TAU

Ninth DOE ACTS Collection Workshop

Building Robust, Scalable, and Portable Software

NERSC OSF, Thursday, 9:30am, August 21, 2008

Sameer Shende

Director, Performance Research Laboratory

University of Oregon, Eugene, OR

sameer@cs.uoregon.edu

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

UNIVERSITY
OF OREGON

Acknowledgements: University of Oregon

- Dr. Allen D. Malony, Professor
- Alan Morris, Senior software engineer
- Wyatt Spear, Software engineer
- Scott Biersdorff, Software engineer
- Dr. Matt Sottile, Research faculty
- Dr. Robert Yelle, Research faculty
- Kevin Huck, Ph.D. student
- Aroon Nataraj, Ph.D. student
- Shangkar Mayanglambam, Ph.D. student
- Brad Davidson, Systems administrator

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.)
- TAU runs on all HPC platforms and it is free (BSD style license)
- TAU has instrumentation, measurement and analysis tools
- Using TAU requires setting a couple of environment variables and substituting the name of the compiler with a TAU shell script

Using TAU: A Tutorial

Using TAU: A brief Introduction

- To instrument source code using PDT
  - Choose an appropriate TAU stub makefile in <arch>/lib:

% setenv TAU_MAKEFILE /usr/common/acts/TAU/tau-2.17.2/x86_64/lib/Makefile.tau-pathcc-mpi-pdt

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

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 – Examples

% cd /usr/common/acts/TAU/tau-2.17.2/x86_64/lib; ls Makefile.*
Makefile.tau-pathcc-pdt
Makefile.tau-pathcc-mpi-pdt
Makefile.tau-pathcc-callpath-mpi-pdt
Makefile.tau-pathcc-mpi-pdt-trace
Makefile.tau-pathcc-mpi-compensate-pdt
Makefile.tau-multiplecounters-pathcc-mpi-papi-pdt
Makefile.tau-multiplecounters-pathcc-mpi-papi-pdt-trace
Makefile.tau-pathcc-pthread-pdt…

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

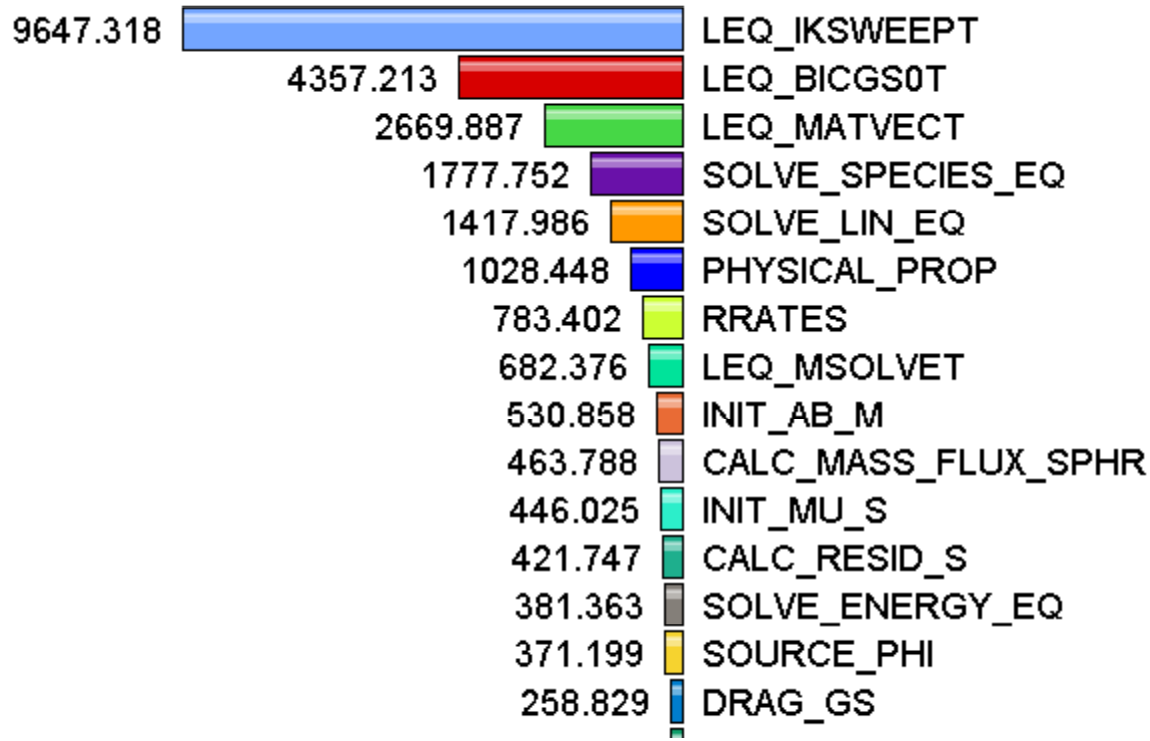
Makefile.tau-mpi-pdt

  - Supports MPI instrumentation & PDT for automatic source instrumentation
  - % setenv TAU_MAKEFILE /usr/common/acts/TAU/tau-2.17.2/x86_64/lib/Makefile.tau-pathcc-mpi-pdt

Usage Scenarios: Routine Level Profile

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

Metric: P_VIRTUAL_TIME
Value: Exclusive
Units: seconds



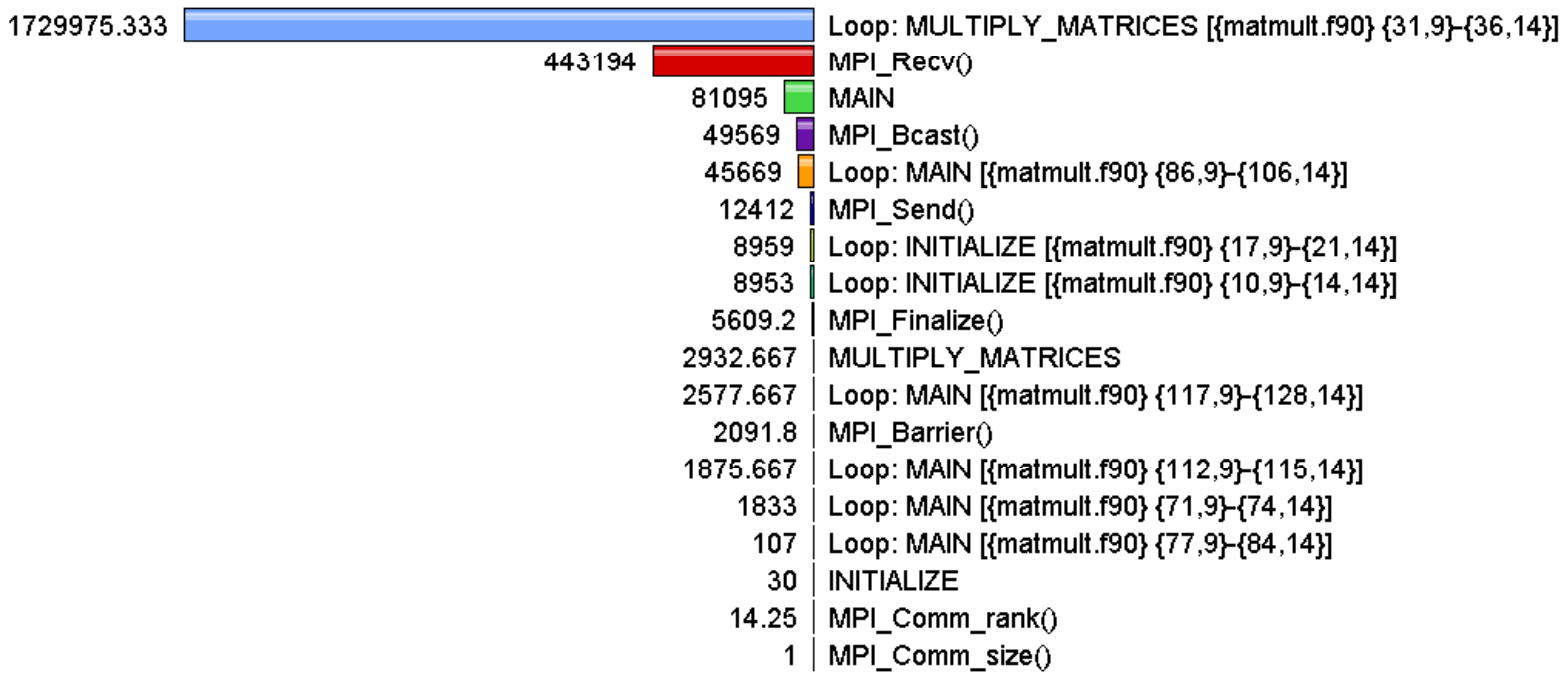
Solution: Generating a flat profile with MPI

```
% setenv TAU_MAKEFILE /usr/common/acts/TAU/tau-2.17.2/x86_64
                                /lib/Makefile.tau-pathcc-mpi-pdt
% set path=(/usr/common/acts/TAU/tau-2.17.2/x86_64/bin $path)
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)
% setenv TAU_THROTTLE 1
% qsub run.job
% paraprof --pack app.ppk
  Move the app.ppk file to your desktop.

% paraprof app.ppk
```

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



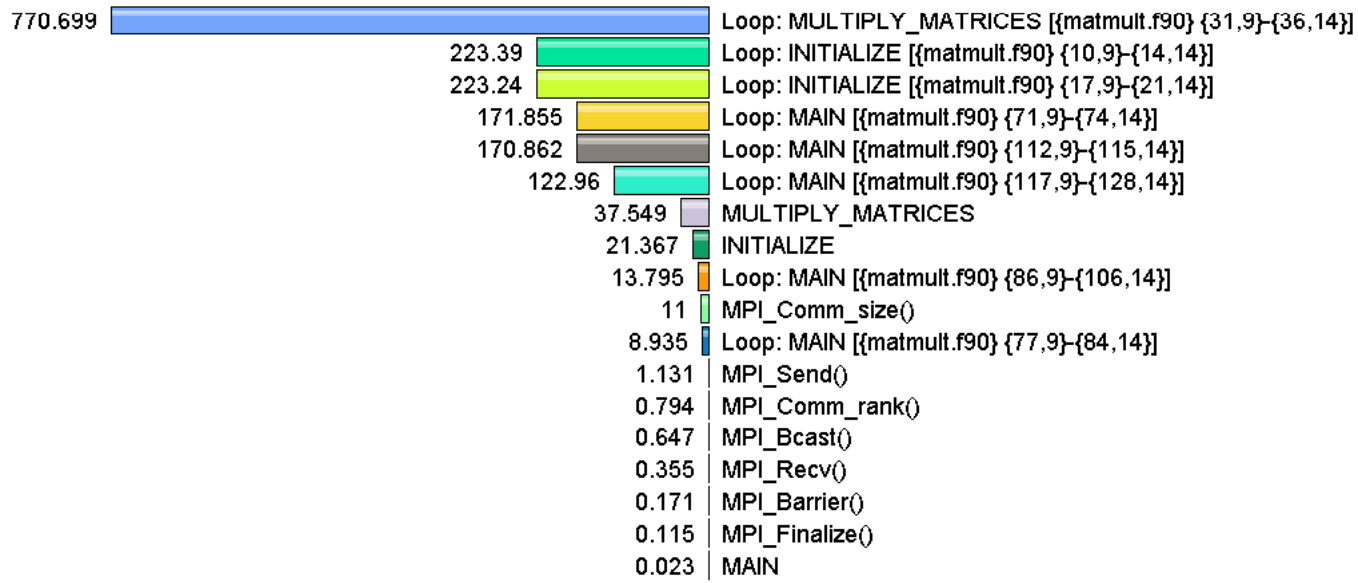
Solution: Generating a loop level profile

```
% setenv TAU_MAKEFILE /usr/common/acts/TAU/tau-2.17.2/x86_64
                                /lib/Makefile.tau-pathcc-mpi-pdt
% setenv TAU_OPTIONS '-optTauSelectFile=select.tau –optVerbose'
% cat select.tau
  BEGIN_INSTRUMENT_SECTION
  loops routine="#"
  END_INSTRUMENT_SECTION

% set path=(/usr/common/acts/TAU/tau-2.17.2/x86_64/bin $path)
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)
% qsub  run.job
% paraprof --pack app.ppk
  Move the app.ppk file to your desktop.

% paraprof app.ppk
```

Usage Scenarios: MFlops in Loops

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

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

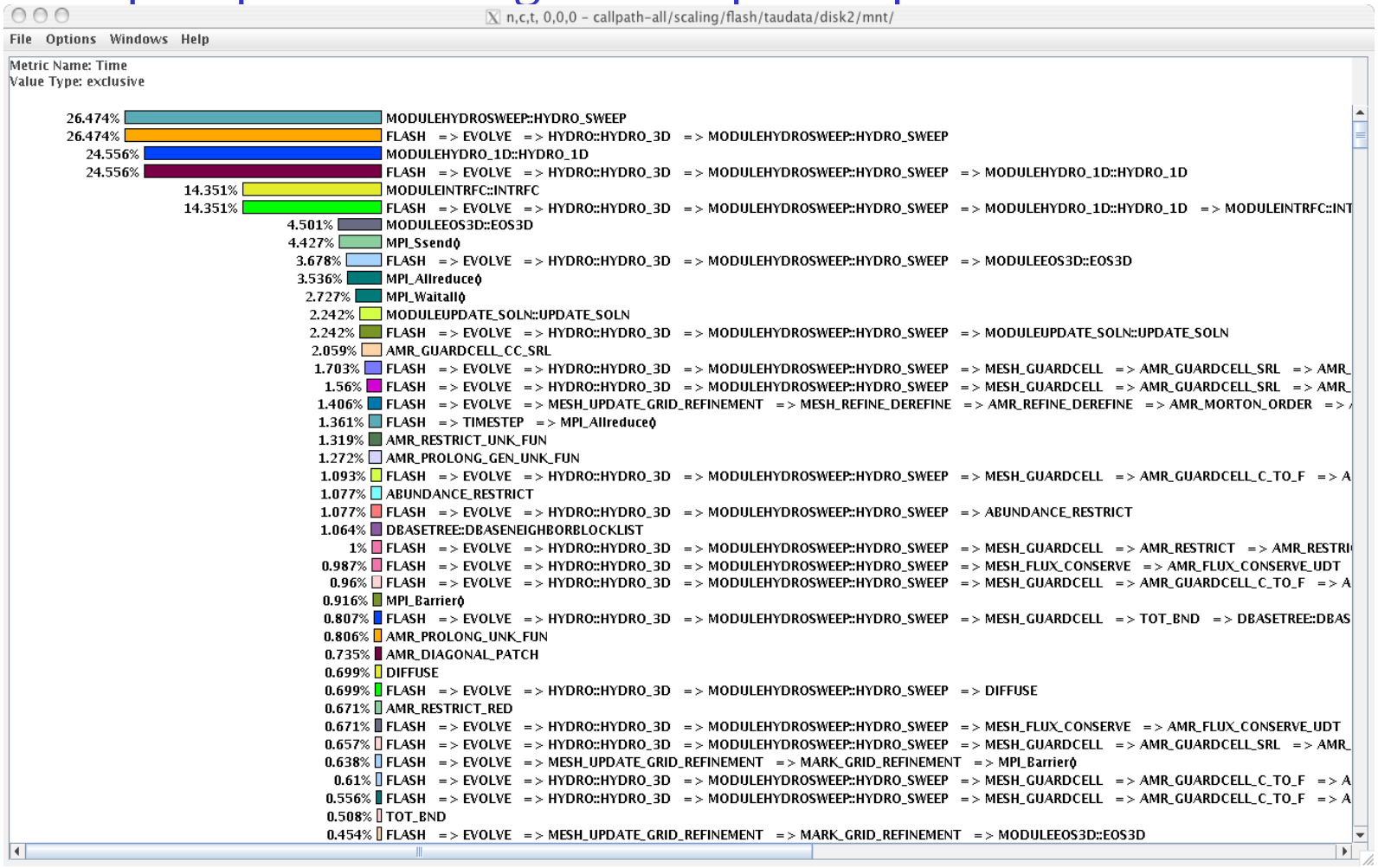


Generate a PAPI profile with 2 or more counters

```
% setenv TAU_MAKEFILE /usr/common/acts/TAU/tau-2.17.2/x86_64
              /lib/Makefile.tau-multiplecounters-pathcc-papi-mpi-pdt
% setenv TAU_OPTIONS '-optTauSelectFile=select.tau –optVerbose'
% cat select.tau
  BEGIN_INSTRUMENT_SECTION
  loops routine="#"
  END_INSTRUMENT_SECTION

% set path=(/usr/common/acts/TAU/tau-2.17.2/x86_64/bin $path)
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)
% setenv COUNTER1 GET_TIME_OF_DAY
% setenv COUNTER2 PAPI_FP_INS
% qsub  run.job
% paraprof --pack app.ppk
  Move the app.ppk file to your desktop.
% paraprof app.ppk
  Choose Options -> Show Derived Panel -> Arg 1 = PAPI_FP_INS,
 Arg 2 = GET_TIME_OF_DAY, Operation = Divide -> Apply, choose.
```

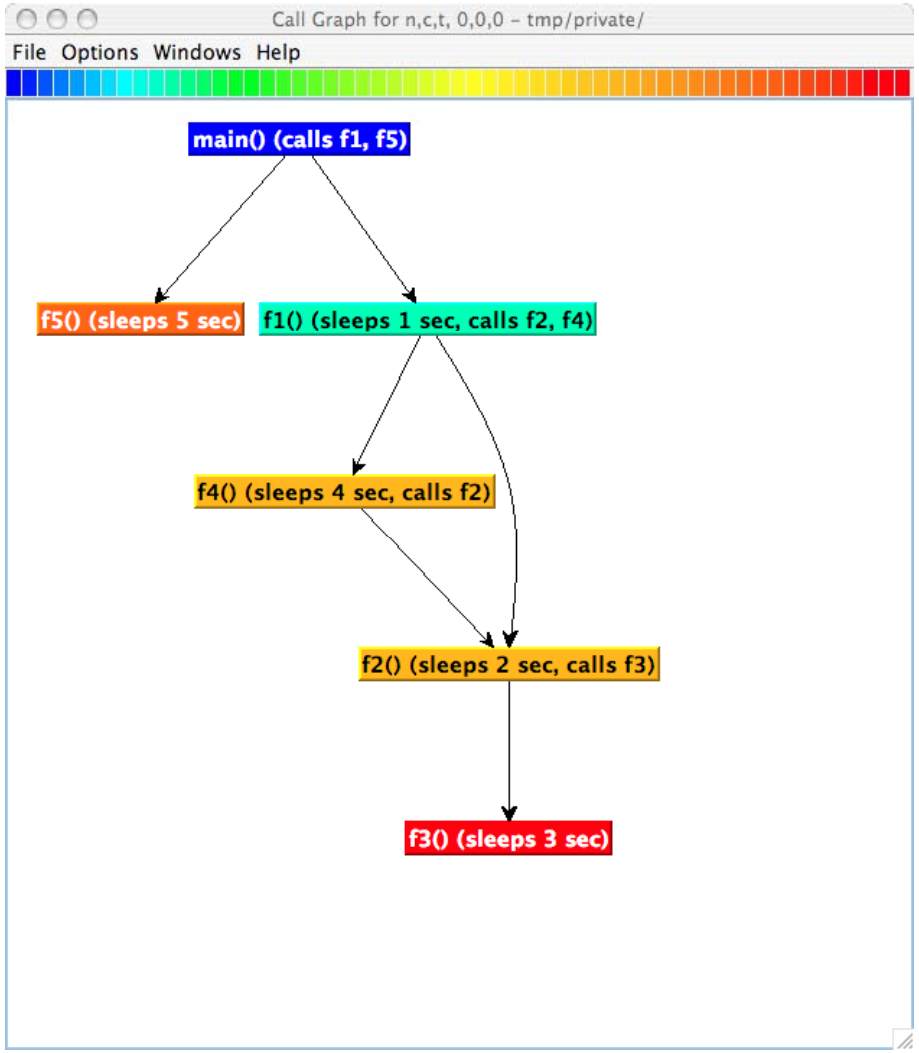
Usage Scenarios: Generating Callpath Profile

