION

% set path=(/usr/local/packages/tau/i386_linux/bin $path)
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)
% setenv TAU_METRICS TIME:PAPI_FP_INS:PAPI_L1_DCM
% mpirun -np 4  ./a.out
% paraprof --pack app.ppk
   Move the app.ppk file to your desktop.
% paraprof app.ppk
  Choose Options -> Show Derived Panel -> "PAPI_FP_INS", click "/", "TIME", click "Apply"
     choose.
```

Derived Metrics in ParaProf

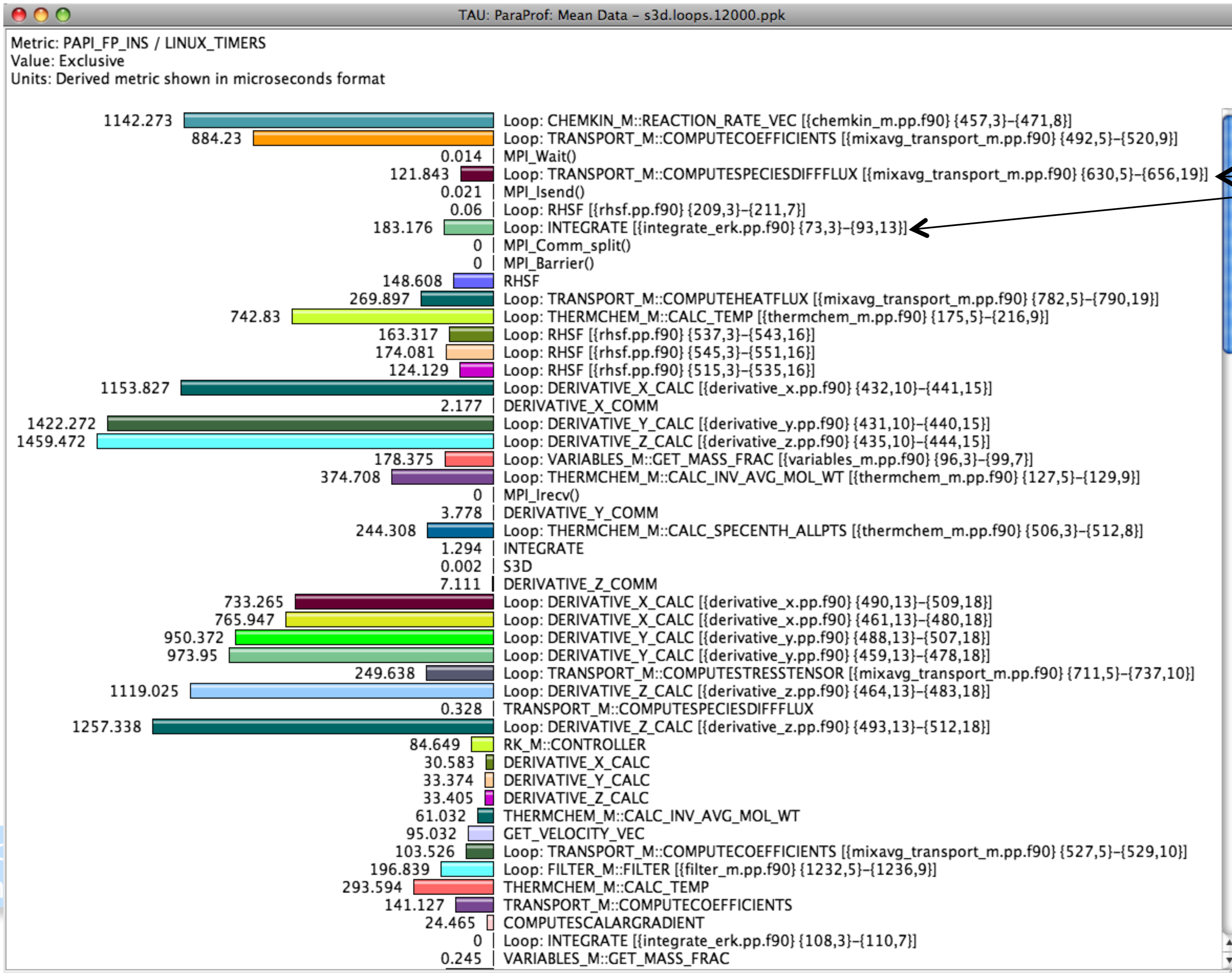


Comparing Effects of Multi-Core Processors



AORSA2D
- magnetized plasma simulation
- Automatic loop level instrumentation
- Blue is single node
- Red is dual core
- Cray XT3 (4K cores)

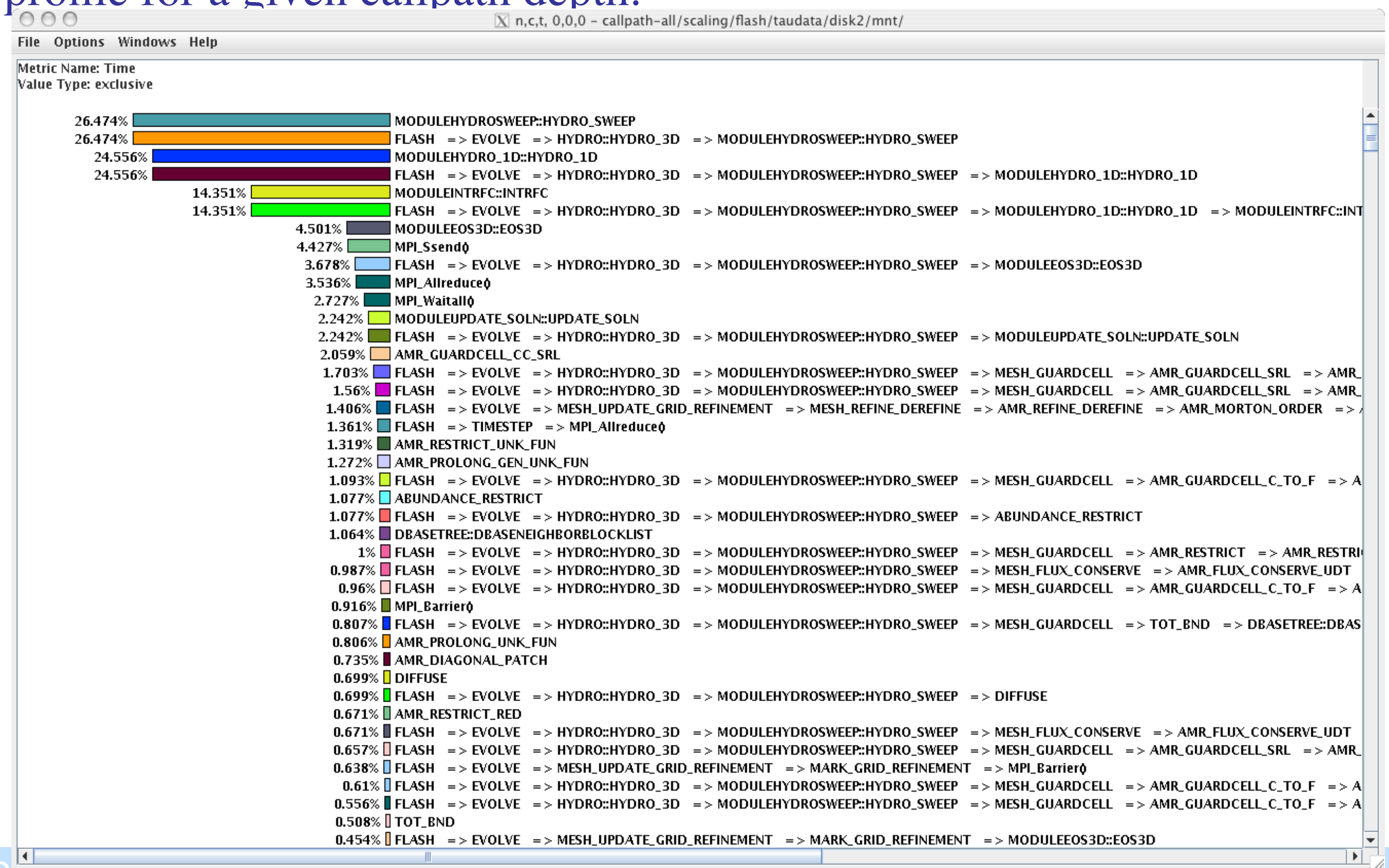
Mflops Sorted by Exclusive Time



low mflops?

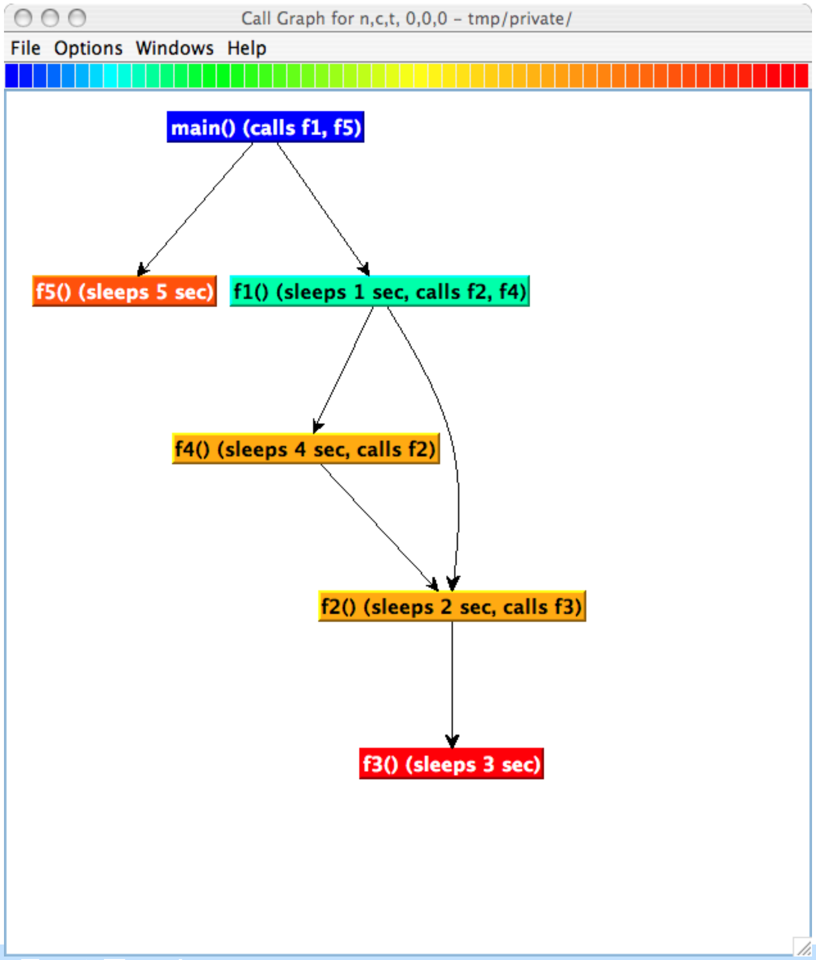
Generating Callpath Profiles

- Goal: To expose the calling sequence. E.g., what routine calls an MPI_Barrier()? Where?
- Callpath profile for a given callpath depth:



Callpath Profile

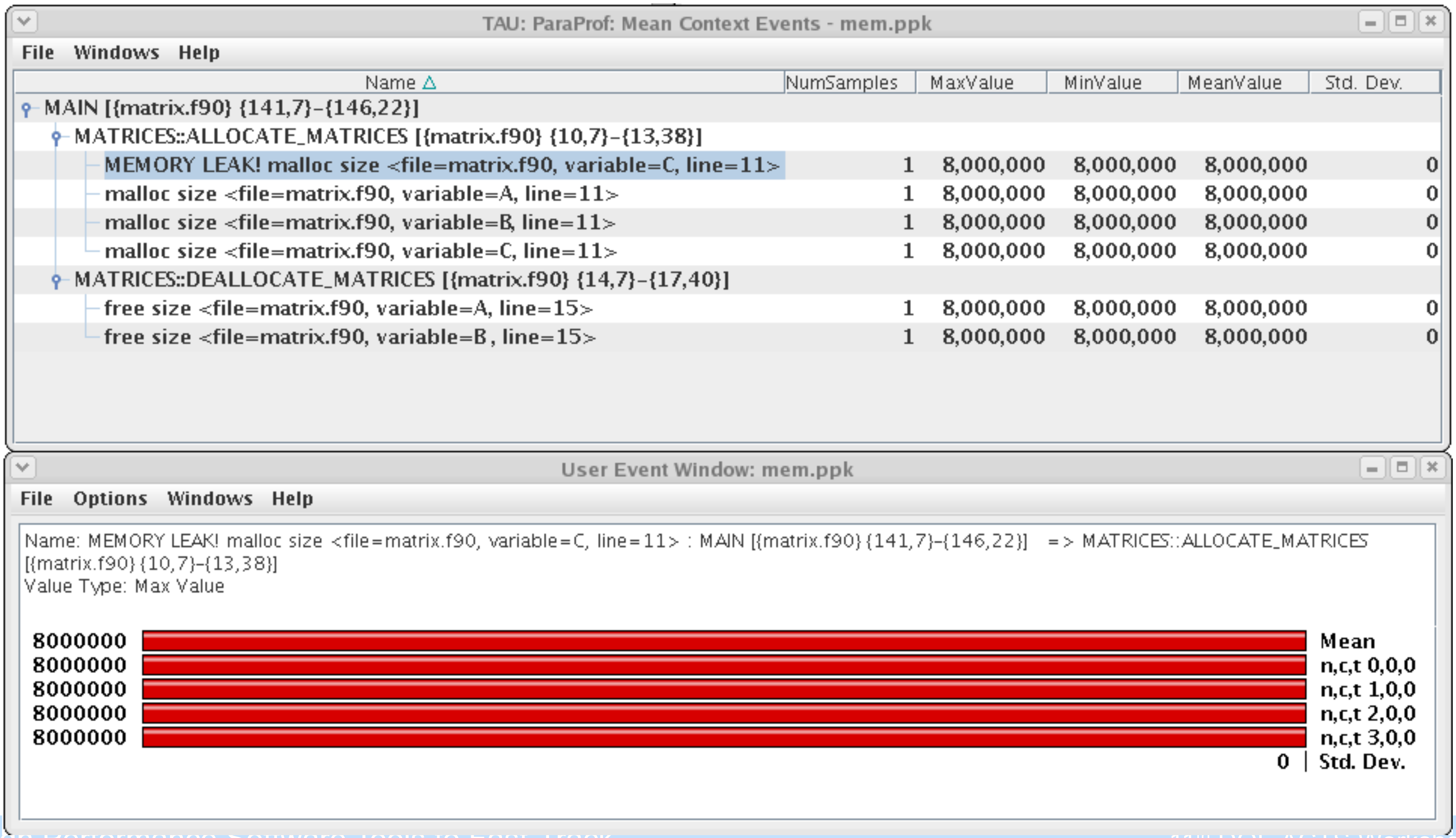
- Generates program callgraph



Generate a Callpath Profile

```
% setenv TAU_MAKEFILE /usr/local/packages/tau/i386_linux
                      /lib/Makefile.tau-mpi-pdt
% set path=(/usr/local/packages/tau/i386_linux/bin $path)
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)
% export TAU_CALLPATH_DEPTH 1
% export TAU_CALLPATH_DEPTH=100

