TAU Performance Tools

Mira Conference at Argonne National Laboratory,

ALCF, Bldg. 240,# 1416, 10am, May 21, 2014, Argonne, IL

Sameer Shende, ParaTools, Inc. and U. Oregon

sameer@paratools.com

http://tau.uoregon.edu/tau_mbc14.ppt

Slides: /soft/perftools/tau/ppt/tau_mbc14.ppt (.pdf)

Acknowledgements: U. Oregon, ParaTools, Inc.

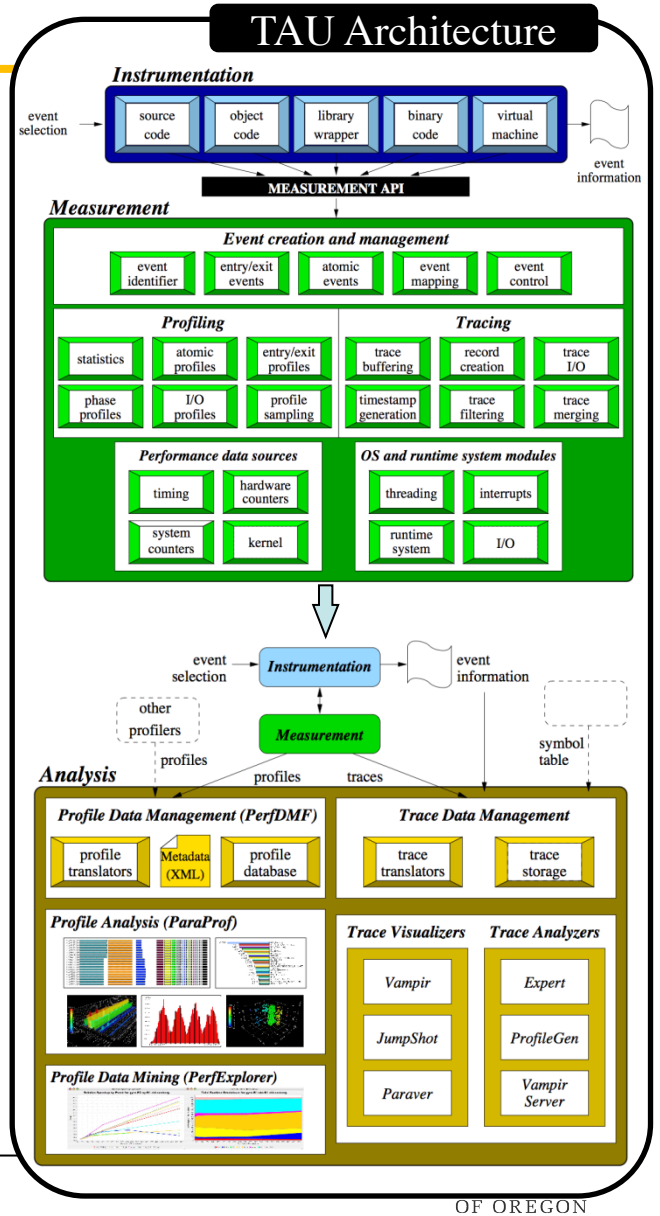
- Dr. Allen D. Malony, Professor, CIS Dept, and Director, NeuroInformatics Center, and CEO, ParaTools, Inc.
- Dr. Kevin Huck, Research Associate, U. Oregon
- Dr. John Linford, Computer Scientist, ParaTools, Inc.
- Dr. Tyler Simon, Computer Scientist, ParaTools, Inc.
- Wyatt Spear, Software engineer, UO, ParaTools, Inc.
- Daniel Ellsworth, Ph.D. student, UO
- Nick Chaimov, Ph.D. student, UO
- Ender Dai, Ph.D. student, UO
- David Ozog, Ph.D. student, UO
- David Poliakoff, Ph.D. student, UO
- Dr. Robert Yelle, Research faculty, UO

What is TAU?

- TAU is a performance evaluation tool
- It supports parallel profiling and tracing toolkit
- 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
- 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.)
- It supports C++, C, UPC, Fortran, Python, and Java
- TAU runs on all HPC platforms and it is free (BSD style license)
- TAU has instrumentation, measurement and analysis tools
- To use TAU, you need to set a couple of environment variables and substitute the name of the compiler with a TAU shell script

TAU Performance System®

- Integrated toolkit for performance problem solving
  - Instrumentation, measurement, analysis, visualization
  - Portable performance profiling and tracing facility
  - Performance data management and data mining
- Supports both direct as well as indirect (sampling) performance measurement approach
- Open source
- Available on all HPC platforms
- http://tau.uoregon.edu

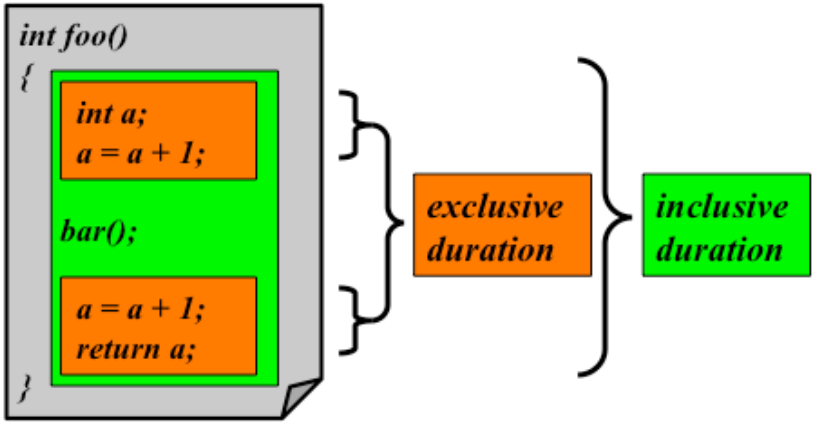


Performance Evaluation

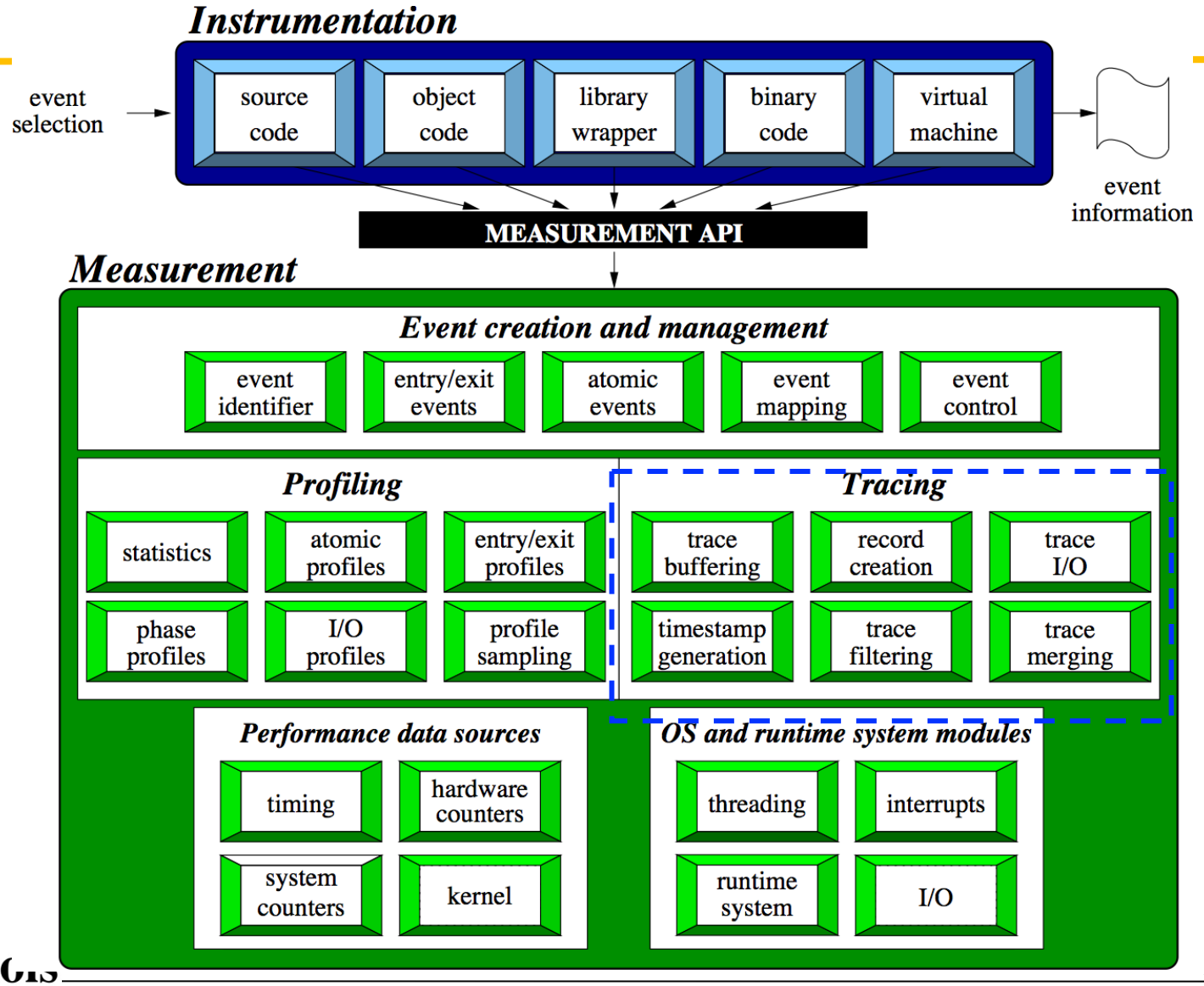
- Profiling
  - Presents summary statistics of performance metrics
    - number of times a routine was invoked
    - exclusive, inclusive time/hpm counts spent executing it
    - number of instrumented child routines invoked, etc.
    - structure of invocations (calltrees/callgraphs)
    - memory, message communication sizes also tracked
- Tracing
  - Presents when and where events took place along a global timeline
    - timestamped log of events
    - message communication events (sends/receives) are tracked
      - shows when and where messages were sent
    - large volume of performance data generated leads to more perturbation in the program

TAU Performance Profiling

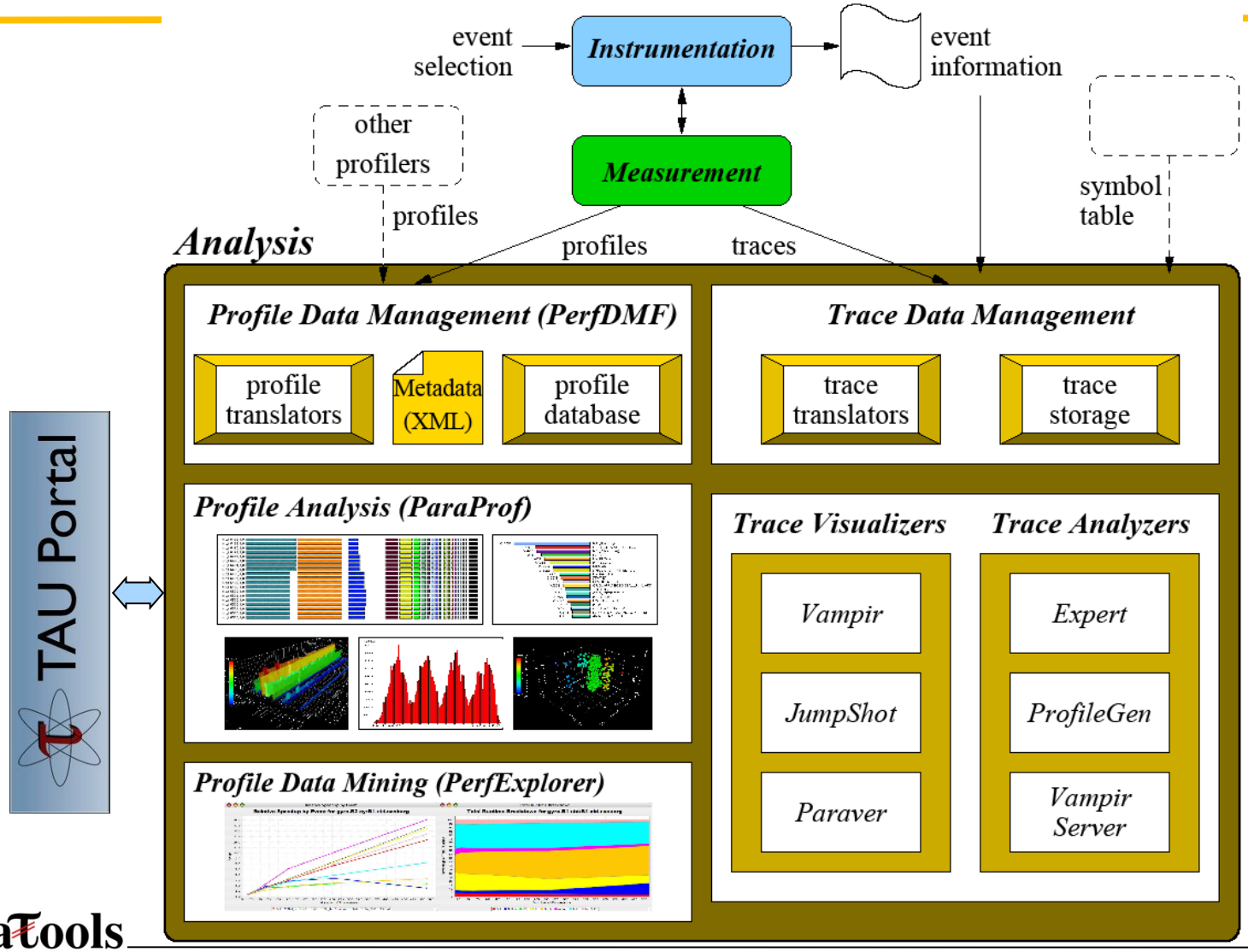
- Performance with respect to nested event regions
  - Program execution event stack (begin/end events)
- Profiling measures inclusive and exclusive data
- Exclusive measurements for region only performance
- Inclusive measurements includes nested “child” regions
- Support multiple profiling types
  - Flat, callpath, and phase profiling



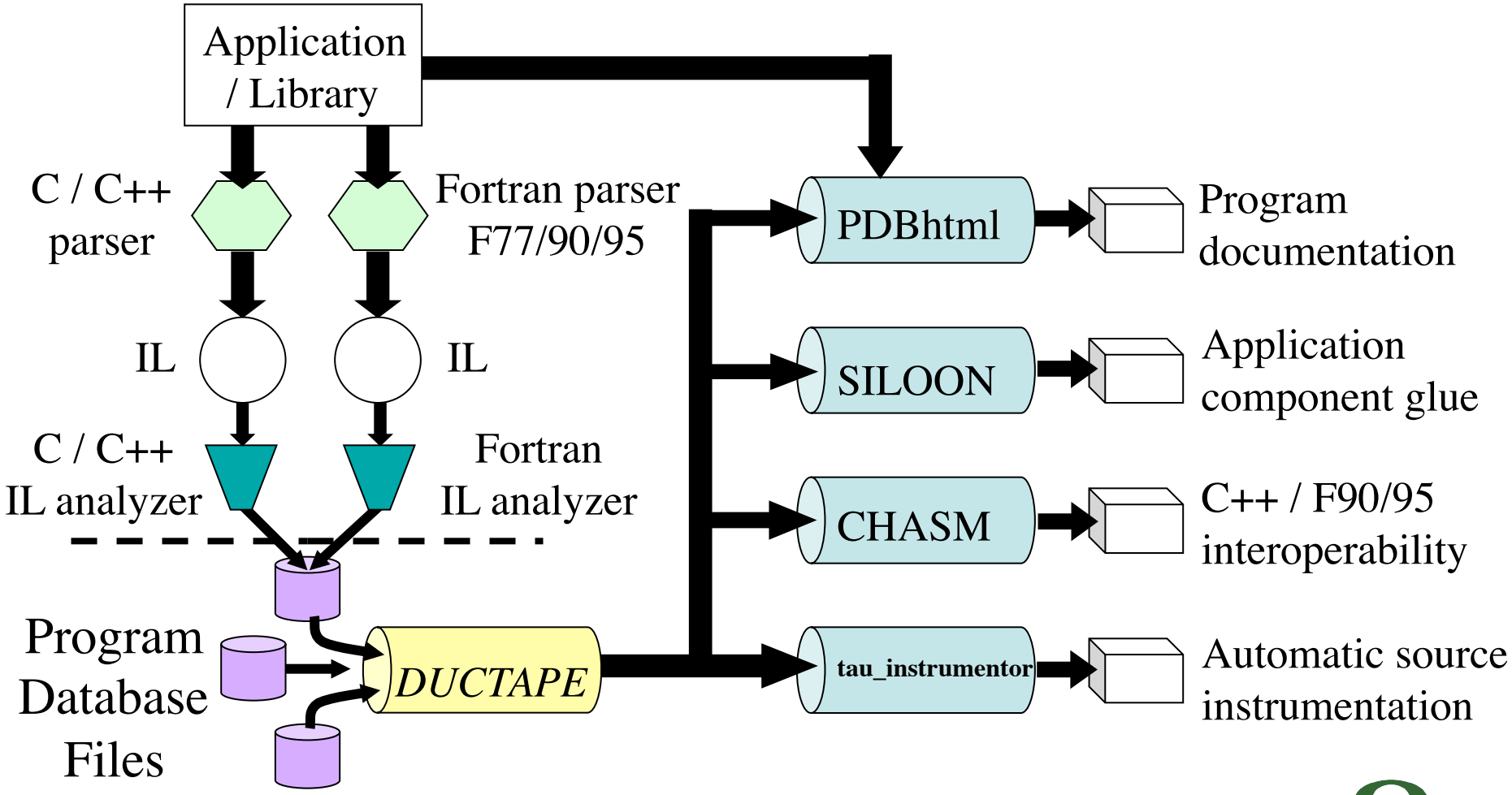
TAU Performance System Architecture



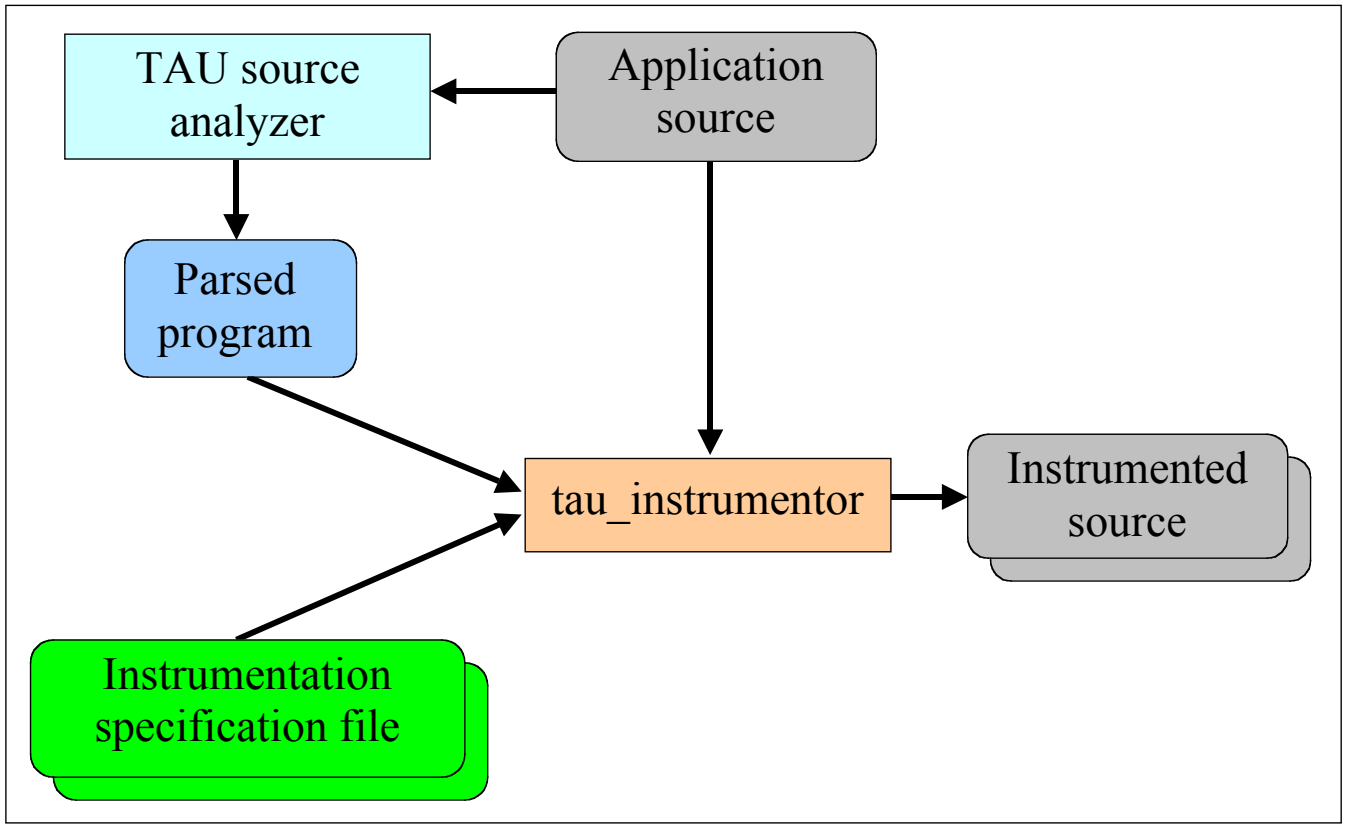
TAU Performance System Architecture



Program Database Toolkit (PDT)



Automatic Source-Level Instrumentation in TAU



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 that is generated when you configure it
- To instrument source code using PDT
  - Choose an appropriate TAU stub makefile in <arch>/lib:

```
% soft add +tau-latest (on BG/Q)
% export TAU_MAKEFILE=/soft/perftools/tau/tau_latest/bgq/lib/Makefile.tau-bgqtimers-mpi-pdt
% export 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:

```
% mpixlf90_r foo.f90
```

  changes to

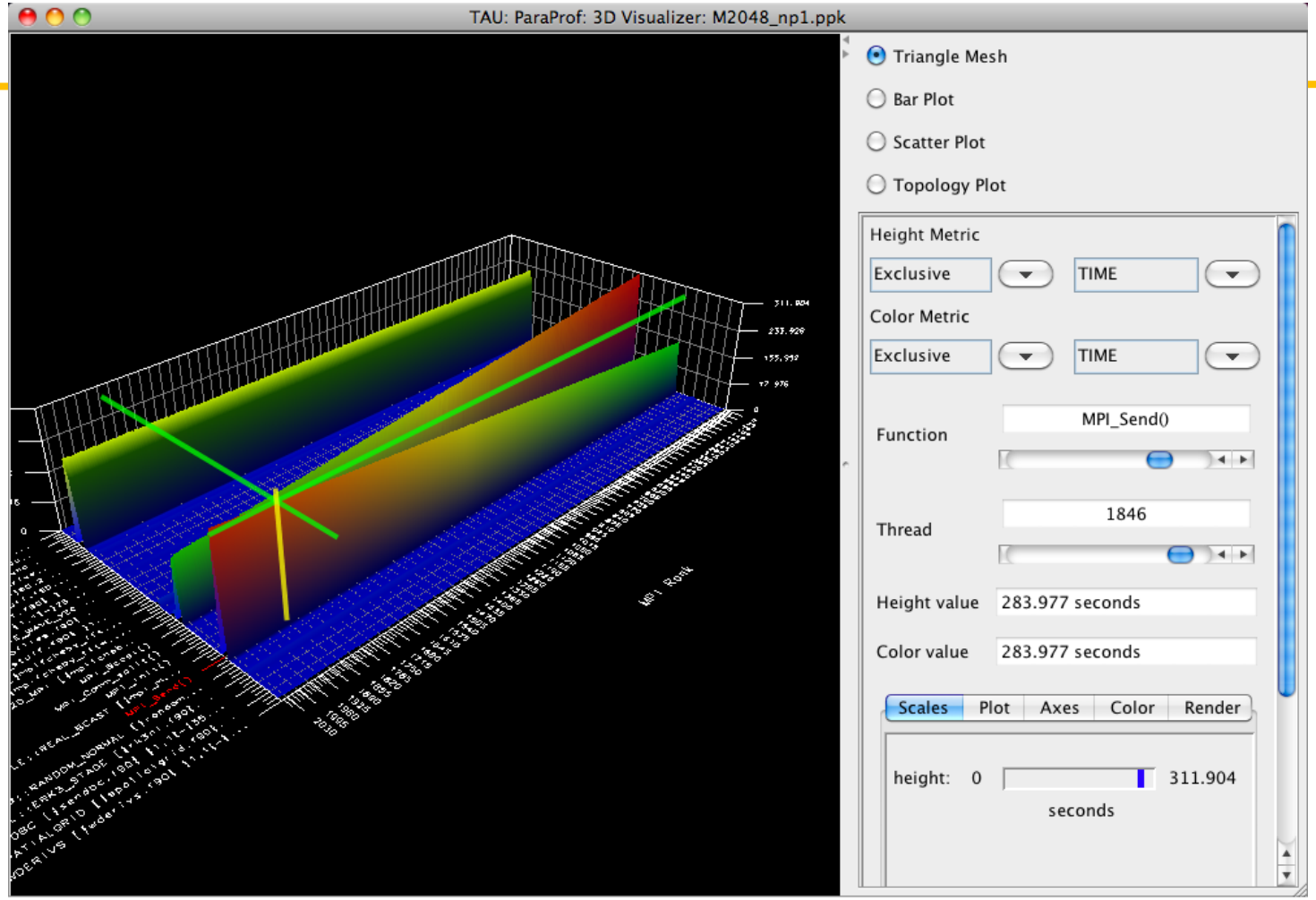
```
% tau_f90.sh foo.f90
% qsub –A <aueue> –q R.bc –n 256 –t 10 ./a.out
```

- Execute application and analyze performance data:

```
% pprof   (for text based profile display)
% paraprof  (for GUI)
```

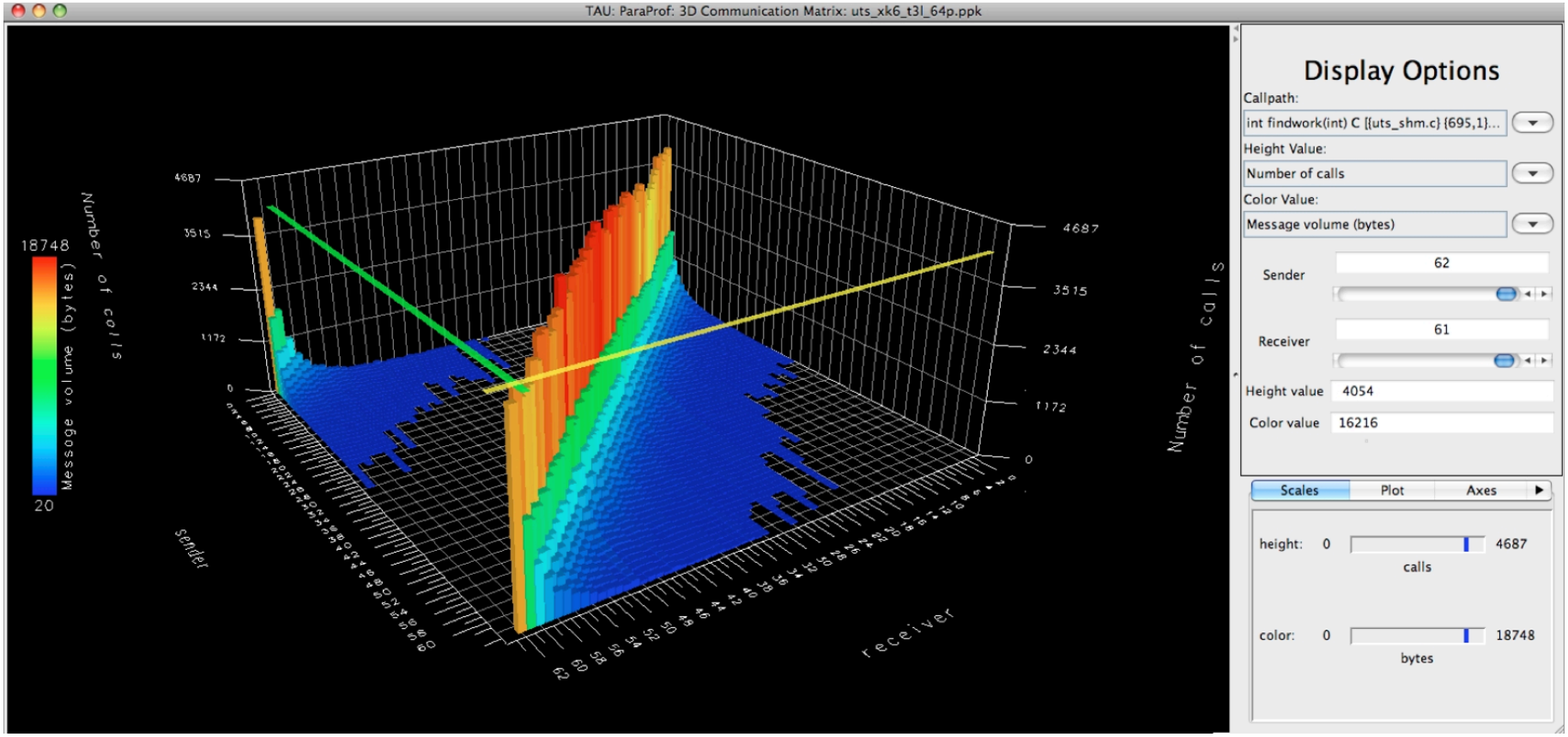
TAU Measurement Configuration on BG/Q

Parallel Profile Visualization: ParaProf



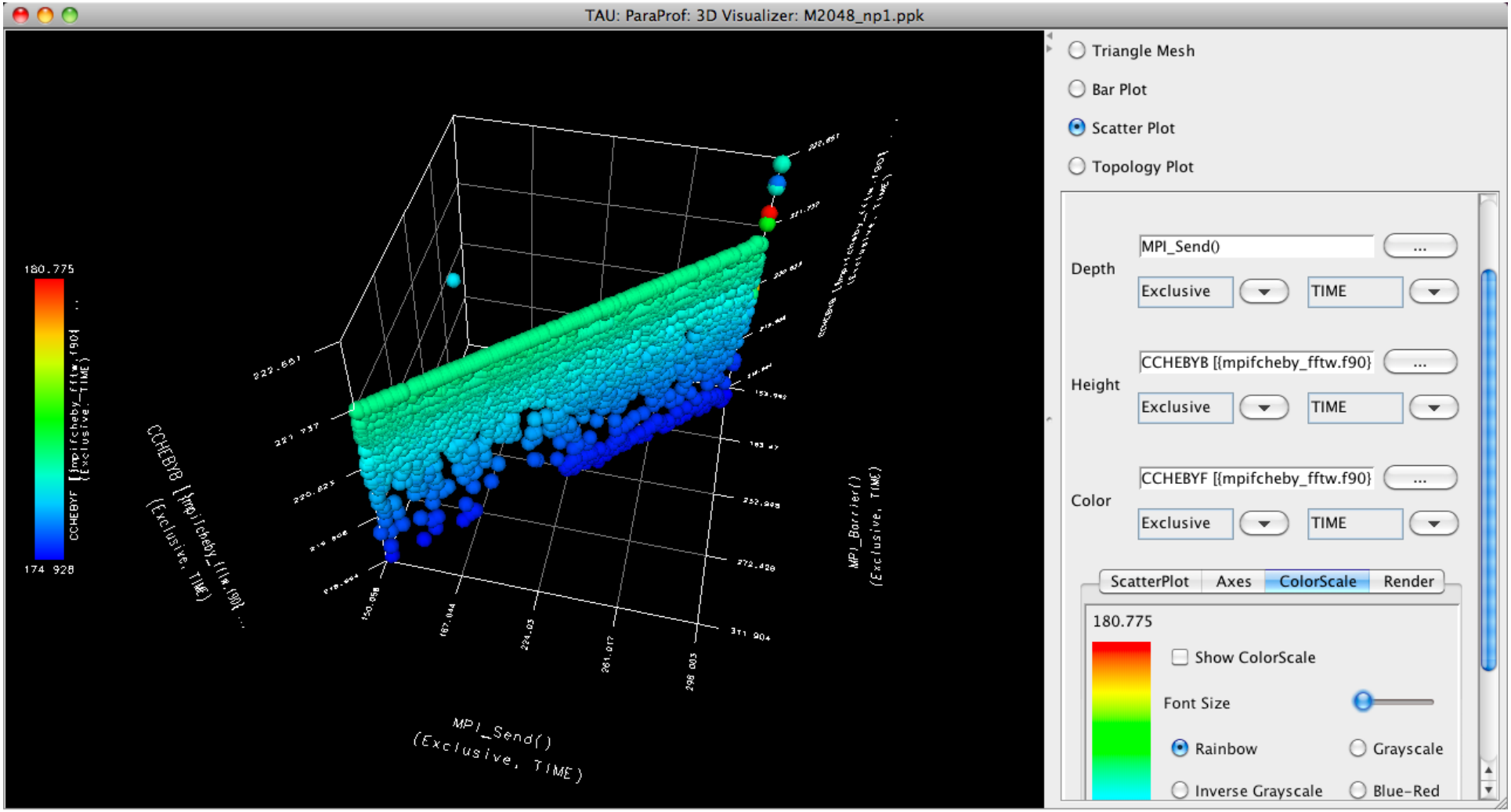

```
% soft add +tau-latest
% paraprof  (Windows -> 3D Visualization)
```

ParaProf: 3D Communication Matrix

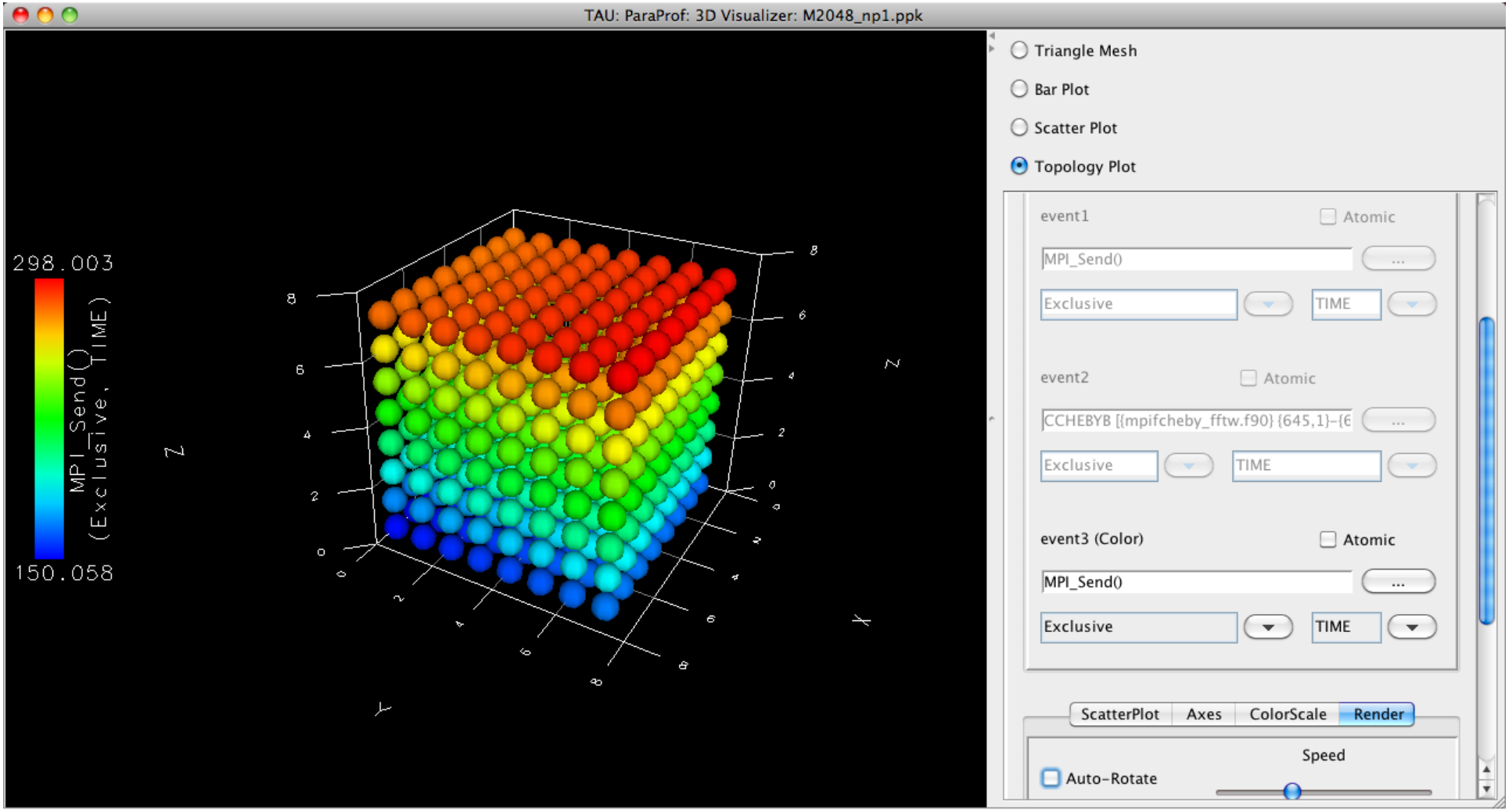



```
% qsub –env TAU_COMM_MATRIX=1 …
% paraprof (Windows -> 3D Communication Matrix)
```

ParaProf: Scatter Plot



ParaProf: Topology View: MPI_Send on BG/P



Interval, Atomic and Context Events in TAU

```
File  Edit  View  Terminal  Tabs  Help
Reading Profile files in profile.*

NODE 0;CONTEXT 0;THREAD 0:
---------------------------------------------------------------------------------------
%Time    Exclusive    Inclusive       #Call      #Subrs  Inclusive Name
              msec   total msec                          usec/call
---------------------------------------------------------------------------------------
100.0        0.007        0.256           1           5        256 MAIN
 97.3        0.132        0.249           5           5         50 FOO
 40.6        0.104        0.104           5           0         21 BAR
 36.3        0.013        0.093           3           3         31 G
---------------------------------------------------------------------------------------

USER EVENTS Profile :NODE 0, CONTEXT 0, THREAD 0
---------------------------------------------------------------------------------------
NumSamples   MaxValue   MinValue  MeanValue  Std. Dev.  Event Name
---------------------------------------------------------------------------------------
         1         16         16         16          0  MEMORY LEAK! malloc size <file=foo.f90, variable=X, line=7> : MAIN => FOO => BAR
         2         52         48         50          2  MEMORY LEAK! malloc size <file=foo.f90, variable=X, line=7> : MAIN => FOO => G => BAR
         1         80         80         80          0  free size <file=foo.f90, variable=X, line=10>
         1         80         80         80          0  free size <file=foo.f90, variable=X, line=10> : MAIN => FOO => G => BAR
         1        180        180        180          0  free size <file=foo.f90, variable=X, line=15>
         1        180        180        180          0  free size <file=foo.f90, variable=X, line=15> : MAIN => FOO => BAR
         1        180        180        180          0  malloc size <file=foo.f90, variable=X, line=13>
         1        180        180        180          0  malloc size <file=foo.f90, variable=X, line=13> : MAIN => FOO => BAR
         4         80         16         49      22.69  malloc size <file=foo.f90, variable=X, line=7>
         1         16         16         16          0  malloc size <file=foo.f90, variable=X, line=7> : MAIN => FOO => BAR
         3         80         48         60      14.24  malloc size <file=foo.f90, variable=X, line=7> : MAIN => FOO => G => BAR
---------------------------------------------------------------------------------------
                                                                                1,1           All
```

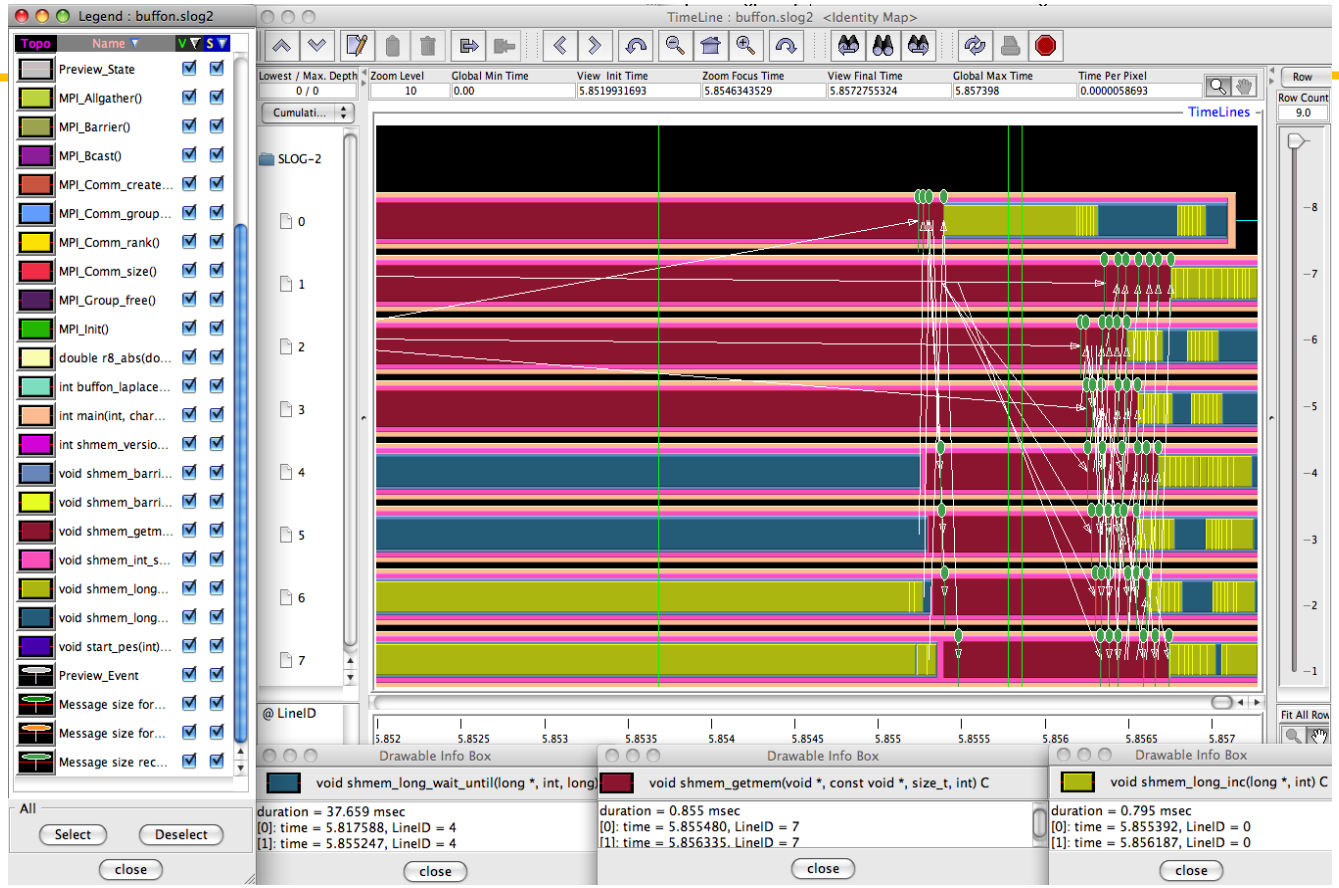
Interval Event

Context Event

Atomic Event