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



Callpath Profile

- Generates program callgraph

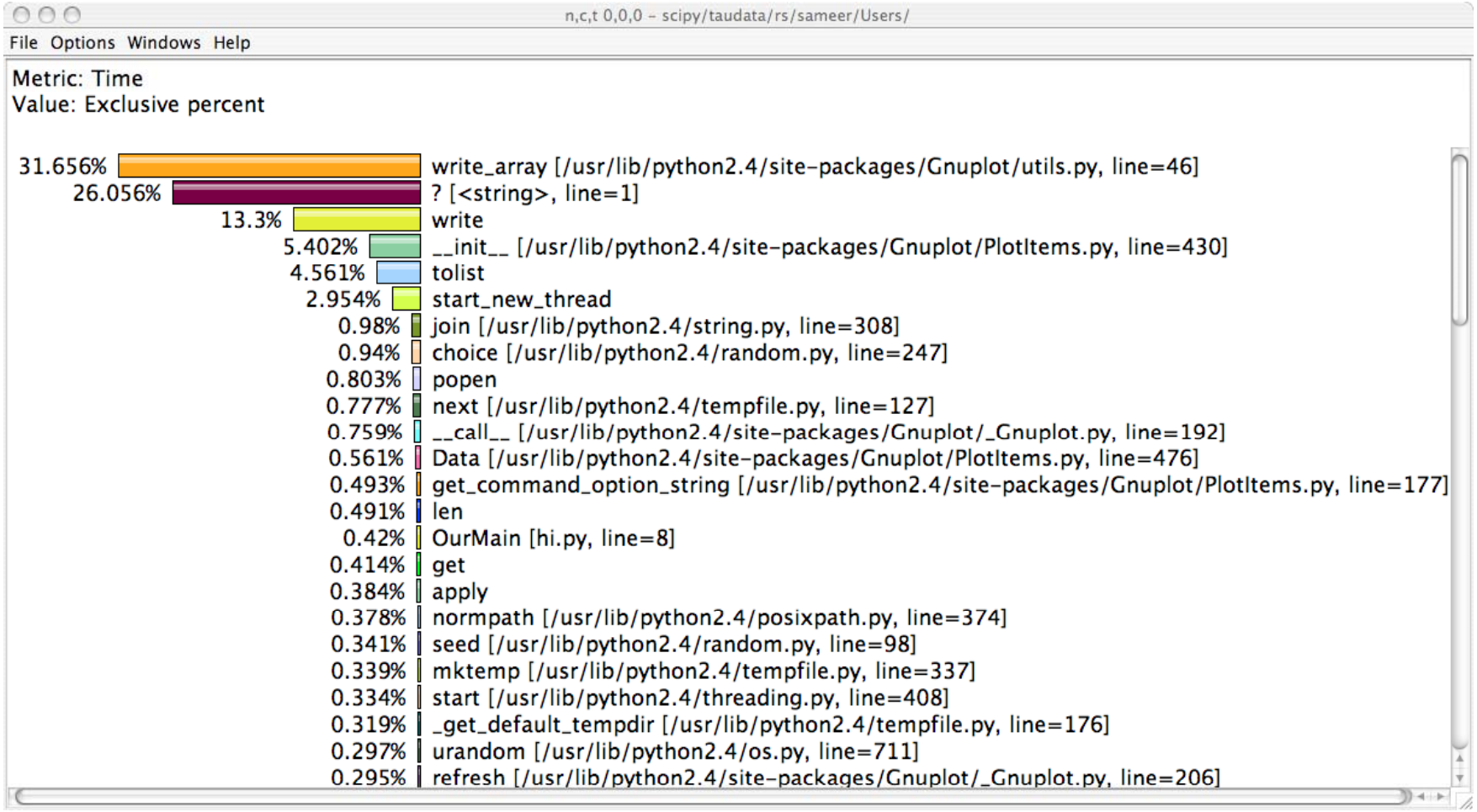


Generate a Callpath Profile

```
% setenv TAU_MAKEFILE /usr/common/acts/TAU/tau-2.17.2/x86_64
                /lib/Makefile.tau-pathcc-callpath-mpi-pdt
% set path=(/usr/common/acts/TAU/tau-2.17.2/x86_64/bin $path)
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)
% setenv TAU_CALLPATH_DEPTH 100

% qsub  run.job
% paraprof --pack app.ppk
  Move the app.ppk file to your desktop.
% paraprof app.ppk
(Windows -> Thread -> Call Graph)
```

Usage Scenarios: Mixed Python+F90+C+pyMPI

- Goal: Generate multi-level instrumentation for Python+MPI+…

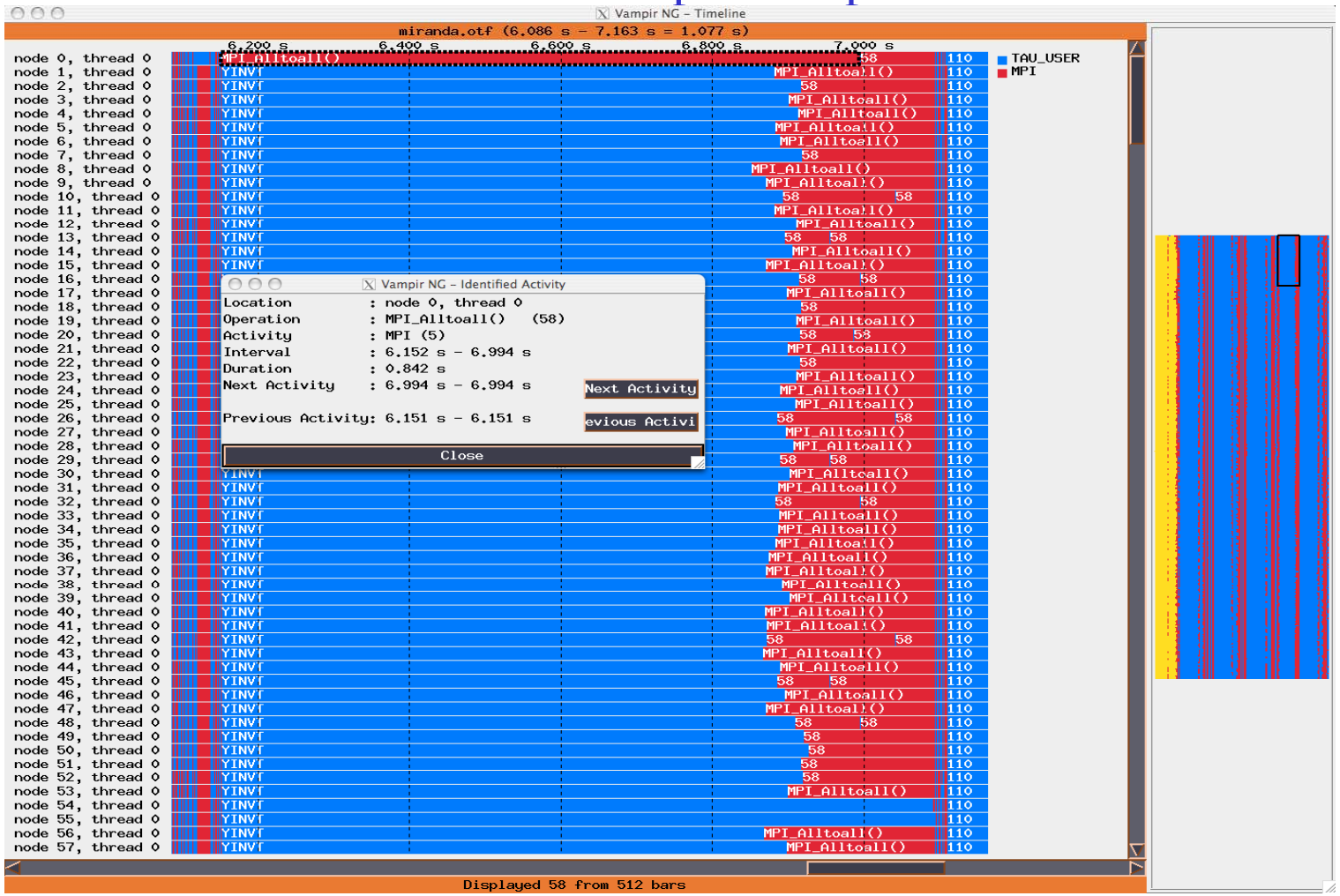


Generate a Multi-Language Profile w/ Python

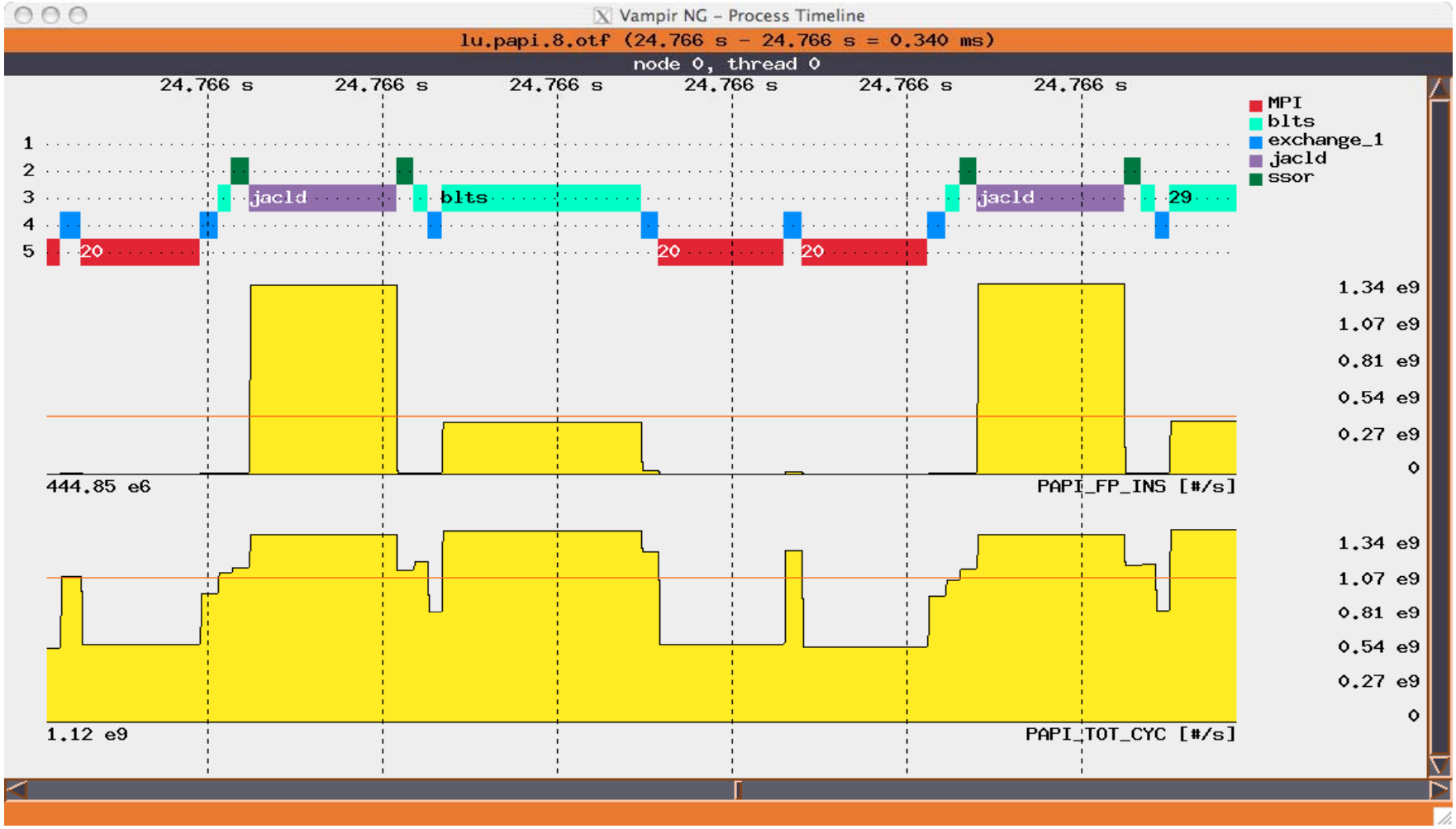
```
% setenv TAU_MAKEFILE /usr/common/acts/TAU/tau-2.17.2/x86_64
                /lib/Makefile.tau-pathcc-python-mpi-pdt
% set path=(/usr/common/acts/TAU/tau-2.17.2/x86_64/bin $path)
% setenv TAU_OPTIONS '-optShared -optVerbose…'
(Python needs shared object based TAU library)
% make F90=tau_f90.sh CXX=tau_cxx.sh CC=tau_cc.sh  (build pyMPI w/TAU)
% cat wrapper.py
  import tau
  def OurMain():
      import App
  tau.run('OurMain()')
Uninstrumented:
% mpirun.lsf /pyMPI-2.4b4/bin/pyMPI ./App.py
Instrumented:
% setenv PYTHONPATH <taudir>/x86_64/<lib>/bindings-python-mpi-pdt-pgi
(same options string as TAU_MAKEFILE)
setenv LD_LIBRARY_PATH <taudir>/x86_64/lib/bindings-icpc-python-mpi-pdt-
pgi\:$LD_LIBRARY_PATH
% mpirun –np 4 <dir>/pyMPI-2.4b4-TAU/bin/pyMPI ./wrapper.py
(Instrumented pyMPI with wrapper.py)
```

Usage Scenarios: Generating a Trace File

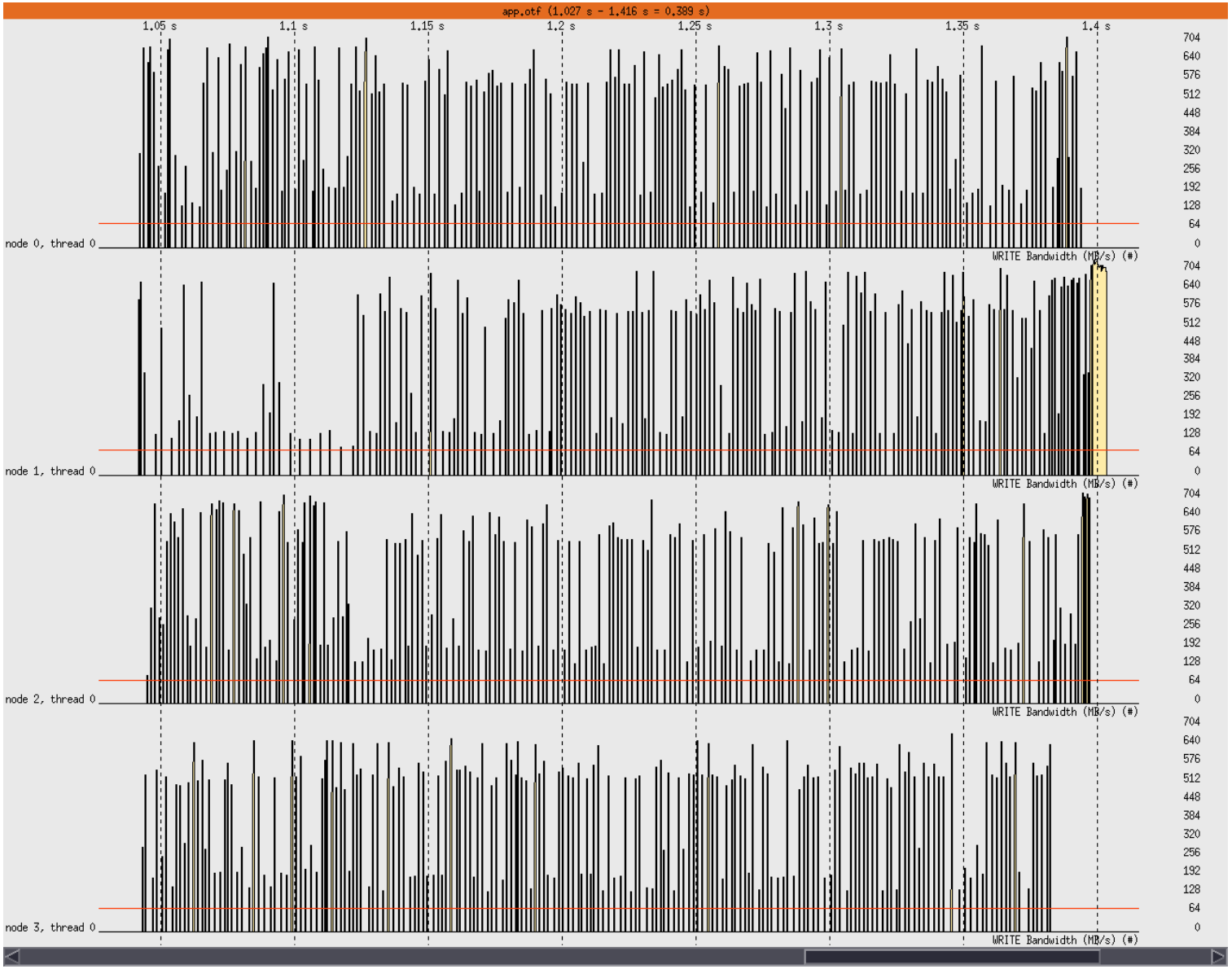
- Goal: What happens in my code at a given time? When?
- Event trace visualized in Vampir/Jumpshot



VNG Process Timeline with PAPI Counters



Vampir Counter Timeline Showing I/O BW

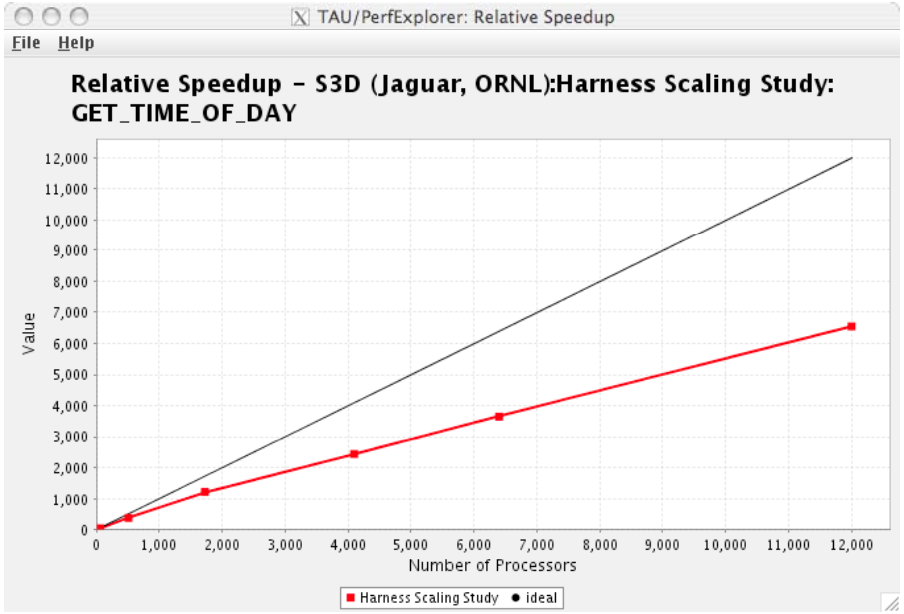


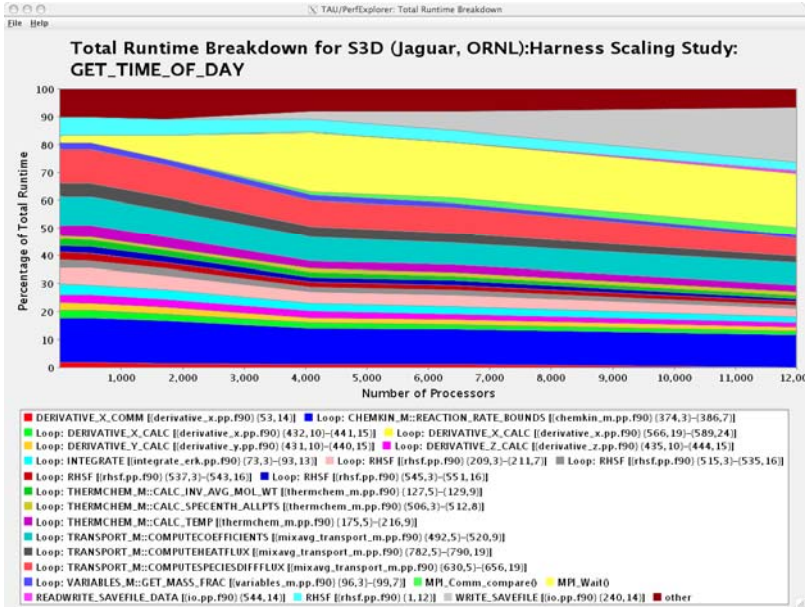
Generate a Trace File

```
% setenv TAU_MAKEFILE /usr/common/acts/TAU/tau-2.17.2/x86_64
                /lib/Makefile.tau-pathcc-mpi-pdt-trace
% set path=(/usr/common/acts/TAU/tau-2.17.2/x86_64/bin $path)
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)
% qsub  run.job
% 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
(or vng client with vngd server).
```

Usage Scenarios: Evaluate Scalability

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





Evaluate Scalability using PerfExplorer Charts

```
% setenv TAU_MAKEFILE /usr/common/acts/TAU/tau-2.17.2/x86_64
                /lib/Makefile.tau-pathcc-mpi-pdt
% set path=(/usr/common/acts/TAU/tau-2.17.2/x86_64/bin $path)
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)
% qsub  run1p.job
% paraprof --pack 1p.ppk
% qsub  run2p.job …
% paraprof --pack 2p.ppk … and so on.
On your client:
% perfdmf_configure
(Choose 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)
% perfexplorer
(Charts -> Speedup)
```

Part II: Rest of the talk

The Details...