% mpirun -np 4  ./a.out
% paraprof --pack app.ppk
  Move the app.ppk file to your desktop.
% paraprof app.ppk
(Windows -> Thread -> Call Graph)
```

Usage Scenario: Detect Memory Leaks

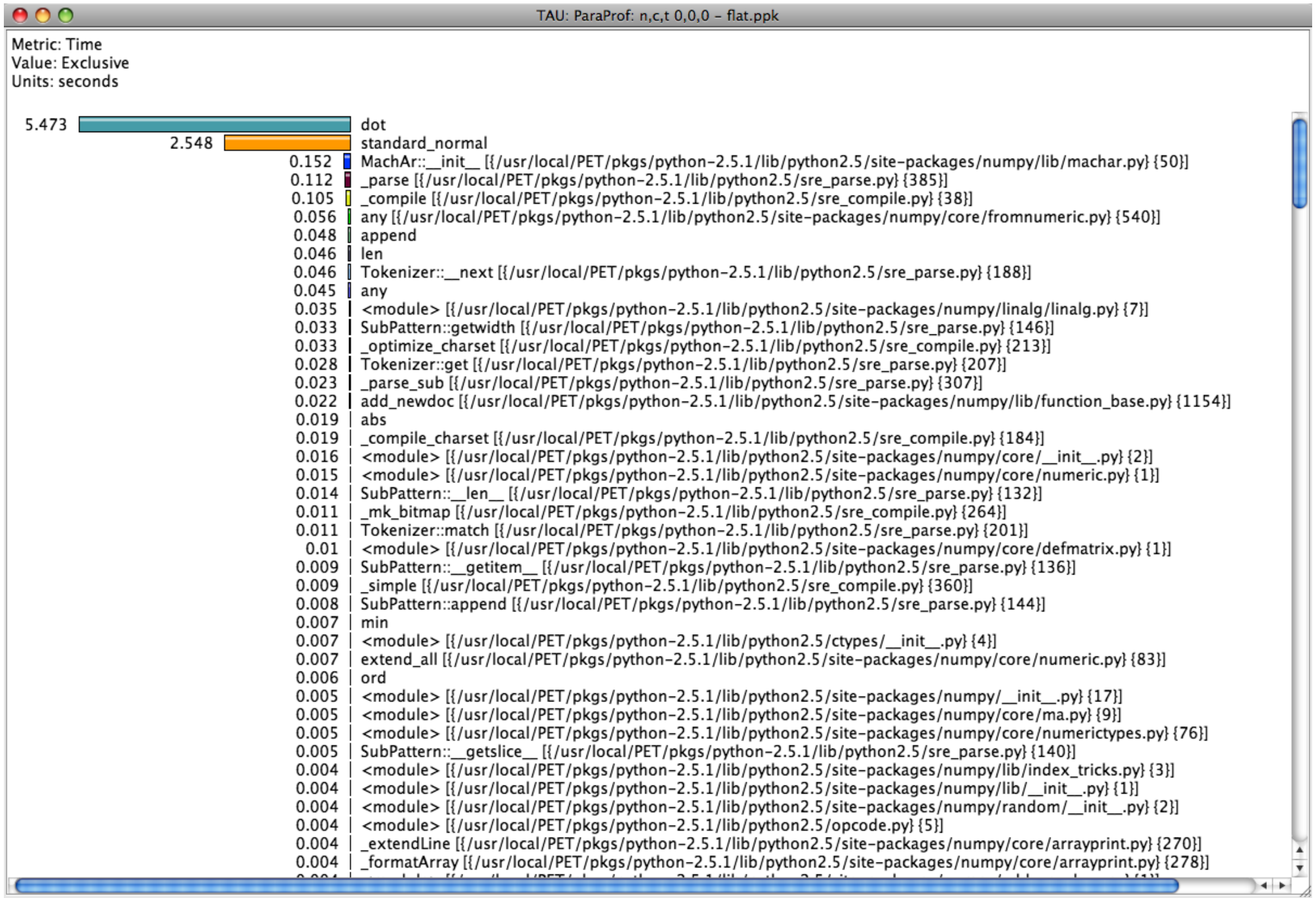


Detect Memory Leaks

```
% setenv TAU_MAKEFILE /usr/local/packages/tau/i386_linux
                          /lib/Makefile.tau-mpi-pdt
% setenv TAU_OPTIONS '-optDetectMemoryLeaks -optVerbose'
% set path=(/usr/local/packages/tau/i386_linux/bin $path)
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)
% setenv TAU_CALLPATH_DEPTH 100

% mpirun -np 4  ./a.out
% paraprof --pack app.ppk
  Move the app.ppk file to your desktop.
% paraprof app.ppk
(Windows -> Thread -> Context Event Window -> Select thread -> select...
   expand tree)
(Windows -> Thread -> User Event Bar Chart -> right click LEAK
-> Show User Event Bar Chart)
```

Instrument a Python program

- Goal: Generate a flat profile for a Python program



Instrumenting a Python program

Original code:

```
% cat foo.py
#!/usr/bin/env python
import numpy
ra=numpy.random
la=numpy.linalg

size=2000
a=ra.standard_normal((size,size))
b=ra.standard_normal((size,size))
c=la.linalg.dot(a,b)
print c
```

Create a wrapper:

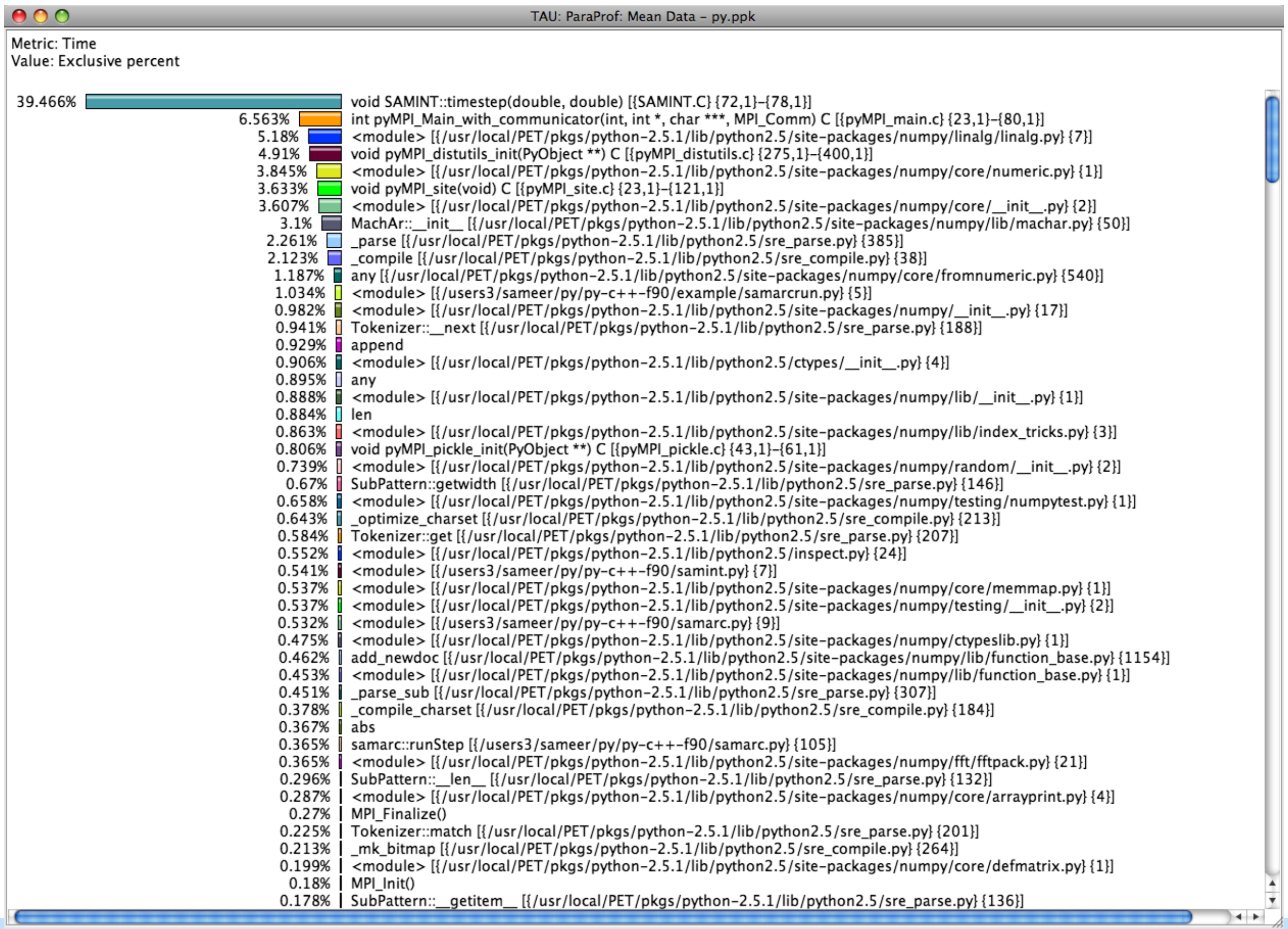
```
% cat wrapper.py
#!/usr/bin/env python

# setenv PYTHONPATH $PET_HOME/pkgs/tau-2.17.3/ppc64/lib/bindings-gnu-python-pdt

import tau

def OurMain():
    import foo

tau.run('OurMain()')
```

Generate a Python Profile

```
% setenv TAU_MAKEFILE /usr/local/packages/tau/i386_linux
              /lib/Makefile.tau-python-pdt
% set path=(/usr/local/packages/tau/i386_linux/bin $path)
% cat wrapper.py
  import tau
  def OurMain():
      import foo
  tau.run('OurMain()')
Uninstrumented:
% ./foo.py
Instrumented:
% export PYTHONPATH= <taudir>/i386_linux/<lib>/bindings-python-pdt
(same options string as TAU_MAKEFILE)
% export LD_LIBRARY_PATH=<taudir>/i386_linux/lib/bindings-python-pdt:
$LD_LIBRARY_PATH
% ./wrapper.py

Wrapper invokes foo and generates performance data
% pprof/paraprof
```

Usage Scenarios: Mixed Python+F90+C MPI

- Goal: Generate multi-level instrumentation for Python+MPI+C+F90+C++ ...



Generate a Multi-Language Profile

```
% setenv TAU_MAKEFILE /usr/local/packages/tau/i386_linux
                 /lib/Makefile.tau-python-mpi-pdt
% set path=(/usr/local/packages/tau/i386_linux/bin $path)
% setenv TAU_OPTIONS '-optShared -optVerbose…'
(Python needs shared object based TAU library)
% make F90=tau_f90.sh CXX=tau_cxx.sh CC=tau_cc.sh  (build libs, pyMPI w/TAU)
% cat wrapper.py
  import tau
  def OurMain():
      import App
  tau.run('OurMain()')
Uninstrumented:
% mpirun -np 4 pyMPI ./App.py
Instrumented:
% export PYTHONPATH= <taudir>/i386_linux/<lib>/bindings-python-mpi-pdt
(same options string as TAU_MAKEFILE)
% export LD_LIBRARY_PATH=<taudir>/i386_linux/lib/bindings-python-mpi-pdt:
$LD_LIBRARY_PATH
% mpirun -np 4 <pkgs>/pyMPI-2.5b0-TAU/bin/pyMPI
./wrapper.py                     (Instrumented pyMPI with wrapper.py)
```

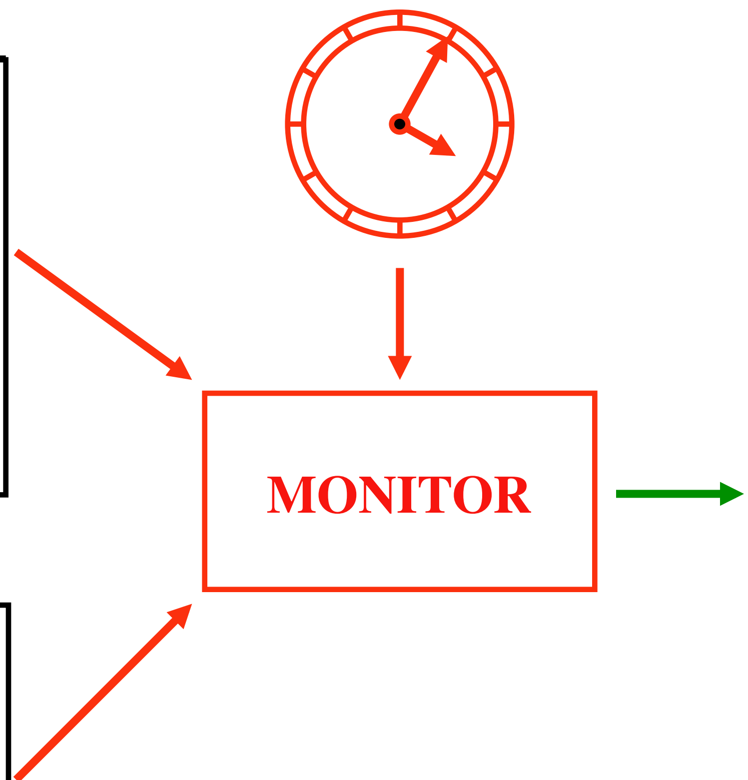
Tracing Measurement

Process A:

```
void master {
  trace(ENTER, 1);
  ...
  trace(SEND, B);
  send(B, tag, buf);
  ...
  trace(EXIT, 1);
}
```

Process B:

```
void worker {

 trace(ENTER, 2);
 ...
 recv(A, tag, buf);

 trace(RECV, A);
 ...