```
% pprof
```

Jumpshot [ANL]: Trace Visualization

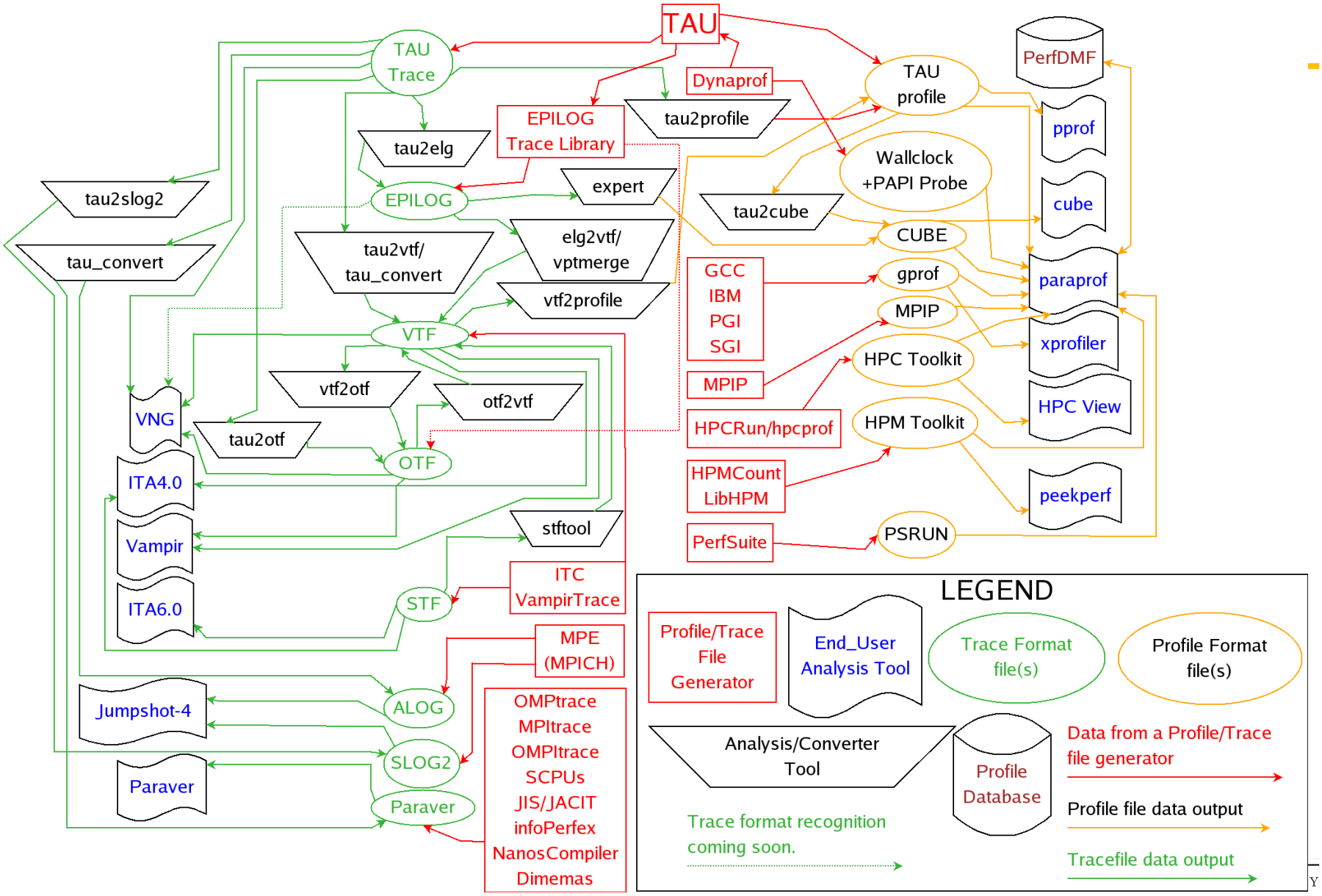



```
% qsub –env TAU_TRACE=1 …
% tau_treemerge.pl
% tau2slog2 tau.trc tau.edf –o app.slog2
% jumpshot app.slog2
```

ParaTools

UNIVERSITY OF OREGON

Building Bridges to Other Tools



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)
  - Selection of event statistics
- Support definition of “semantic” entities for mapping
- Support for event groups (aggregation, selection)
- Instrumentation optimization
  - Eliminate instrumentation in lightweight routines

Interval, Atomic and Context Events in TAU

```
File  Edit  View  Terminal  Tabs  Help
Reading Profile files in profile.*

NODE 0;CONTEXT 0;THREAD 0:
---------------------------------------------------------------------------------------
%Time    Exclusive    Inclusive       #Call      #Subrs  Inclusive Name
              msec   total msec                          usec/call
---------------------------------------------------------------------------------------
100.0        0.007        0.256           1           5        256 MAIN
 97.3        0.132        0.249           5           5         50 FOO
 40.6        0.104        0.104           5           0         21 BAR
 36.3        0.013        0.093           3           3         31 G
---------------------------------------------------------------------------------------

USER EVENTS Profile :NODE 0, CONTEXT 0, THREAD 0
---------------------------------------------------------------------------------------
NumSamples   MaxValue   MinValue  MeanValue  Std. Dev.  Event Name
---------------------------------------------------------------------------------------
         1         16         16         16          0  MEMORY LEAK! malloc size <file=foo.f90, variable=X, line=7> : MAIN => FOO => BAR
         2         52         48         50          2  MEMORY LEAK! malloc size <file=foo.f90, variable=X, line=7> : MAIN => FOO => G => BAR
         1         80         80         80          0  free size <file=foo.f90, variable=X, line=10>
         1         80         80         80          0  free size <file=foo.f90, variable=X, line=10> : MAIN => FOO => G => BAR
         1        180        180        180          0  free size <file=foo.f90, variable=X, line=15>
         1        180        180        180          0  free size <file=foo.f90, variable=X, line=15> : MAIN => FOO => BAR
         1        180        180        180          0  malloc size <file=foo.f90, variable=X, line=13>
         1        180        180        180          0  malloc size <file=foo.f90, variable=X, line=13> : MAIN => FOO => BAR
         4         80         16         49      22.69  malloc size <file=foo.f90, variable=X, line=7>
         1         16         16         16          0  malloc size <file=foo.f90, variable=X, line=7> : MAIN => FOO => BAR
         3         80         48         60      14.24  malloc size <file=foo.f90, variable=X, line=7> : MAIN => FOO => G => BAR
---------------------------------------------------------------------------------------
                                                                                1,1           All
```

Interval Event

Context Event

Atomic Event

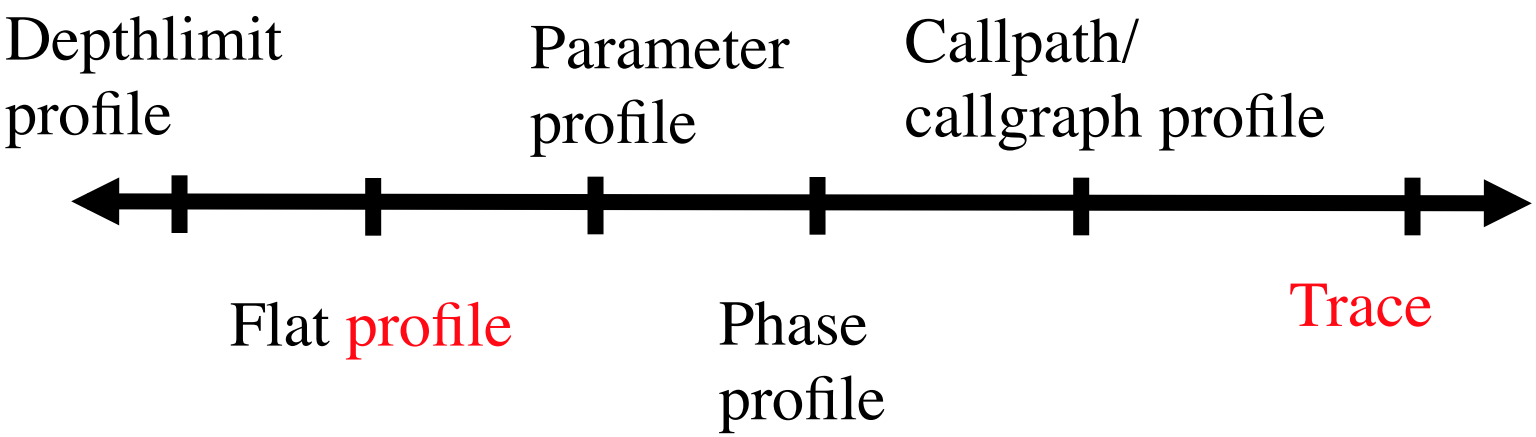
TAU Measurement Mechanisms

- **Parallel profiling**
  - Function-level, block-level, statement-level
  - Direct instrumentation as well as event based sampling
  - Supports user-defined events and mapping events
  - Support for flat, callgraph/callpath, phase profiling
  - Support for memory profiling (headroom, malloc/leaks)
  - Support for tracking I/O (wrappers, read/write/print calls)
  - Parallel profiles written at end of execution
  - Parallel profile snapshots can be taken during execution
- **Tracing**
  - All profile-level events + inter-process communication
  - Inclusion of multiple counter data in traced events

Types of Parallel Performance Profiling

- ***Flat*** profiles
  - Metric (e.g., time) spent in an event (callgraph nodes)
  - Exclusive/inclusive, # of calls, child calls
- ***Callpath*** profiles (***Calldepth*** profiles)