Performance Tools FAQ/Concerns

- Does it automatically instrument my code? At the routine level? At the outer-loop level?
- Can it show me where time is spent in my code? PAPI Flops? L1 data cache misses? Can I measure more than one quantity in a trial?
- Does the tool support profiling (runtime summarization) as well as tracing (time-line based displays)? What about profile snapshots? Callpath (parent-child) profiles? Can I use it to easily benchmark codes?
- Can I observe the performance data at runtime as the application executes?
- Can it show me memory utilization? Memory leaks? Mallocs/frees? When and where?
- What about I/O? Can I observe bandwidth of reads/writes? Volume of I/O? What about Kernel events? User space+Kernel?
- What is the typical overhead? Can I reduce it to < 5%? < 1%? Can it compensate and remove timer overhead from performance data? Can it throttle away instrumentation in lightweight routines at runtime to reduce overhead?
- I already have profile data from <XYZ> tool. Can it import my legacy data?
- I prefer <XYZ> performance tool for visualization. Can it hook up with this tool? Are there converters?

Performance Tools FAQ/Concerns (contd.)

- Can I use it for multi-core CPUs? Compare the performance of application running on a single vs. multi-core processor? Can I observe multi-core data snoops, invalidates?
- Can I share the performance data with my colleagues in a secure manner (web/database)? Can it automatically track progress of my application over time (~ 6 mos)? Can I use it for scalability studies? Over multiple platforms?
- Are the GUI client tools available under Linux? MS Windows? Apple?
- Does it run on all Cray, IBM, SGI, HP … platforms? CNL? Catamount?
- Does it support MPI? MPI2? Threads? Hybrid MPI+Pthreads/MPI+OpenMP?
- Does it support Fortran? C++, C? Java? Python? Python+MPI+F90+C++…?
- Does it support Intel/PGI/PathScale/IBM/Cray/Sun compilers?
- Are tools available in command-line form & GUI? IDE GUI? Web-based? 3D?
- Is it already installed and supported on my HPC system? What about systems at NERSC? ANL? LLNL? LANL? NASA? DoD? NSF sites?...
- Is there support (phone/e-mail) available for the tool? Professional support? For instrumentation? Analysis?
- Will it work on the new <XYZ> HPC platform scheduled for release six months from now?
- Is it free? BSD license? …

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

Definitions – Profiling

- Profiling
  - Recording of summary information during execution
    - inclusive, exclusive time, # calls, hardware statistics, …
  - Reflects performance behavior of program entities
    - functions, loops, basic blocks
    - user-defined "semantic" entities
  - Very good for low-cost performance assessment
  - Helps to expose performance bottlenecks and hotspots
  - Implemented through
    - sampling: periodic OS interrupts or hardware counter traps
    - instrumentation: direct insertion of measurement code

Definitions – Tracing

- Tracing
  - Recording of information about significant points (events) during program execution
    - entering/exiting code region (function, loop, block, …)
    - thread/process interactions (e.g., send/receive message)
  - Save information in event record
    - timestamp
    - CPU identifier, thread identifier
    - Event type and event-specific information
  - Event trace is a time-sequenced stream of event records
  - Can be used to reconstruct dynamic program behavior
  - Typically requires code instrumentation

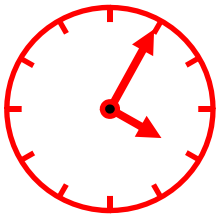
Event Tracing: Instrumentation, Monitor, Trace

CPU A:

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

CPU B:

```
void worker {
  trace(ENTER, 2);
  ...
  recv(A, tag, buf);
  trace(RECV, A);
  ...
  trace(EXIT, 2);
}
```



timestamp

MONITOR

Event definition

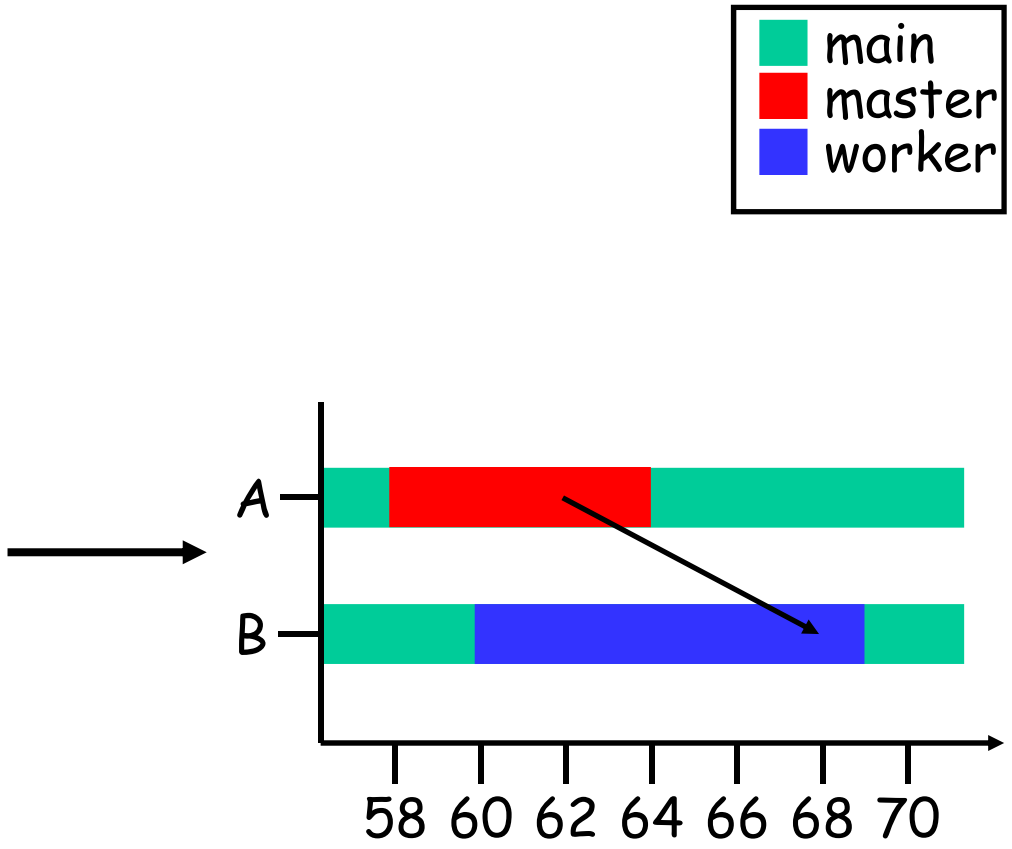
1	master
2	worker
3	...

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

Event Tracing: "Timeline" 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
...			



TAU Performance System Project

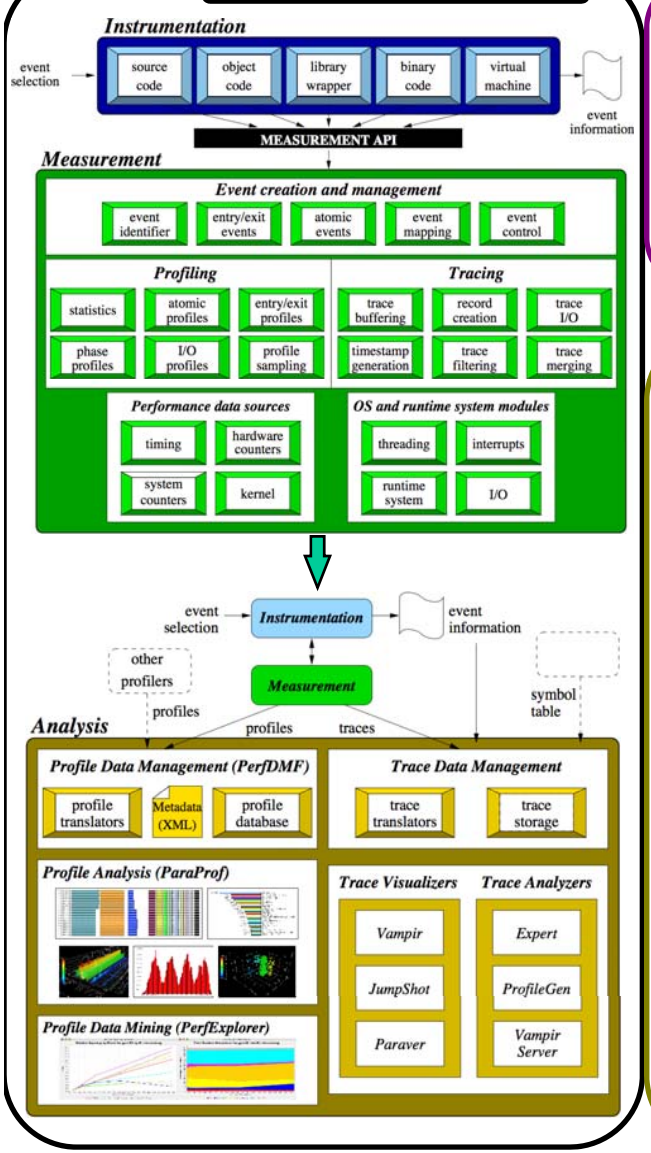
- ***T***uning and ***A***nalysis ***U***tilities (15+ year project effort)
- **Performance system framework for HPC systems**
  - Integrated, scalable, and flexible
  - Target parallel programming paradigms
- **Integrated toolkit for performance problem solving**
  - Instrumentation, measurement, analysis, and visualization
  - Portable performance profiling and tracing facility
  - Performance data management and data mining
- ***Partners***
  - LLNL, ANL, LANL
  - Research Centre Jülich, TU Dresden

TAU Parallel Performance System Goals

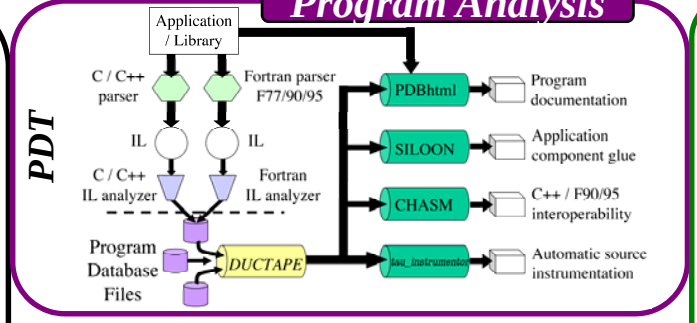
- **Portable (open source) parallel performance system**
  - Computer system architectures and operating systems
  - Different programming languages and compilers
- Multi-level, multi-language performance instrumentation
- **Flexible and configurable performance measurement**
- Support for multiple parallel programming paradigms
  - Multi-threading, message passing, mixed-mode, hybrid, object oriented (generic), component-based
- Support for performance mapping
- Integration of leading performance technology
- **Scalable (very large) parallel performance analysis**