 trace(EXIT, 2);
}
```



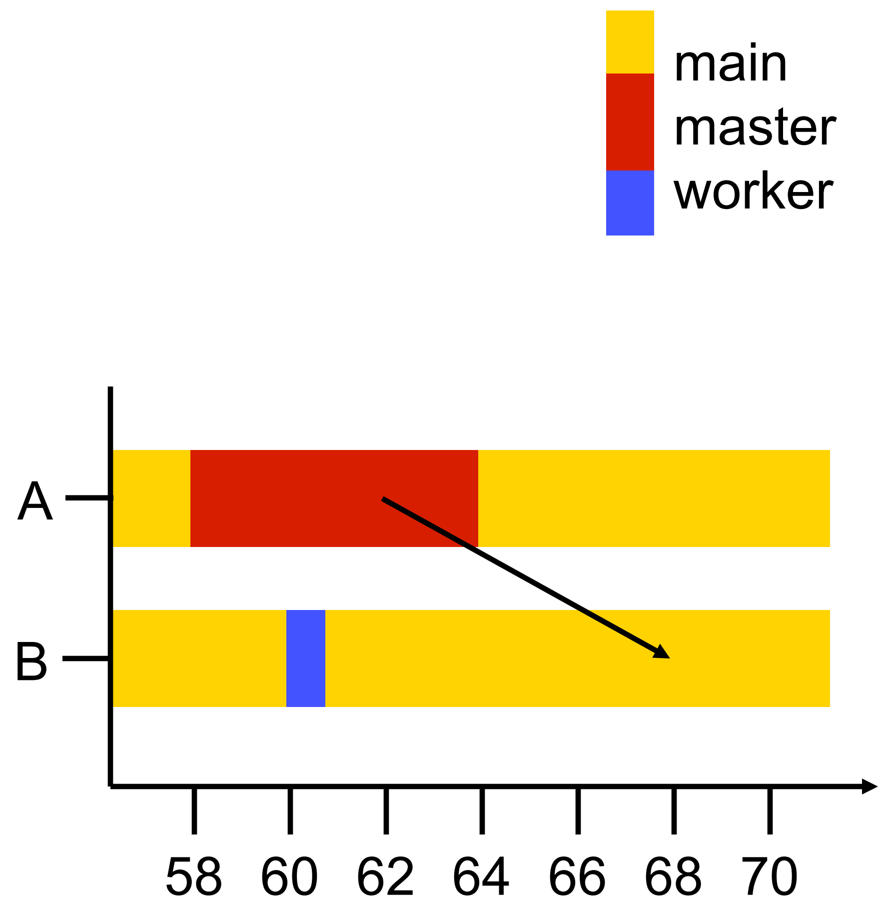
1	master
2	worker
3	...

...			
58	A	ENTER	1
60	B	ENTER	2
62	A	SEND	B
64	A	EXIT	1
68	B	RECV	A
69	B	EXIT	2
...			

Tracing Analysis and Visualization

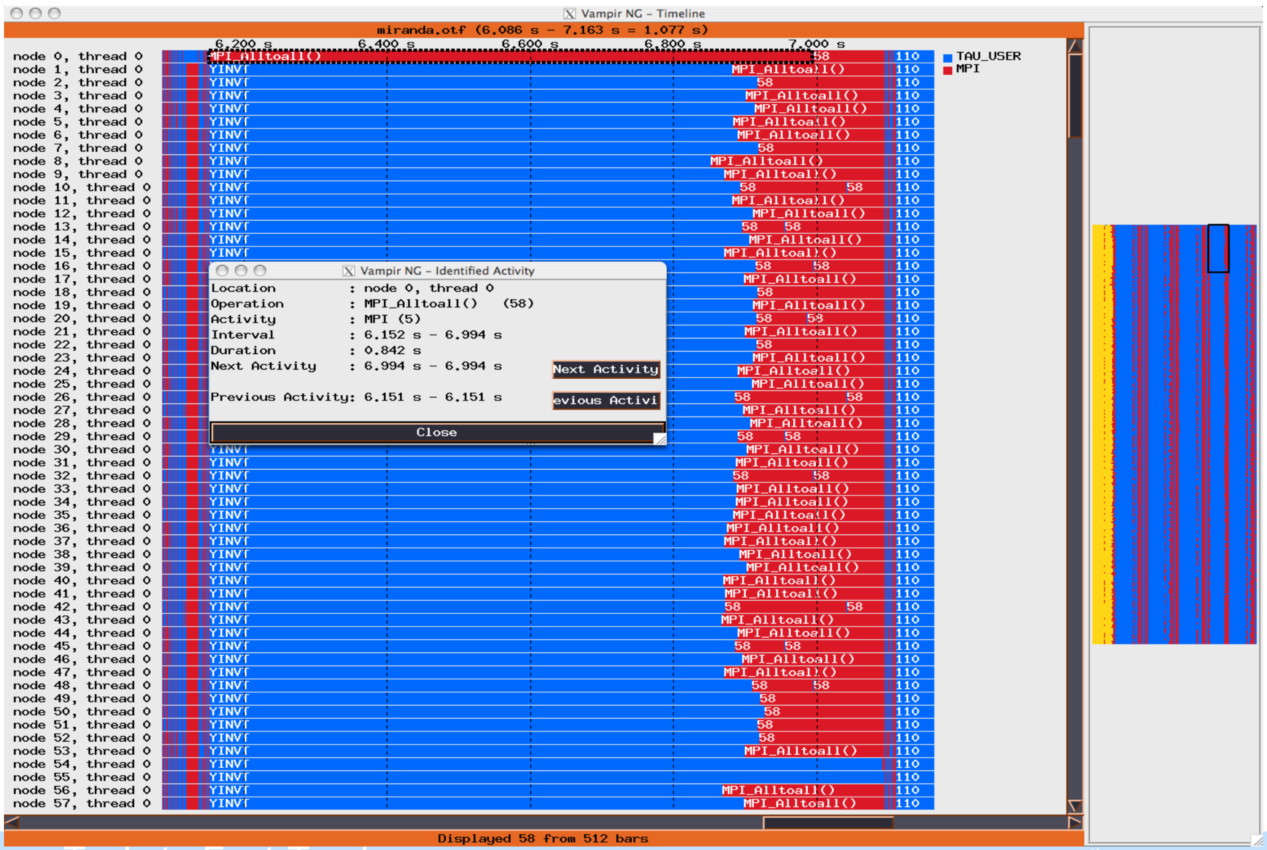
1	master
2	worker
3	...

...			
58	A	ENTER	1
60	B	ENTER	2
62	A	SEND	B
64	A	EXIT	1
68	B	RECV	A
69	B	EXIT	2
...			

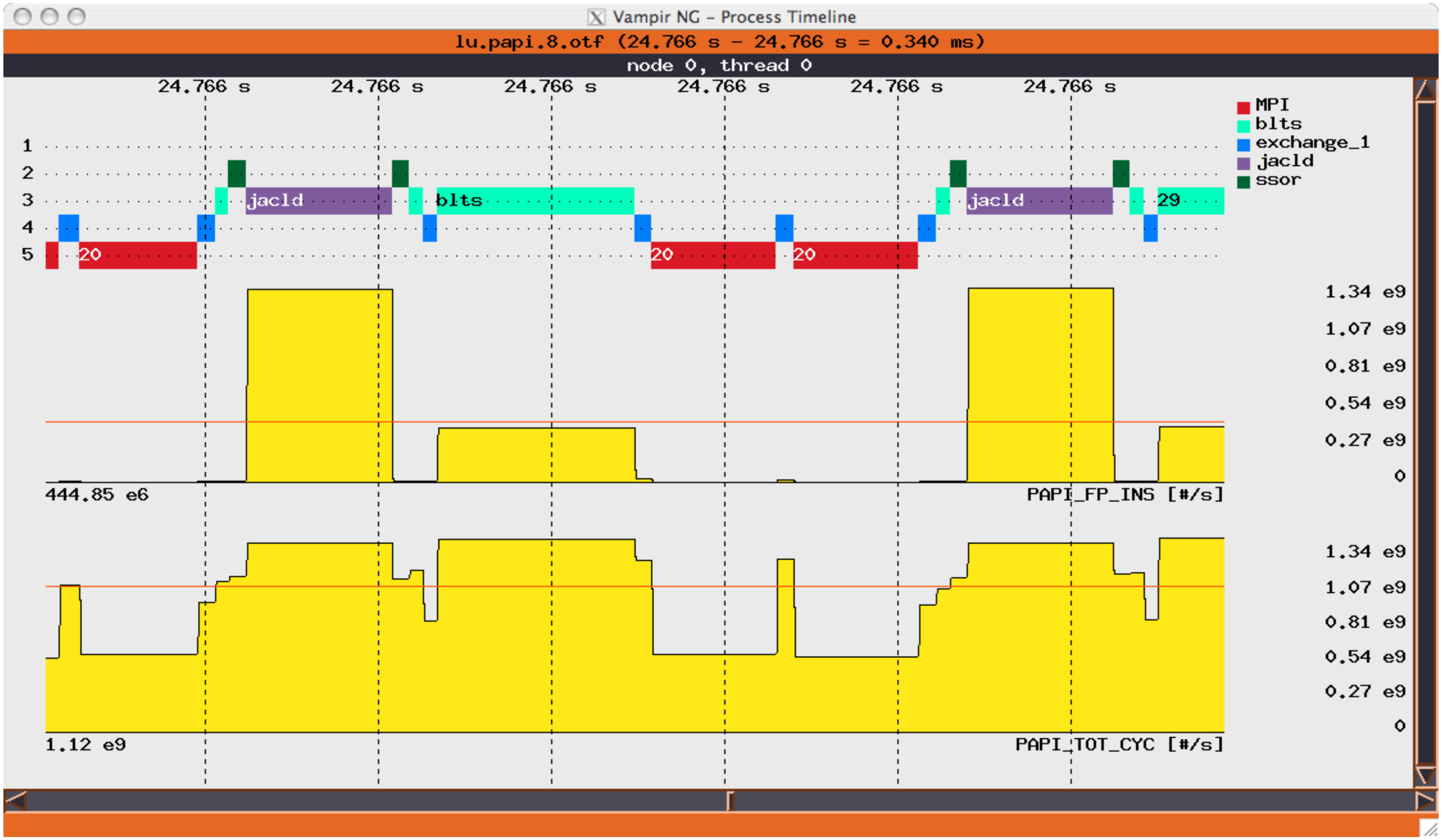


Usage Scenarios: Generating a Trace

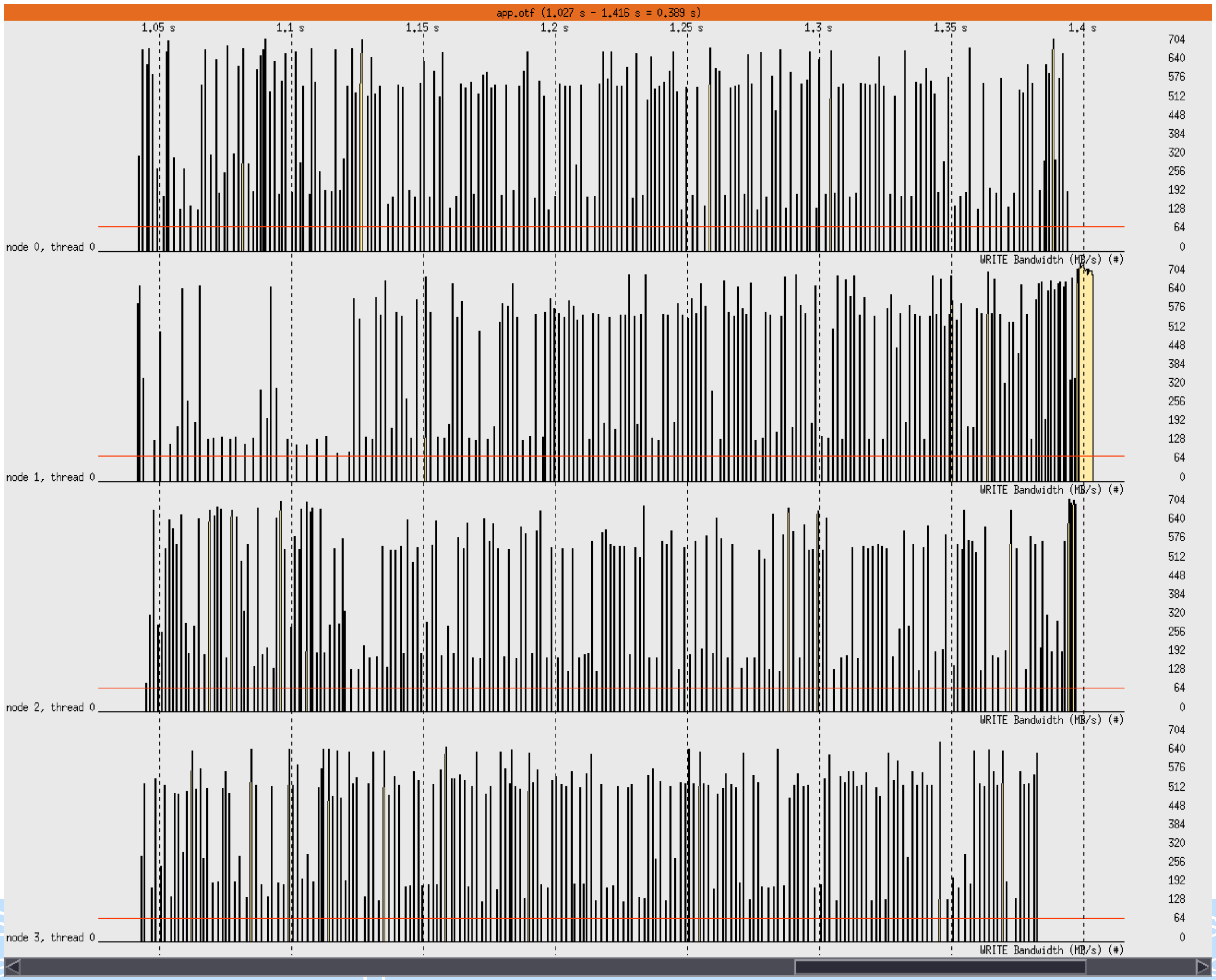
- Goal: Identify the temporal aspect of performance. What happens in my code at a given time? When?
- Event trace visualized in Vampir/Jumpshot



VNG Process Timeline with PAPI



Vampir Counter Timeline Showing I/O BW

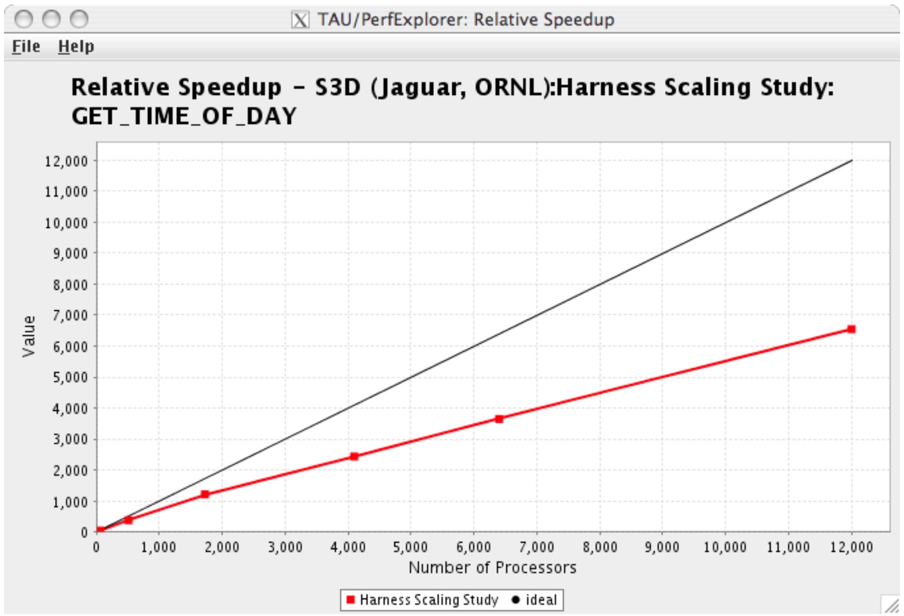


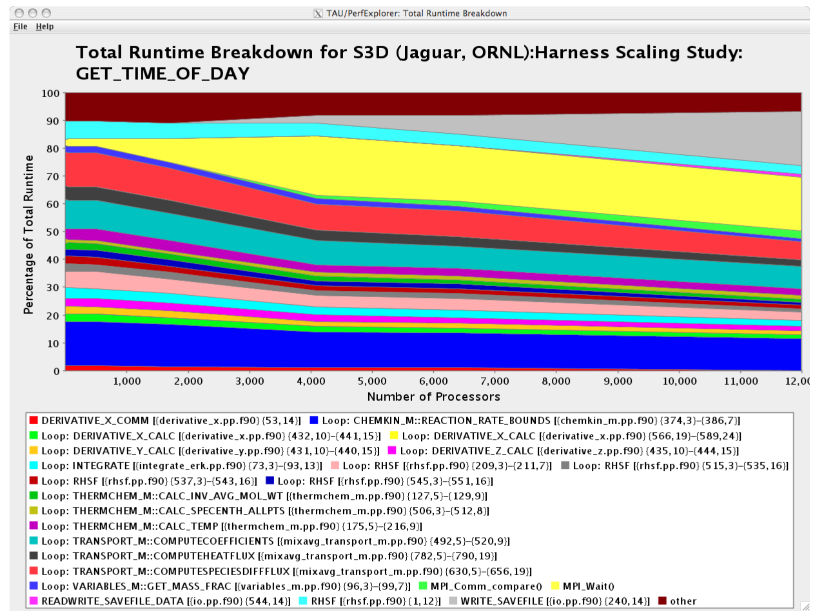
Generate a Trace File

```
% setenv TAU_MAKEFILE /usr/local/packages/tau/i386_linux/
lib/Makefile.tau-mpi-pdt