  - Time spent along a calling path (edges in callgraph)
  - "*main=> f1 => f2 => MPI_Send*" (event name)
  - `TAU_CALLPATH_DEPTH` environment variable
- ***Phase*** profiles
  - Flat profiles under a phase (nested phases are allowed)
  - Default "main" phase
  - Supports static or dynamic (e.g., per-iteration) phases

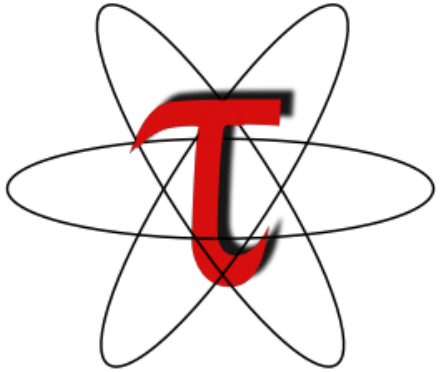
Performance Evaluation Alternatives



Each alternative has:
- one metric/counter
- multiple counters

Volume of performance data

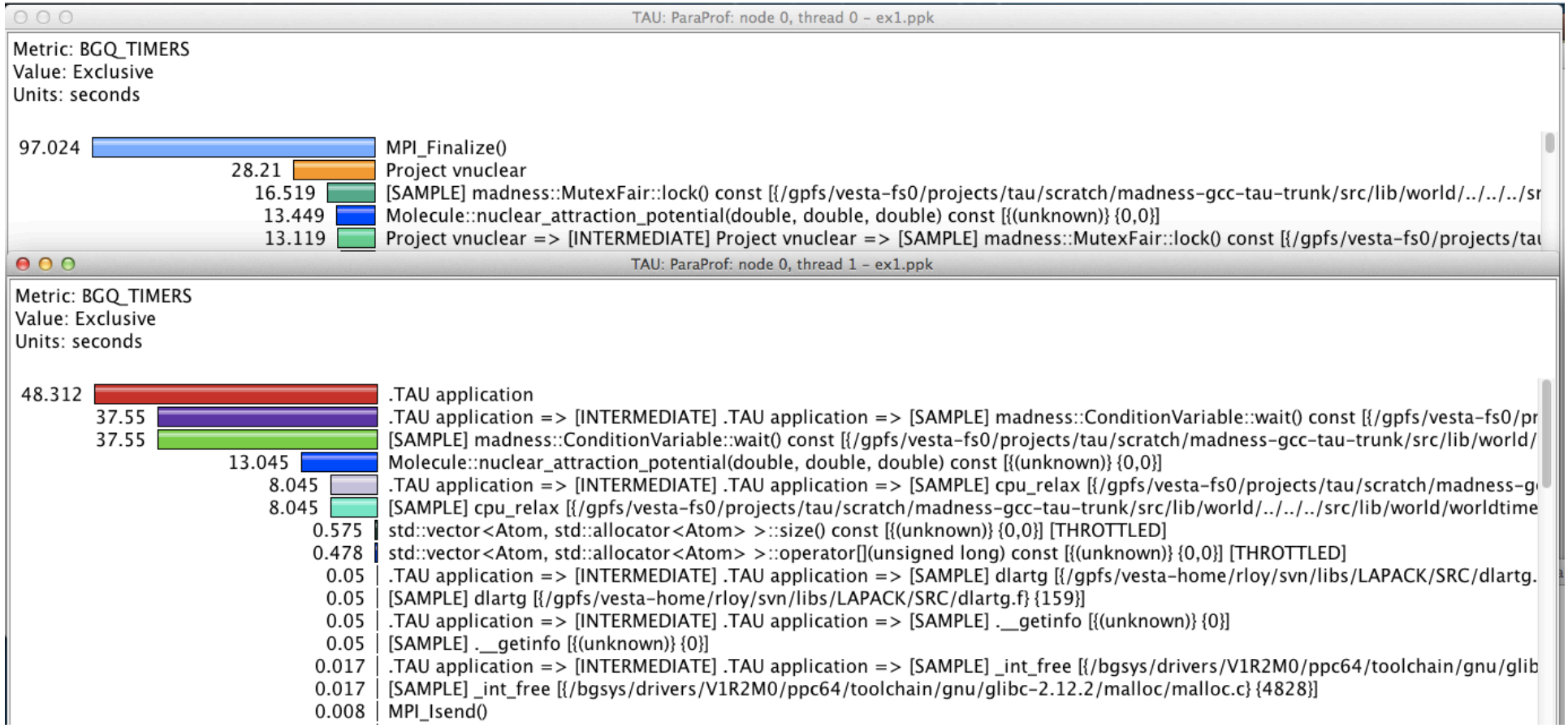
TAU: A Quick Reference



Runtime Environment Variables (tau.conf)

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_TRACK_SIGNALS	0	Setting to 1 generate debugging callstack info when a program crashes
TAU_SAMPLING	0	Setting to 1 generates sample based profilles
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 - tauprofile.xml. “snapshot” generates xml format
TAU_METRICS	TIME	Setting to a comma separted list generates other metrics. (e.g., --env TIME:BGQ_TIMERS:PAPI_FP_INS:PAPI_NATIVE_<event>)

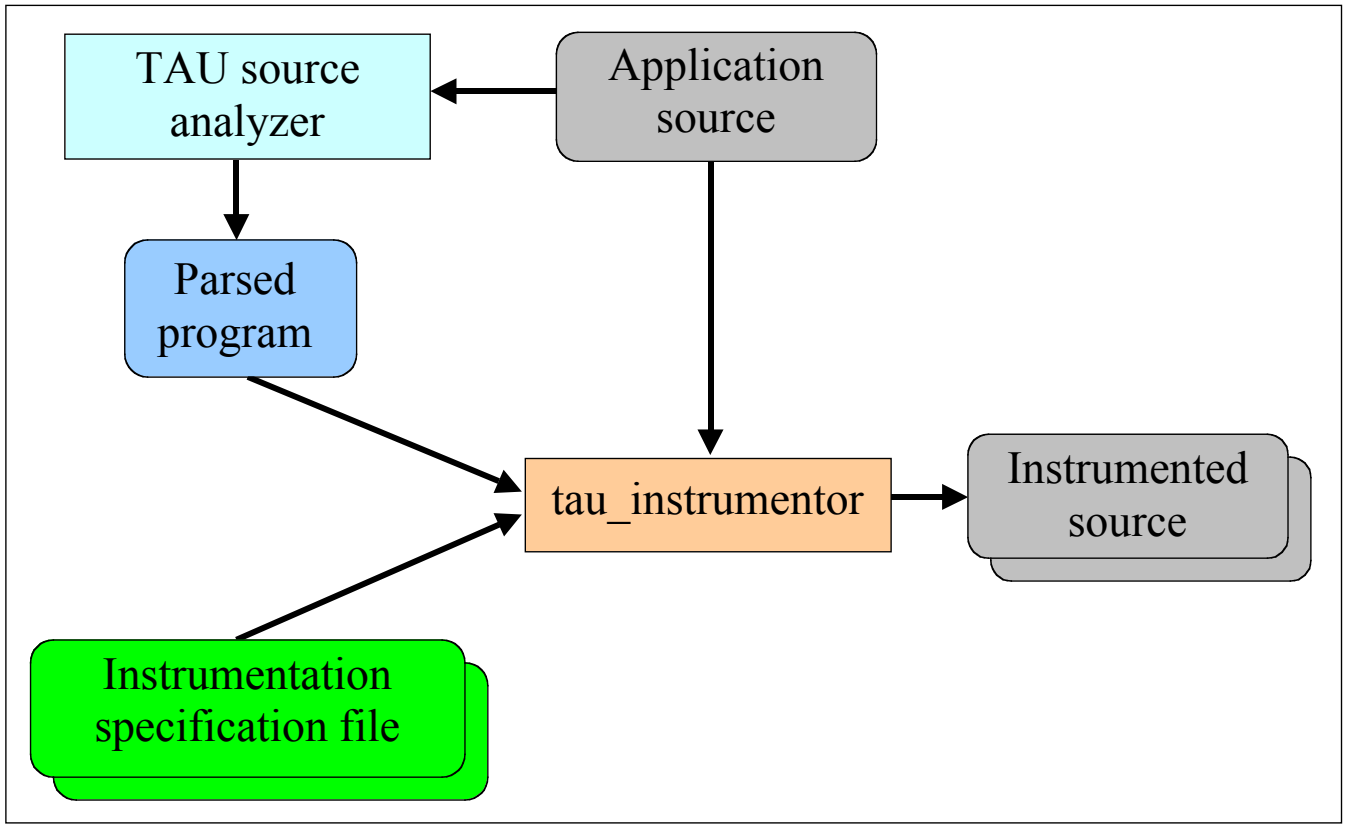
EBS: Setting TAU_SAMPLING=1



% export PATH=/soft/perftools/tau/tau_latest/bgq/bin/compilers:$PATH
No need to set any other TAU variables! Use tau.conf for settings.

TAU's Manual Instrumentation API

Automatic Source-Level Instrumentation in TAU using Program Database Toolkit (PDT)



Automatic Instrumentation

TAU_COMPILER Commandline Options

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.
-optLinkOnly	Do not instrument the source code. Simply link in the TAU libraries
-optTrackIO	Wrap POSIX I/O call and calculates vol/bw of I/O operations (Requires TAU to be configured with –iowrapper)
-optKeepFiles	Does not remove intermediate .pdb and .inst.* files
-optPreProcess	Preprocess Fortran sources before instrumentation
-optTauSelectFile=""	Specify selective instrumentation file for tau_instrumentor
-optTauWrapFile=""	Specify link_options.tau generated by tau_gen_wrapper
-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:
  % 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 -optDetectMemoryLeaks'

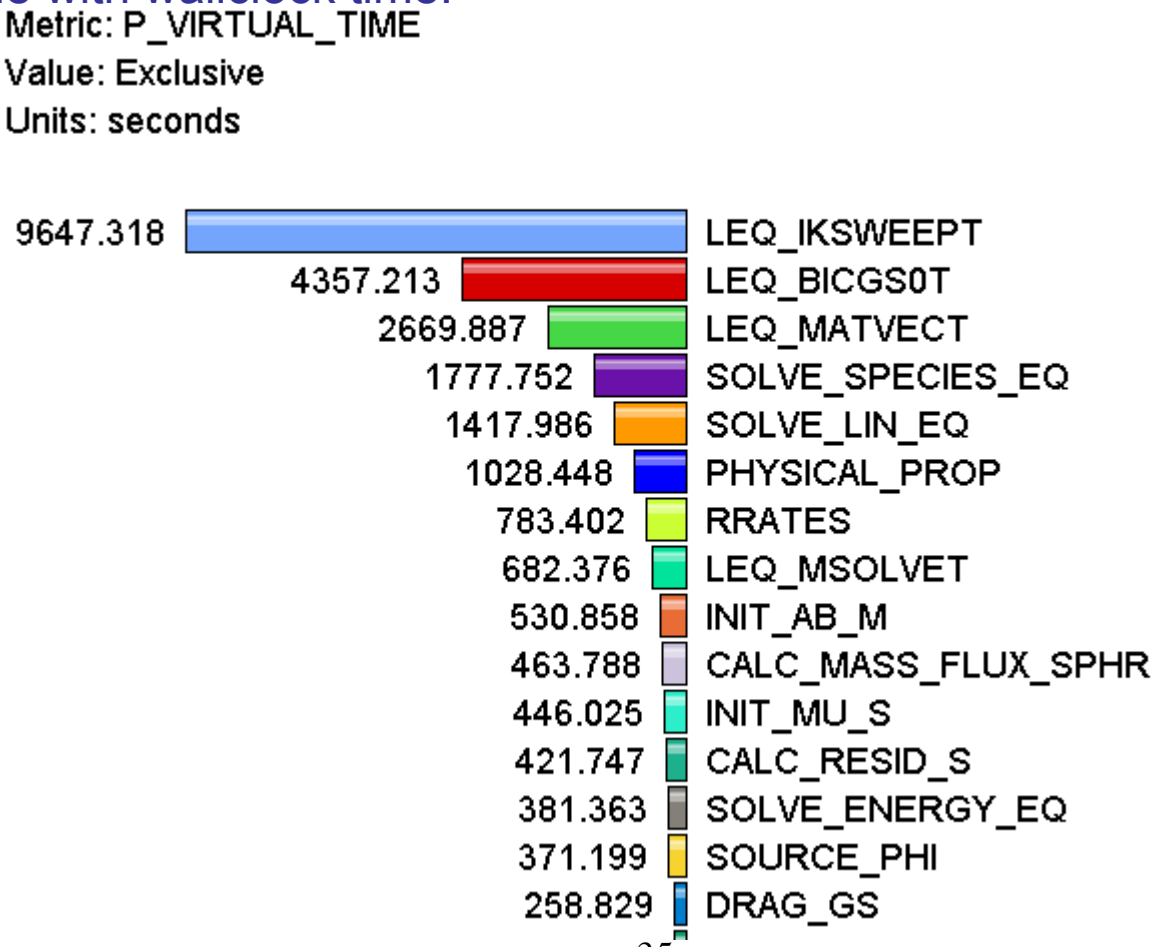
- To use an instrumentation specification file:
  % export TAU_OPTIONS= '-optTauSelectFile=mycmd.tau -optVerbose -optPreProcess'
  % cat mycmd.tau
  BEGIN_INSTRUMENT_SECTION
  memory file="foo.f90" routine="#"
  # instruments all allocate/deallocate statements in all routines in foo.f90
  loops file="*" routine="#"
  io file="abc.f90" routine="FOO"
  END_INSTRUMENT_SECTION

Steps of Performance Evaluation

- Collect basic routine-level timing profile to determine where most time is being spent
- Collect routine-level hardware counter data to determine types of performance problems
- Collect callpath profiles to determine sequence of events causing performance problems
- Conduct finer-grained profiling and/or tracing to pinpoint performance bottlenecks
  - Loop-level profiling with hardware counters
  - Tracing of communication operations

Usage Scenarios: Routine Level Profile

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

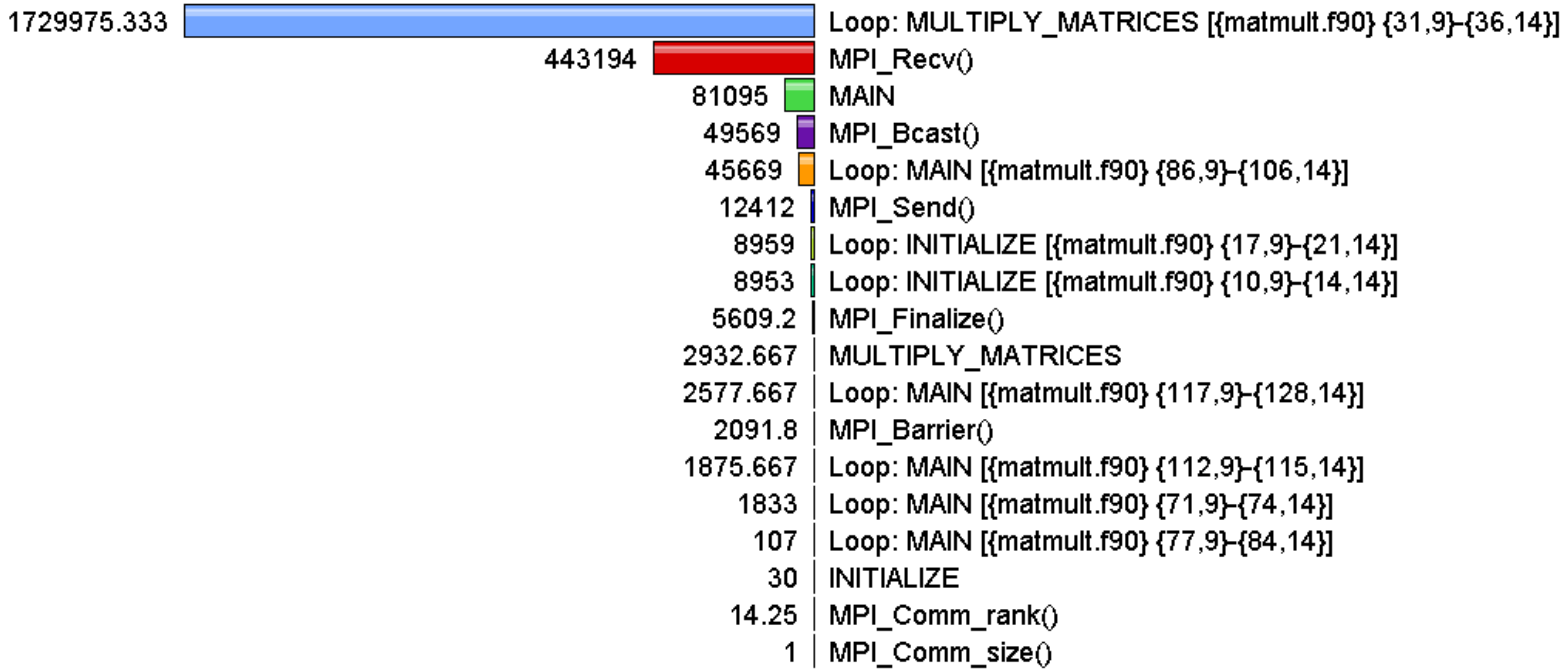


Solution: Generating a flat profile with MPI

Usage Scenarios: Loop Level Instrumentation

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

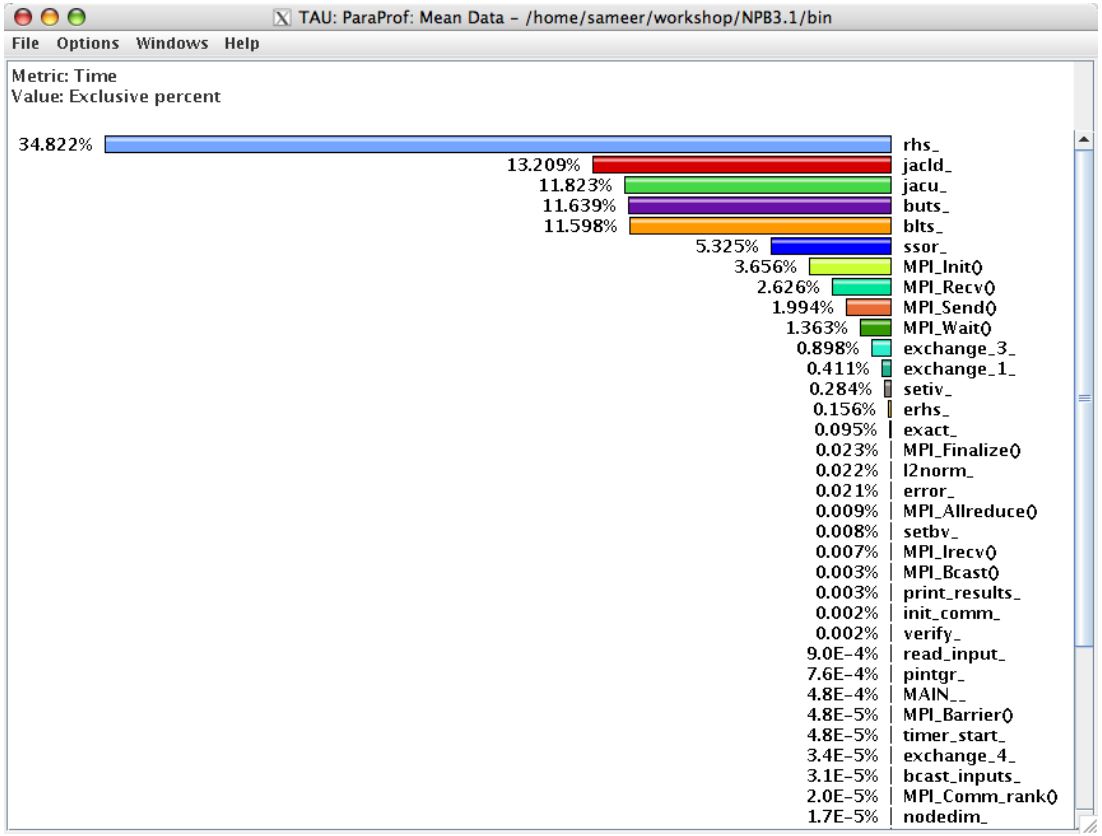
Metric: GET_TIME_OF_DAY
Value: Exclusive
Units: microseconds



Solution: Generating a loop level profile

Usage Scenarios: Compiler-based Instrumentation

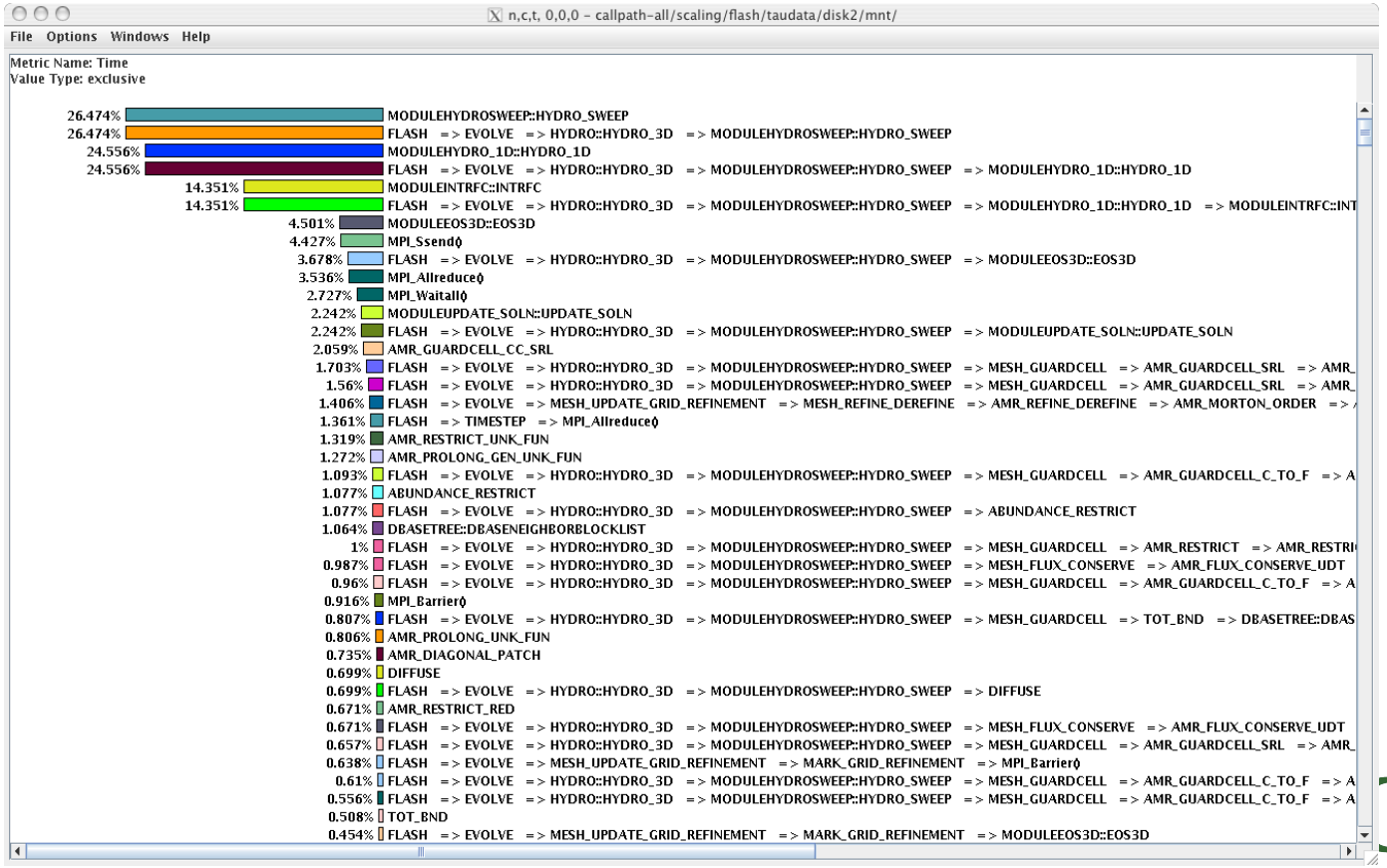
- Goal: Easily generate routine level performance data using the compiler instead of PDT for parsing the source code



Use Compiler-Based Instrumentation

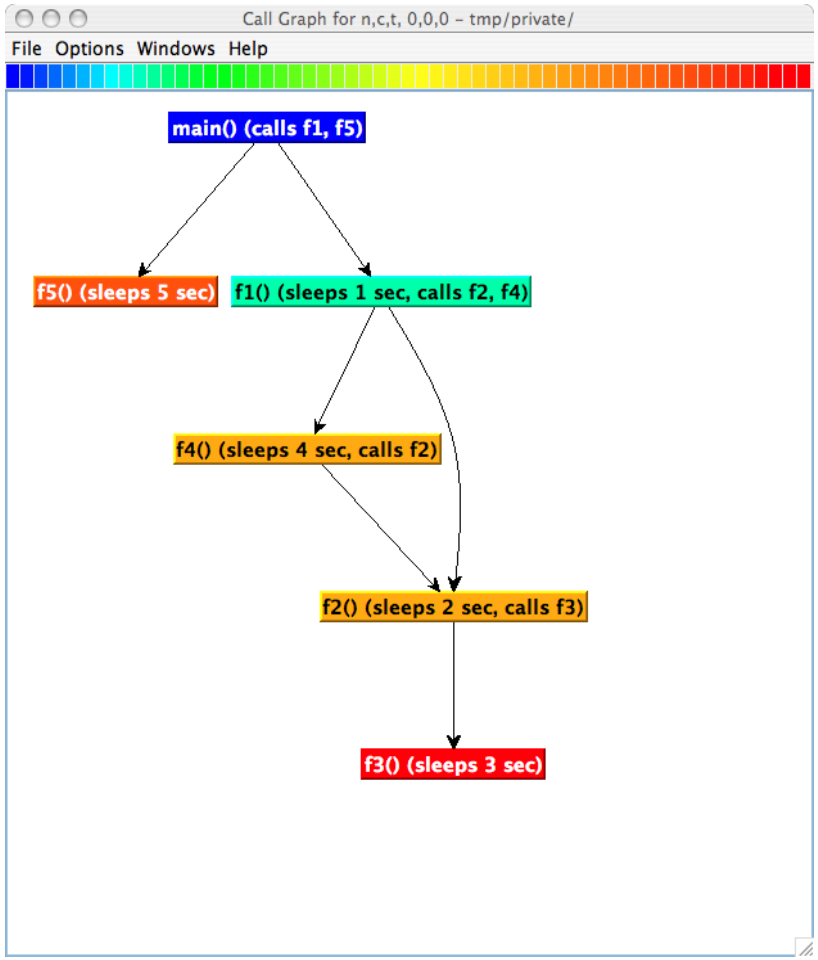
Usage Scenarios: Generating Callpath Profile

- Goal: Who calls my MPI_Barrier()? Where?
- Callpath profile for a given callpath depth:



Callpath Profile

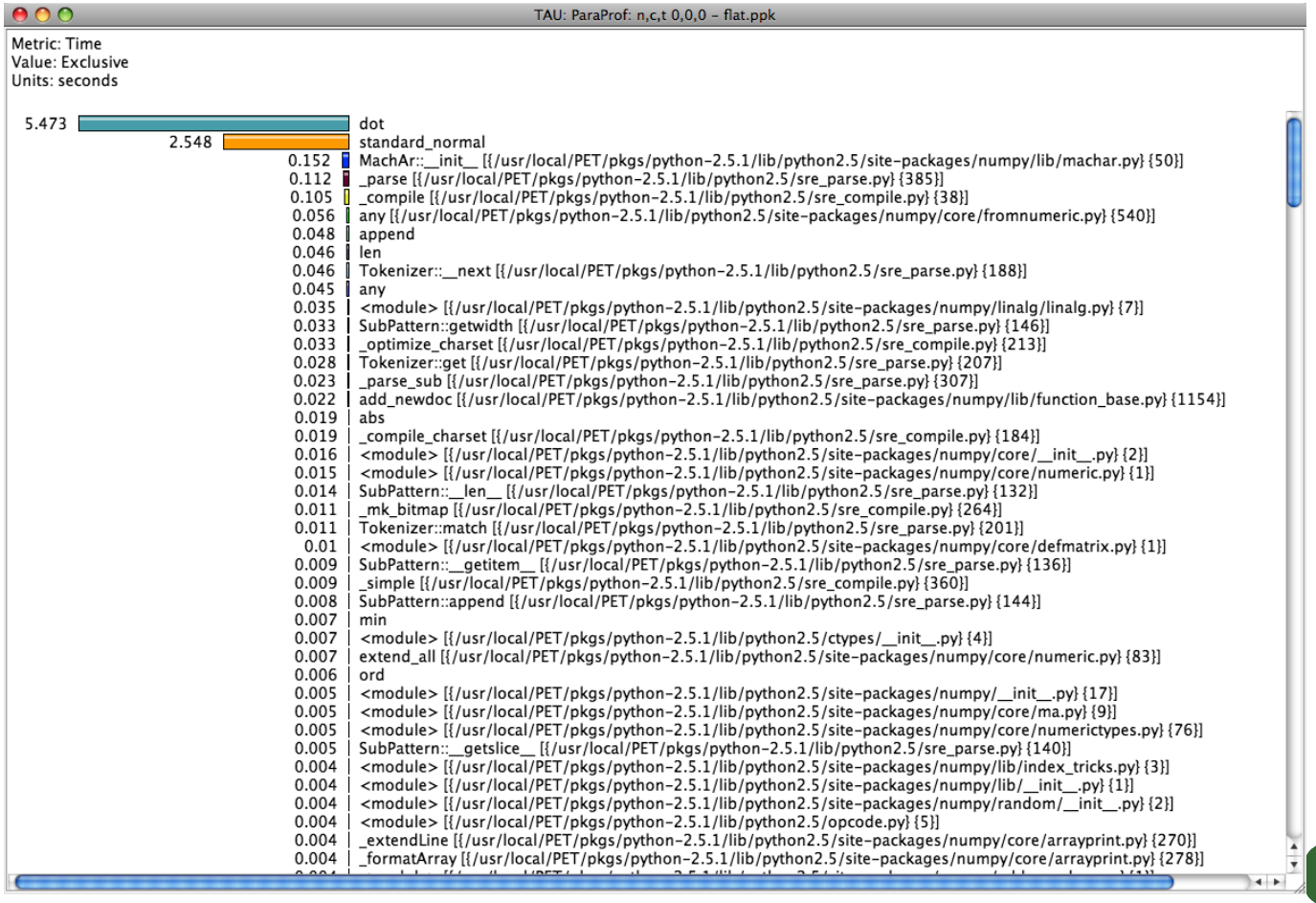
- Generates program callgraph



Generate a Callpath Profile

Usage Scenarios: Instrument a Python program

- Goal: Generate a flat profile for a Python program



Usage Scenarios: Instrument 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
```