TAU Performance System Components

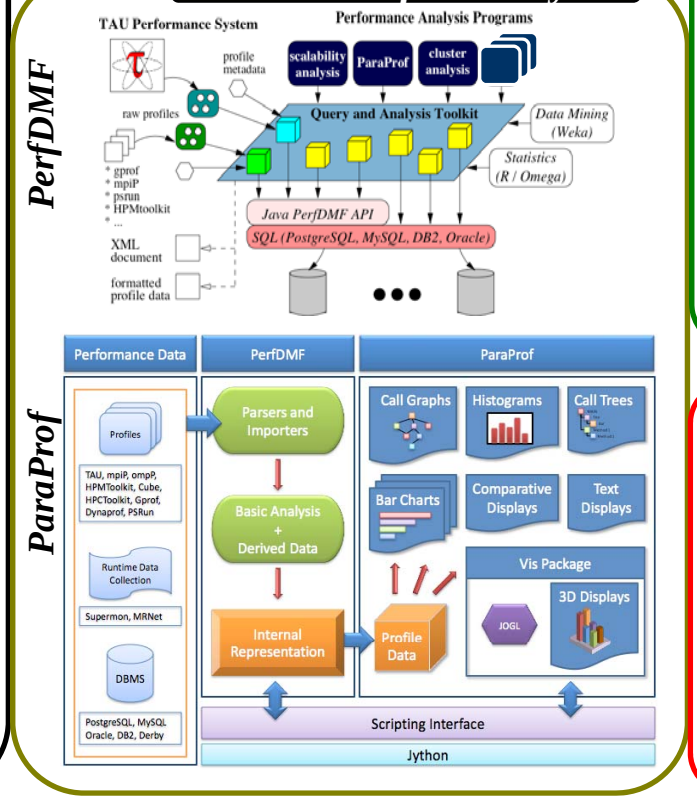
TAU Architecture



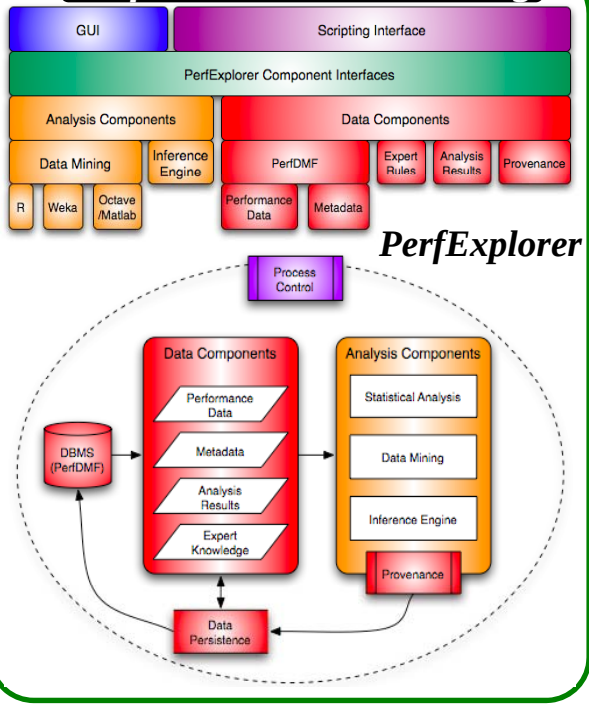
Program Analysis



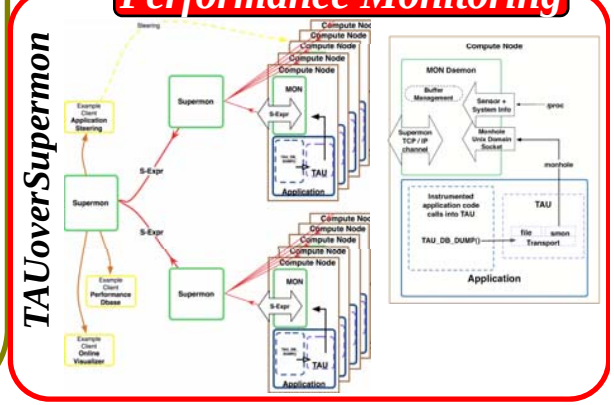
Parallel Profile Analysis



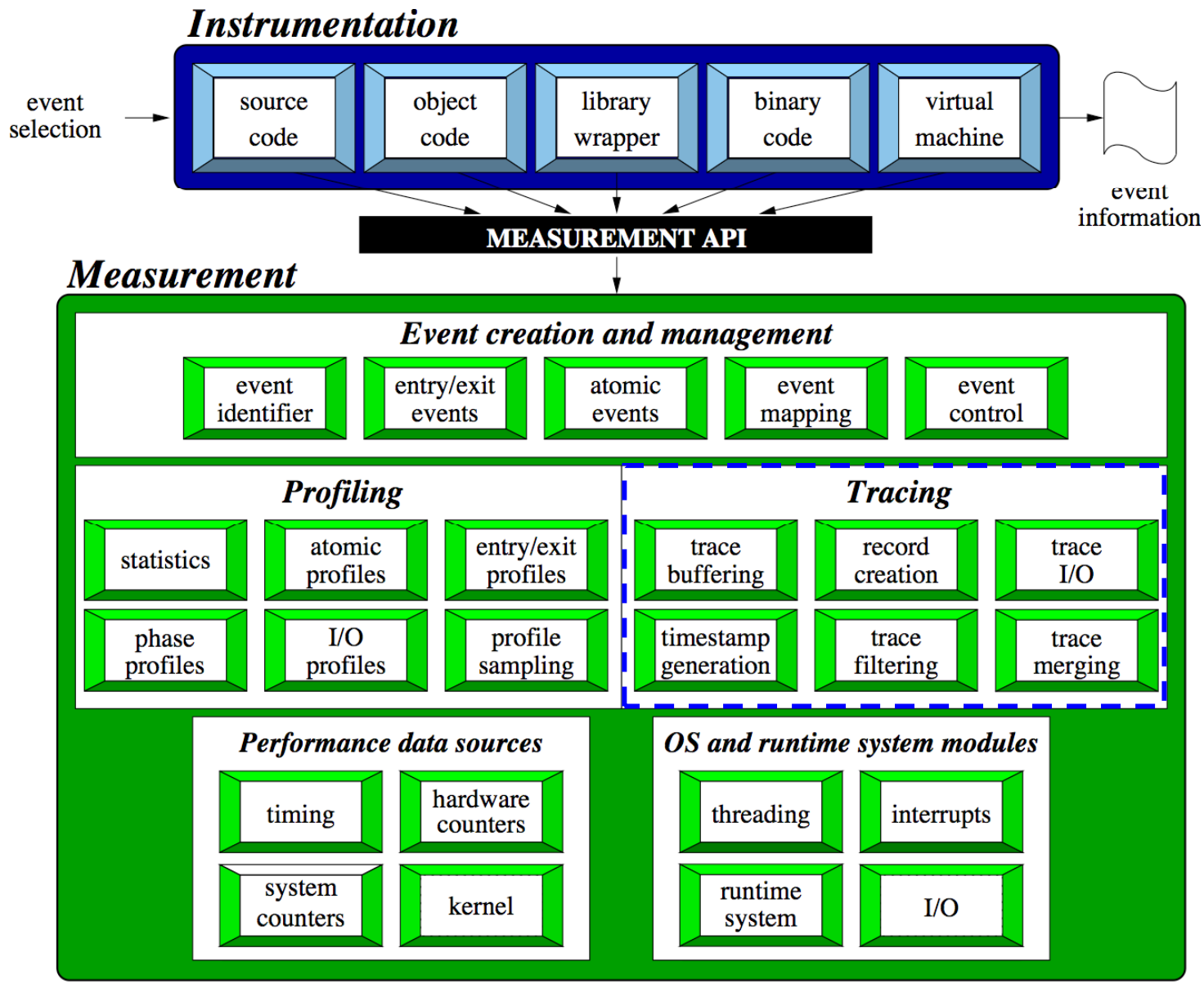
Performance Data Mining



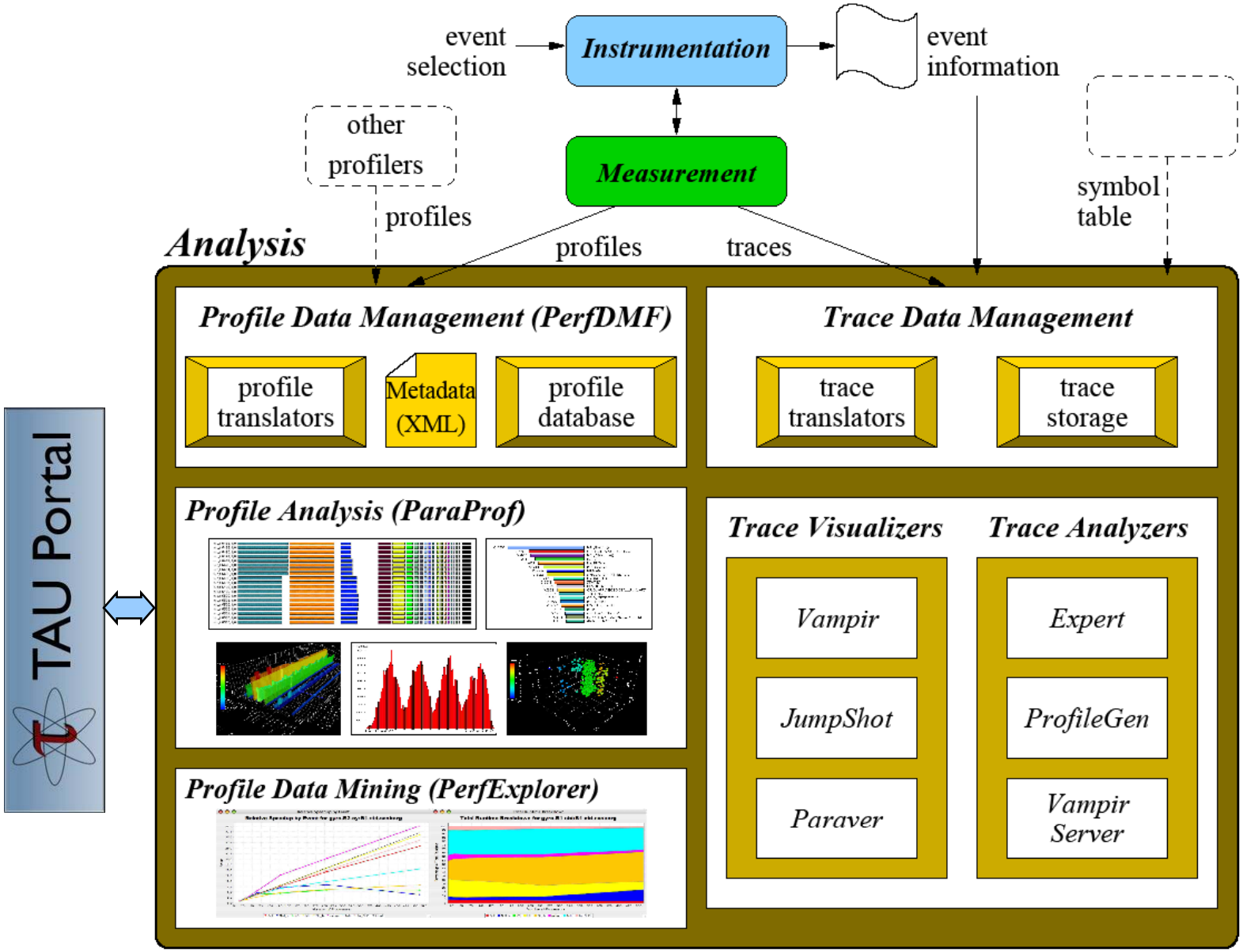
Performance Monitoring



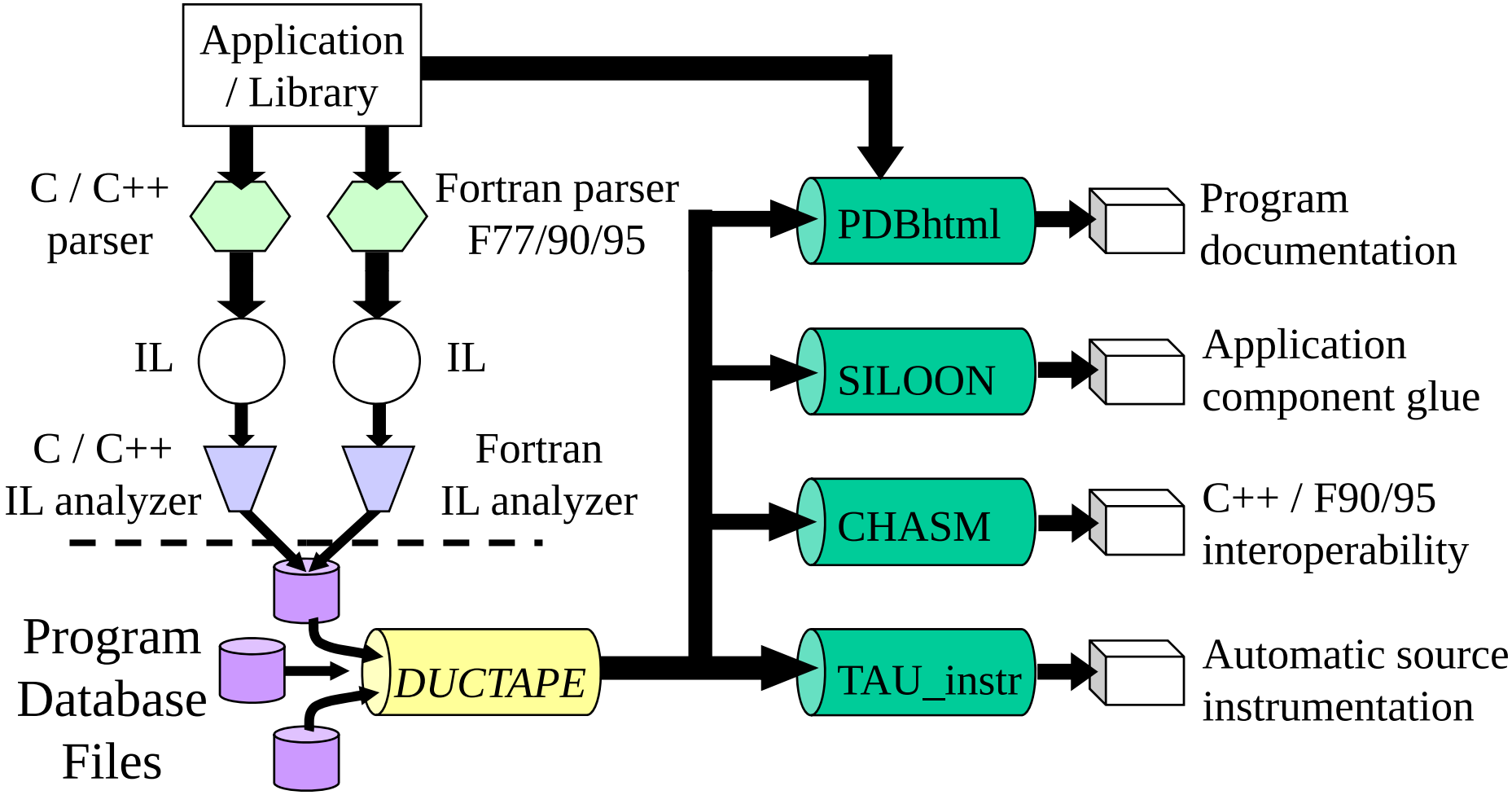
TAU Performance System Architecture



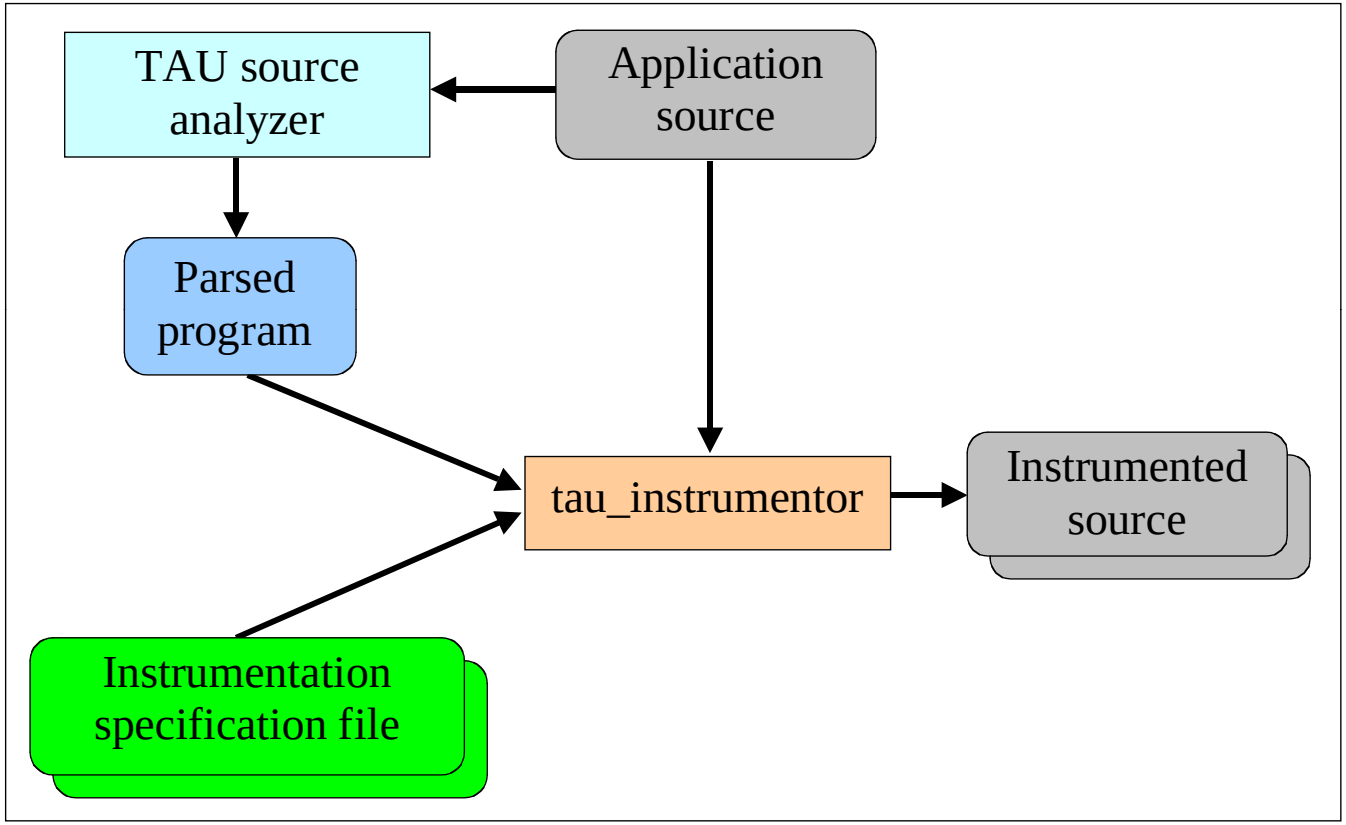
TAU Performance System Architecture



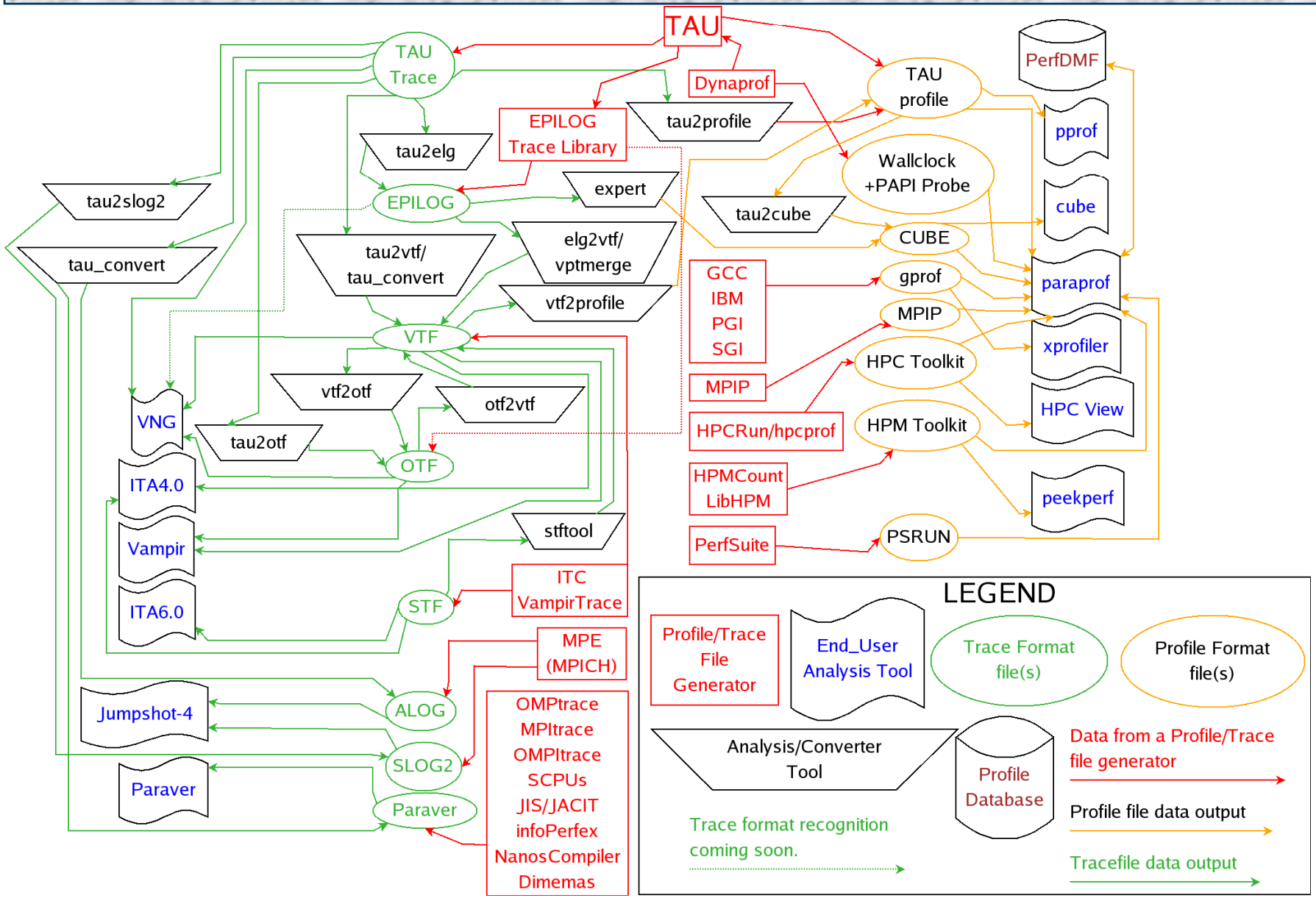
Program Database Toolkit (PDT)



Automatic Source-Level Instrumentation in TAU



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

TAU Instrumentation Mechanisms

- **Source code**
  - Manual (TAU API, TAU component API)
  - Automatic (robust)
    - C, C++, F77/90/95 (Program Database Toolkit (***PDT***))
    - OpenMP (directive rewriting (*Opari*), *POMP2* spec)
- **Object code**
  - Pre-instrumented libraries (e.g., MPI using *PMPI*)
  - Statically-linked and dynamically-linked
- **Executable code**
  - Binary and dynamic instrumentation (*Dyninst*)
  - Virtual machine instrumentation (e.g., Java using *JVMPI*)
- *TAU_COMPILER* to automate instrumentation process

Using TAU: A brief Introduction

- To instrument source code using PDT
  - Choose an appropriate TAU stub makefile (measurement option) from <taudir>/<arch>/lib directory:

% setenv TAU_MAKEFILE /usr/common/acts/TAU/tau-2.17.2/x86_64/lib/Makefile.tau-pathcc-mpi-pdt

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

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

% mpxlf90_r 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 – Examples

% cd /usr/common/acts/TAU/tau-2.17.2/x86_64/lib; ls Makefile.*
Makefile.tau-pathcc-pdt
Makefile.tau-pathcc-mpi-pdt
Makefile.tau-pathcc-callpath-mpi-pdt
Makefile.tau-pathcc-mpi-pdt-trace
Makefile.tau-pathcc-mpi-compensate-pdt
Makefile.tau-multiplecounters-pathcc-mpi-papi-pdt
Makefile.tau-multiplecounters-pathcc-mpi-papi-pdt-trace
Makefile.tau-pathcc-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/common/acts/TAU/tau-2.17.2/x86_64/lib/Makefile.tau-pathcc-mpi-pdt

Installing TAU on 64bit AIX

- Install PAPI and PDT
  - PAPI:
    - ./configure –prefix=$HOME/pkgs/papi-3.5.0;
    - Make ; make install
  - PDT:
    - ./configure –prefix=$HOME/pkgs/pdt-3.13 –PGI
    - make; make install
- Install TAU:
  - ./installtau –pdt=$HOME/pkgs/pdt-3.13 –papi=$HOME/pkgs/papi-3.5.0 -c++=pgCC -cc=pgcc -fortran=pgi –mpiinc=<dir> -mpilib=<dir>
    - Configures multiple typically requested versions for you in ia64/lib/Makefile.tau-* configurations
  - tau_validate –html –build x86_64 >& results.html
  - mozilla results.html

Validating an Install

SeaMonkey

File Edit View Go Bookmarks Tools Window Help

Back Forward Reload Stop file:///home/sameer/tmp/resibm64.html Search Print

Home Bookmarks CentOS Support

```
Debug: Compiling with Instrumented Code

Executing> xlf90_r -I. -q64 -c simple.inst.f90 -qsuffix=f=f90 -I/u/sameer/pkgs/tau-2.17.1/include -I/usr/lpp/ppe.poe/include -I/usr/lpp/ssp/css/include -I/usr/lpp/ppe.poe/include/thread64 -o simple.o

** func   === End of Compilation 1 ===

** main   === End of Compilation 2 ===

1501-510  Compilation successful for file simple.inst.f90.

Looking for file: simple.o


        /u/sameer/pkgs/tau-2.17.1/ibm64/bin/tau_compiler.sh -optPdtDir="/u/sameer/pkgs/pdtoolkit-3.12/ibm64" -optPdtCOpts="-I/u/sameer/pkgs/tau-2.17.1/include -DPROFILING_ON           -DTAU_PAPI -I/u/sameer/pkgs/papi-3.5.0/src -I/u/sameer/pkgs/papi-3.5.0/inclu



Debug: Linking with TAU Options

Executing> xlf90_r -q64 simple.o -o simple -L/u/sameer/pkgs/tau-2.17.1/ibm64/lib -lTauMpi-phase-multiplecounters-mpi-papi-pdt -L/usr/lpp/ppe.poe/lib -lmpi_r -L/usr/lpp/ppe.poe/lib/threads -L/usr/lpp/ssp/css/lib -llapi_r -L/u/sameer/pkgs/tau-2.17.1/ibm64/lib -lt


Target "all" is up to date.

/gpfs/home/sameer/pkgs/tau-2.17.1/examples/validate/pdtmpif> mkdir build-Makefile.tau-phase-multiplecounters-mpi-papi-pdt
/gpfs/home/sameer/pkgs/tau-2.17.1/examples/validate/pdtmpif> cp simple build-Makefile.tau-phase-multiplecounters-mpi-papi-pdt
pass/na   : PDT-MPI (Fortran)    : Makefile.tau-phase-multiplecounters-mpi-papi-pdt
```

PdtTest (/gpfs/home/sameer/pkgs/tau-2.17.1/ibm64/lib/Makefile.tau-phase-multiplecounters-mpi-papi-pdt)