% export TAU_TRACE=1
% set path=(/usr/local/packages/tau/i386_linux/bin $path)
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)
% mpirun -np 4  ./a.out
% tau_treemerge.pl
(merges binary traces to create tau.trc and tau.edf files)
JUMPSHOT:
% tau2slog2 tau.trc tau.edf -o app.slog2
% jumpshot app.slog2
   OR
VAMPIR:
% tau2otf tau.trc tau.edf app.otf -n 4 -z
(4 streams, compressed output trace)
% vampir app.otf
```

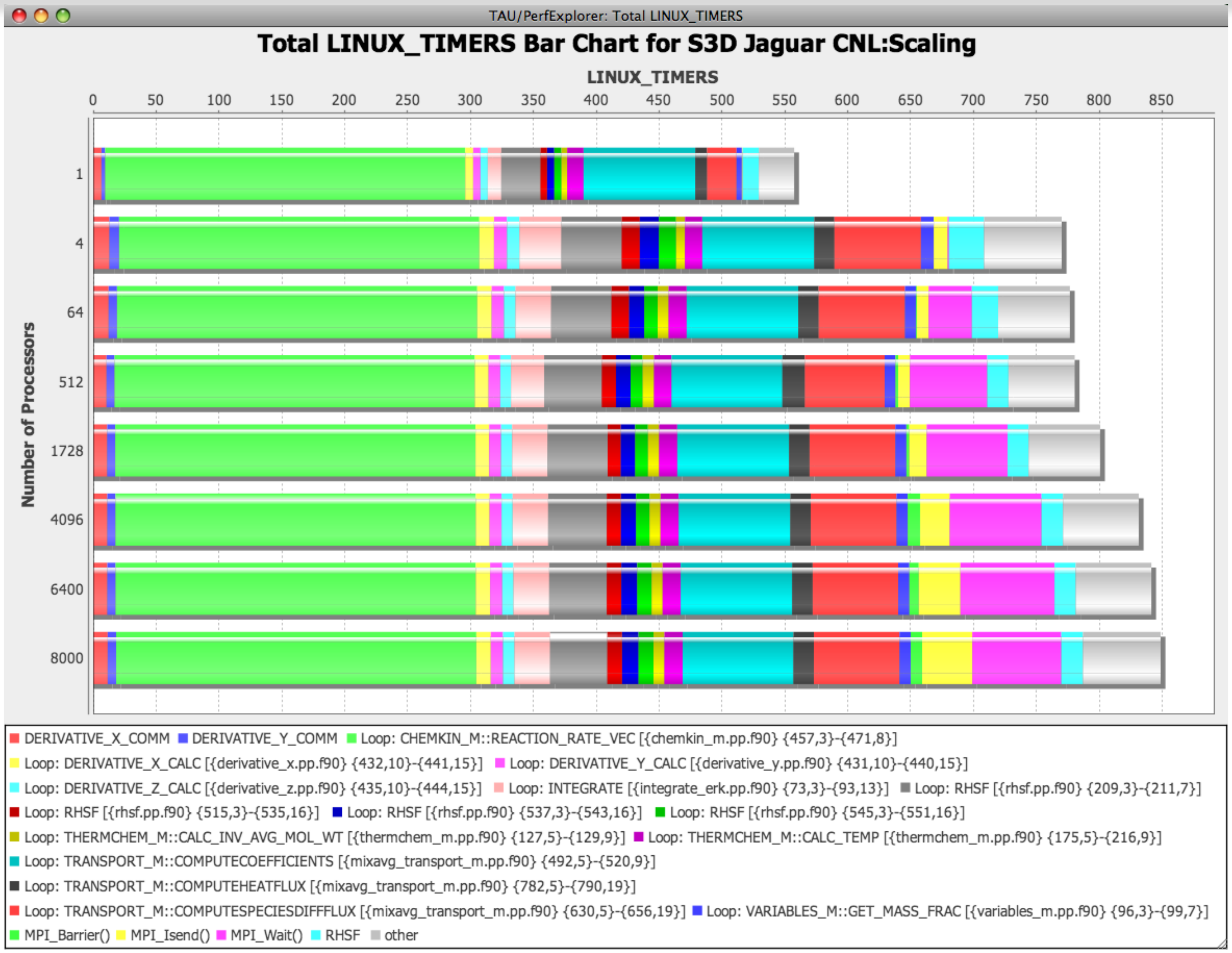
Usage Scenarios: Evaluate Scalability

- Goal: How does my application scale? What bottlenecks occur at what core counts?
- Load profiles in PerfDMF database and examine with PerfExplorer

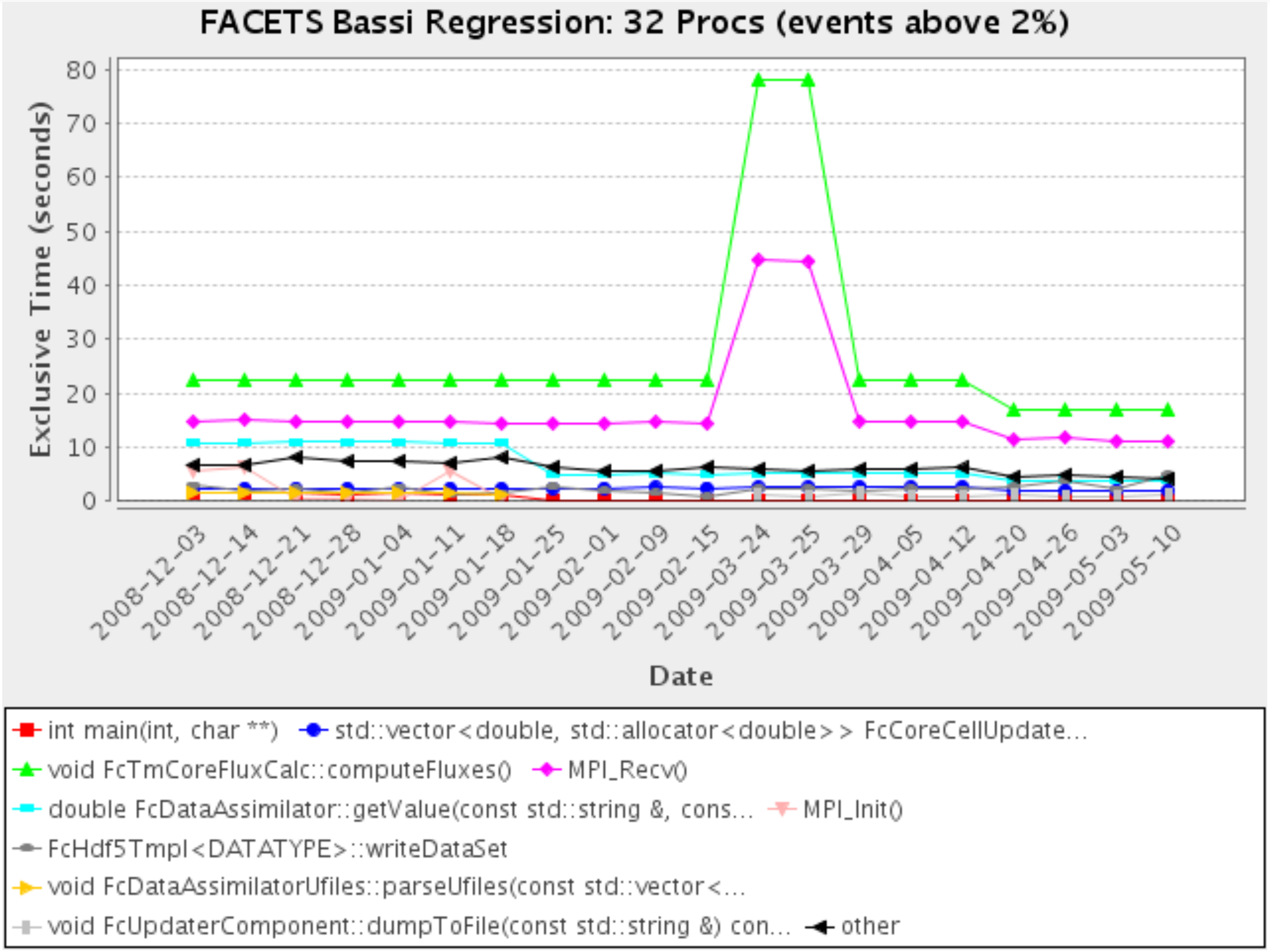




Usage Scenarios: Evaluate Scalability



Performance Regression Testing



ACTS COLLECTION

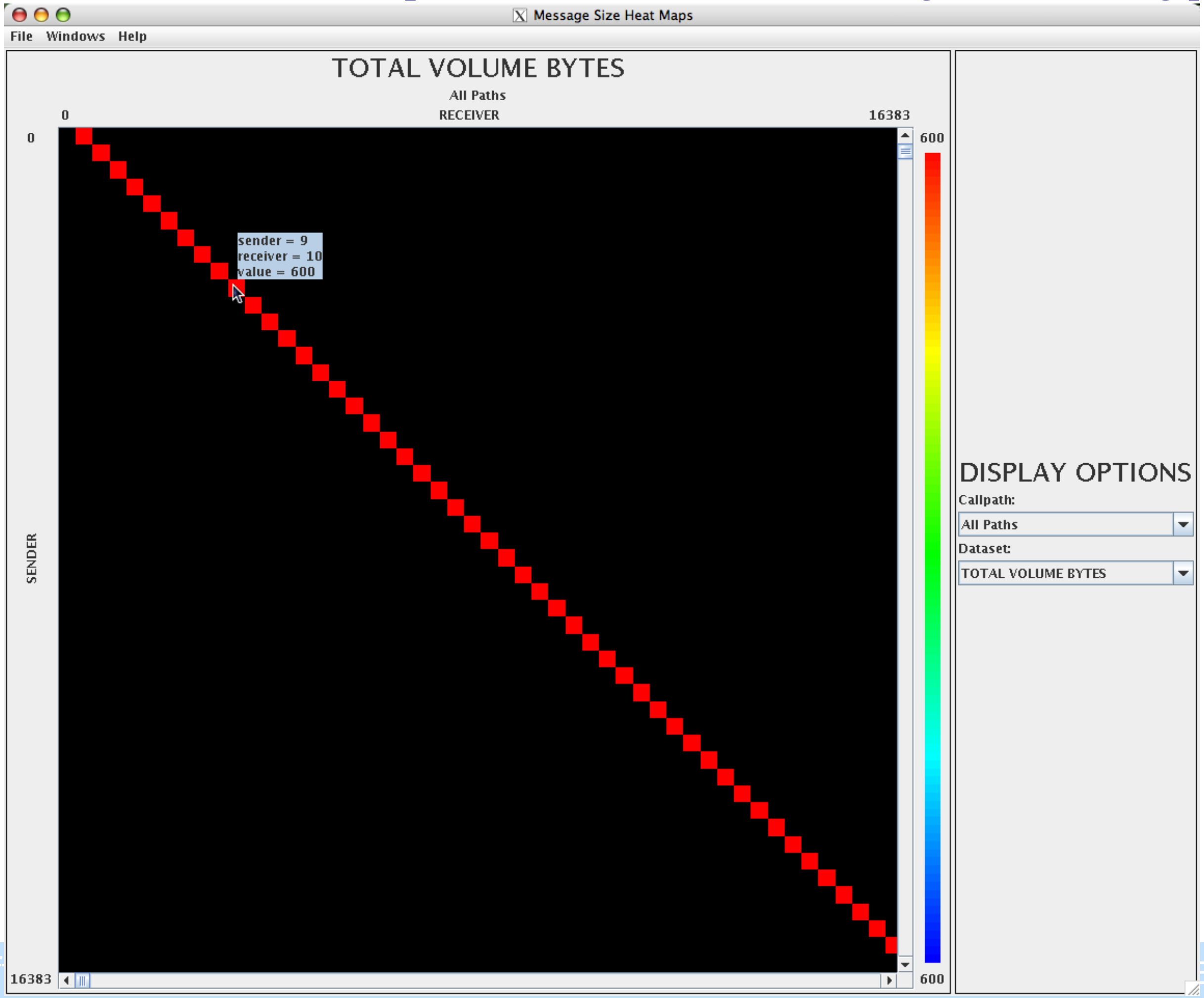
UNIVERSITY OF OREGON

Evaluate Scalability using PerfExplorer Charts

```
% setenv TAU_MAKEFILE /usr/local/packages/tau/i386_linux
                          /lib/Makefile.tau-mpi-pdt
% set path=(/usr/local/packages/tau/i386_linux/bin $path)
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)
% mpirun -np 1 ./a.out
% paraprof --pack 1p.ppk
% mpirun -np 2 ./a.out …
% paraprof --pack 2p.ppk … and so on.
On your client:
% perfdmf_configure --create-default
(Chooses derby, blank user/passwd, yes to save passwd, defaults)
% perfexplorer_configure
(Yes to load schema, defaults)
% paraprof
(load each trial: DB -> Add Trial -> Type (Paraprof Packed Profile) ->
   OK) OR use perfdmf_loadtrial.
% perfdmf_loadtrial -a “NWChem” -x “Scaling on i386_linux” -n “32p”
   32p.ppk
Then,
% perfexplorer
(Select experiment, Menu: Charts -> Speedup)
```

Communication Matrix Display

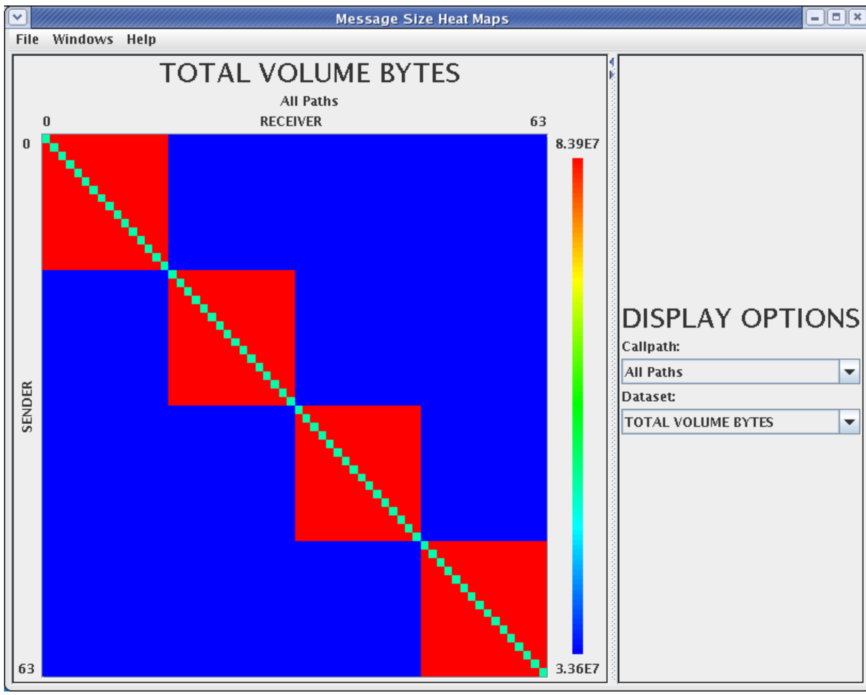
- Goal: What is the volume of inter-process communication? Along which calling path?

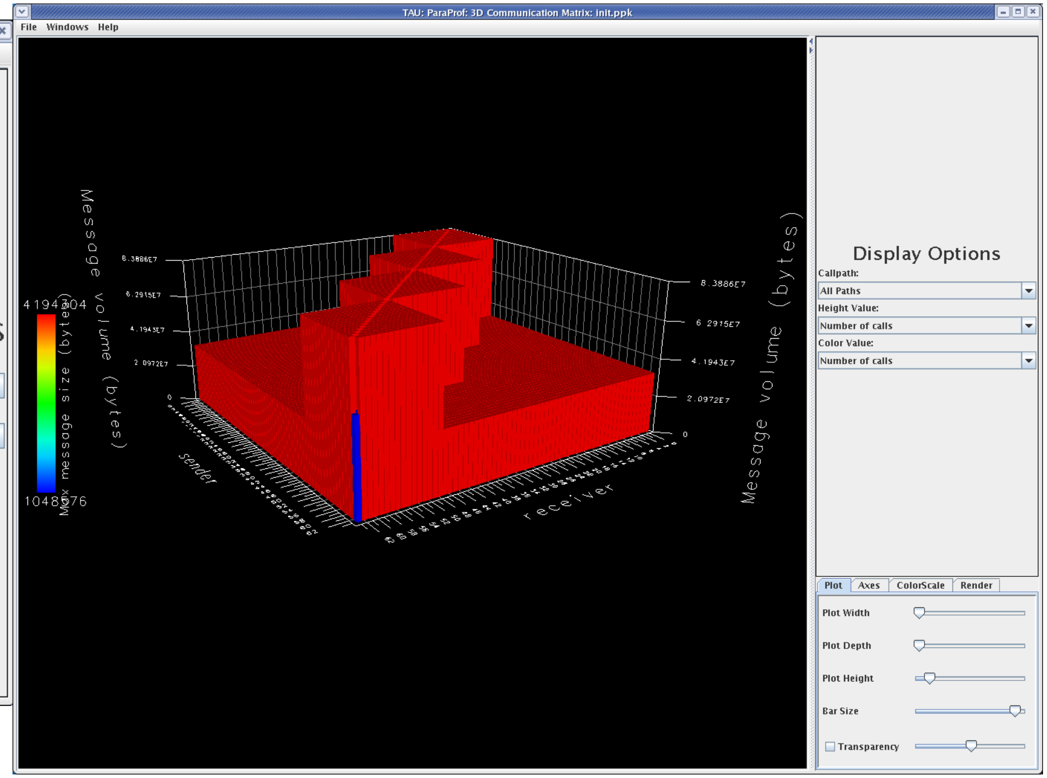


Communication Matrix

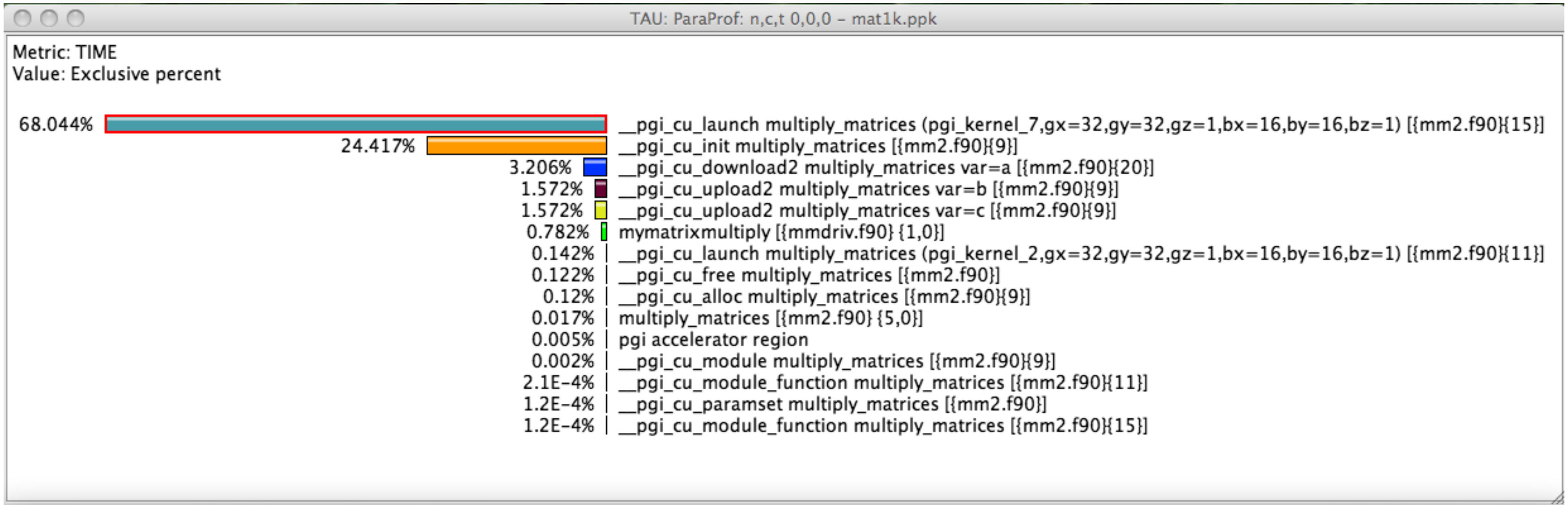
```
% setenv TAU_MAKEFILE /usr/local/packages/tau/i386_linux
                      /lib/Makefile.tau-mpi-pdt
% set path=(/usr/local/packages/tau/i386_linux/bin $path)
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)
% export TAU_COMM_MATRIX=1

% mpirun -np 4 ./a.out (setting the environment variables)