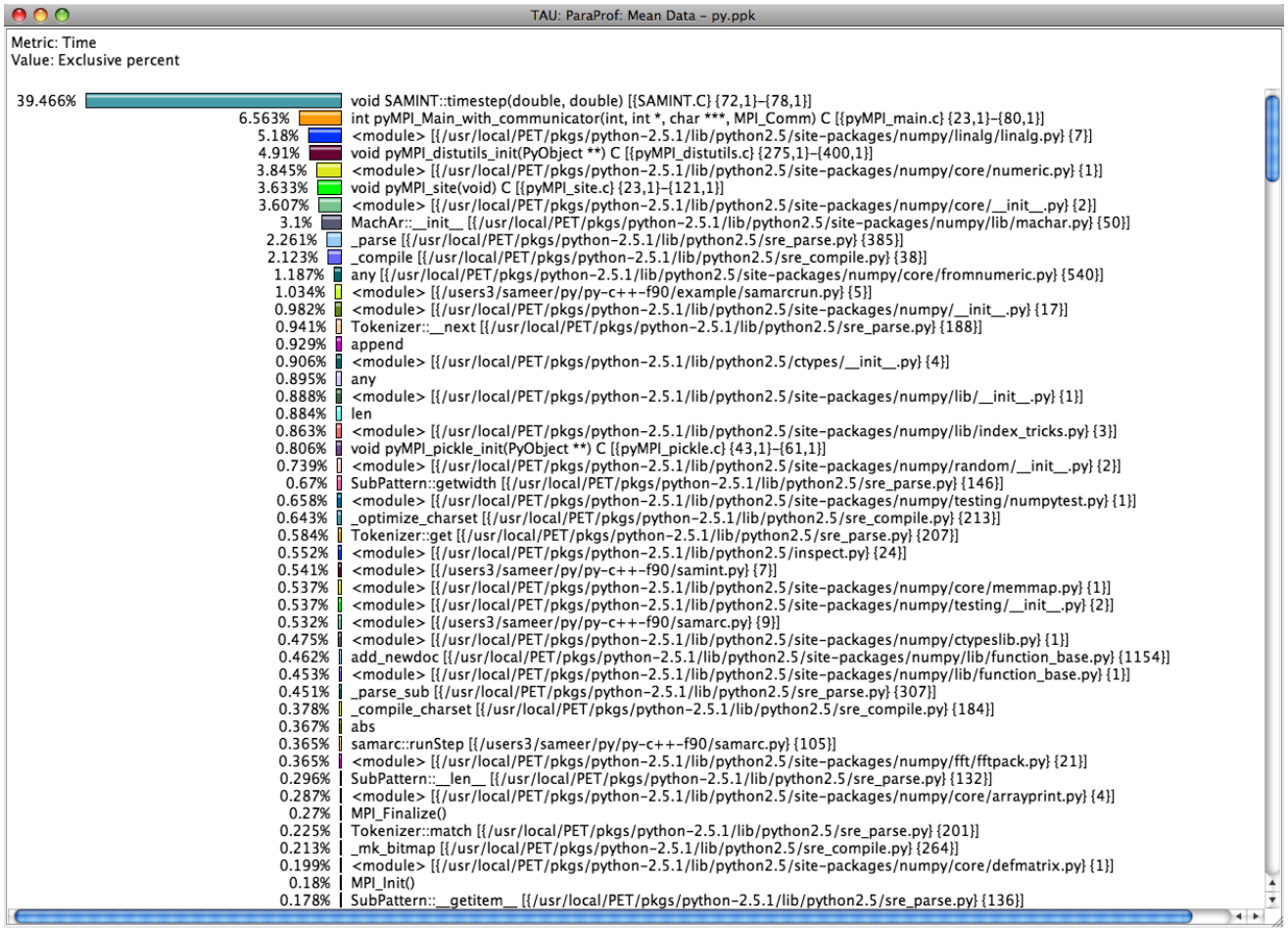
Create a wrapper:

```
% 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

Usage Scenarios: Mixed Python+F90+C+pyMPI

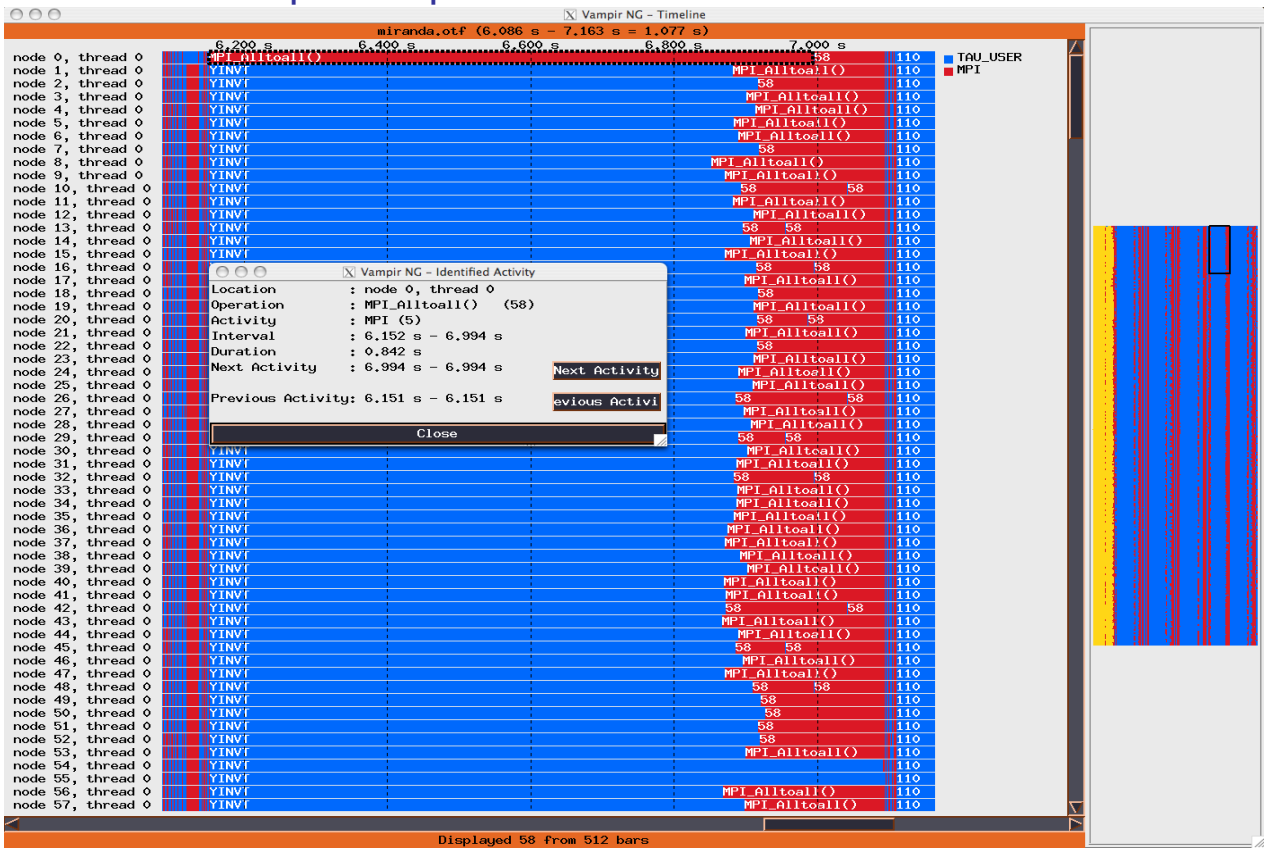
- Goal: Generate multi-level instrumentation for Python+MPI+C+F90+C++ ...



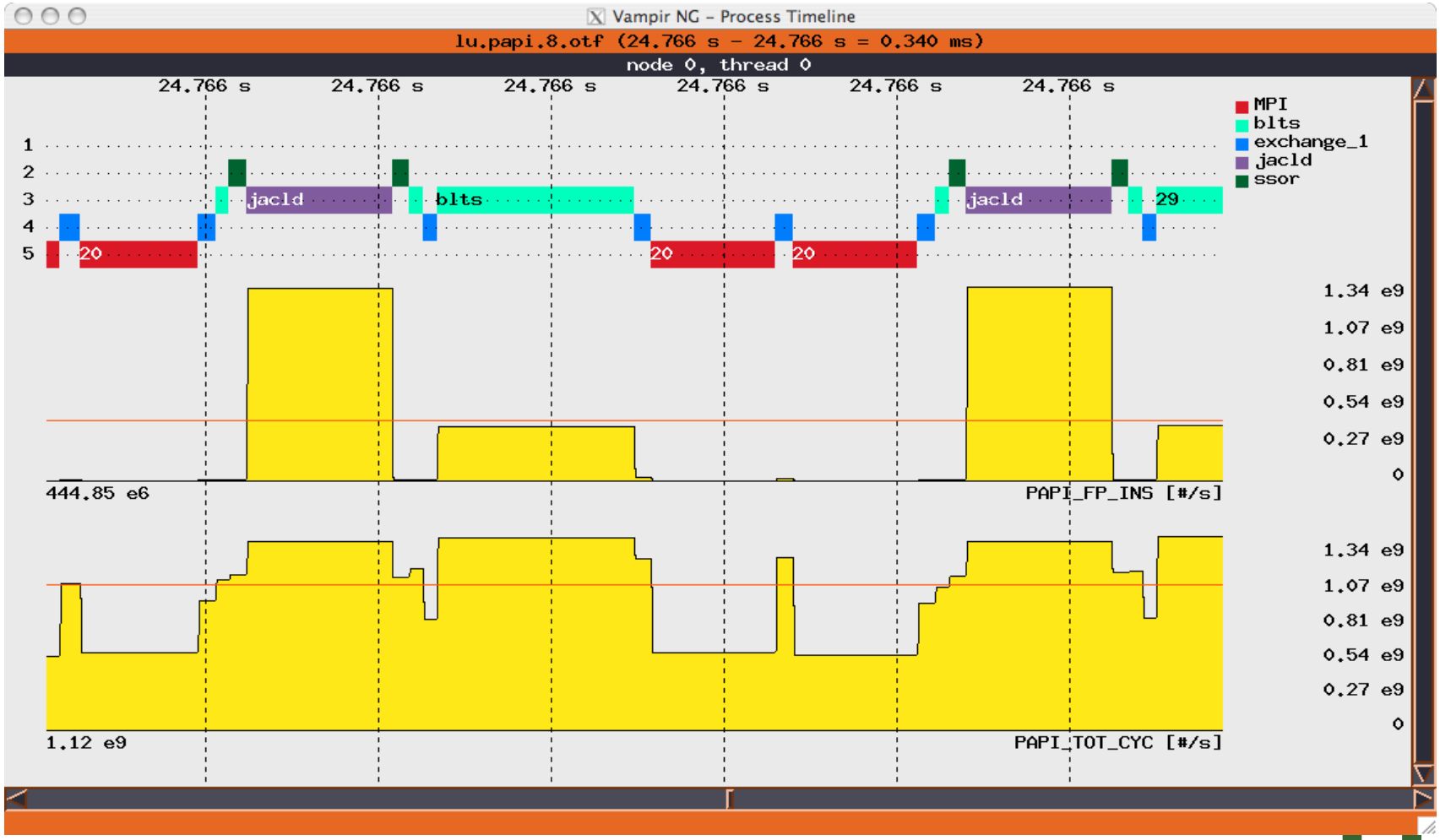
Generate a Multi-Language Profile w/ Python

Usage Scenarios: Generating a Trace File

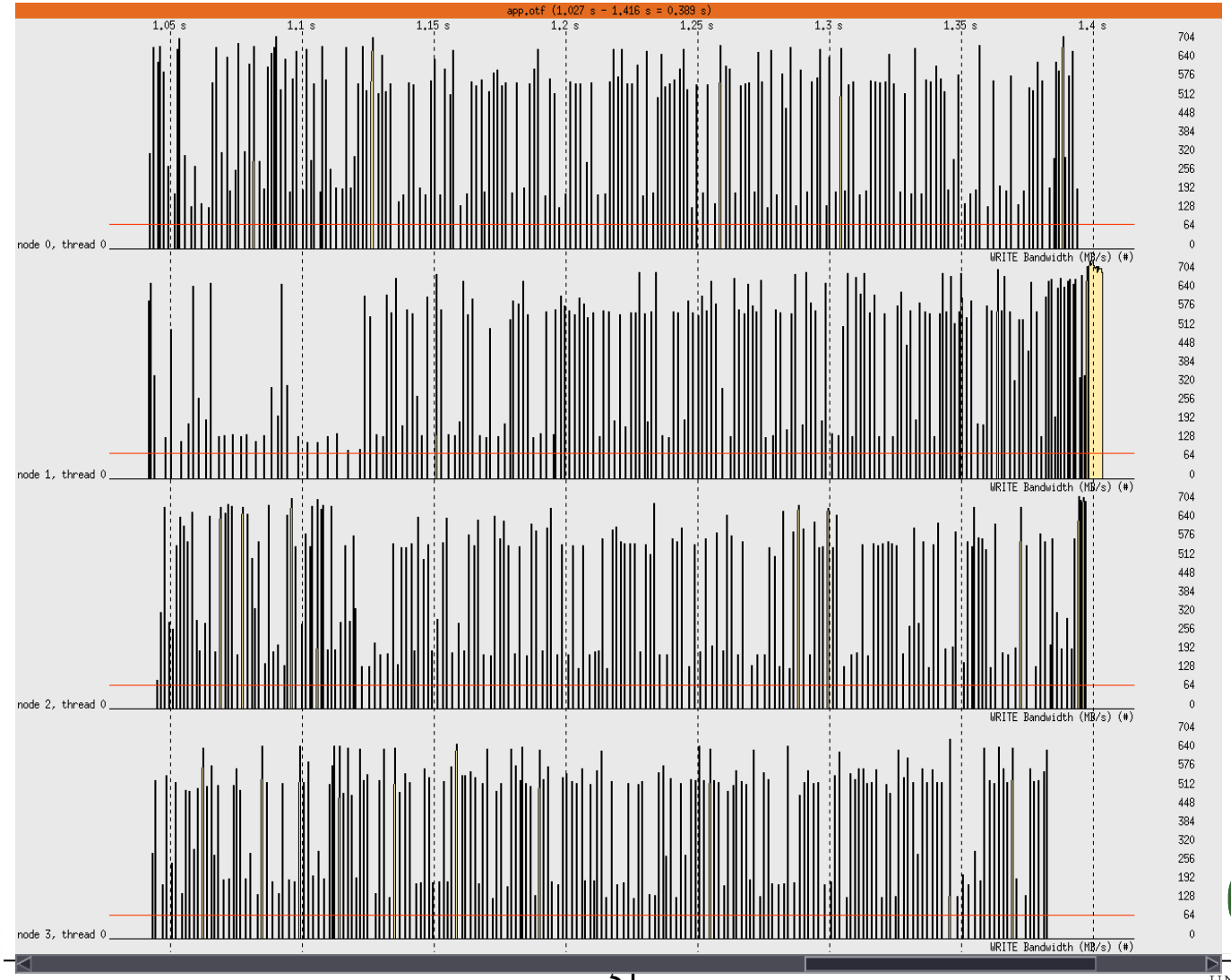
- Goal: Identify the temporal aspect of performance. What happens in my code at a given time? When?
- Event trace visualized in Vampir/Jumpshot



VNG Process Timeline with PAPI Counters



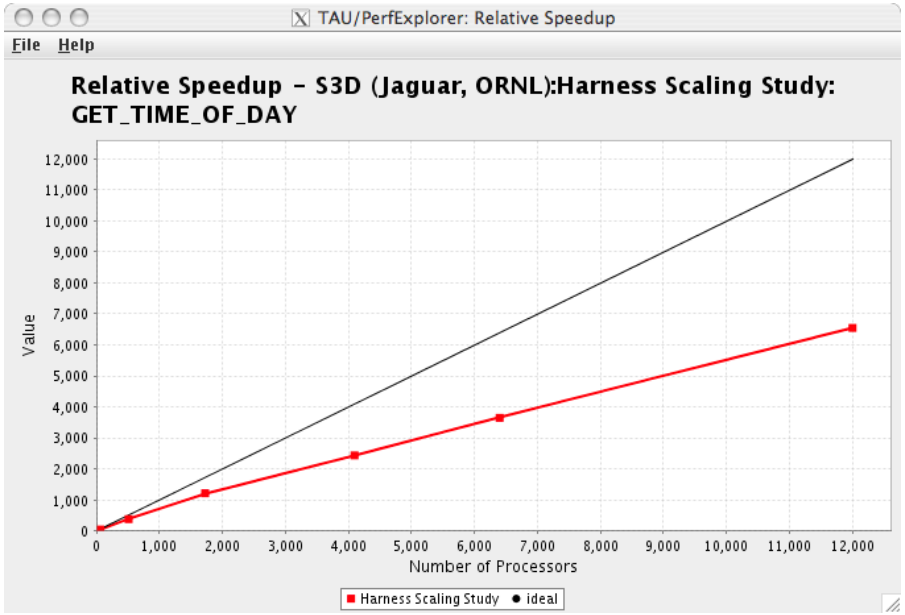
Vampir Counter Timeline Showing I/O BW

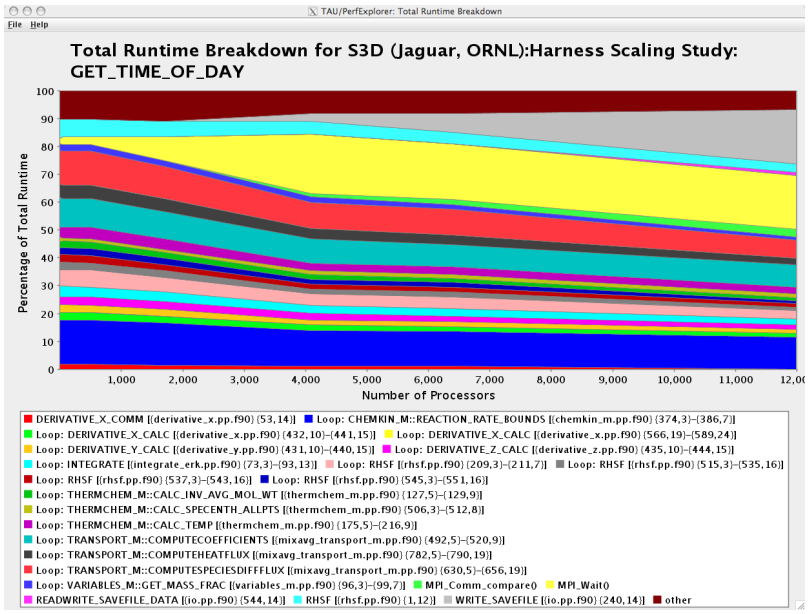


Generate a Trace File

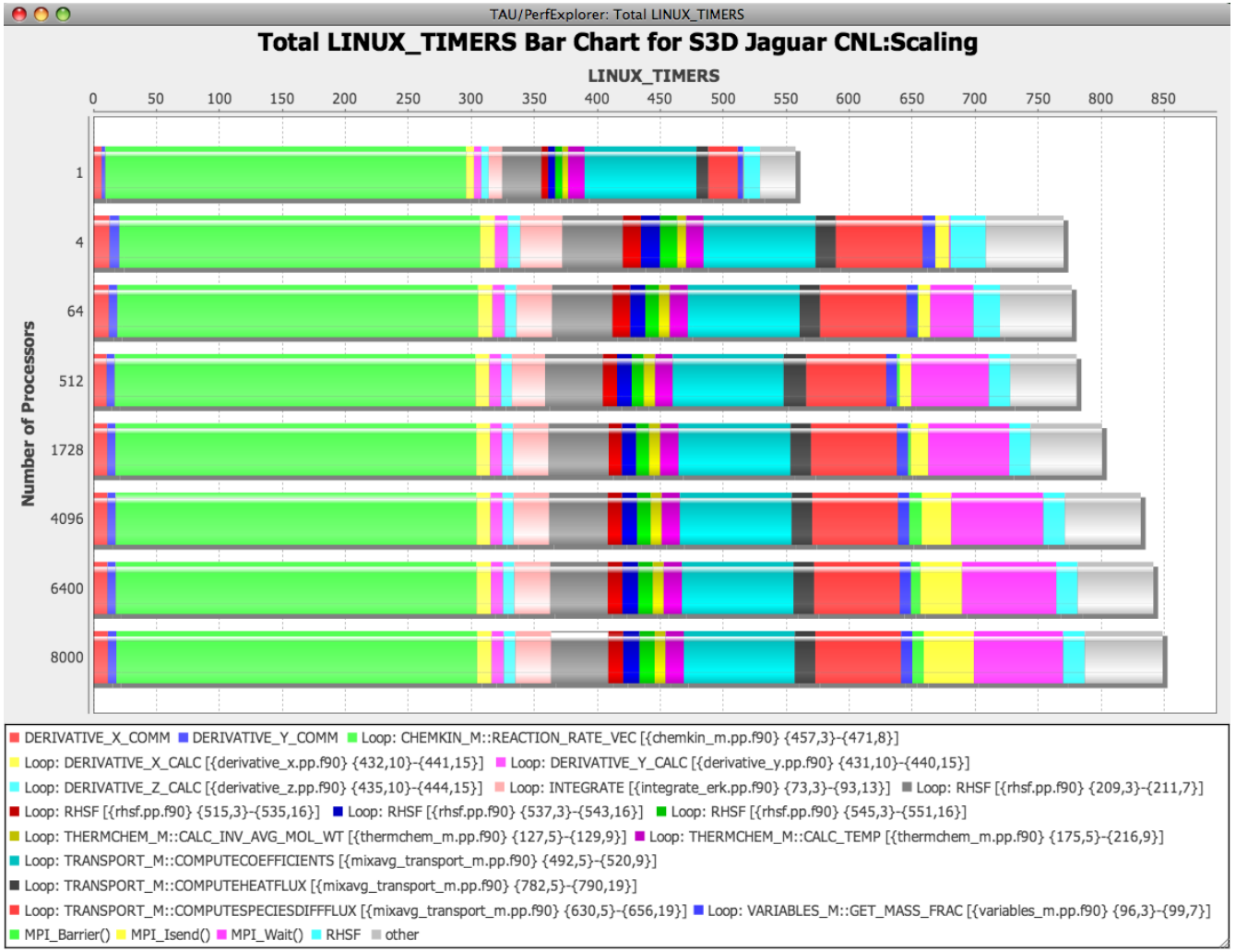