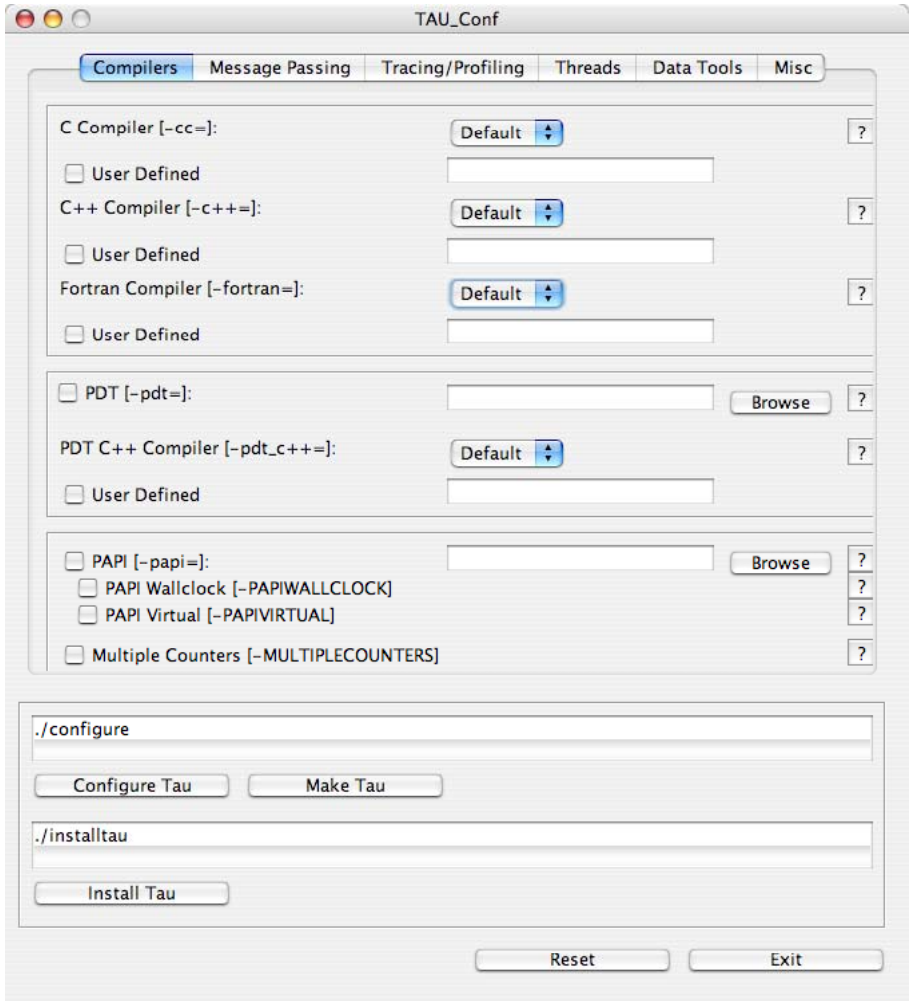
```
/gpfs/home/sameer/pkgs/tau-2.17.1/examples/validate/pdtmpif> grep "^TAU_F90 " /gpfs/home/sameer/pkgs/tau-2.17.1/ibm64/lib/Makefile.tau-phase-multiplecounters-mpi-papi-pdt
TAU_F90       = xlf90$(TAU_R)    $(F90_ABI) $(TAU_F90_OPT)    #ENDIF##IBM64_FORTRAN#

  na/na   : PDT-MPI (GFortran)    : Makefile.tau-phase-multiplecounters-mpi-papi-pdt
```

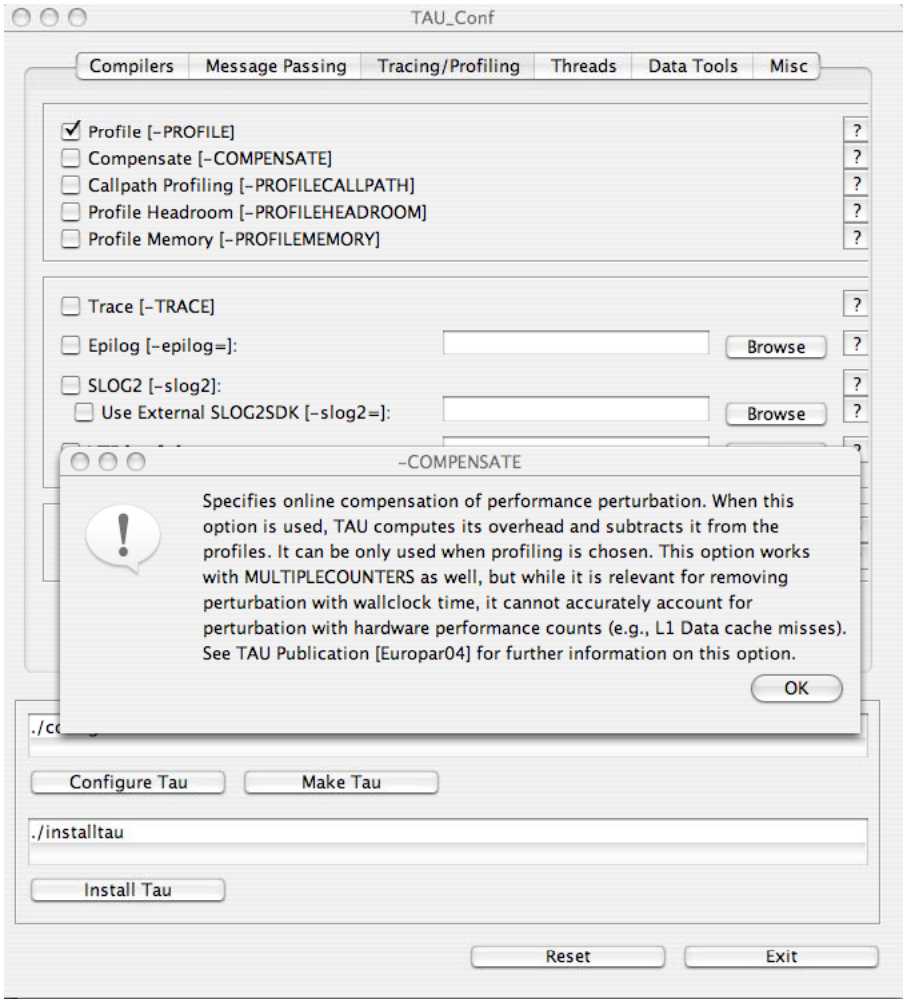
Stub Makefile	C		PDT (C)		PDT (C++)		PDT (Fortran)		PDT (GFortran)		Fortran (flink)		Fortran (cpplink)		Fortran (clink)		MPI (C)		MPI (Fortran)		PDT-MPI (C)		PDT-MPI (C++)		PDT-MPI (Fortran)		PDT-MPI (GFortran)	
	build	run	build	run	build	run	build	run	build	run	build	run	build	run	build	run	build	run	build	run	build	run	build	run	build	run	build	run
Makefile.tau-mpi-pdt	pass	N/A	pass	N/A	pass	N/A	pass	N/A	N/A	N/A	pass	N/A	pass	N/A	pass	N/A	pass	N/A	pass	N/A	pass	N/A	pass	N/A	pass	N/A	N/A	N/A
Makefile.tau-mpi-pdt-trace	pass	N/A	pass	N/A	pass	N/A	pass	N/A	N/A	N/A	pass	N/A	pass	N/A	pass	N/A	pass	N/A	pass	N/A	pass	N/A	pass	N/A	pass	N/A	N/A	N/A
Makefile.tau-pdt	pass	N/A	pass	N/A	pass	N/A	pass	N/A	N/A	N/A	pass	N/A	pass	N/A	pass	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Makefile.tau-callpath-mpi-compensate-pdt	pass	N/A	pass	N/A	pass	N/A	pass	N/A	N/A	N/A	pass	N/A	pass	N/A	pass	N/A	pass	N/A	pass	N/A	pass	N/A	pass	N/A	pass	N/A	N/A	N/A
Makefile.tau-mpi-compensate-pdt	pass	N/A	pass	N/A	pass	N/A	pass	N/A	N/A	N/A	pass	N/A	pass	N/A	pass	N/A	pass	N/A	pass	N/A	pass	N/A	pass	N/A	pass	N/A	N/A	N/A
Makefile.tau-phase-multiplecounters-mpi-papi-compensate-pdt	pass	N/A	pass	N/A	pass	N/A	pass	N/A	N/A	N/A	pass	N/A	pass	N/A	pass	N/A	pass	N/A	pass	N/A	pass	N/A	pass	N/A	pass	N/A	N/A	N/A
Makefile.tau-multiplecounters-papi-pthread-pdt	pass	N/A	pass	N/A	pass	N/A	pass	N/A	N/A	N/A	pass	N/A	pass	N/A	pass	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Makefile.tau-multiplecounters-mpi-papi-pdt	pass	N/A	pass	N/A	pass	N/A	pass	N/A	N/A	N/A	pass	N/A	pass	N/A	pass	N/A	pass	N/A	pass	N/A	pass	N/A	pass	N/A	pass	N/A	N/A	N/A
Makefile.tau-multiplecounters-papi-pdt	pass	N/A	pass	N/A	pass	N/A	pass	N/A	N/A	N/A	pass	N/A	pass	N/A	pass	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Makefile.tau-multiplecounters-mpi-papi-pdt-trace	pass	N/A	pass	N/A	pass	N/A	pass	N/A	N/A	N/A	pass	N/A	pass	N/A	pass	N/A	pass	N/A	pass	N/A	pass	N/A	pass	N/A	pass	N/A	N/A	N/A
Makefile.tau-depthlimit-mpi-pdt	pass	N/A	pass	N/A	pass	N/A	pass	N/A	N/A	N/A	pass	N/A	pass	N/A	pass	N/A	pass	N/A	pass	N/A	pass	N/A	pass	N/A	pass	N/A	N/A	N/A
Makefile.tau-pthread-pdt	pass	N/A	pass	N/A	pass	N/A	pass	N/A	N/A	N/A	pass	N/A	pass	N/A	pass	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Makefile.tau-callpath-mpi-pdt	pass	N/A	pass	N/A	pass	N/A	pass	N/A	N/A	N/A	pass	N/A	pass	N/A	pass	N/A	pass	N/A	pass	N/A	pass	N/A	pass	N/A	pass	N/A	N/A	N/A
Makefile.tau-phase-multiplecounters-mpi-papi-pdt	pass	N/A	pass	N/A	pass	N/A	pass	N/A	N/A	N/A	pass	N/A	pass	N/A	pass	N/A	pass	N/A	pass	N/A	pass	N/A	pass	N/A	pass	N/A	N/A	N/A

No Errors!

Done

TAU_SETUP: A GUI for Installing TAU





Upgrading TAU v2.17.1 configurations to 2.17.2

- Upgrade TAU
  - Previous installation in $HOME/pkgs/tau-2.17.2
    - cd tau-2.17.3
    - ./upgradetau $HOME/pkgs/tau-2.17.2
      - Builds all previous configurations in the current dir
      - You may also upgrade with a new package say PDT 3.14
    - ./upgradetau $HOME/pkgs/tau-2.17.3 –pdt=$HOME/pkgs/pdt-3.14

- Validate your new installation
    - ./tau_validate –html –build ia64 >& results.html
    - mozilla results.html

TAU Configuration – Examples on Ranger

- ./configure -pdt=<dir> -c++=pgCC -cc=pgcc -fortran=pgi –mpi
  - on AIX 64 bit with PDT, MPI for IBM xlC compilers
- ./configure –papi=<dir> -MULTIPLECOUNTERS –pdt=<dir> -mpi -c++=pgCC –cc=pgcc –fortran=mpi; make clean install
  - Use PAPI counters (one or more) with C/C++/F90 automatic instrumentation for AIX. Also instrument the MPI library.
- Typically configure multiple measurement libraries
  - .all_configs, .last_config files contain all and last configuration
  - tau_validate --html --build x86_64 >& results.html
  - ./upgradetau /path/to/old/tau-2.17
- Each configuration creates a unique <arch>/lib/Makefile.tau<options> stub makefile. It corresponds to the configuration options used. e.g.,
  - /usr/common/acts/TAU/tau-2.17.2/x86_64/lib/Makefile.tau-mpi-pdt-pgi
  - /usr/common/acts/TAU/tau-2.17.2/x86_64/lib/Makefile.tau-multiplecounters-mpi-papi-pdt-pgi

Using TAU

- Install TAU

  % ./configure [options]; make clean install
- Replace the names of your compiler with tau_f90.sh, tau_cxx.sh and tau_cc.sh in your makefiles
- Set environment variables
  - Choose the measurement option and compile your code:
    - setenv TAU_MAKEFILE $TAU/Makefile.tau-mpi-pdt
    - setenv TAU_OPTIONS '-optVerbose -optKeepFiles -optPreProcess'
    - setenv TAU_THROTTLE 1
      - At runtime to keep instrumentation overhead in check
  - At runtime, if more than one metric is measured (-multiplecounters):
    - setenv COUNTER1 GET_TIME_OF_DAY
    - setenv COUNTER2 PAPI_FP_INS
    - setenv COUNTER3 PAPI_NATIVE_<native_name>
      - Use papi_native_avail, papi_avail, and papi_event_chooser to select these preset and native event names
- Build the application, run it, analyze performance data

PAPI

- **P**erformance **A**pplication **P**rogramming **I**nterface
  - The purpose of the PAPI project is to design, standardize and implement a portable and efficient API to access the hardware performance monitor counters found on most modern microprocessors.
- Parallel Tools Consortium project started in 1998
- Developed by University of Tennessee, Knoxville
- http://icl.cs.utk.edu/papi/

Papi_avail

- List of PAPI preset counters:

```
cfe1.sameer 66> ./papi_avail | more
Available events and hardware information.
--------------------------------------------------------------------------------
Vendor string and code   : GenuineIntel (1)
Model string and code    : Itanium 2 (1)
CPU Revision             : 5.000000
CPU Megahertz            : 1500.000000
CPU's in this Node       : 28
Nodes in this System     : 1
Total CPU's              : 28
Number Hardware Counters : 4
Max Multiplex Counters   : 32
--------------------------------------------------------------------------------
The following correspond to fields in the PAPI_event_info_t structure.

Name            Code            Avail   Deriv   Description (Note)
PAPI_L1_DCM     0x80000000      Yes     No      Level 1 data cache misses
PAPI_L1_ICM     0x80000001      Yes     No      Level 1 instruction cache misses
PAPI_L2_DCM     0x80000002      Yes     Yes     Level 2 data cache misses
PAPI_L2_ICM     0x80000003      Yes     No      Level 2 instruction cache misses
PAPI_L3_DCM     0x80000004      Yes     Yes     Level 3 data cache misses
PAPI_L3_ICM     0x80000005      Yes     No      Level 3 instruction cache misses
PAPI_L1_TCM     0x80000006      Yes     Yes     Level 1 cache misses
PAPI_L2_TCM     0x80000007      Yes     No      Level 2 cache misses


...
```

Papi_native_avail

- List of PAPI native counters:

```
cfe1.sameer 67> ./papi_native_avail | more
Available native events and hardware information.
--------------------------------------------------------------------------------
Vendor string and code   : GenuineIntel (1)
Model string and code    : Itanium 2 (1)
CPU Revision             : 5.000000
CPU Megahertz            : 1500.000000
CPU's in this Node       : 28
Nodes in this System     : 1
Total CPU's              : 28
Number Hardware Counters : 4
Max Multiplex Counters   : 32
--------------------------------------------------------------------------------
The following correspond to fields in the PAPI_event_info_t structure.

Symbol                           Event Code   Long Description
 Register Name[n]
 Register Value[n]
ALAT_CAPACITY_MISS_ALL           0x40000000   ALAT Entry Replaced -- both integer and floating point i
nstructions
ALAT_CAPACITY_MISS_FP            0x40000001   ALAT Entry Replaced -- only floating point instructions
ALAT_CAPACITY_MISS_INT           0x40000002   ALAT Entry Replaced -- only integer instructions
```

Papi_event_chooser on IA-64

- List of PAPI PRESET counters that may be counted together:

```
cfe1.sameer 68> ./papi_event_chooser
Usage: eventChooser NATIVE|PRESET evt1 evet2 ...

cfe1.sameer 72> ./papi_event_chooser PRESET PAPI_FP_OPS PAPI_L1_DCM PAPI_TOT_CYC
Test case eventChooser: Available events which can be added with given events.
--------------------------------------------------------------------------
Vendor string and code   : GenuineIntel (1)
Model string and code    : Itanium 2 (1)
CPU Revision             : 5.000000
CPU Megahertz            : 1500.000000
CPU's in this Node       : 28
Nodes in this System     : 1
Total CPU's              : 28
Number Hardware Counters : 4
Max Multiplex Counters   : 32
--------------------------------------------------------------------------
Name            Derived Description (Mgr. Note)
PAPI_L1_ICM     No      Level 1 instruction cache misses ()
PAPI_L2_ICM     No      Level 2 instruction cache misses ()
PAPI_L3_ICM     No      Level 3 instruction cache misses

...
```

PAPI_L1_ICM may be counted with these counters on IA64

Automatic Instrumentation

- TAU provides compiler wrapper scripts
  - Simply replace `mpxlf90_r` with `tau_f90.sh`
  - Automatically instruments Fortran source code, links with TAU MPI Wrapper libraries.
- Use `tau_cc.sh` and `tau_cxx.sh` for C/C++

Before

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

app: $(OBJS)
      $(F90) $(LDFLAGS) $(OBJS) -o $@
$(LIBS)
.f90.o:
      $(F90) $(CFLAGS) -c $<
```

After

```
F90 = tau_f90.sh
CXX = tau_cxx.sh
CFLAGS =
LIBS = -lm
OBJS = f1.o f2.o f3.o … fn.o

app: $(OBJS)
      $(F90) $(LDFLAGS) $(OBJS) -o $@
$(LIBS)
.f90.o:
      $(F90) $(CFLAGS) -c $<
```

TAU_COMPILER Commandline Options

- See `<taudir>/<arch>/bin/tau_compiler.sh –help`
- Compilation:

```
% mpxlf90 -c foo.f90
```

  Changes to

```
% f95parse foo.f90 $(OPT1)
% tau_instrumentor foo.pdb foo.f90 –o foo.inst.f90 $(OPT2)
% mpxlf90 –c foo.f90 $(OPT3)
```

- Linking:

```
% mpxlf90 foo.o bar.o –o app
```

  Changes to

```
% mpxlf90 foo.o bar.o –o app $(OPT4)
```

- Where options OPT[1-4] default values may be overridden by the user:

```
% setenv TAU_OPTIONS '...'
% make F90=tau_f90.sh
```

TAU_COMPILER Options

- Optional parameters for $(TAU_COMPILER): [tau_compiler.sh –help]

-optVerbose	Turn on verbose debugging messages
-optDetectMemoryLeaks	Turn on debugging memory allocations/ de-allocations to track leaks
-optPdtGnuFortranParser	Use gfparse (GNU) instead of f95parse (Cleanscape) for parsing Fortran source code
-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)

...

Overriding Default Options: TAU_OPTIONS

```
% cat Makefile
F90 = tau_f90.sh
OBJS = f1.o f2.o f3.o …
LIBS = -Lappdir –lapplib1 –lapplib2 …

app: $(OBJS)
    $(F90) $(OBJS) –o app $(LIBS)
.f90.o:
    $(F90) –c $<
% setenv TAU_OPTIONS '-optVerbose -
optTauSelectFile=select.tau'

% setenv TAU_MAKEFILE <taudir>/x86_64/lib/Makefile.tau-icpc-
mpi-pdt
```

Compiling Fortran Codes with TAU: Tips

- 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 '

- If it uses several module files, you may switch from the default Cleanscape Inc. parser in PDT to the GNU gfortran parser to generate PDB files:

  % setenv TAU_OPTIONS '-optPdtGnuFortranParser -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

TAU Measurement Mechanisms

- **Parallel profiling**
  - Function-level, block-level, statement-level
  - 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

Optimization of Program Instrumentation

- Need to eliminate instrumentation in frequently executing lightweight routines
- Throttling of events at runtime (set by default in TAU 2.17.2+):

```
% setenv TAU_THROTTLE 1
```

  Turns off instrumentation in routines that execute over 100000 times (TAU_THROTTLE_NUMCALLS) and take less than 10 microseconds of inclusive time per call (TAU_THROTTLE_PERCALL)

- Selective instrumentation file to filter events