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

ParaProf: Communication Matrix Display





Measuring Performance of PGI GPGPU Accelerated Code

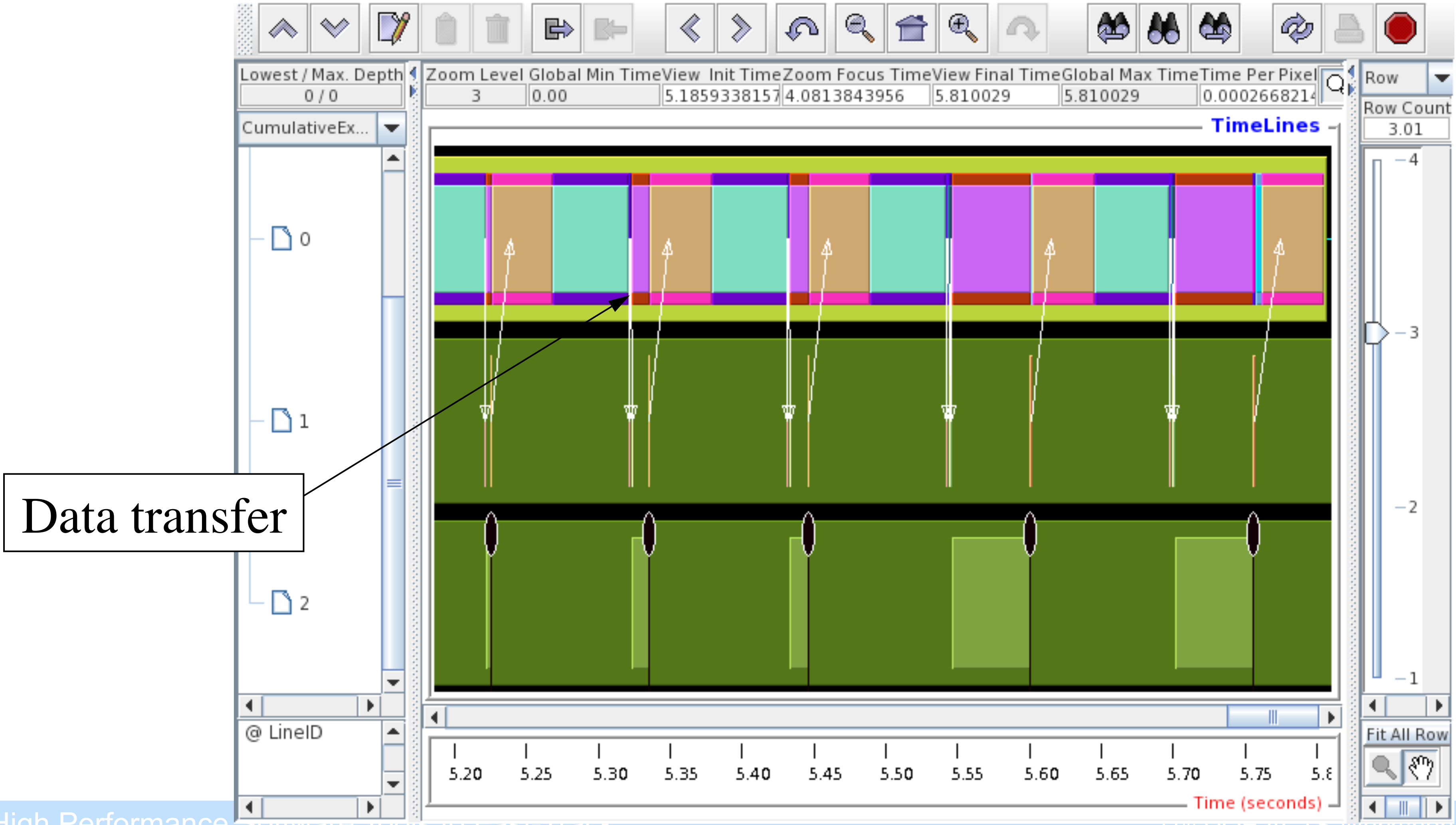


TAU: ParaProf: Thread Statistics: n,c,t, 0,0,0 – mat1k.ppk

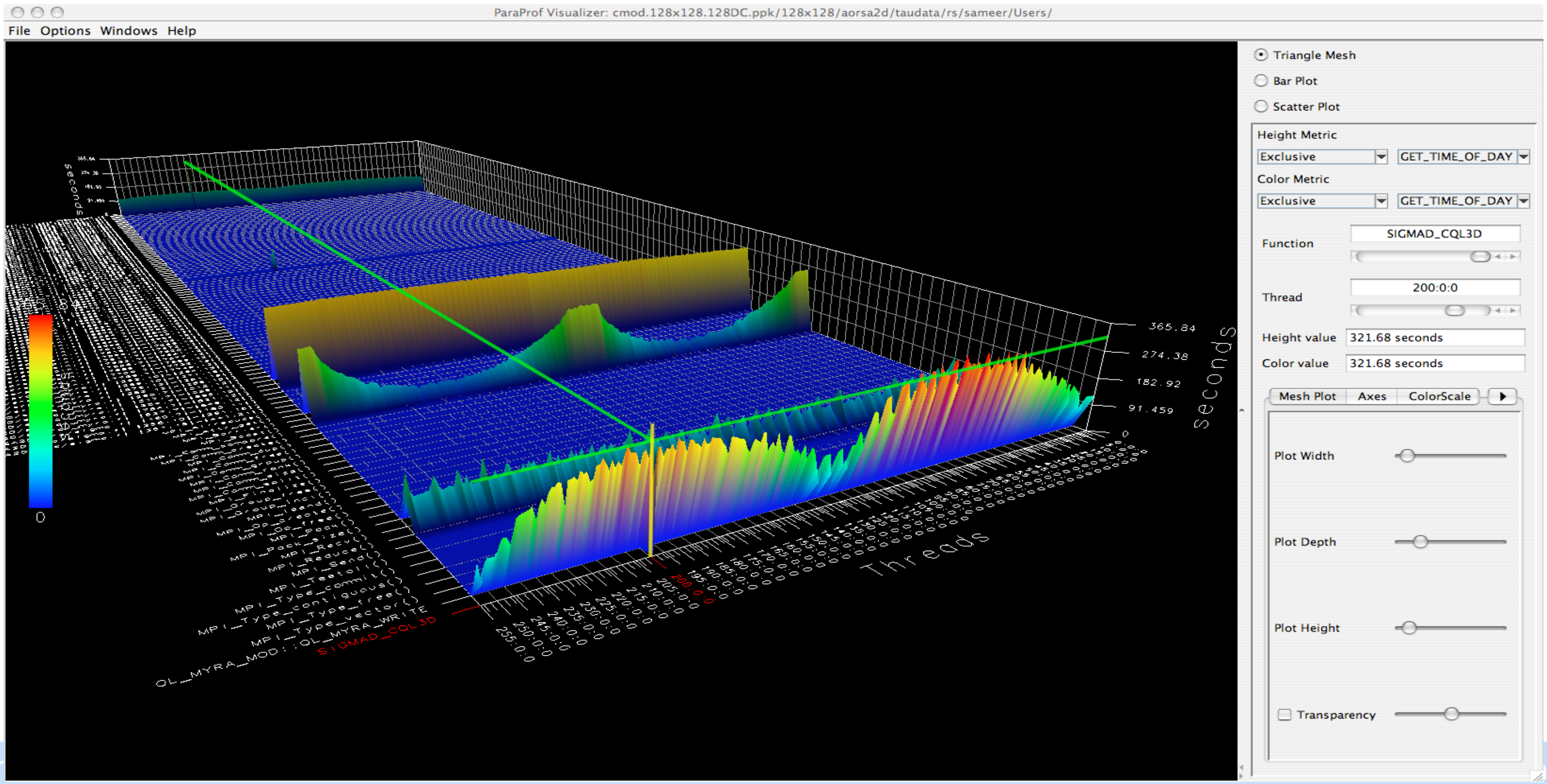
Name	Exclusive TIME ▽	Inclusive TIME	Calls	Child Calls
__pgi_cu_launch multiply_matrices (pgi_kernel_7,gx=32,gy=32,gz=1,bx=16,by=16,bz=1) [{mm2.f90}{15}]	10.901	10.901	5	0
__pgi_cu_init multiply_matrices [{mm2.f90}{9}]	3.912	3.912	5	0
__pgi_cu_download2 multiply_matrices var=a [{mm2.f90}{20}]	0.514	0.514	5	0
__pgi_cu_upload2 multiply_matrices var=b [{mm2.f90}{9}]	0.252	0.252	5	0
__pgi_cu_upload2 multiply_matrices var=c [{mm2.f90}{9}]	0.252	0.252	5	0
mymatrixmultiply [{mmdriv.f90} {1,0}]	0.125	16.021	1	1
__pgi_cu_launch multiply_matrices (pgi_kernel_2,gx=32,gy=32,gz=1,bx=16,by=16,bz=1) [{mm2.f90}{11}]	0.023	0.023	5	0
__pgi_cu_free multiply_matrices [{mm2.f90}]	0.02	0.02	15	0
__pgi_cu_alloc multiply_matrices [{mm2.f90}{9}]	0.019	0.019	15	0
multiply_matrices [{mm2.f90} {5,0}]	0.003	15.895	5	9
pgi accelerator region	0.001	15.893	5	85
__pgi_cu_module multiply_matrices [{mm2.f90}{9}]	0	0	5	0
__pgi_cu_module_function multiply_matrices [{mm2.f90}{11}]	0	0	5	0
__pgi_cu_paramset multiply_matrices [{mm2.f90}]	0	0	10	0

Show Source Code
Show Function Bar Chart
Show Function Histogram
Assign Function Color
Reset to Default Color

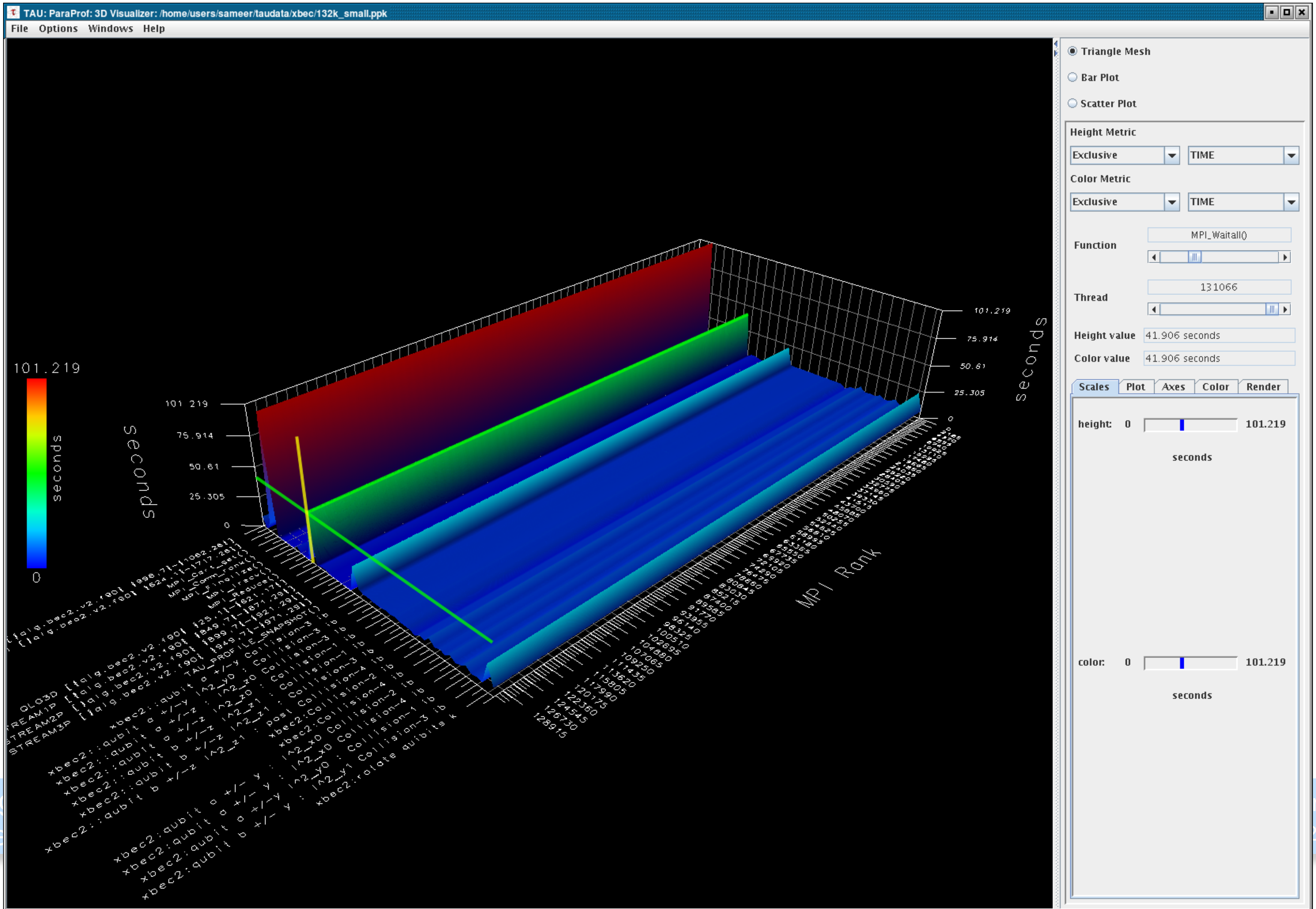
Scaling NAMD with CUDA (Jumpshot with TAU)



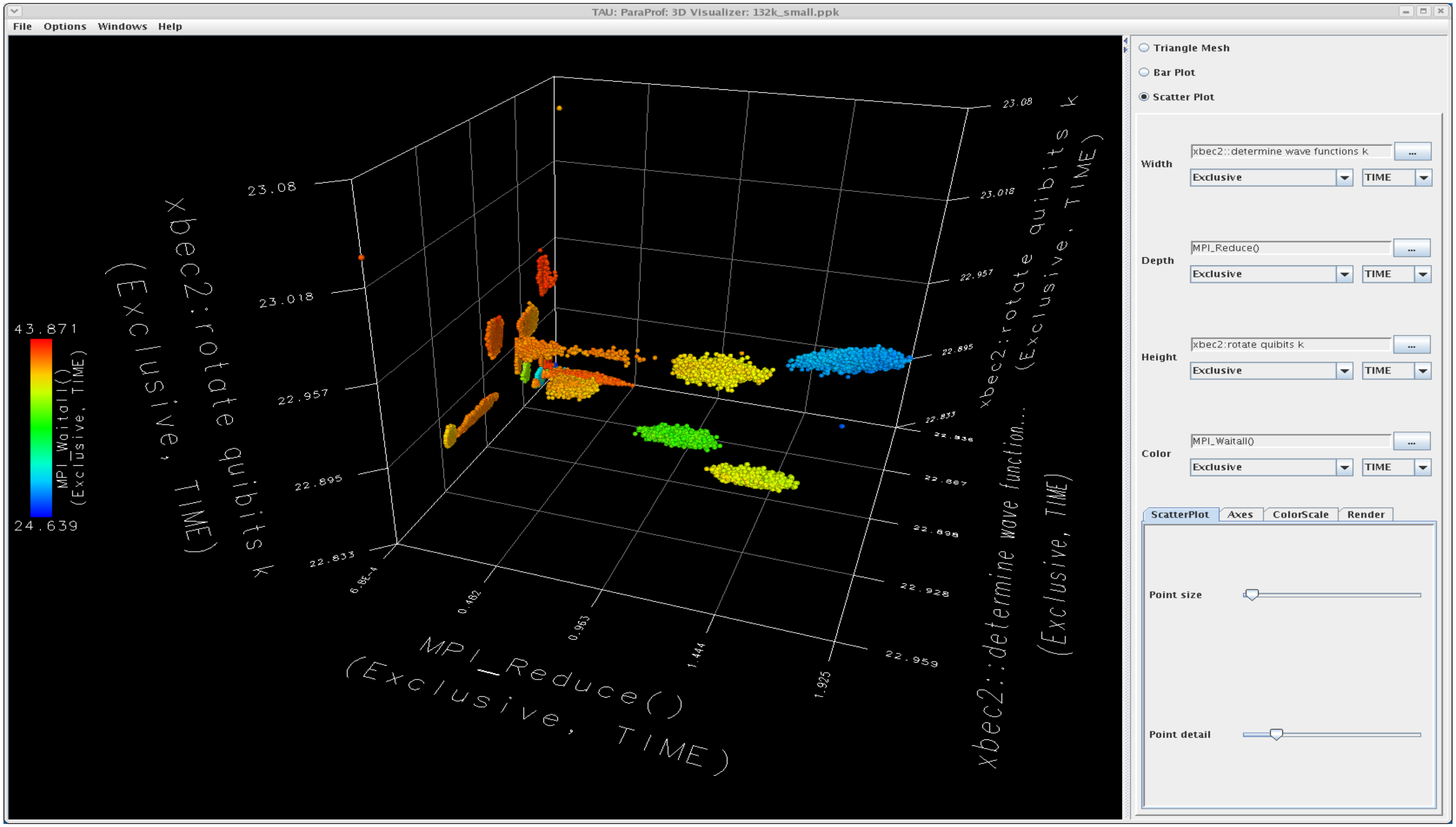
Parallel Profile Visualization: ParaProf



Scalable Visualization: ParaProf (128k cores)