Usage Scenarios: Evaluate Scalability

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

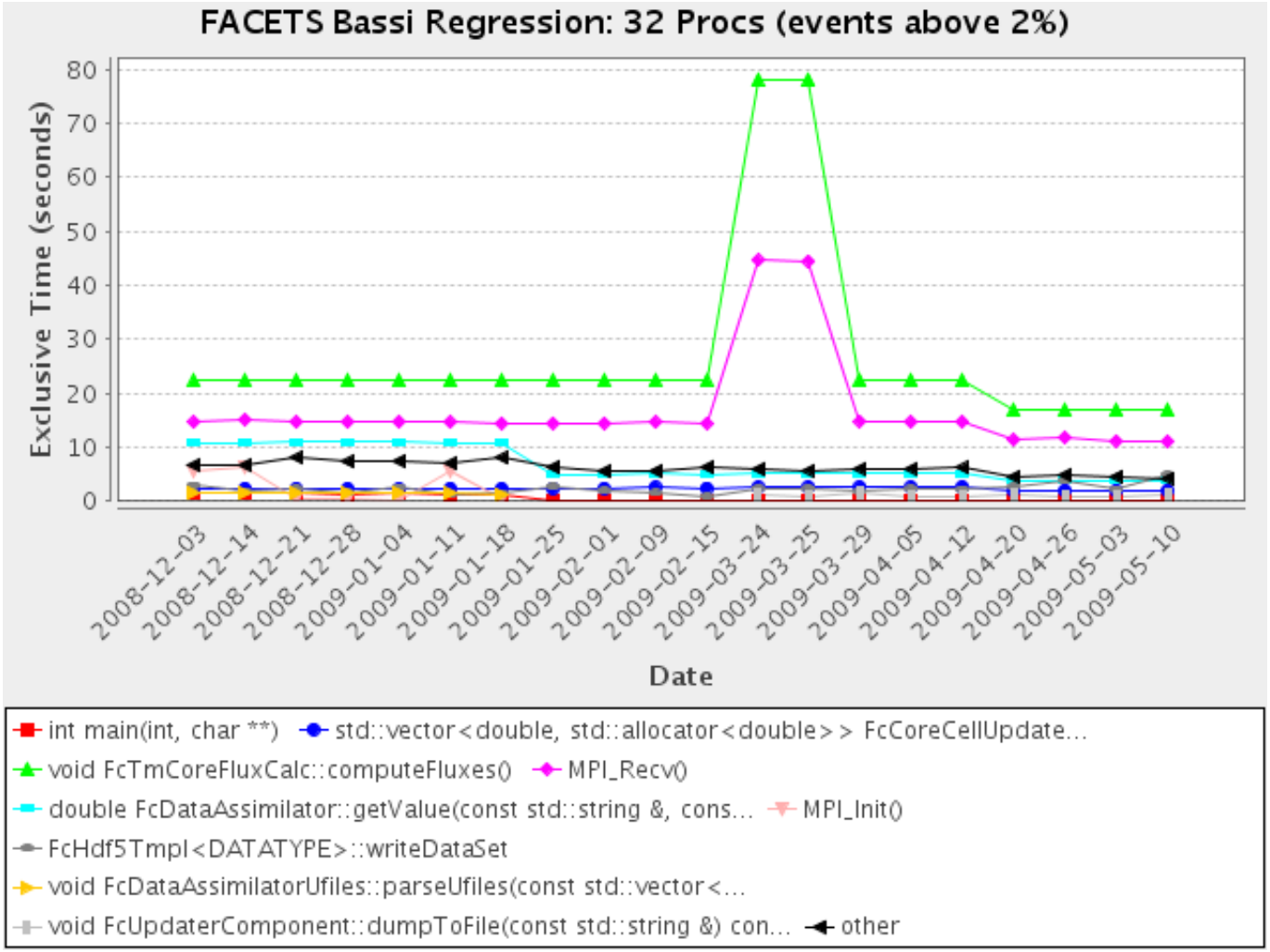




Usage Scenarios: Evaluate Scalability



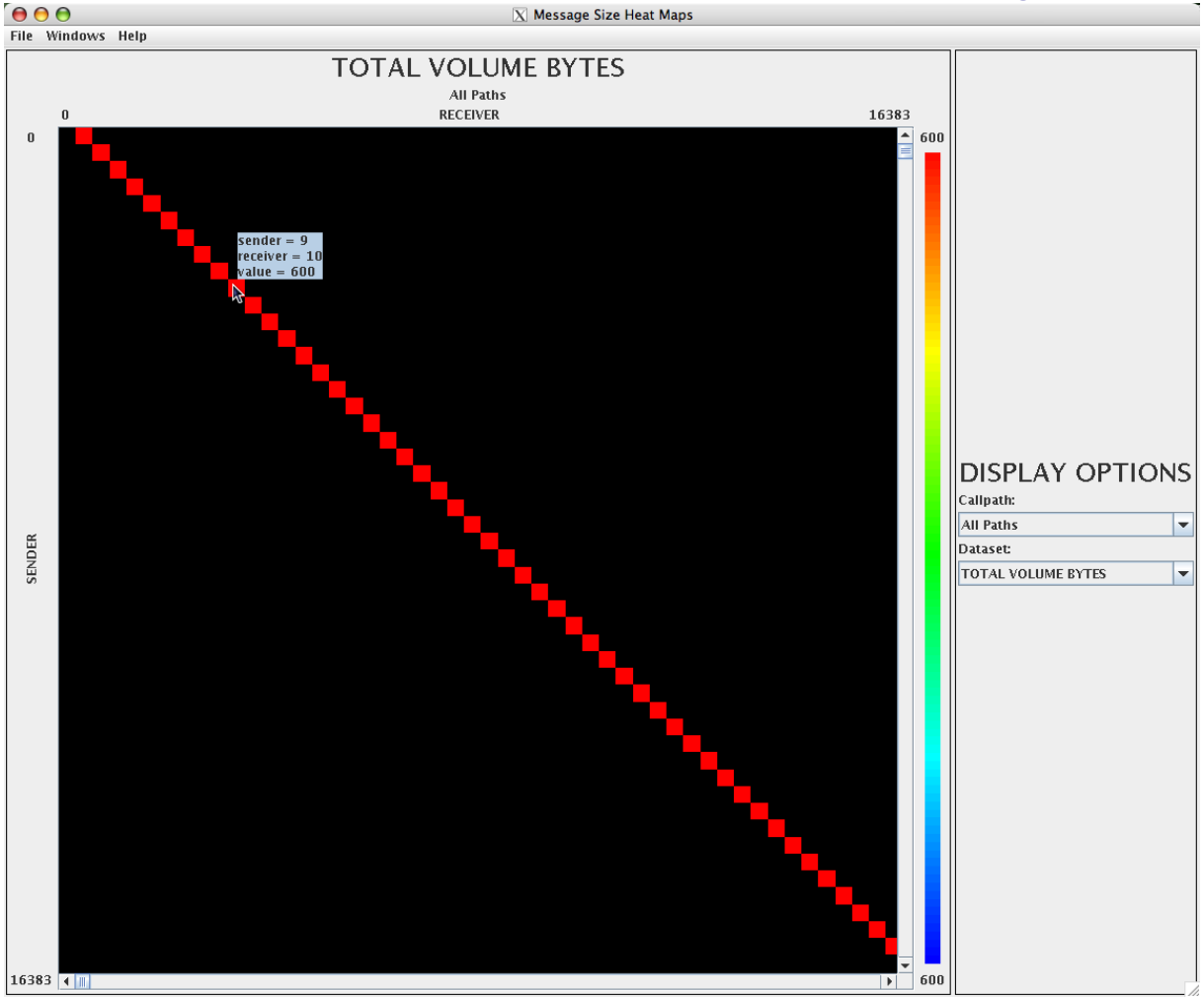
Performance Regression Testing



Evaluate Scalability using PerfExplorer Charts

Communication Matrix Display

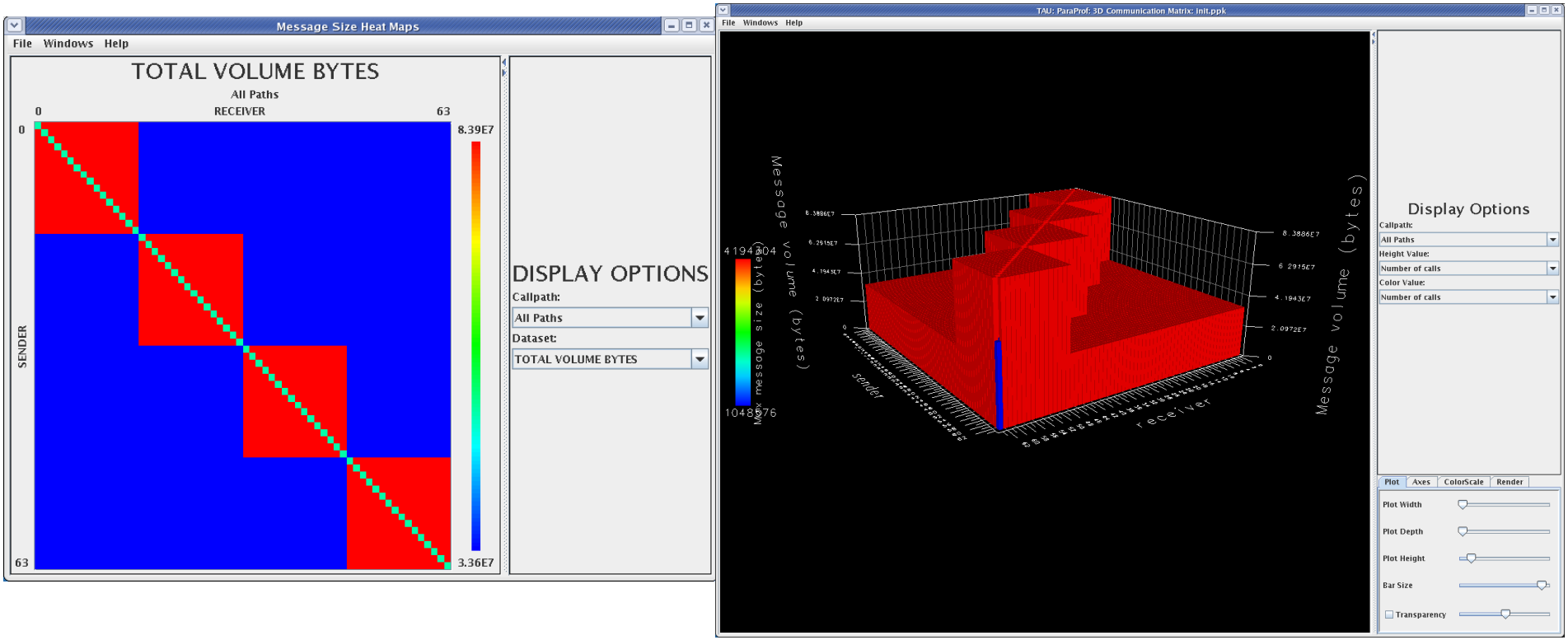
- Goal: What is the volume of inter-process communication? Along which calling path?



Evaluate Scalability using PerfExplorer Charts

Communication Matrix Display

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



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

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

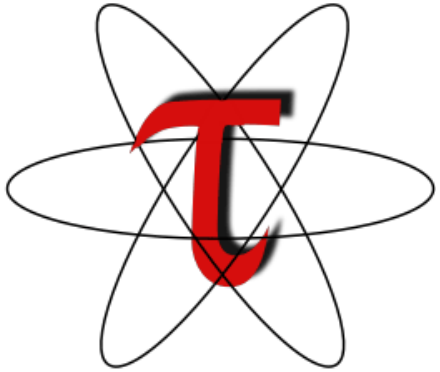
% setenv TAU_CALLPATH_DEPTH 2

% setenv TAU_TRACK_HEAP 1

Binary Rewriting: DyninstAPI [U.Wisc] and TAU