```
% tau_instrumentor [options] –f <file>  OR
% setenv TAU_OPTIONS '-optTauSelectFile=tau.txt'
```

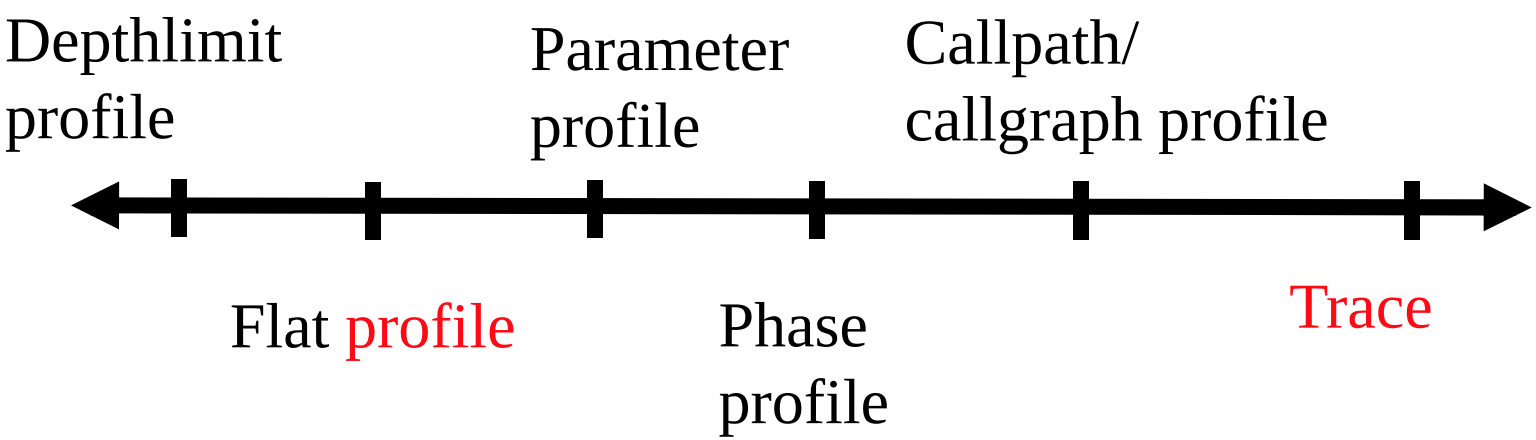
- Compensation of local instrumentation overhead