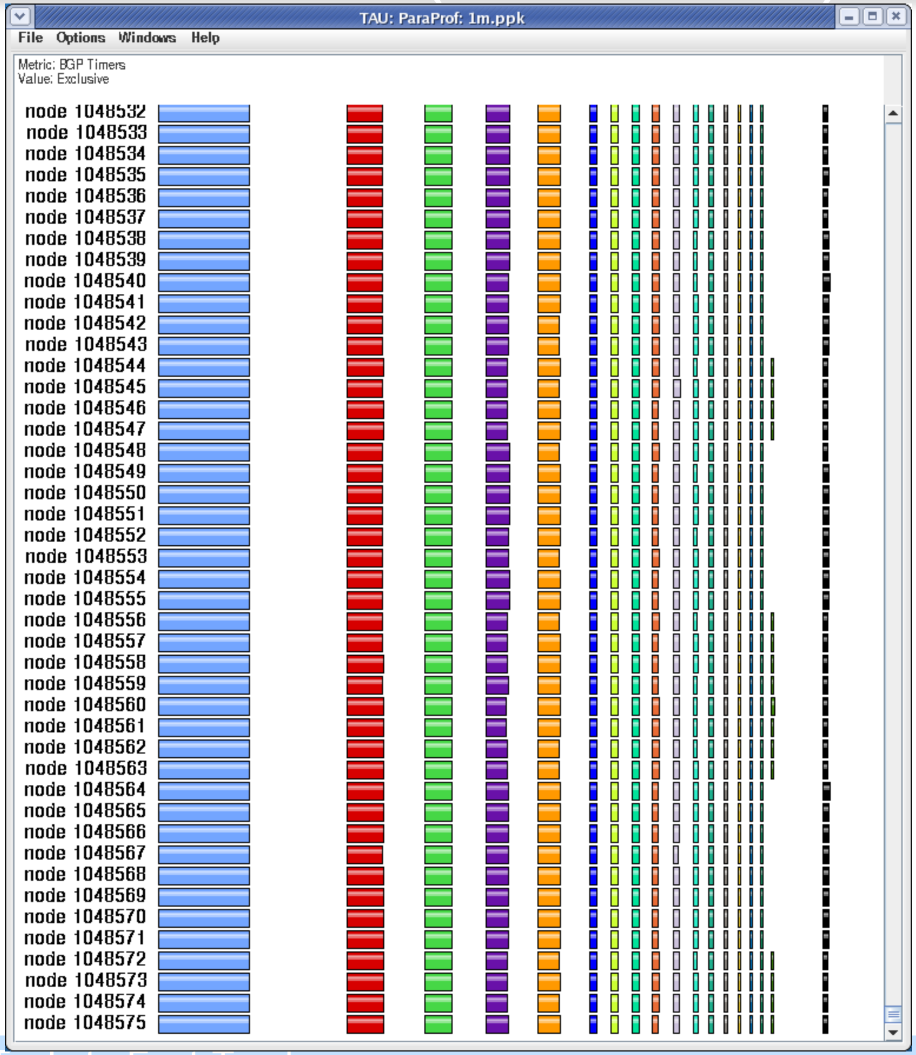
Scatter Plot: ParaProf (128k cores)



ACTS COLLECTION

UNIVERSITY OF OREGON

ParaProf (1m cores*)

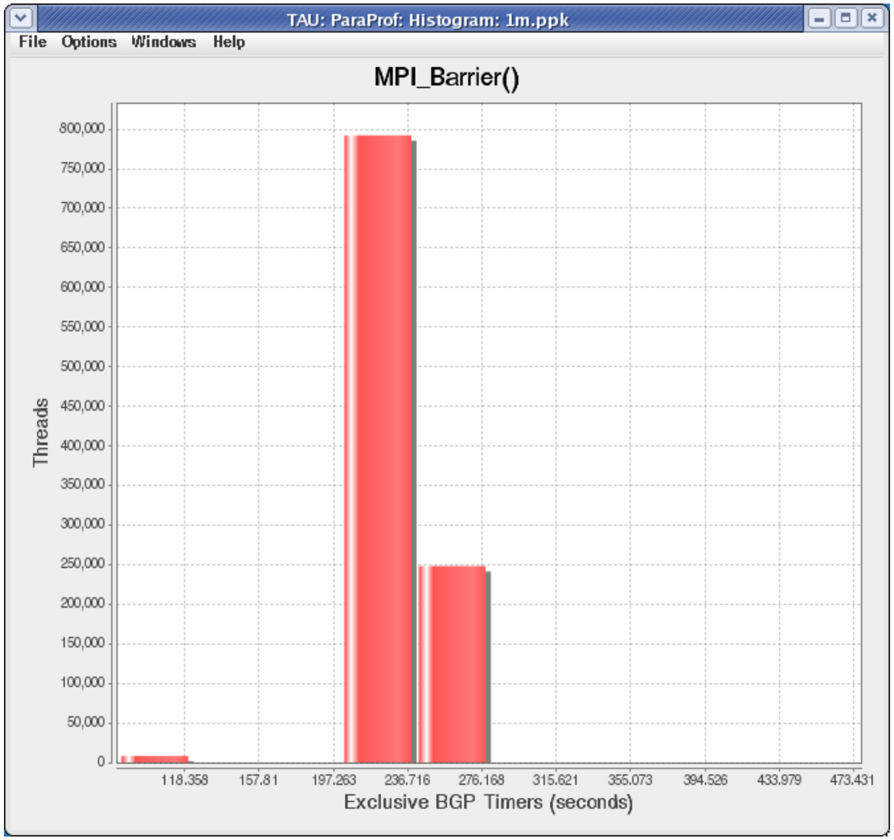


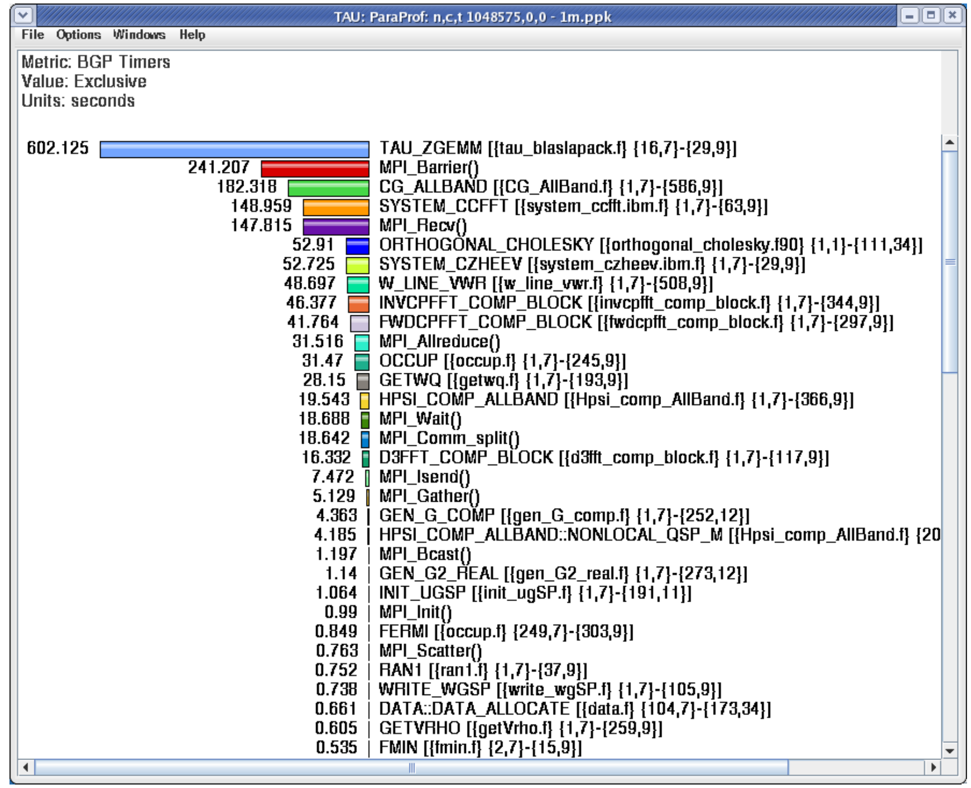
*1m core dataset generated by replicating a 32 k core dataset

ACTS COLLECTION

UNIVERSITY OF OREGON

Histogram: ParaProf (1m cores*)





***1m core dataset generated by replicating a 32 k core dataset**

Labs: LiveDVD

- Add one of

  source /usr/local/packages/etc/point.bashrc

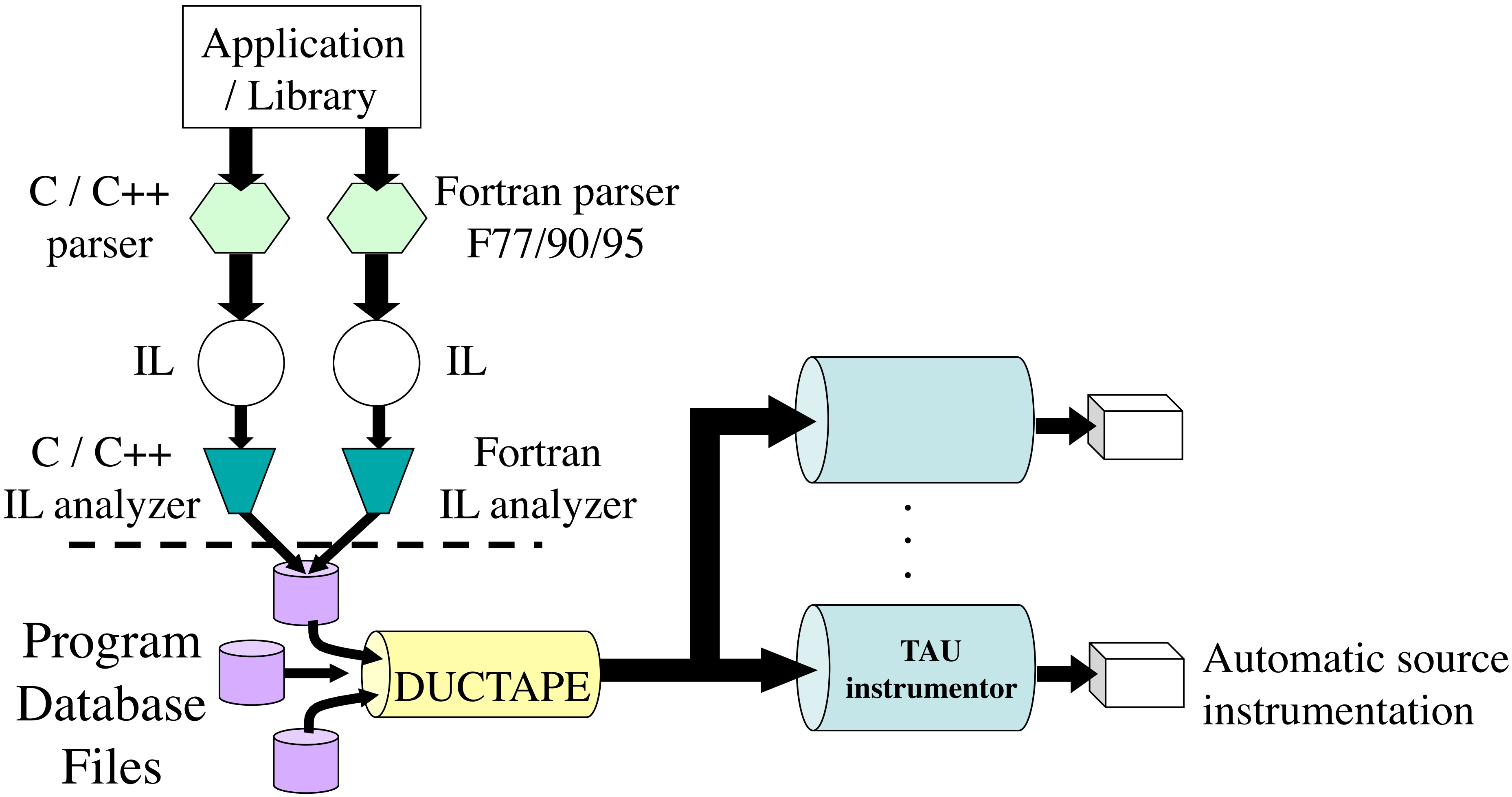
  or

  source /usr/local/packages/etc/point.cshrc

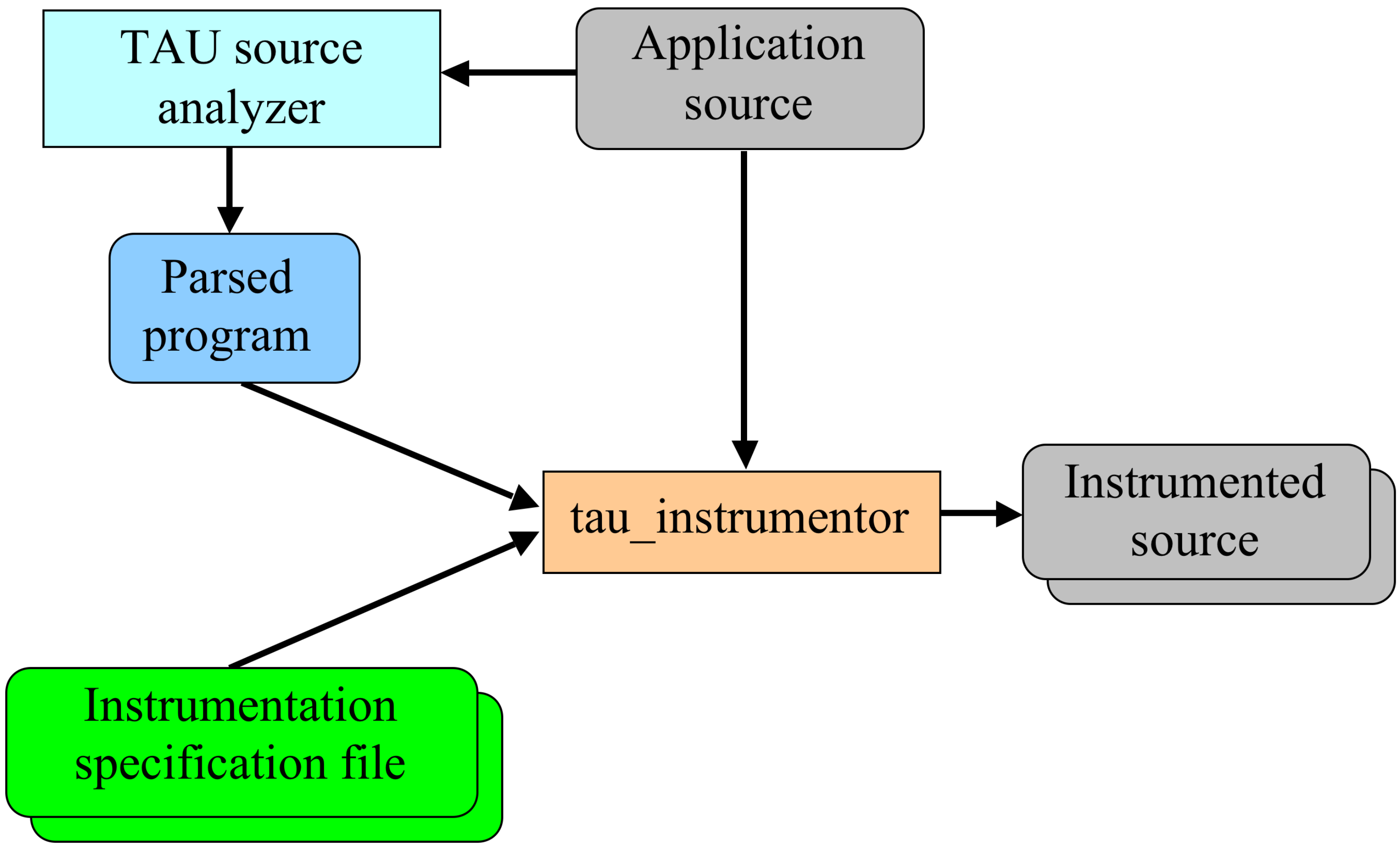
  to the end of your **.login** file (for bash or csh/tcsh users respectively).