```
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%
```

Using PAPI and TAU



Hardware Counters

Hardware performance counters available on most modern microprocessors can provide insight into:

1. Whole program timing
2. Cache behaviors
3. Branch behaviors
4. Memory and resource access patterns
5. Pipeline stalls
6. Floating point efficiency
7. Instructions per cycle

Hardware counter information can be obtained with:

1. Subroutine or basic block resolution
2. Process or thread attribution

What's PAPI?

- Open Source software from U. Tennessee, Knoxville
- http://icl.cs.utk.edu/papi
- Middleware to provide a consistent programming interface for the performance counter hardware found in most major micro-processors.
- Countable events are defined in two ways:
  - Platform-neutral *preset* events
  - Platform-dependent native events
- Presets can be derived from multiple *native events*
- All events are referenced by name and collected in EventSets

PAPI Utilities: *papi_avail*

PAPI Utilities: *papi_avail*

PAPI Utilities: *papi_avail*

PAPI Utilities: *papi_native_avail*

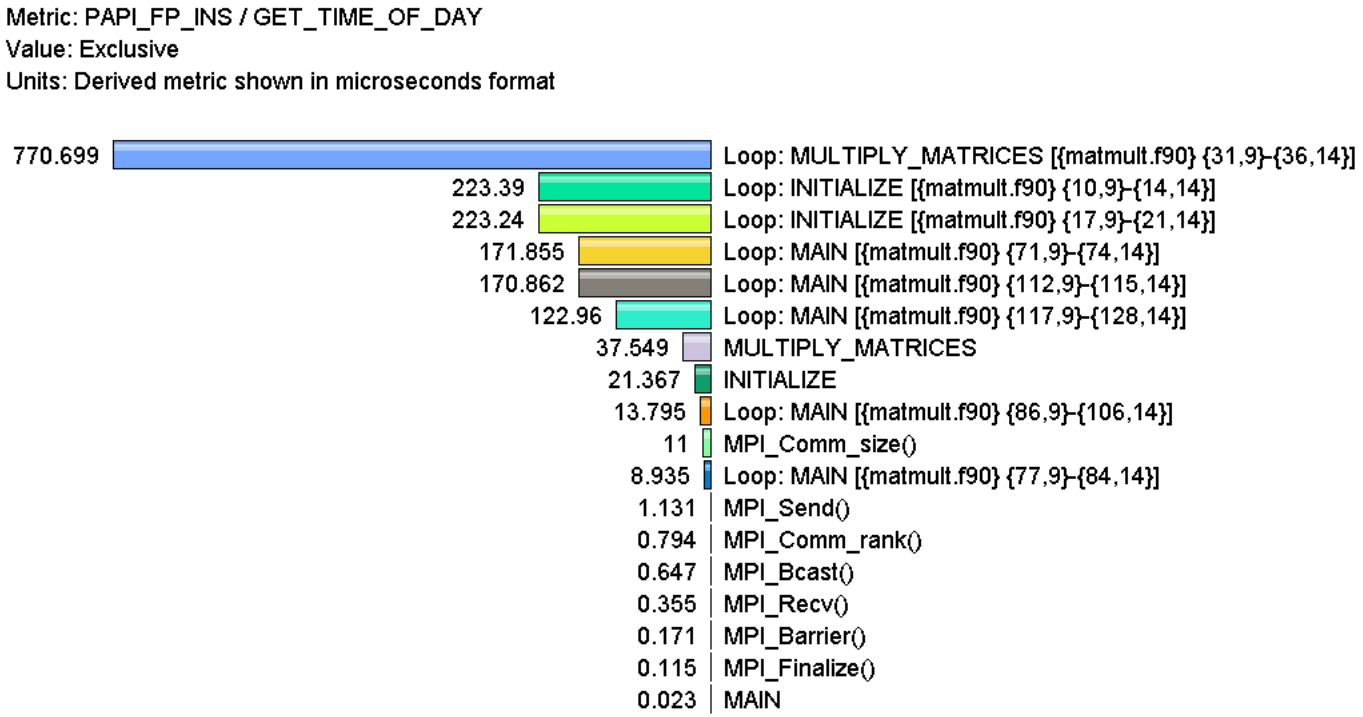
PAPI Utilities: *papi_event_chooser*

PAPI Utilities: *papi_event_chooser*

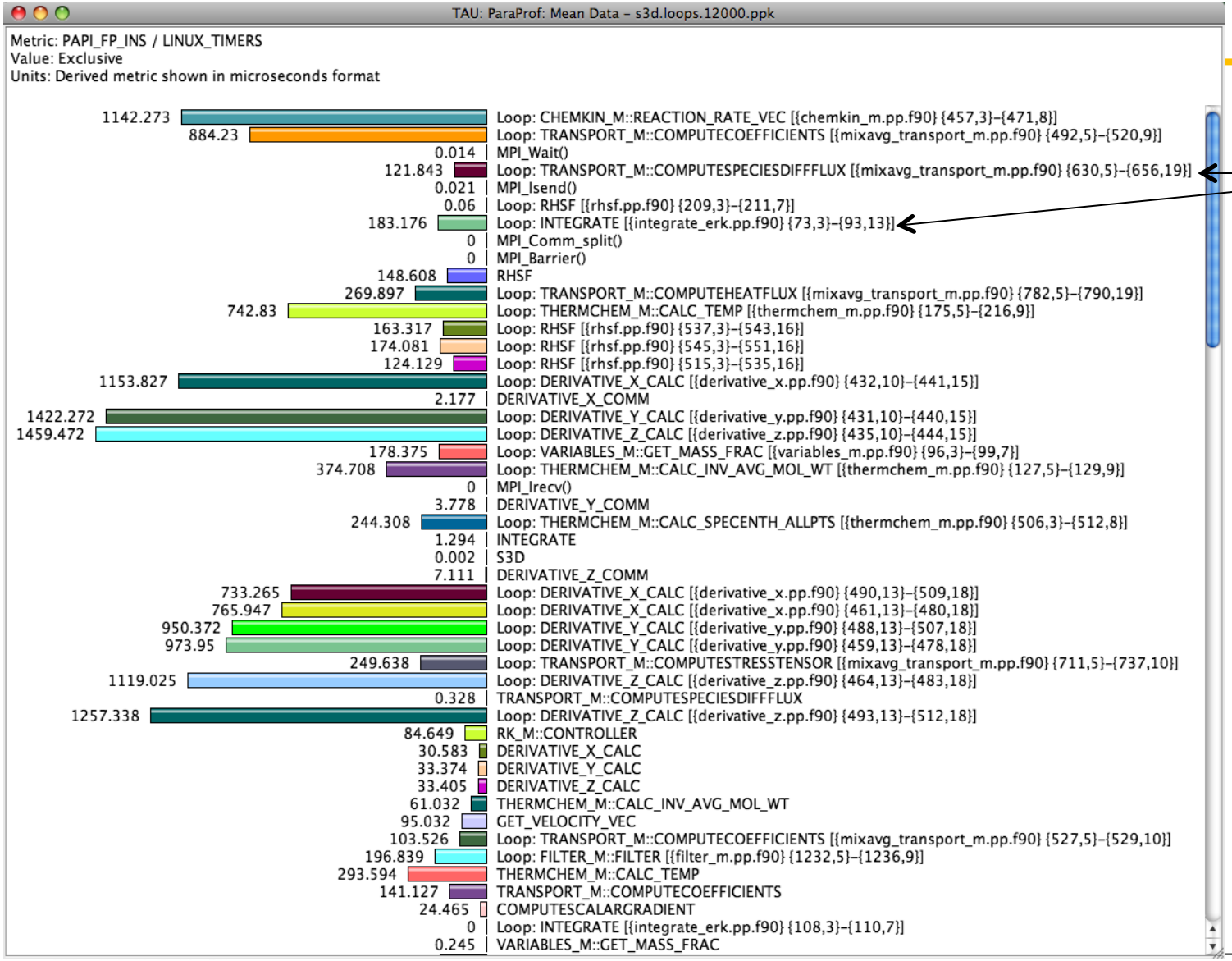
PAPI Utilities: *papi_event_chooser*

Usage Scenarios: Calculate mflops in Loops

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



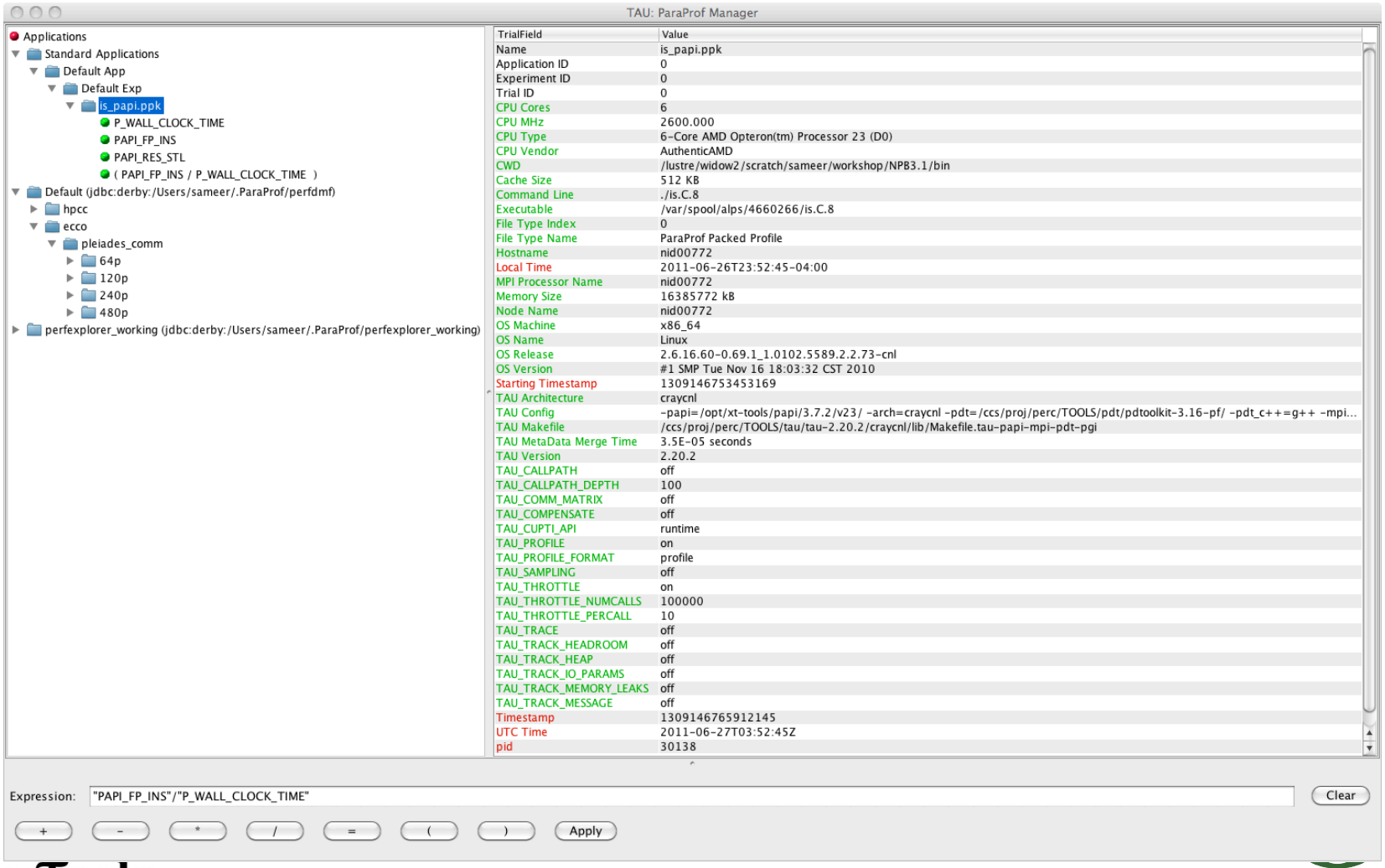
ParaProf: Mflops Sorted by Exclusive Time



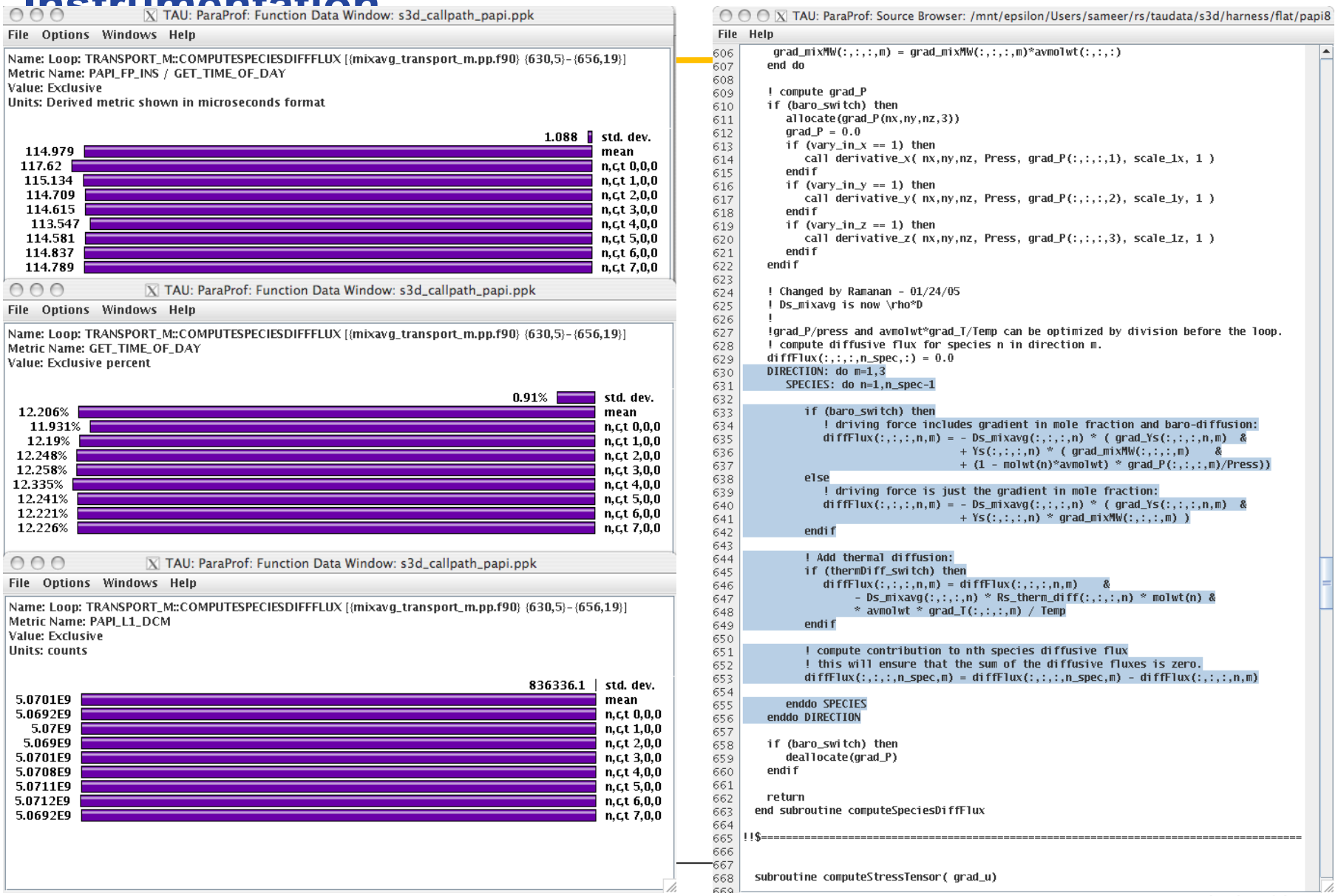
low mflops?

Generate a PAPI profile with 2 or more counters

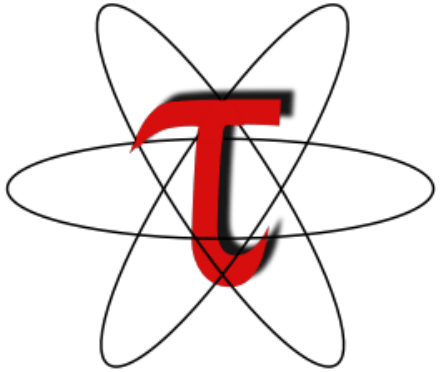
Derived Metrics in ParaProf



ParaProf's Source Browser: Loop Level Instrumentation



Estimation of tool intrusiveness



PAPI Utilities: *papi_cost*

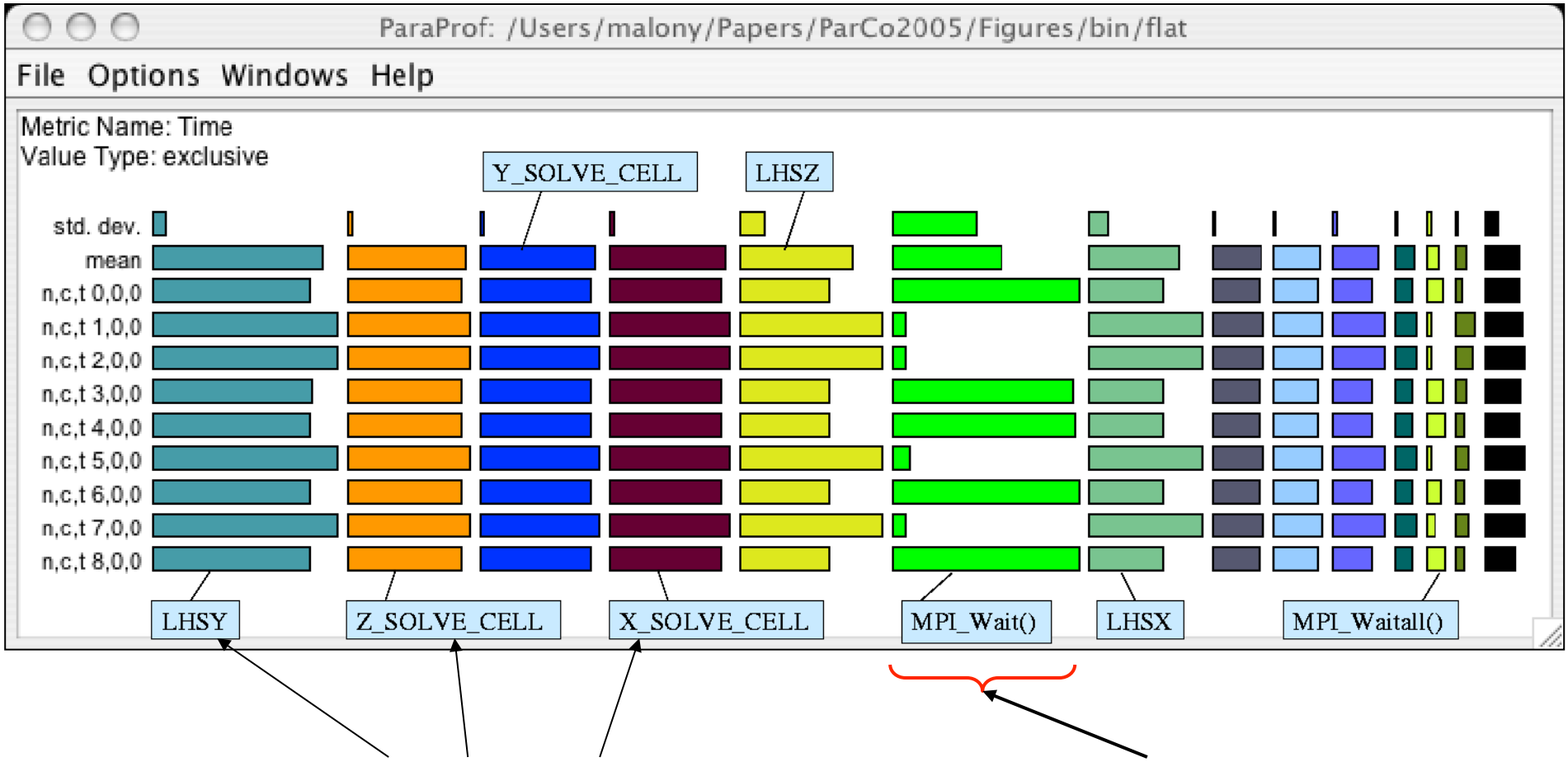
PAPI Utilities: *papi_cost*

PAPI Utilities: *papi_cost*

Profile Measurement – Three Flavors

- Flat profiles
  - Time (or counts) spent in each routine (nodes in callgraph).
  - Exclusive/inclusive time, no. of calls, child calls
  - E.g,: MPI_Send, foo, …
- Callpath Profiles
  - Flat profiles, **plus**
  - Sequence of actions that led to poor performance
  - Time spent along a calling path (edges in callgraph)
  - E.g., "main=> f1 => f2 => MPI_Send" shows the time spent in MPI_Send when called by f2, when f2 is called by f1, when it is called by main. Depth of this callpath = 4 (TAU_CALLPATH_DEPTH environment variable)
- Phase based profiles
  - Flat profiles, **plus**
  - Flat profiles under a phase (nested phases are allowed)
  - Default "main" phase has all phases and routines invoked outside phases
  - Supports static or dynamic (per-iteration) phases
  - E.g., "IO => MPI_Send" is time spent in MPI_Send in IO phase

Phase Profiling (NAS BT, Flat Profile)

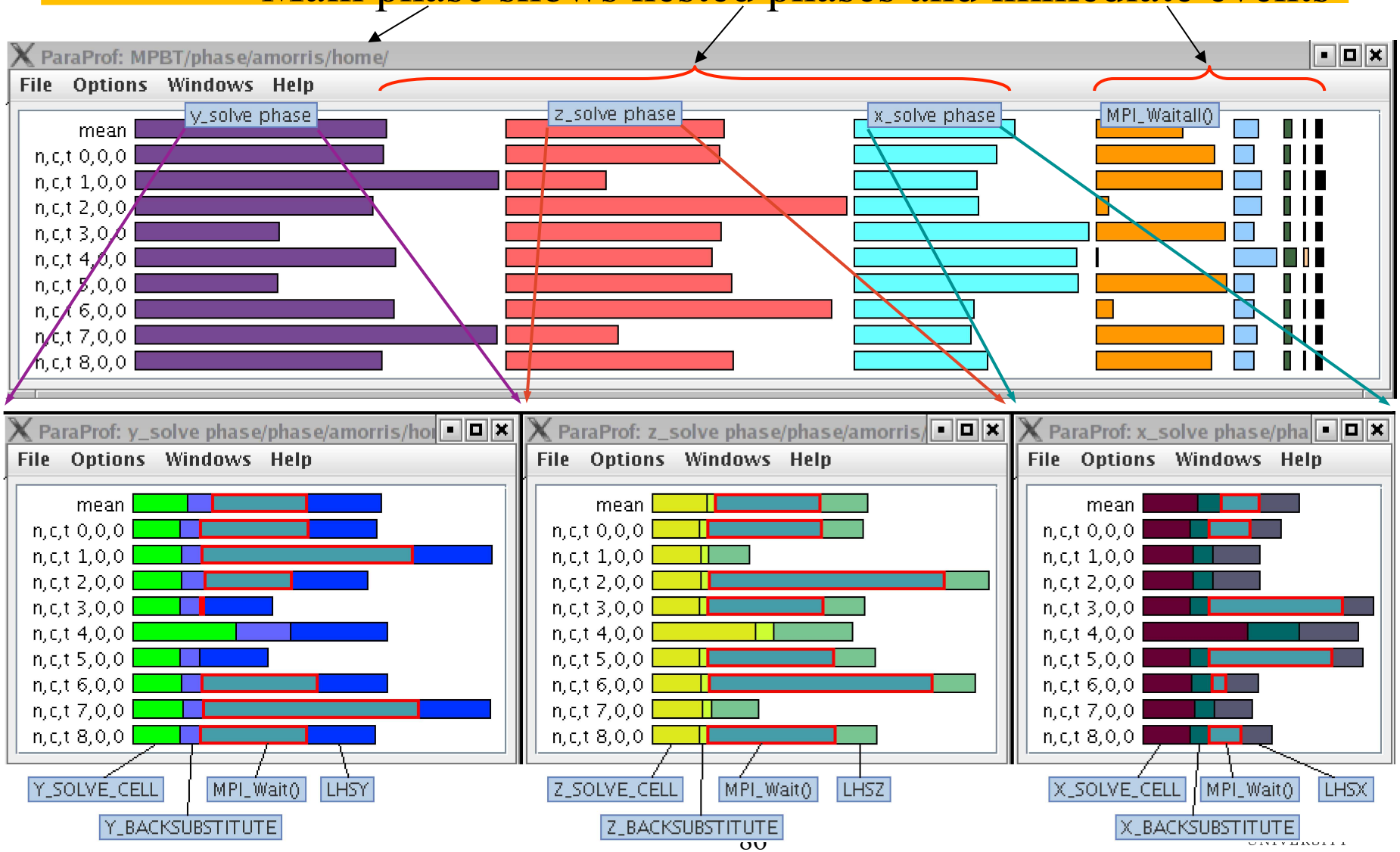


Application routine names reflect phase semantics

How is MPI_Wait() distributed relative to solver direction?

NAS BT – Phase Profile (Main and X, Y, Z)

Main phase shows nested phases and immediate events

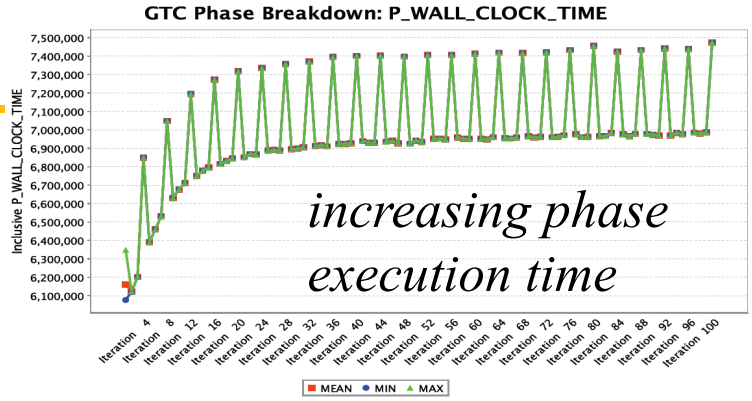


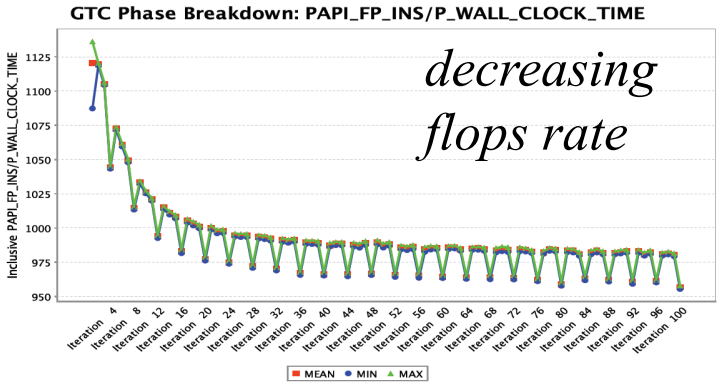
TAU Timers and Phases

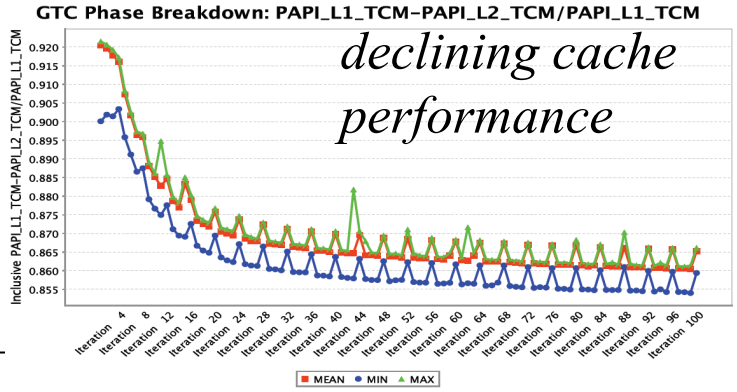
- Static timer
  - Shows time spent in all invocations of a routine (foo)
  - E.g., “foo()” 100 secs, 100 calls
- Dynamic timer
  - Shows time spent in each invocation of a routine
  - E.g., “foo() 3” 4.5 secs, “foo 10” 2 secs (invocations 3 and 10 respectively)
- Static phase
  - Shows time spent in all routines called (directly/indirectly) by a given routine (foo)
  - E.g., “foo() => MPI_Send()” 100 secs, 10 calls shows that a total of 100 secs were spent in MPI_Send() when it was called by foo.
- Dynamic phase
  - Shows time spent in all routines called by a given invocation of a routine.
  - E.g., “foo() 4 => MPI_Send()” 12 secs, shows that 12 secs were spent in MPI_Send when it was called by the 4th invocation of foo.

Performance Dynamics: Phase-Based Profiling

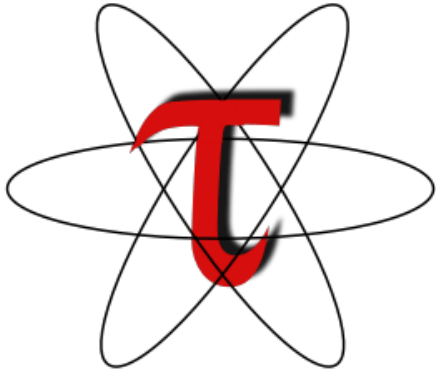
- Profile phases capture performance with respect to application-defined ‘phases’ of execution
  - Separate full profile produce for each phase
- GTC particle-in-cell simulation of fusion turbulence
- Phases assigned to iterations
- Data change affects cache







Memory and I/O evaluation



Library interposition/wrapping: tau_exec, tau_wrap

- TAU provides a wealth of options to measure the performance of an application
- Need to simplify TAU usage to easily evaluate performance properties, including I/O, memory, and communication
- Designed a new tool (*tau_exec*) that leverages runtime instrumentation by pre-loading measurement libraries
- Works on dynamic executables (default under Linux, not on IBM Blue Gene where we must compile with -dynamic)
- Substitutes I/O, MPI, and memory allocation/deallocation routines with instrumented calls
  - Interval events (e.g., time spent in write())
  - Atomic events (e.g., how much memory was allocated)
- Measure I/O and memory usage

TAU Execution Command (tau_exec)

- Uninstrumented execution (compiled with –Wl,–Bdynamic on BG/P)
  - % qsub –n 256 –t 10 ./a.out
- Track MPI performance (-T <options>)
  - % tau_exec –qsub -T bgqtimers,mpi,pdt -- qsub –n 256 –t 10 ./a.out
- Track I/O and MPI performance (MPI by default, use –T serial for serial)
  - % tau_exec –io qsub -T bgqtimers,mpi,pdt -- qsub –n 256 –t 10 ./a.out
- Track memory operations
  - % tau_exec –memory –env TAU_TRACK_MEMORY_LEAKS=1 qsub -T bgqtimers,mpi,pdt -- qsub –n 256 –t 10 ./a.out

Library wrapping: tau_gen_wrapper

- How to instrument an external library without source?
  - Source may not be available
  - Library may be too cumbersome to build (with instrumentation)
- Build a library wrapper tools
  - Used PDT to parse header files
  - Generate new header files with instrumentation files
  - Three methods to instrument: runtime preloading, linking, redirecting headers to re-define functions
- Application is instrumented
- Add the –optTauWrapFile=<wrapperdir>/link_options.tau file to TAU_OPTIONS env var while compiling with tau_cc.sh, etc.
- Wrapped library
  - Redirects references at routine callsite to a wrapper call
  - Wrapper internally calls the original
  - Wrapper has TAU measurement code

HDF5 Library Wrapping