```
% configure -COMPENSATE
```

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



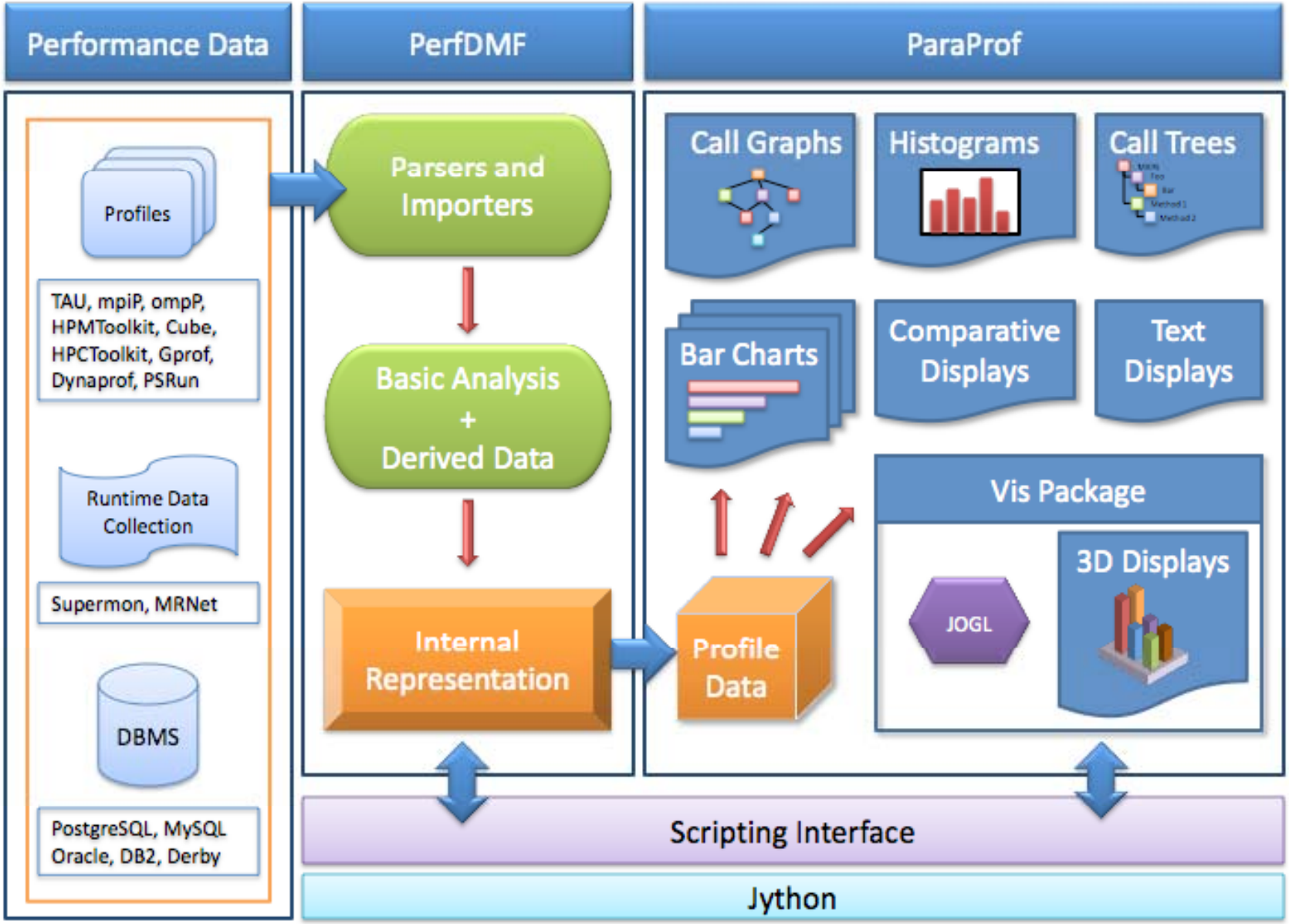
Each alternative has:
- one metric/counter
- multiple counters

Volume of performance data

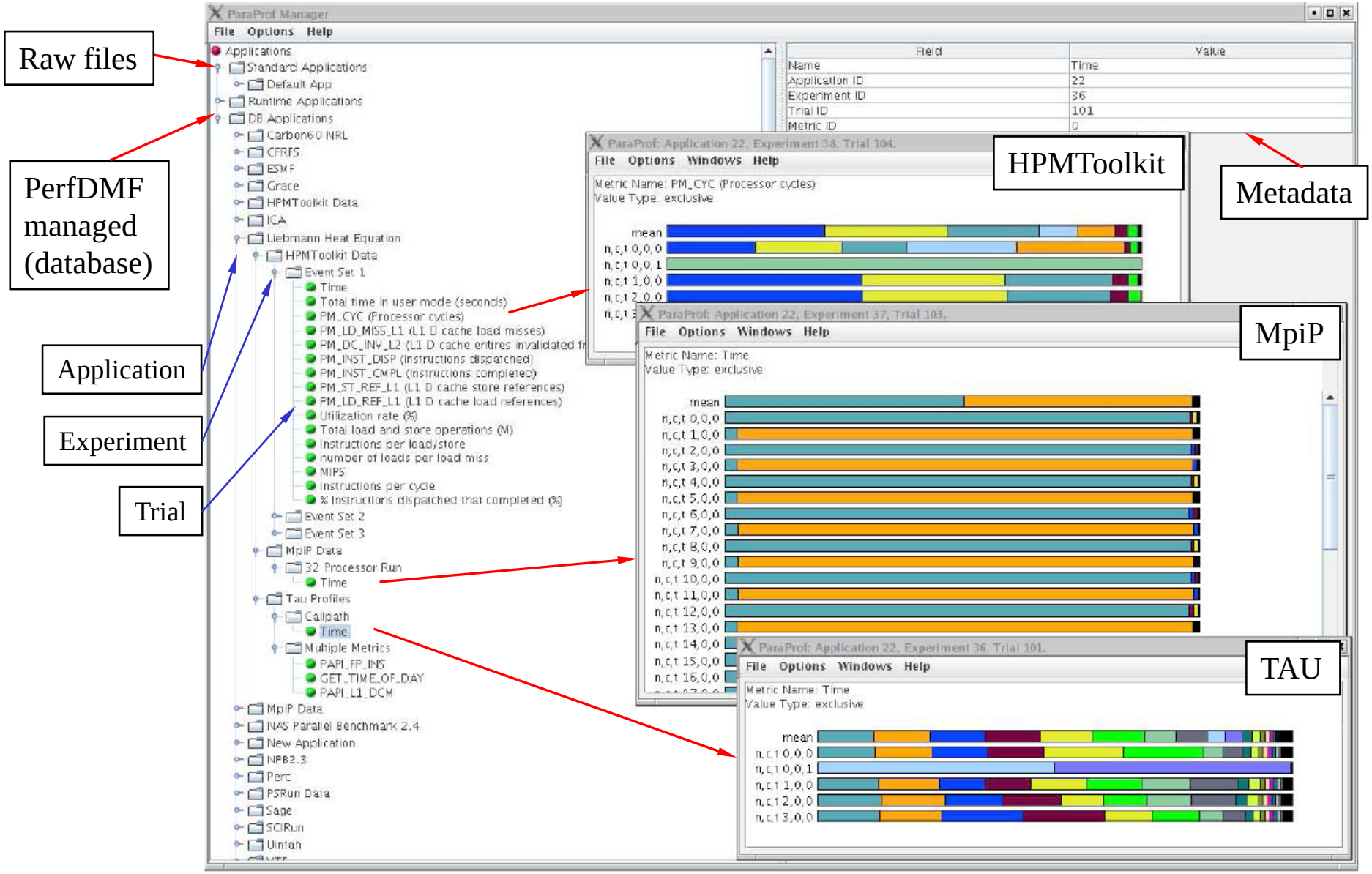
Performance Analysis and Visualization

- Analysis of parallel profile and trace measurement
- **Parallel profile analysis (*ParaProf*)**
  - Java-based analysis and visualization tool
  - Support for large-scale parallel profiles
- Performance data management framework (***PerfDMF***)
- **Parallel trace analysis**
  - Translation to *VTF* (V3.0), *EPILOG*, *OTF* formats
  - Integration with *Vampir / Vampir Server* (TU Dresden)
  - Profile generation from trace data
- Online parallel analysis and visualization
- Integration with *CUBE* browser (KOJAK, UTK, FZJ)

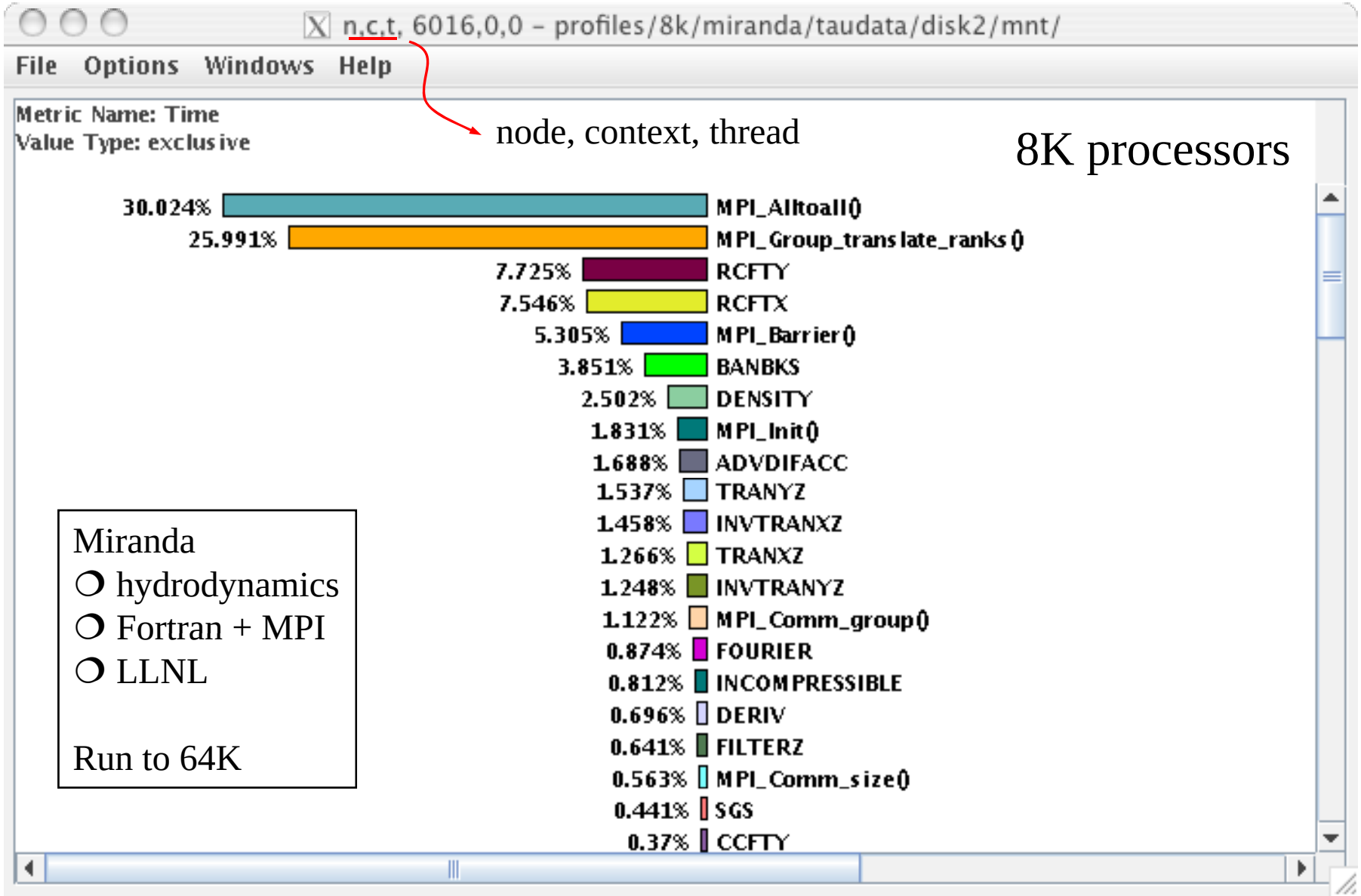
ParaProf – Parallel Performance Profile Analysis



ParaProf – Manager Window



ParaProf – Flat Profile (Miranda, BG/L)

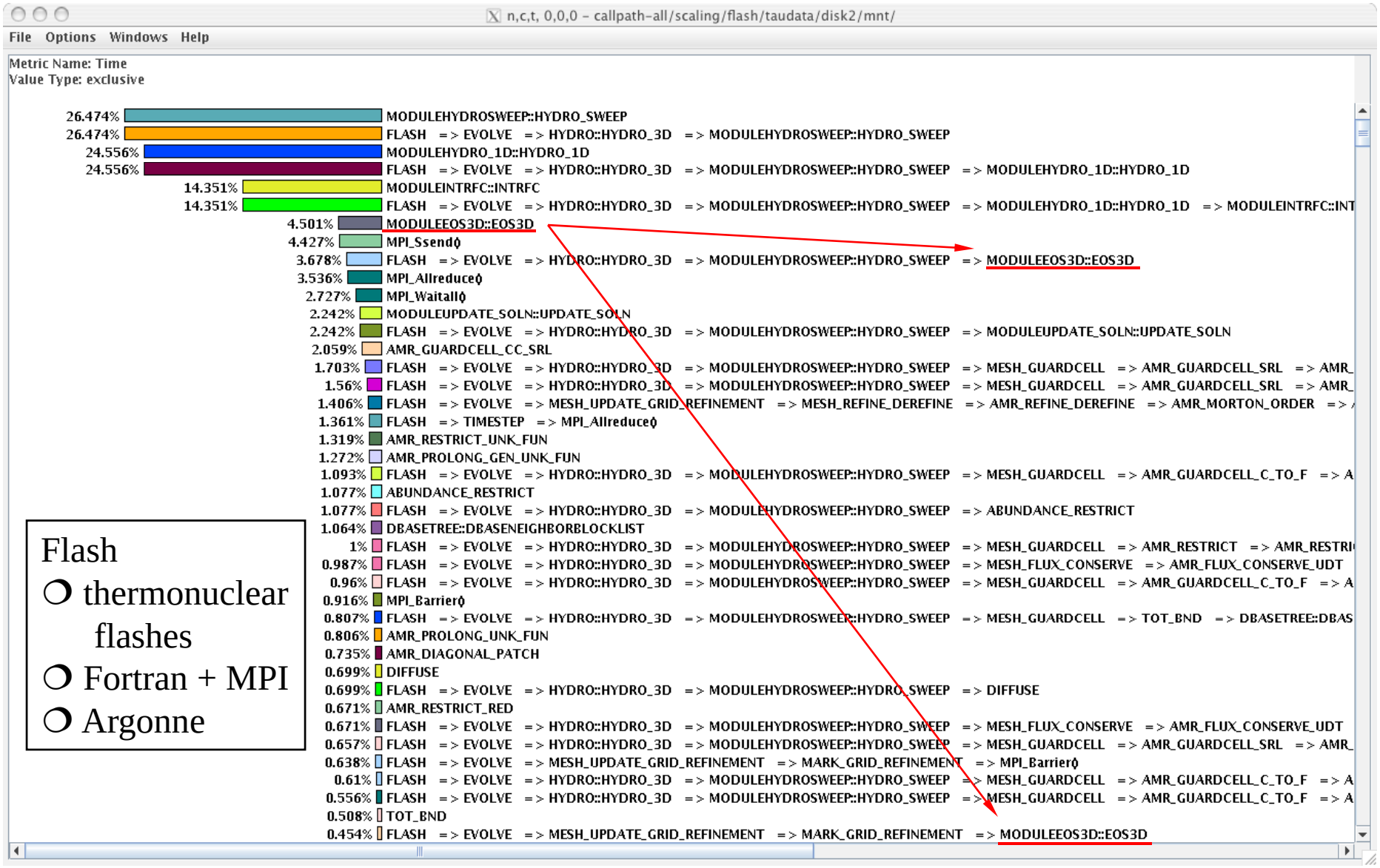


Miranda
- hydrodynamics
- Fortran + MPI
- LLNL

Run to 64K

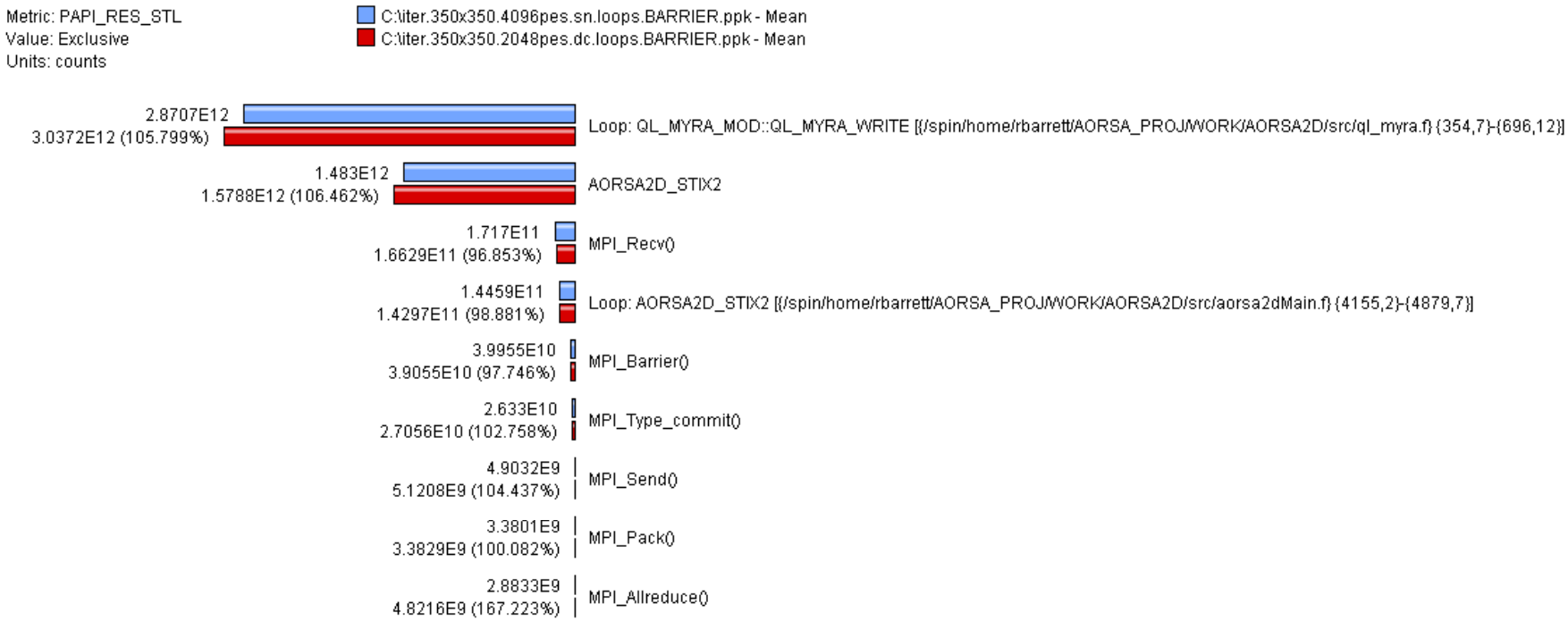
ParaProf – Stacked View (Miranda)

ParaProf – Callpath Profile (Flash)



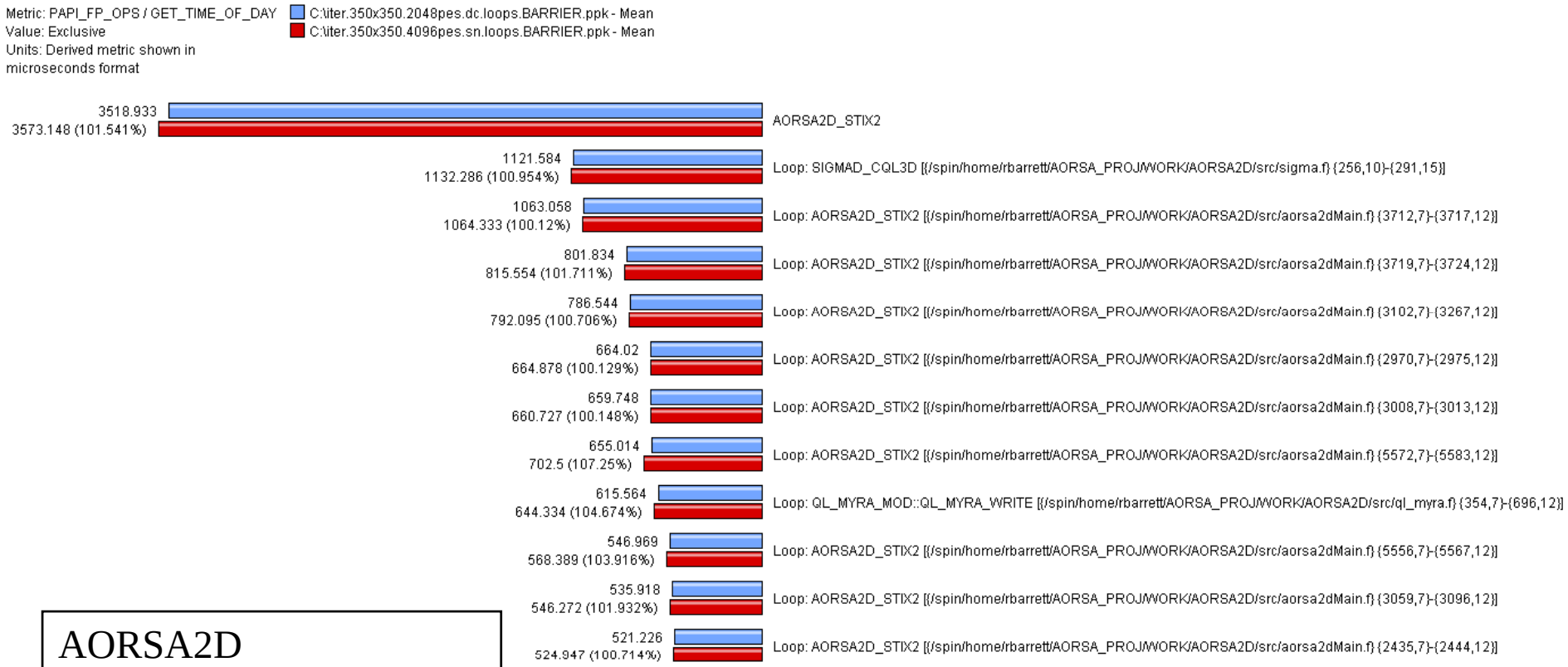
Flash
- thermonuclear flashes
- Fortran + MPI
- Argonne

Comparing Effects of Multi-Core Processors



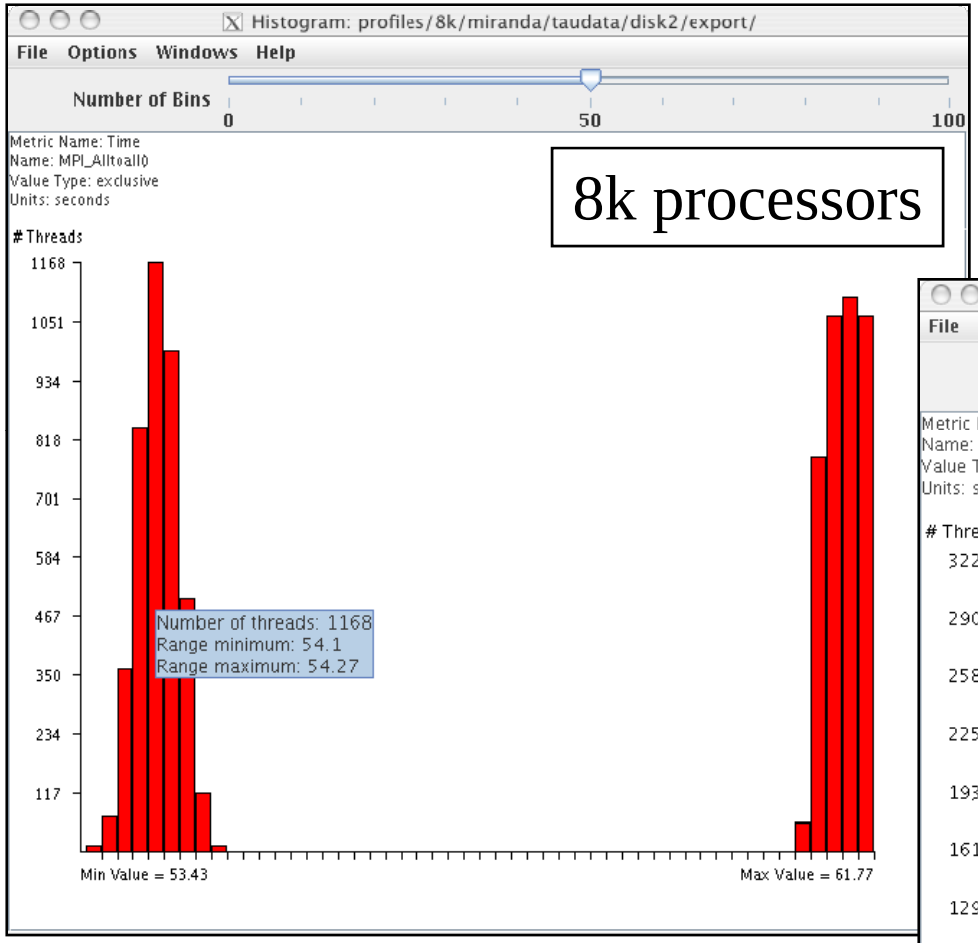
AORSA2D