  On the LiveDVD, please see the ~/point-workshop directory

Program Database Toolkit (PDT)



Automatic Source-level Instrumentation



Selective Instrumentation File

- Specify a list of events to exclude or include
- # is a wildcard in a routine name

```
BEGIN_EXCLUDE_LIST
Foo
Bar
D#EMM
END_EXCLUDE_LIST

BEGIN_INCLUDE_LIST
int main(int, char **)
F1
F3
END_INCLUDE_LIST
```

Selective Instrumentation File

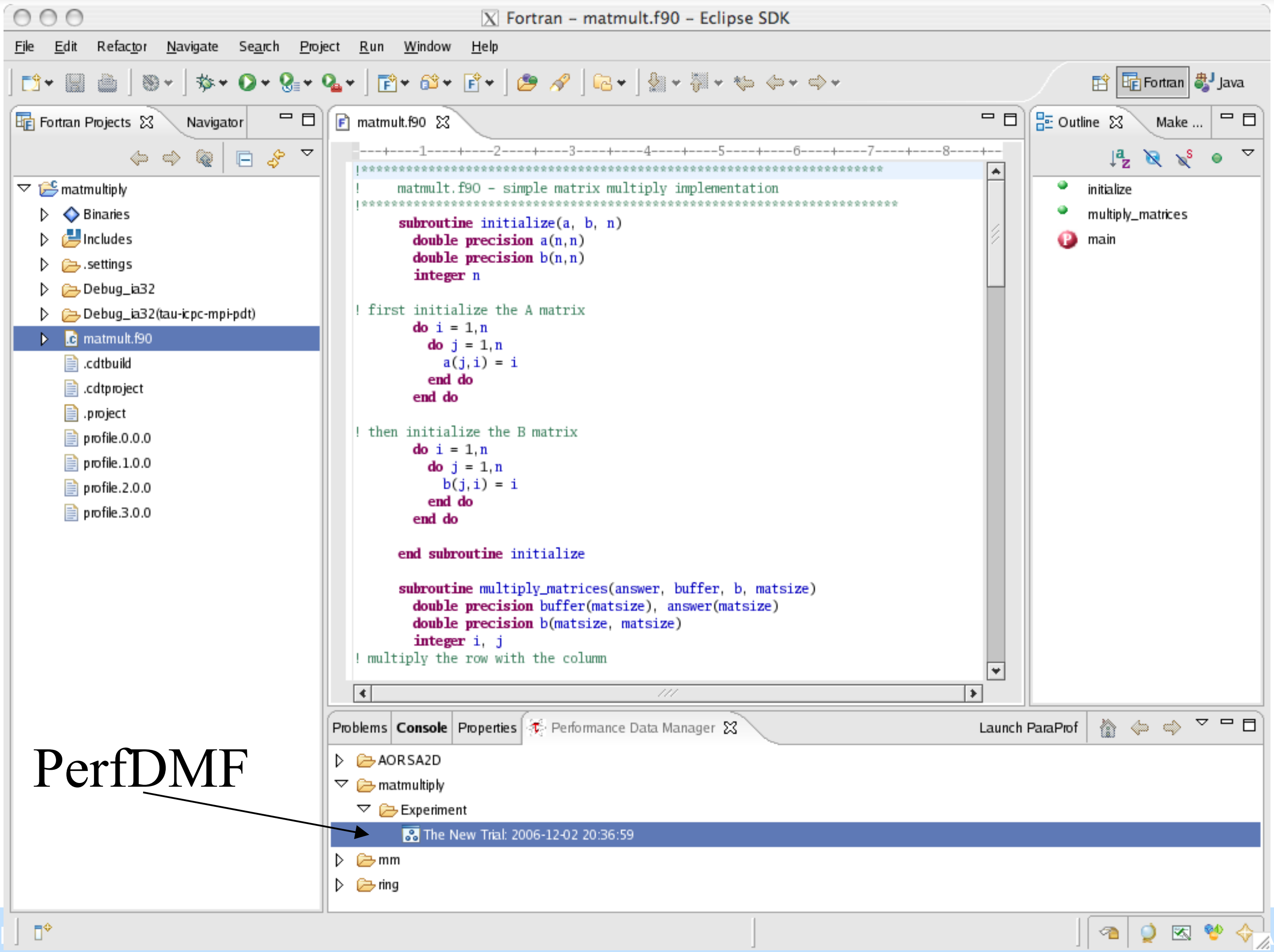
- Optionally specify a list of files
- * and ? may be used as wildcard characters

```
BEGIN_FILE_EXCLUDE_LIST
f*.f90
Foo?.cpp
END_FILE_EXCLUDE_LIST
BEGIN_FILE_INCLUDE_LIST
main.cpp
foo.f90
END_FILE_INCLUDE_LIST
```

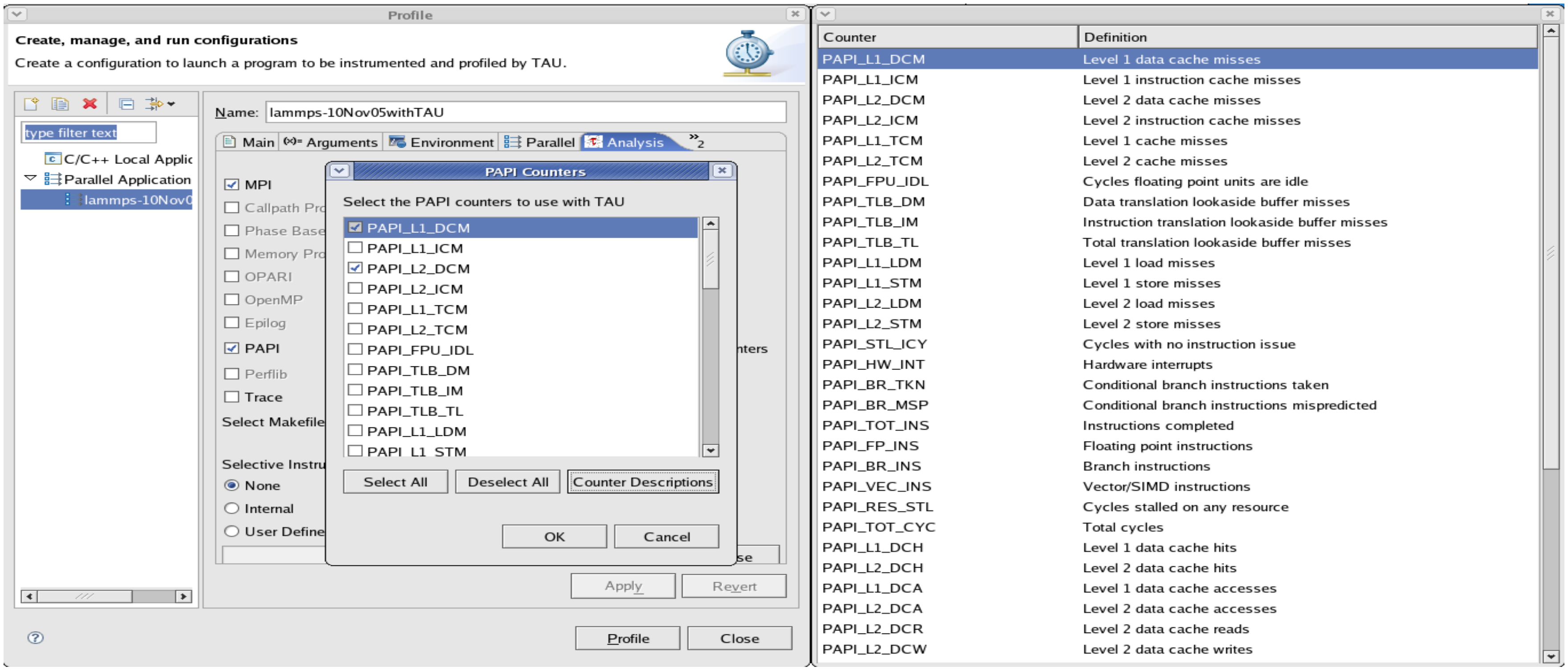
TAU Integration with IDEs

- High performance software development environments
  - Tools may be complicated to use
  - Interfaces and mechanisms differ between platforms / OS
- Integrated development environments
  - Consistent development environment
  - Numerous enhancements to development process
  - Standard in industrial software development
- Integrated performance analysis
  - Tools limited to single platform or programming language
  - Rarely compatible with 3rd party analysis tools
  - Little or no support for parallel projects

TAU and Eclipse



Choosing PAPI Counters with TAU in Eclipse

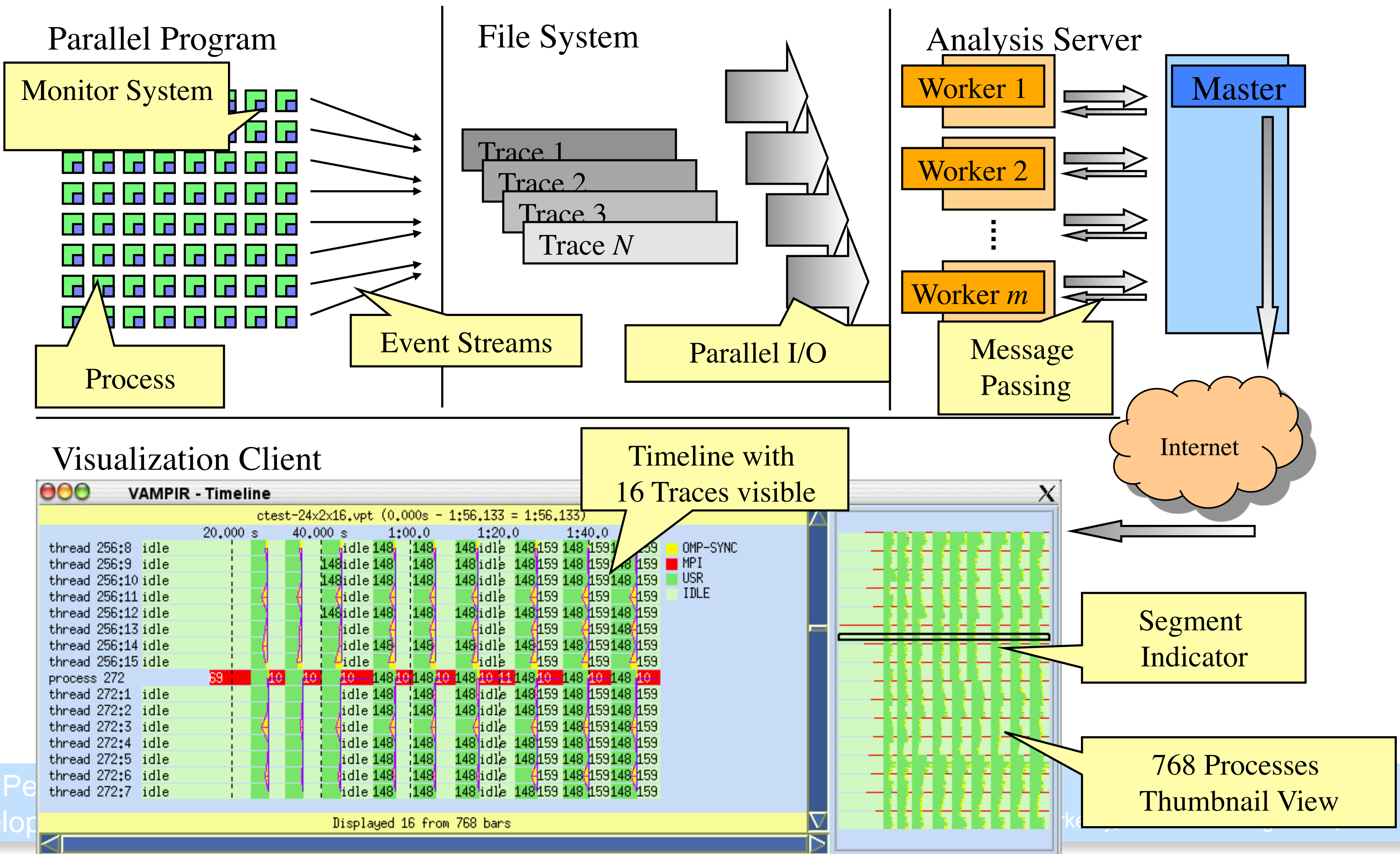



```
% /usr/local/packages/eclipse/eclipse
```

VampirTrace and Vampir

- Introduction
- Event Trace Visualization
- Vampir & VampirServer
- The Vampir Displays
  - Timeline
  - Process Timeline with Performance Counters
  - Summary Display
  - Message Statistics
- VampirTrace
  - Instrumentation & Run-Time Measurement
- Conclusions

VampirServer Architecture

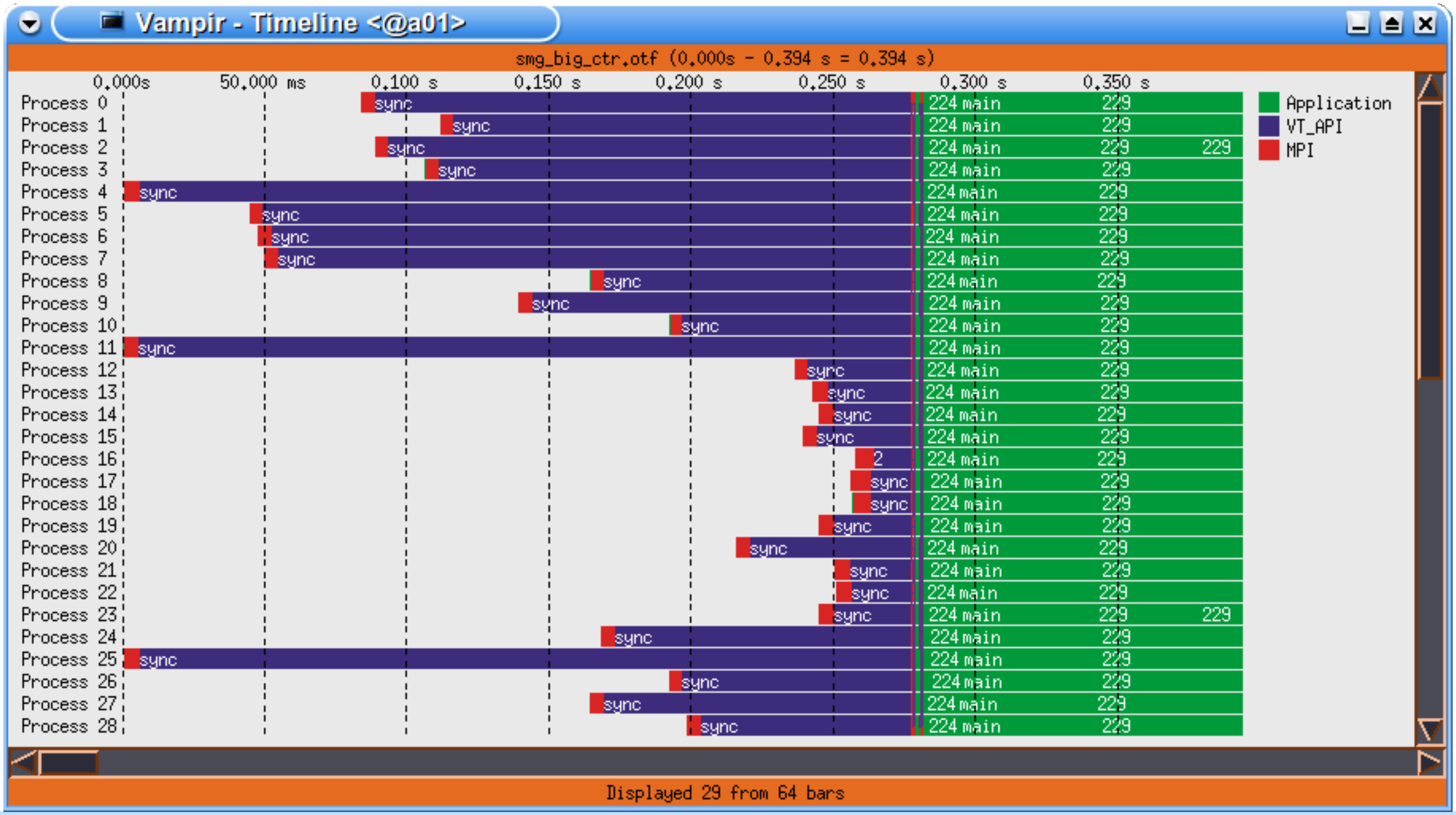


Vampir Displays

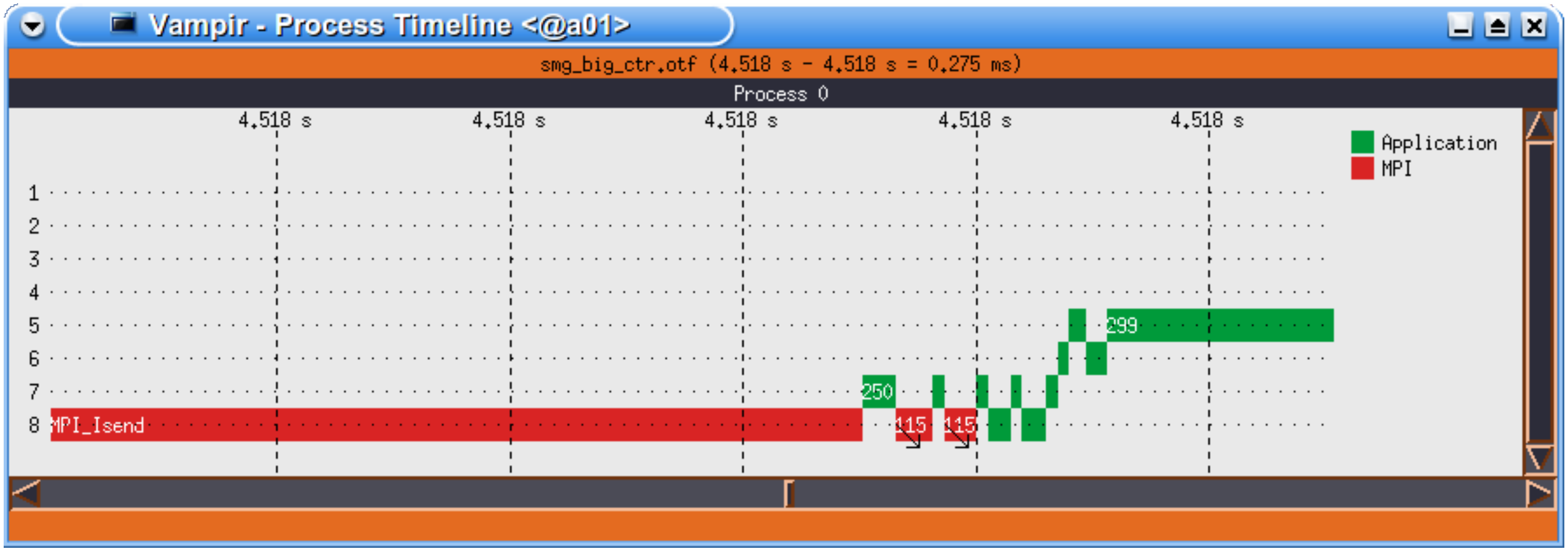
The main displays of Vampir:

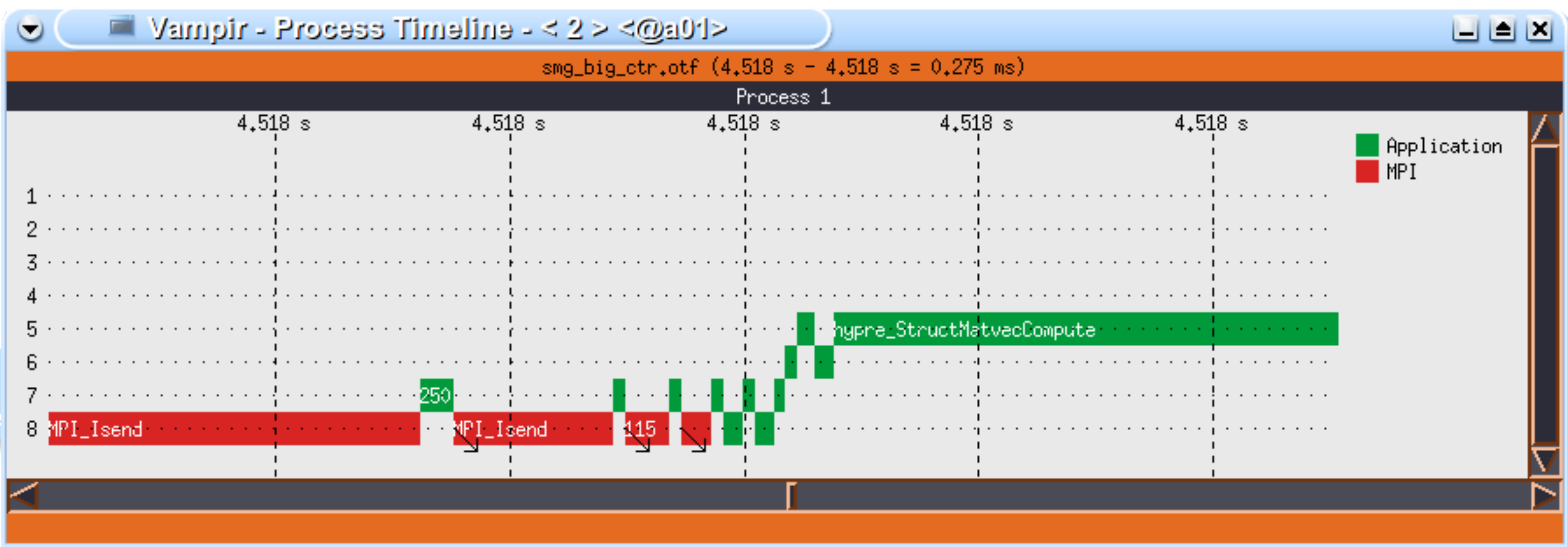
- Global Timeline
- Process Timeline w/o Counters
- Statistic Summary
- Summary Timeline
- Message Statistics
- Collective Operation Statistics
- Counter Timeline
- Call Tree

Vampir Global Timeline Display

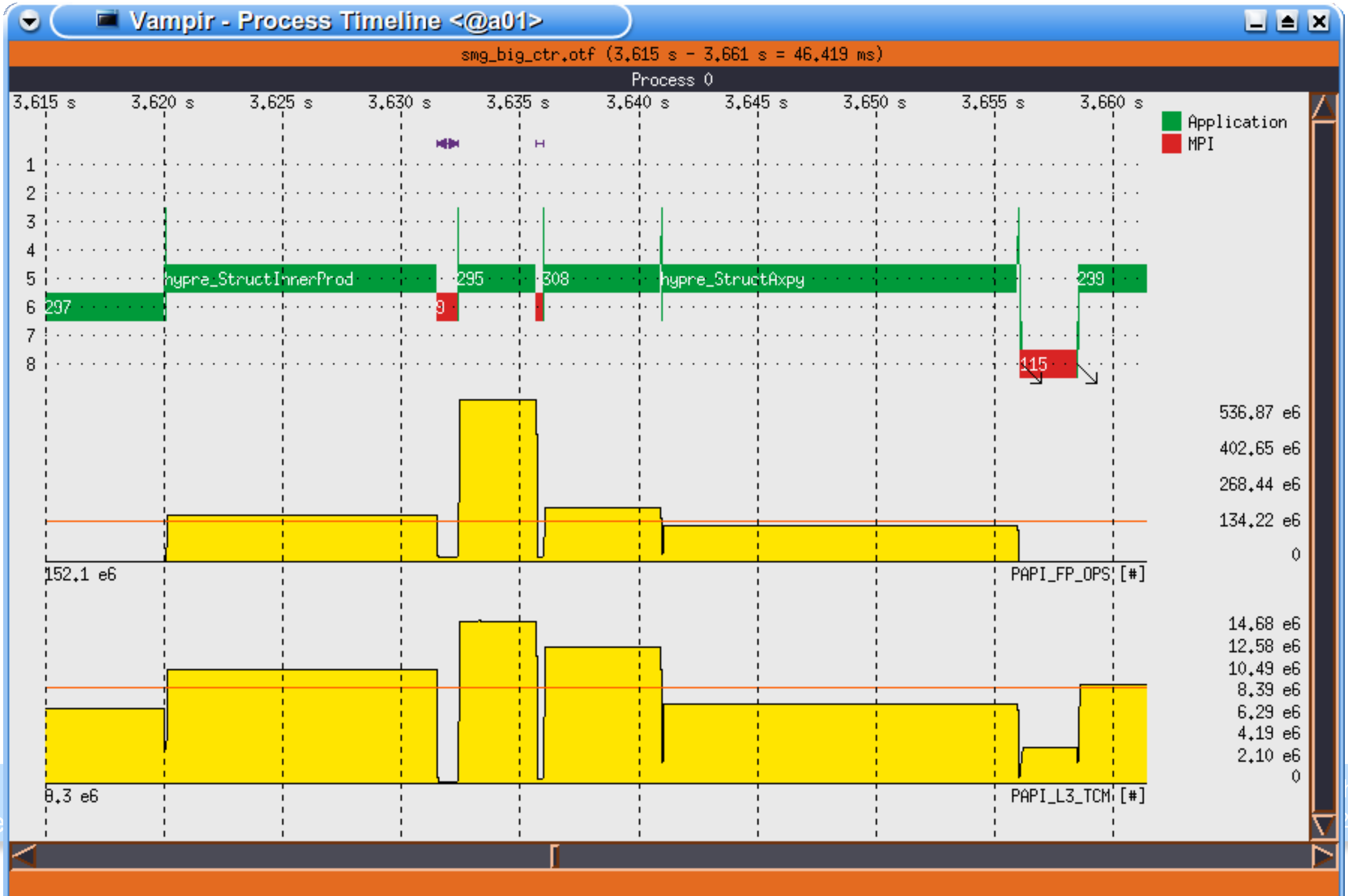


Process Timeline Display

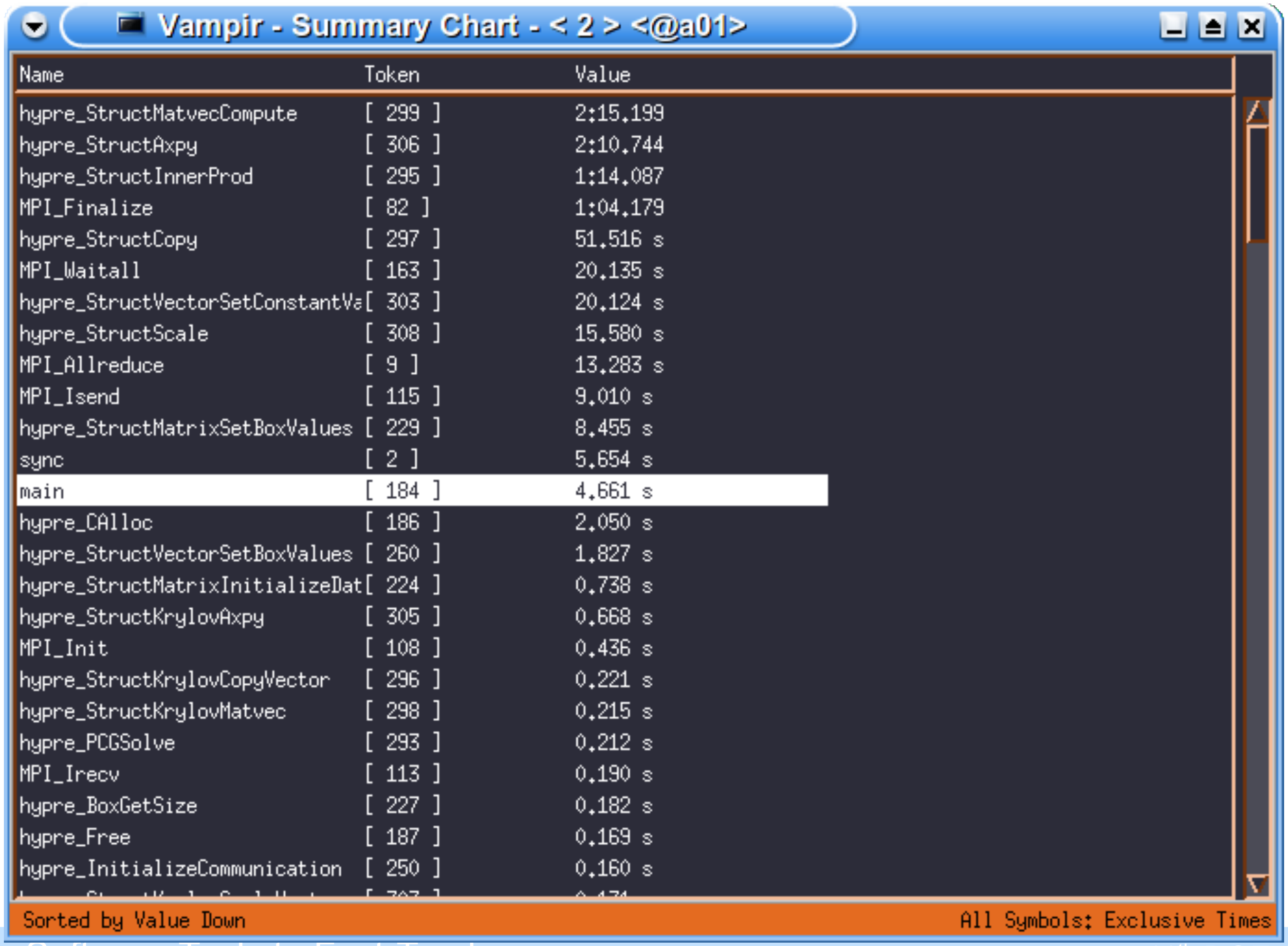




Process Timeline with Counters



Statistic Summary Display



Vampir - Summary Chart - < 2 > <@a01>

Name	Token	Value
hypre_StructMatvecCompute	[299]	2:15.199
hypre_StructAxpy	[306]	2:10.744
hypre_StructInnerProd	[295]	1:14.087
MPI_Finalize	[82]	1:04.179
hypre_StructCopy	[297]	51.516 s
MPI_Waitall	[163]	20.135 s
hypre_StructVectorSetConstantVa	[303]	20.124 s
hypre_StructScale	[308]	15.580 s
MPI_Allreduce	[9]	13.283 s
MPI_Isend	[115]	9.010 s
hypre_StructMatrixSetBoxValues	[229]	8.455 s
sync	[2]	5.654 s
main	[184]	4.661 s
hypre_CAlloc	[186]	2.050 s
hypre_StructVectorSetBoxValues	[260]	1.827 s
hypre_StructMatrixInitializeDat	[224]	0.738 s
hypre_StructKrylovAxpy	[305]	0.668 s
MPI_Init	[108]	0.436 s
hypre_StructKrylovCopyVector	[296]	0.221 s
hypre_StructKrylovMatvec	[298]	0.215 s
hypre_PCGSolve	[293]	0.212 s
MPI_Irecv	[113]	0.190 s
hypre_BoxGetSize	[227]	0.182 s
hypre_Free	[187]	0.169 s
hypre_InitializeCommunication	[250]	0.160 s

Sorted by Value Down — All Symbols: Exclusive Times

Vampir and VampirTrace are

available at http://www.vampir.eu and

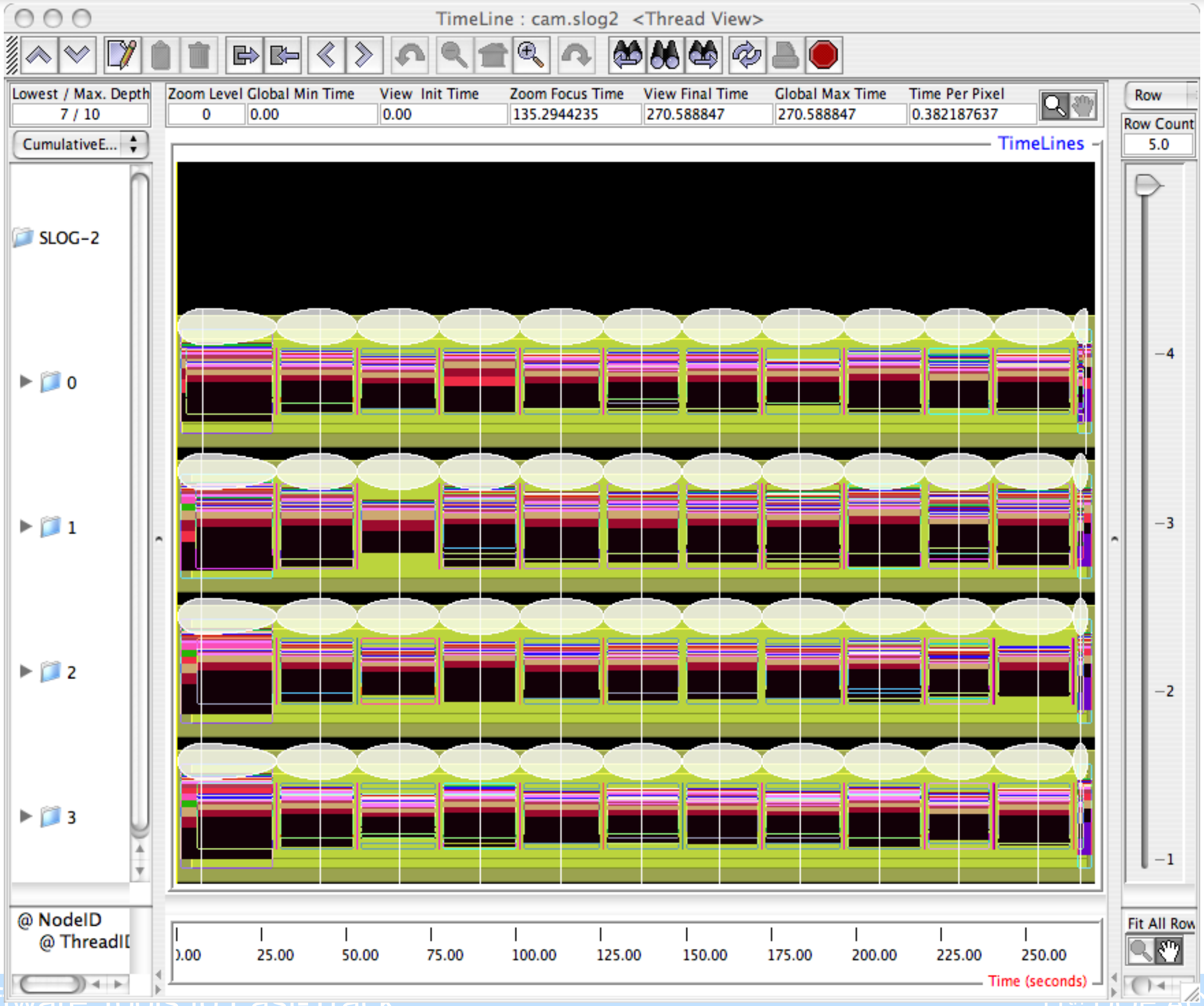
http://www.tu-dresden.de/zih/vampirtrace/ ,

get support via vampirsupport@zih.tu-dresden.de

Jumpshot

- http://www-unix.mcs.anl.gov/perfvis/software/viewers/index.htm
- Developed at Argonne National Laboratory as part of the MPICH project
  - Also works with other MPI implementations
  - Installed on IBM BG/P
  - Jumpshot is bundled with the TAU package
- Java-based tracefile visualization tool for postmortem performance analysis of MPI programs
- Latest version is Jumpshot-4 for SLOG-2 format
  - Scalable level of detail support
  - Timeline and histogram views
  - Scrolling and zooming
  - Search/scan facility

Jumpshot


TimeLine : cam.slog2 <Thread View>
Lowest / Max. Depth
7 / 10
Zoom Level
0
Global Min Time
0.00
View Init Time
0.00
Zoom Focus Time
135.2944235
View Final Time
270.588847
Global Max Time
270.588847
Time Per Pixel
0.382187637
Row
Row Count
5.0
CumulativeE...
TimeLines
SLOG-2
0
1
2
3
@ NodeID
@ ThreadID
25.00
50.00
75.00
100.00
125.00
150.00
175.00
200.00
225.00
250.00
Time (seconds)
Fit All Row
-4
-3
-2
-1

Support Acknowledgements

- Department of Energy (DOE)
  - Office of Science
    - MICS, Argonne National Lab
  - ASC/NNSA
    - University of Utah ASC/NNSA Level 1
    - ASC/NNSA, Lawrence Livermore National Lab
- Department of Defense (DoD)
- NSF Software Development for Cyberinfrastructure (SDCI)
- Research Centre Juelich
- ANL, LBL, PNNL, LLNL, LANL, SNL
- TU Dresden
- ParaTools, Inc.