```
$ tau_gen_wrapper hdf5.h /usr/lib/libhdf5.a -f select.tau

Usage : tau_gen_wrapper <header> <library> [-r|-d|-w (default)] [-g groupname] [-i
headerfile] [-c|-c++|-fortran] [-f <instr_req_file> ]
• instruments using runtime preloading (-r), or -Wl,-wrap linker (-w), redirection
of header file to redefine the wrapped routine (-d)
• instrumentation specification file (select.tau)
• -g group may be specified (hdf5)
• tau_exec loads libhdf5_wrap.so shared library using –loadlib=<libwrap_pkg.so>
• creates the wrapper/ directory with linkoptions.tau passed to the TAU_OPTIONS
environment variable using –optTauWrapFile=<file>

NODE 0;CONTEXT 0;THREAD 0:
---------------------------------------------------------------------------------------
%Time    Exclusive    Inclusive       #Call      #Subrs  Inclusive Name
              msec   total msec                          usec/call
---------------------------------------------------------------------------------------
100.0        0.057            1           1          13       1236 .TAU Application
 70.8        0.875        0.875           1           0        875 hid_t H5Fcreate()
  9.7         0.12         0.12           1           0        120 herr_t H5Fclose()
  6.0        0.074        0.074           1           0         74 hid_t H5Dcreate()
  3.1        0.038        0.038           1           0         38 herr_t H5Dwrite()
  2.6        0.032        0.032           1           0         32 herr_t H5Dclose()
  2.1        0.026        0.026           1           0         26 herr_t H5check_version()
  0.6        0.008        0.008           1           0          8 hid_t H5Screate_simple()
  0.2        0.002        0.002           1           0          2 herr_t H5Tset_order()
  0.2        0.002        0.002           1           0          2 hid_t H5Tcopy()
  0.1        0.001        0.001           1           0          1 herr_t H5Sclose()
```

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

Tracking I/O in static binaries (IBM Blue Gene)

- The linker can substitute TAU's I/O wrapper and intercept POSIX I/O Calls
- We can track parameters that flow through the I/O calls
- Configure TAU with –iowrappers
- Use –optTrackIO in TAU_OPTIONS

Tracking I/O in static binaries

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

I/O Calls Supported

Unbuffered I/O	Buffered I/O	Communication	Control	Asynchronous I/O
open	fopen	socket	fcntl	aio_read
open64	fopen64	pipe	rewind	aio_write
close	fdopen	socketpair	lseek	aio_suspend
read	freopen	bind	lseek64	aio_cancel
write	fclose	accept	fseek	aio_return
readv	fprintf	connect	dup	lio_listio
writev	fscanf	recv	dup2	
creat	fwrite	send	mkstep	
creat64	fread	sendto	tmpfile	
		recvfrom		
		pclose		

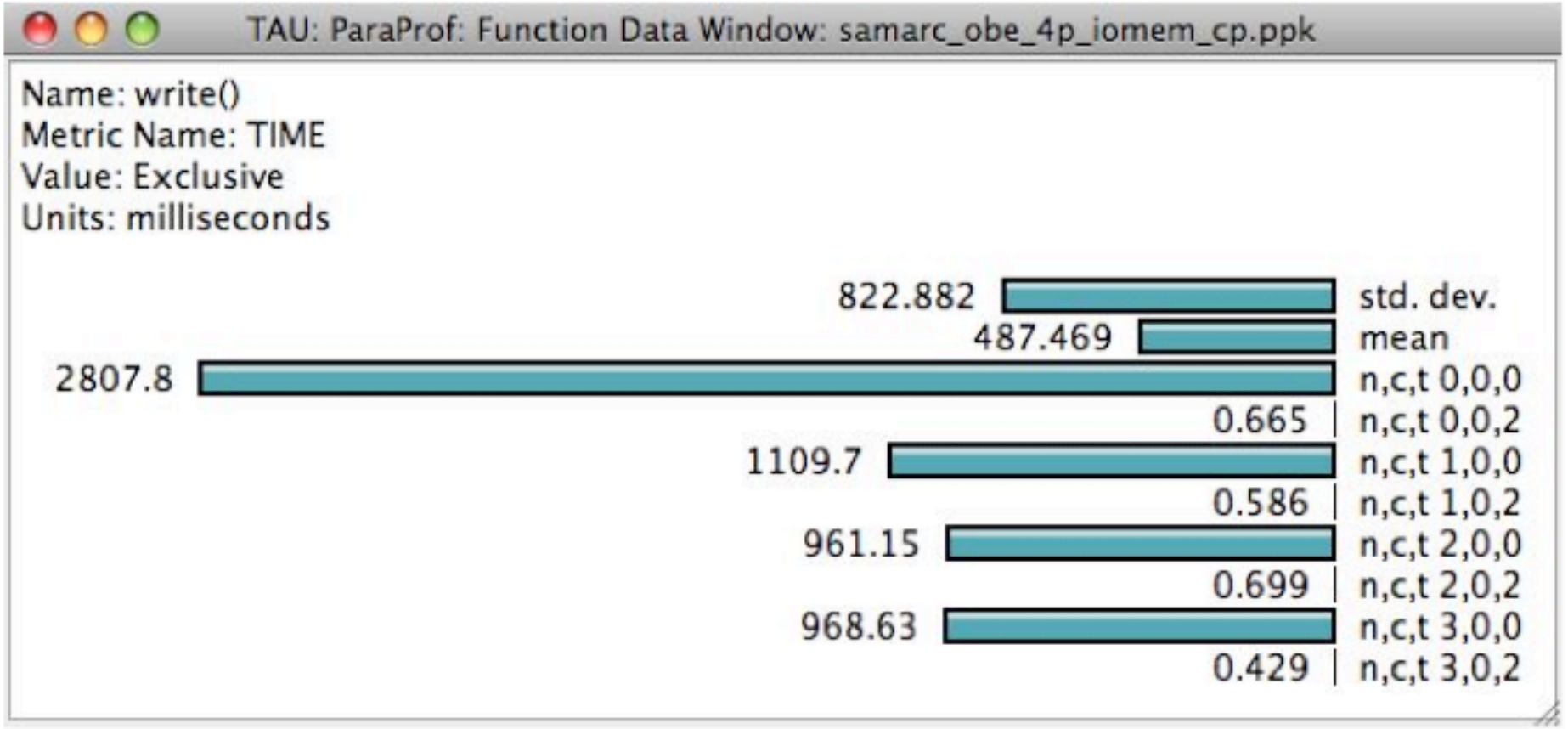
Tracking I/O in Each File

TAU: ParaProf: Context Events for thread: n,c,t, 1,0,0 - IOR_mana_iothreads_posix.ppk

File Windows Help

Name	Total	NumSamples	MaxValue	MinValue	MeanValue	Std. Dev.
.TAU application						
MPI_Finalize()						
MPI_Init()						
fscanf()						
read()						
Bytes Read	20,024	32	8,192	4	625.75	2,014.699
Bytes Read <file="/opt/openmpi/tm/intel/1.4/etc/openmpi-mca-params.conf">	2,812	1	2,812	2,812	2,812	0
Bytes Read <file="/opt/openmpi/tm/intel/1.4/share/openmpi/help-mpi-btl-openib.txt">	8,192	1	8,192	8,192	8,192	0
Bytes Read <file="/opt/openmpi/tm/intel/1.4/share/openmpi/mca-btl-openib-device-params.ini">	8,727	2	8,192	535	4,363.5	3,828.5
Bytes Read <file="/sys/class/infiniband/mthca0/node_type">	8	1	8	8	8	0
Bytes Read <file="/sys/class/infiniband/mthca0/ports/1/gids/0">	41	1	41	41	41	0
Bytes Read <file="/sys/class/infiniband_verbs/abi_version">	8	1	8	8	8	0
Bytes Read <file="/sys/class/infiniband_verbs/uverbs0/abi_version">	8	1	8	8	8	0
Bytes Read <file="/sys/class/infiniband_verbs/uverbs0/device/device">	24	3	8	8	8	0
Bytes Read <file="/sys/class/infiniband_verbs/uverbs0/device/vendor">	24	3	8	8	8	0
Bytes Read <file="/sys/class/infiniband_verbs/uverbs0/ibdev">	64	1	64	64	64	0
Bytes Read <file="/sys/devices/system/cpu/cpu0/topology/core_id">	7	1	7	7	7	0
Bytes Read <file="/sys/devices/system/cpu/cpu0/topology/physical_package_id">	7	1	7	7	7	0
Bytes Read <file="/sys/devices/system/cpu/cpu1/topology/core_id">	7	1	7	7	7	0
Bytes Read <file="/sys/devices/system/cpu/cpu1/topology/physical_package_id">	7	1	7	7	7	0
Bytes Read <file="/sys/devices/system/cpu/cpu2/topology/core_id">	7	1	7	7	7	0
Bytes Read <file="/sys/devices/system/cpu/cpu2/topology/physical_package_id">	7	1	7	7	7	0
Bytes Read <file="/sys/devices/system/cpu/cpu3/topology/core_id">	7	1	7	7	7	0
Bytes Read <file="/sys/devices/system/cpu/cpu3/topology/physical_package_id">	7	1	7	7	7	0
Bytes Read <file="/sys/devices/system/cpu/cpu4/topology/core_id">	7	1	7	7	7	0
Bytes Read <file="/sys/devices/system/cpu/cpu4/topology/physical_package_id">	7	1	7	7	7	0
Bytes Read <file="/sys/devices/system/cpu/cpu5/topology/core_id">	7	1	7	7	7	0
Bytes Read <file="/sys/devices/system/cpu/cpu5/topology/physical_package_id">	7	1	7	7	7	0
Bytes Read <file="/sys/devices/system/cpu/cpu6/topology/core_id">	7	1	7	7	7	0
Bytes Read <file="/sys/devices/system/cpu/cpu6/topology/physical_package_id">	7	1	7	7	7	0
Bytes Read <file="/sys/devices/system/cpu/cpu7/topology/core_id">	7	1	7	7	7	0
Bytes Read <file="/sys/devices/system/cpu/cpu7/topology/physical_package_id">	7	1	7	7	7	0
Bytes Read <file="pipe">	4	1	4	4	4	0
READ Bandwidth (MB/s)	2,932.118	32	1,170.286	0.001	91.629	269.282
READ Bandwidth (MB/s) <file="/opt/openmpi/tm/intel/1.4/etc/openmpi-mca-params.conf">	312.444	1	312.444	312.444	312.444	0
READ Bandwidth (MB/s) <file="/opt/openmpi/tm/intel/1.4/share/openmpi/help-mpi-btl-openib.txt">	1,170.286	1	1,170.286	1,170.286	1,170.286	0
READ Bandwidth (MB/s) <file="/opt/openmpi/tm/intel/1.4/share/openmpi/mca-btl-openib-device-params.i	1,291.5	2	1,024	267.5	645.75	378.25
READ Bandwidth (MB/s) <file="/sys/class/infiniband/mthca0/node_type">	4	1	4	4	4	0
READ Bandwidth (MB/s) <file="/sys/class/infiniband/mthca0/ports/1/gids/0">	0.304	1	0.304	0.304	0.304	0
READ Bandwidth (MB/s) <file="/sys/class/infiniband_verbs/abi_version">	4	1	4	4	4	0
READ Bandwidth (MB/s) <file="/sys/class/infiniband_verbs/uverbs0/abi_version">	4	1	4	4	4	0
READ Bandwidth (MB/s) <file="/sys/class/infiniband_verbs/uverbs0/device/device">	16	3	8	4	5.333	1.886
READ Bandwidth (MB/s) <file="/sys/class/infiniband_verbs/uverbs0/device/vendor">	20	3	8	4	6.667	1.886
READ Bandwidth (MB/s) <file="/sys/class/infiniband_verbs/uverbs0/ibdev">	32	1	32	32	32	0

Time Spent in POSIX I/O write()

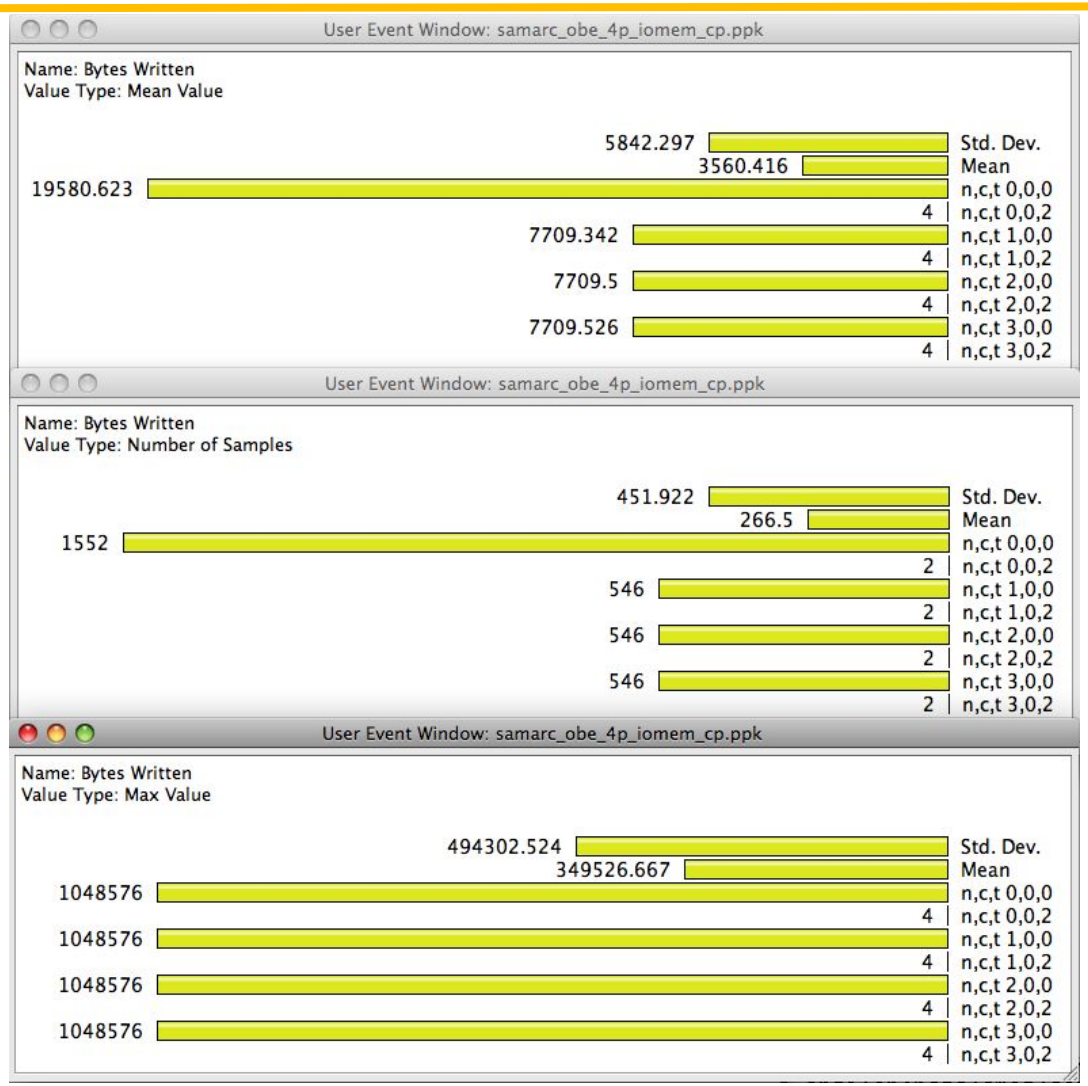


Volume of I/O by File, Memory

TAU: ParaProf: Context Events for thread: n,c,t, 1,0,0 – samarc_obe_4p_iomem_cp.ppk

Name	Total	MeanValue	NumSamples	MinValue	MaxValue	Std. Dev.
.TAU application						
read()						
fopen64()						
fclose()						
OurMain()						
malloc size	25,235	1,097.174	23	11	12,032	2,851.143
free size	22,707	1,746.692	13	11	12,032	3,660.642
OurMain [{wrapper.py}{3}]						
read()						
malloc size	3,877	323.083	12	32	981	252.72
free size	1,536	219.429	7	32	464	148.122
fopen64()						
fclose()						
<module> [{obe.py}{8}]						
writeRestartData [{samarcInterface.py}{145}]						
samarcWriteRestartData						
write()						
WRITE Bandwidth (MB/s) <file="samarc/restore.00002/nodes.00004/proc.00001">		74.565	117	0	2,156.889	246.386
WRITE Bandwidth (MB/s) <file="samarc/restore.00001/nodes.00004/proc.00001">		77.594	117	0	1,941.2	228.366
WRITE Bandwidth (MB/s)		76.08	234	0	2,156.889	237.551
Bytes Written <file="samarc/restore.00002/nodes.00004/proc.00001">	2,097,552	17,927.795	117	1	1,048,576	133,362.946
Bytes Written <file="samarc/restore.00001/nodes.00004/proc.00001">	2,097,552	17,927.795	117	1	1,048,576	133,362.946
Bytes Written	4,195,104	17,927.795	234	1	1,048,576	133,362.946
open64()						

Bytes Written



Memory Leaks in MPI

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

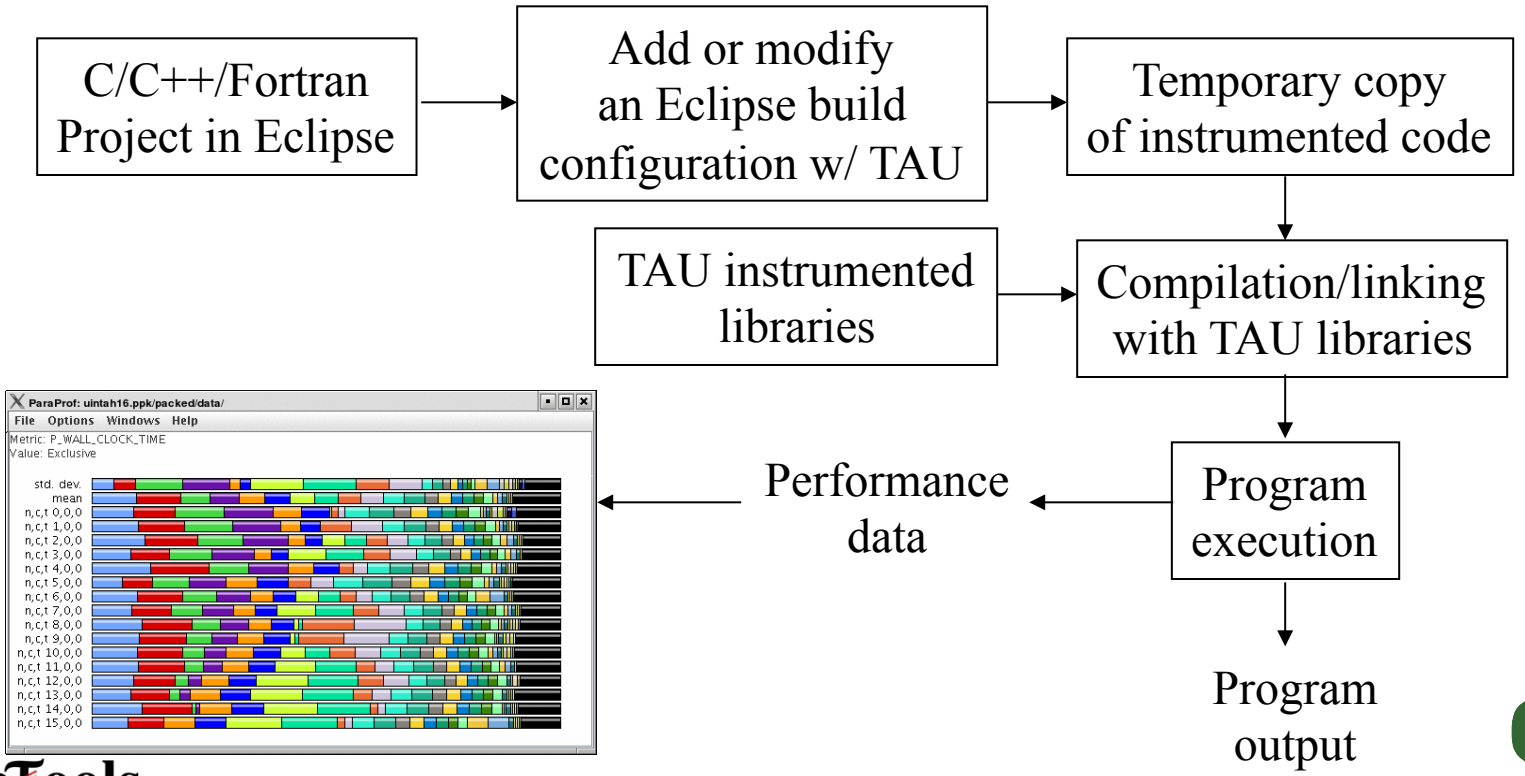
Name △	Total	MeanValue	NumSamples	MaxValue	MinValue	Std. Dev.
▼ .TAU application						
▼ MPI_Finalize()						
free size	23,901,253	22,719.822	1,052	2,099,200	2	186,920.948
malloc size	5,013,902	65,972.395	76	5,000,000	2	569,732.815
MEMORY LEAK!	5,000,264	500,026.4	10	5,000,000	3	1,499,991.2
▼ read()						
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()						

TAU Integration with IDEs

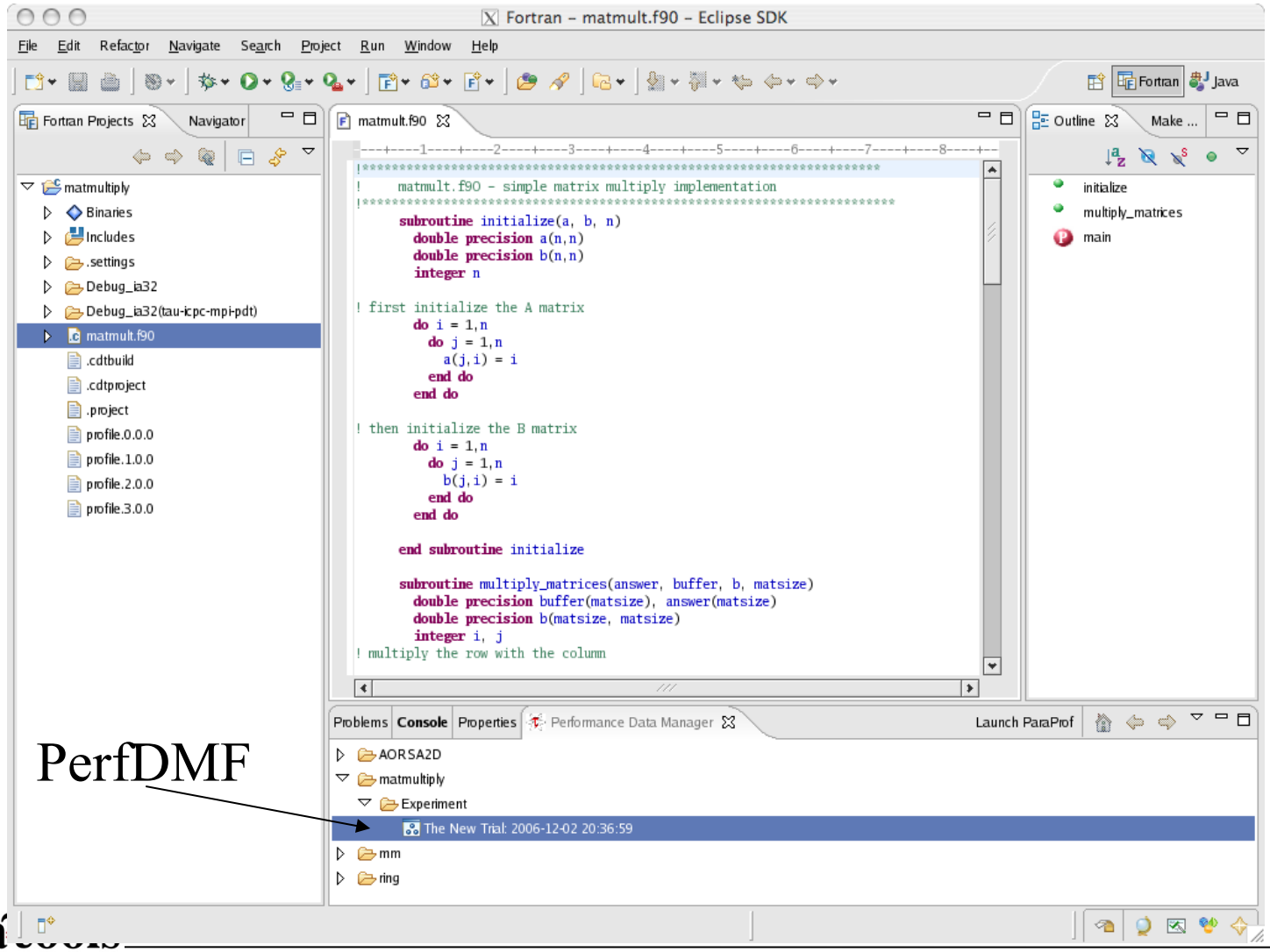
- High performance software development environments
  - Tools may be complicated to use
  - Interfaces and mechanisms differ between platforms / OS
- Integrated development environments
  - Consistent development environment
  - Numerous enhancements to development process
  - Standard in industrial 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

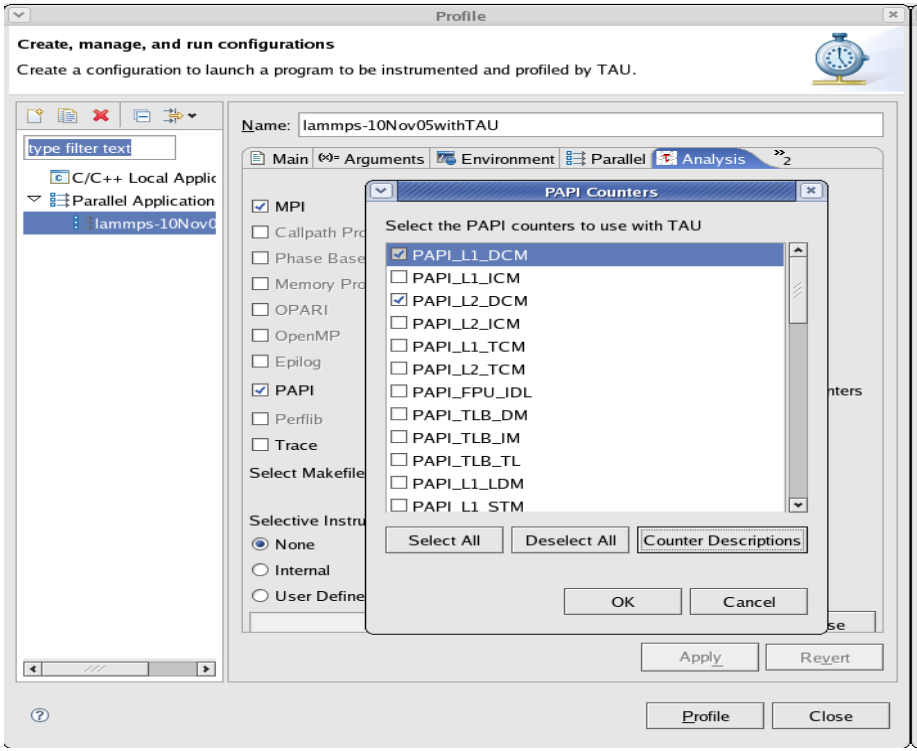
- Provide an interface for configuring TAU's automatic instrumentation within Eclipse's build system
- Manage runtime configuration settings and environment variables for execution of TAU instrumented programs



TAU and Eclipse



Choosing PAPI Counters with TAU in Eclipse



Counter	Definition
PAPI_L1_DCM	Level 1 data cache misses
PAPI_L1_ICM	Level 1 instruction cache misses
PAPI_L2_DCM	Level 2 data cache misses
PAPI_L2_ICM	Level 2 instruction cache misses
PAPI_L1_TCM	Level 1 cache misses
PAPI_L2_TCM	Level 2 cache misses
PAPI_FPU_IDL	Cycles floating point units are idle
PAPI_TLB_DM	Data translation lookaside buffer misses
PAPI_TLB_IM	Instruction translation lookaside buffer misses
PAPI_TLB_TL	Total translation lookaside buffer misses
PAPI_L1_LDM	Level 1 load misses
PAPI_L1_STM	Level 1 store misses
PAPI_L2_LDM	Level 2 load misses
PAPI_L2_STM	Level 2 store misses
PAPI_STL_ICY	Cycles with no instruction issue
PAPI_HW_INT	Hardware interrupts
PAPI_BR_TKN	Conditional branch instructions taken
PAPI_BR_MSP	Conditional branch instructions mispredicted
PAPI_TOT_INS	Instructions completed
PAPI_FP_INS	Floating point instructions
PAPI_BR_INS	Branch instructions
PAPI_VEC_INS	Vector/SIMD instructions
PAPI_RES_STL	Cycles stalled on any resource
PAPI_TOT_CYC	Total cycles
PAPI_L1_DCH	Level 1 data cache hits
PAPI_L2_DCH	Level 2 data cache hits
PAPI_L1_DCA	Level 1 data cache accesses
PAPI_L2_DCA	Level 2 data cache accesses
PAPI_L2_DCR	Level 2 data cache reads
PAPI_L2_DCW	Level 2 data cache writes

Labs!

Lab Instructions

Lab Instructions

More Information

- PAPI References:
  - PAPI documentation page available from the PAPI website:
    http://icl.cs.utk.edu/papi/
- TAU References:
  - TAU Users Guide and papers available from the TAU website:
    http://tau.uoregon.edu/
- VAMPIR References
  - VAMPIR website
    http://www.vampir.eu/
- Scalasca/KOJAK References
  - Scalasca documentation page
    http://www.scalasca.org/
- Eclipse PTP References
  - Documentation available from the Eclipse PTP website:
    http://www.eclipse.org/ptp/

Acknowledgements

- Department of Energy
  - Office of Science
  - Argonne National Laboratory
  - ORNL
  - NNSA/ASC Trilabs (SNL, LLNL, LANL)
- HPCMP DoD PETTT Program
- National Science Foundation
- University of Tennessee

- University of Oregon
  - Allen D. Malony, K. Huck, W. Spear
- TU Dresden
  - Holger Brunst, Andreas Knupfer
  - Wolfgang Nagel
- Research Centre Juelich, Germany
  - Bernd Mohr
  - Felix Wolf

UNIVERSITY
OF OREGON