- magnetized plasma simulation
- Blue is single node
- Red is dual core
- Cray XT3 (4K cores)

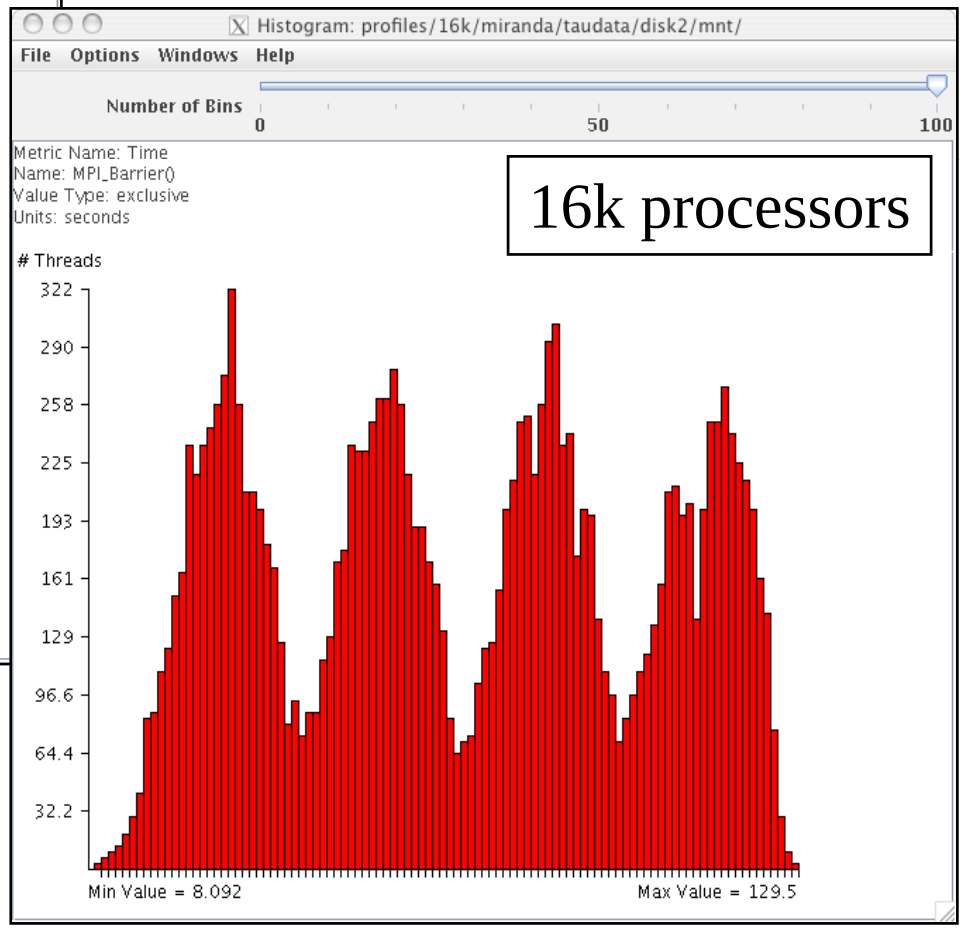
Comparing FLOPS (AORSA2D, Cray XT3)



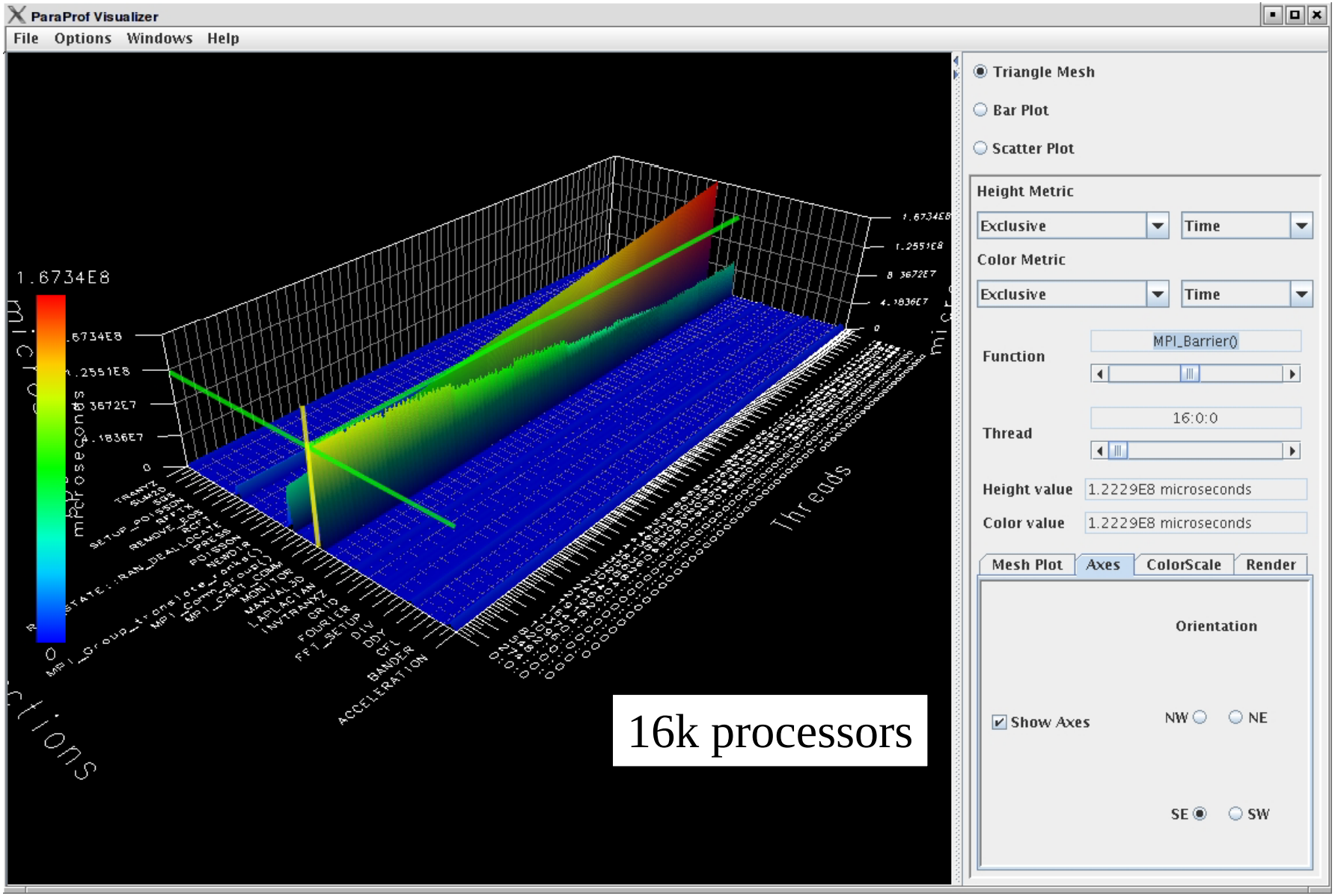
AORSA2D
- Blue is dual core
- Red is single node
- Cray XT3 (4K cores)
- Data generated by Richard Barrett, ORNL

ParaProf – Scalable Histogram View (Miranda)

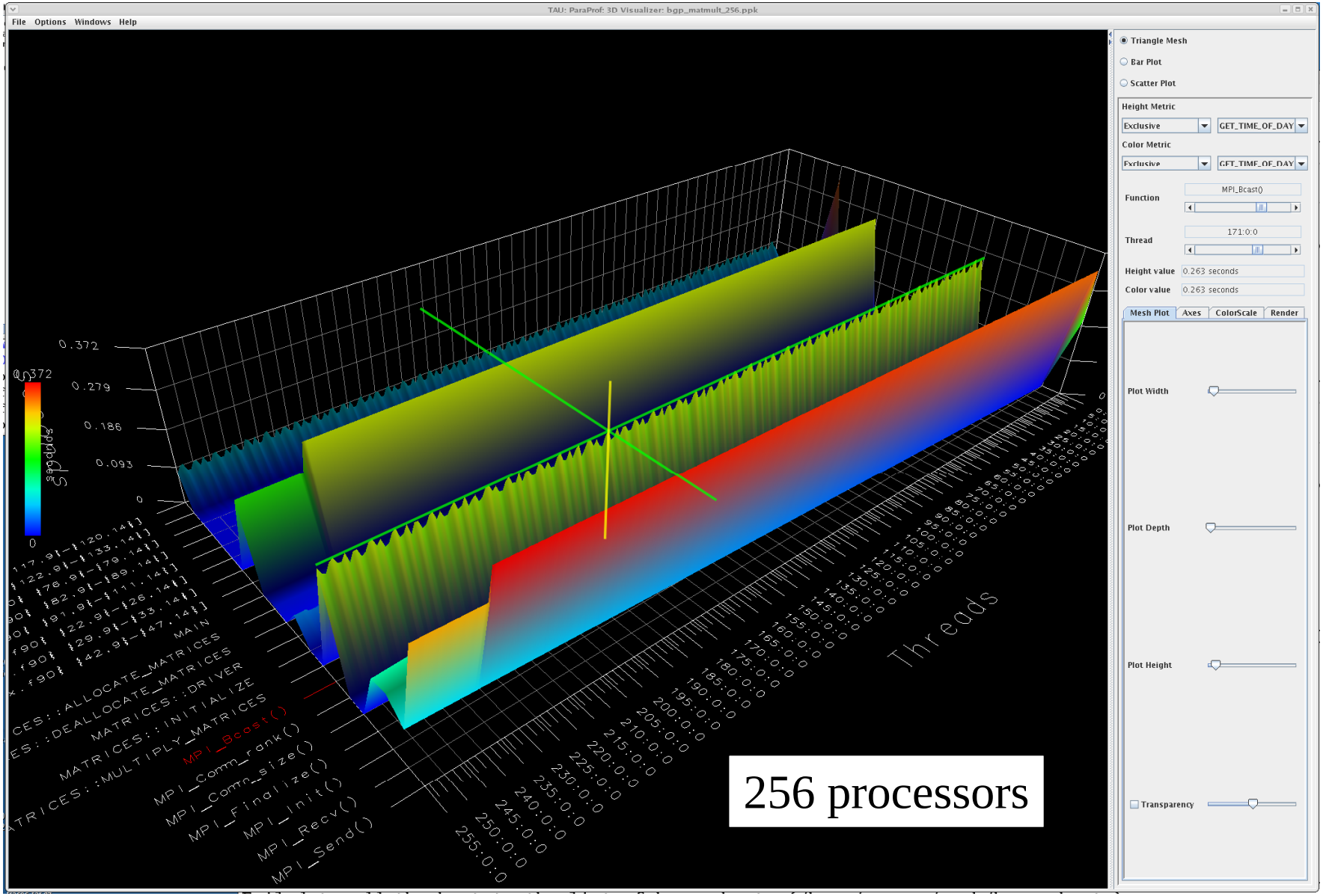




ParaProf – Full Profile (Miranda)

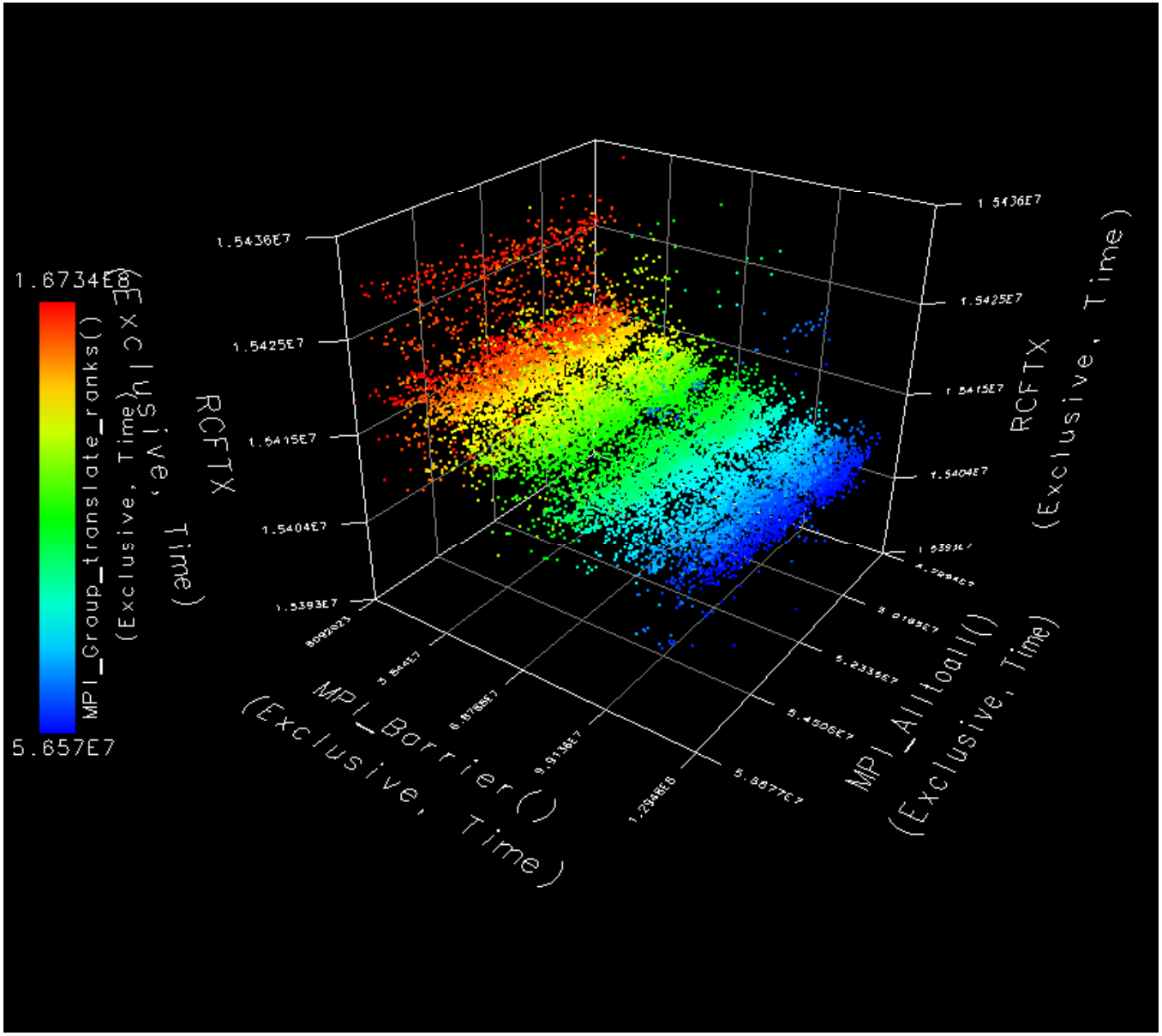


ParaProf –Full Profile (Matmult, ANL BGP)



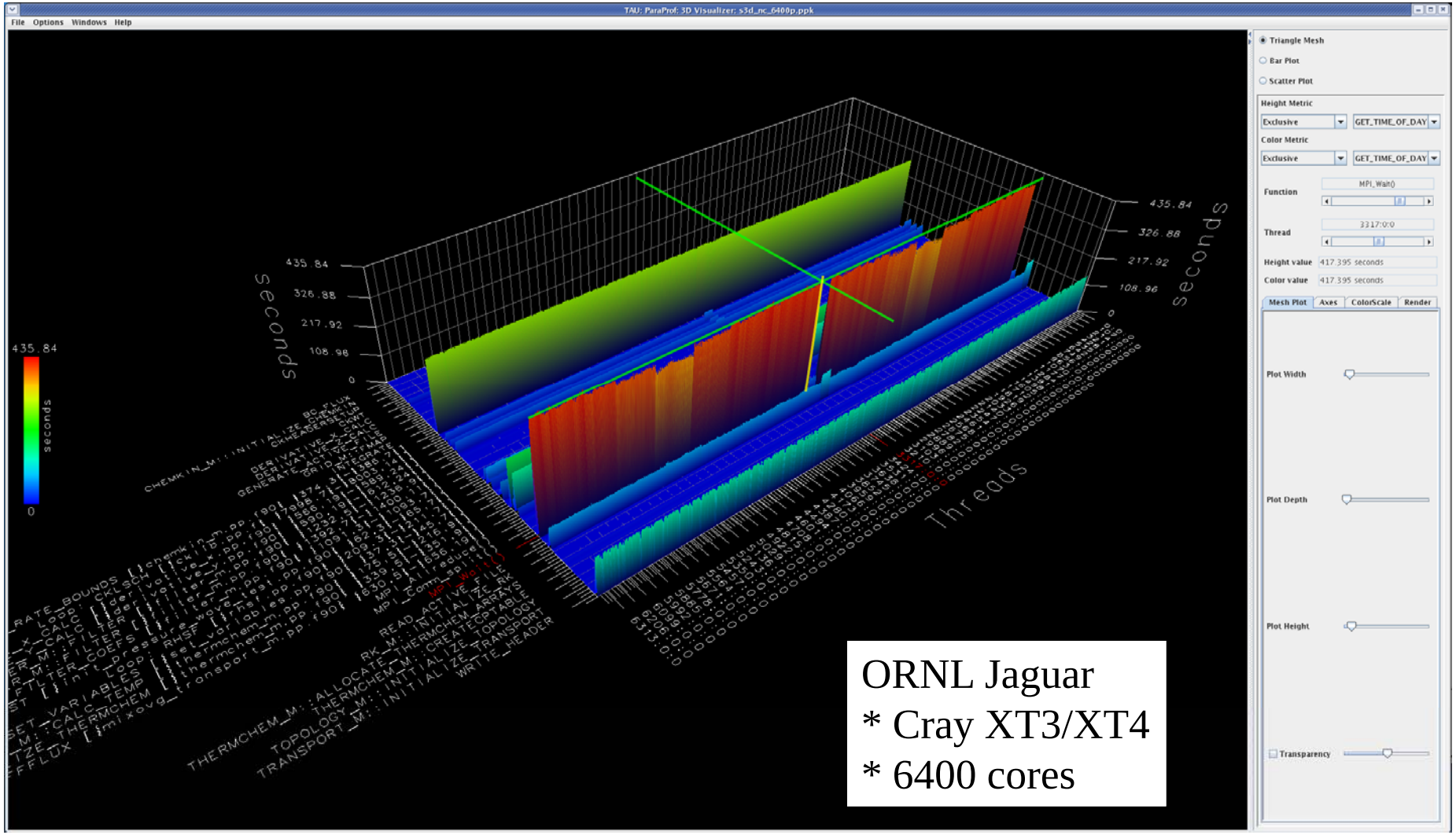
ParaProf – 3D Scatterplot (Miranda)

- Each point is a "thread" of execution
- A total of four metrics shown in relation
- ParaProf's visualization library
  - JOGL

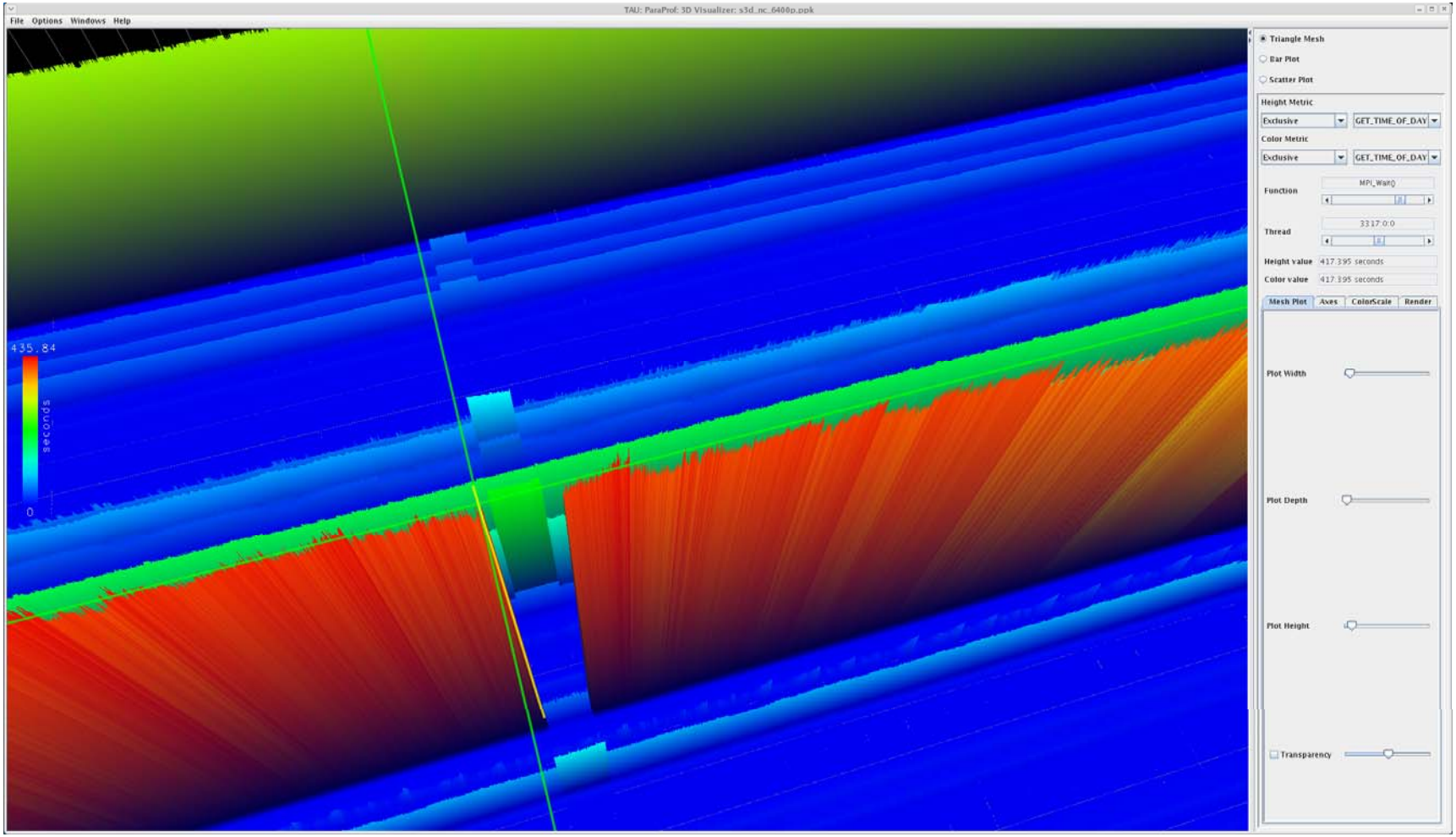


Visualizing Hybrid Problems (S3D, XT3+XT4)

- S3D combustion simulation (DOE SciDAC PERI)



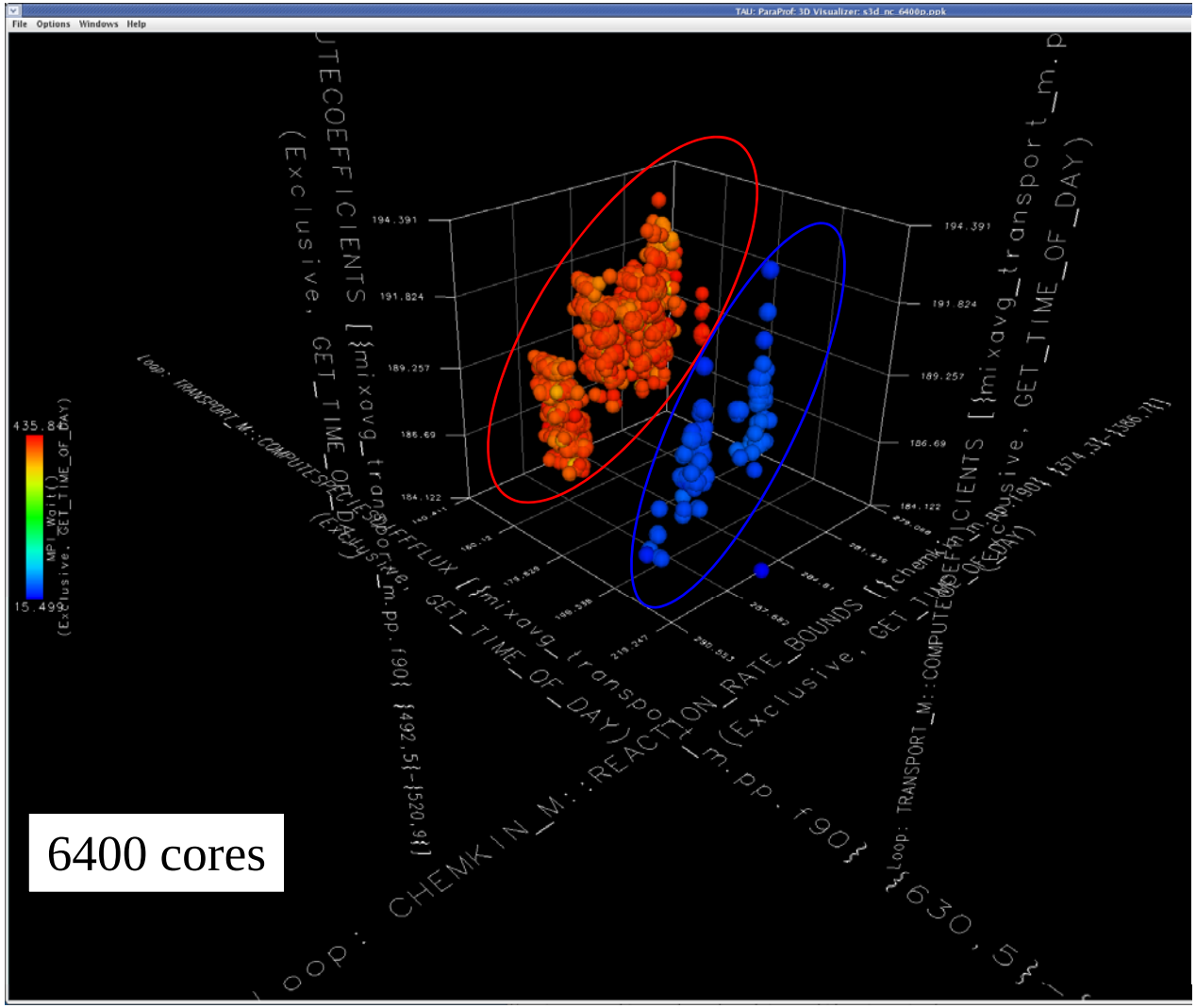
Zoom View of Hybrid Execution (S3D, XT3+XT4)



- Gap represents XT3 nodes
  - MPI_Wait takes less time, other routines take more time

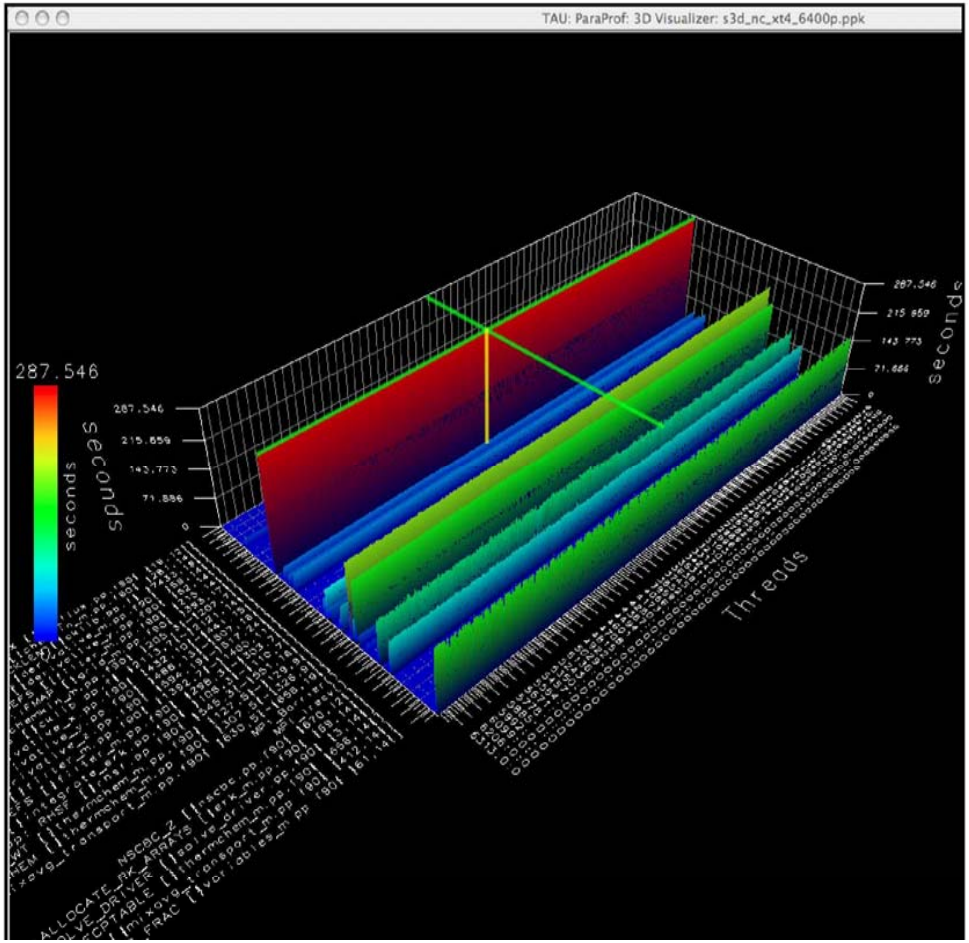
Visualizing Hybrid Execution (S3D, XT3+XT4)

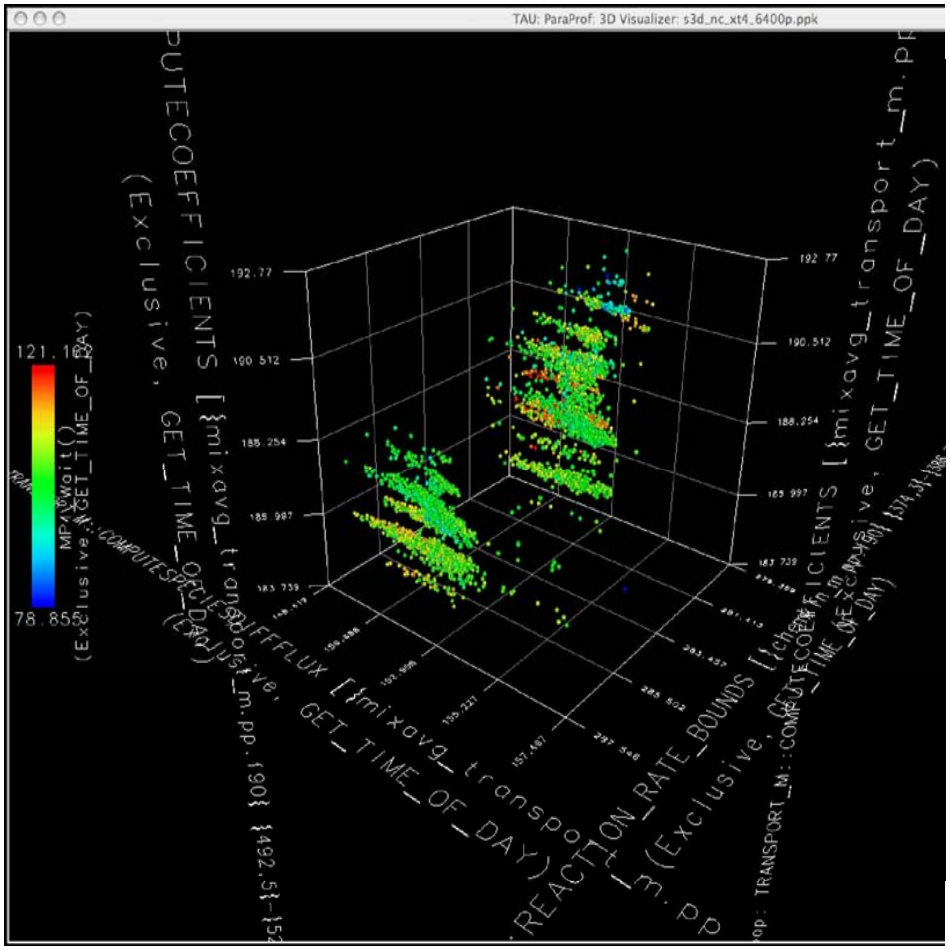
- Hybrid execution
- Process metadata is used to map performance to machine type
- Memory speed accounts for performance difference



S3D Run on XT4 Only

- Better balance across nodes
- More performance uniformity





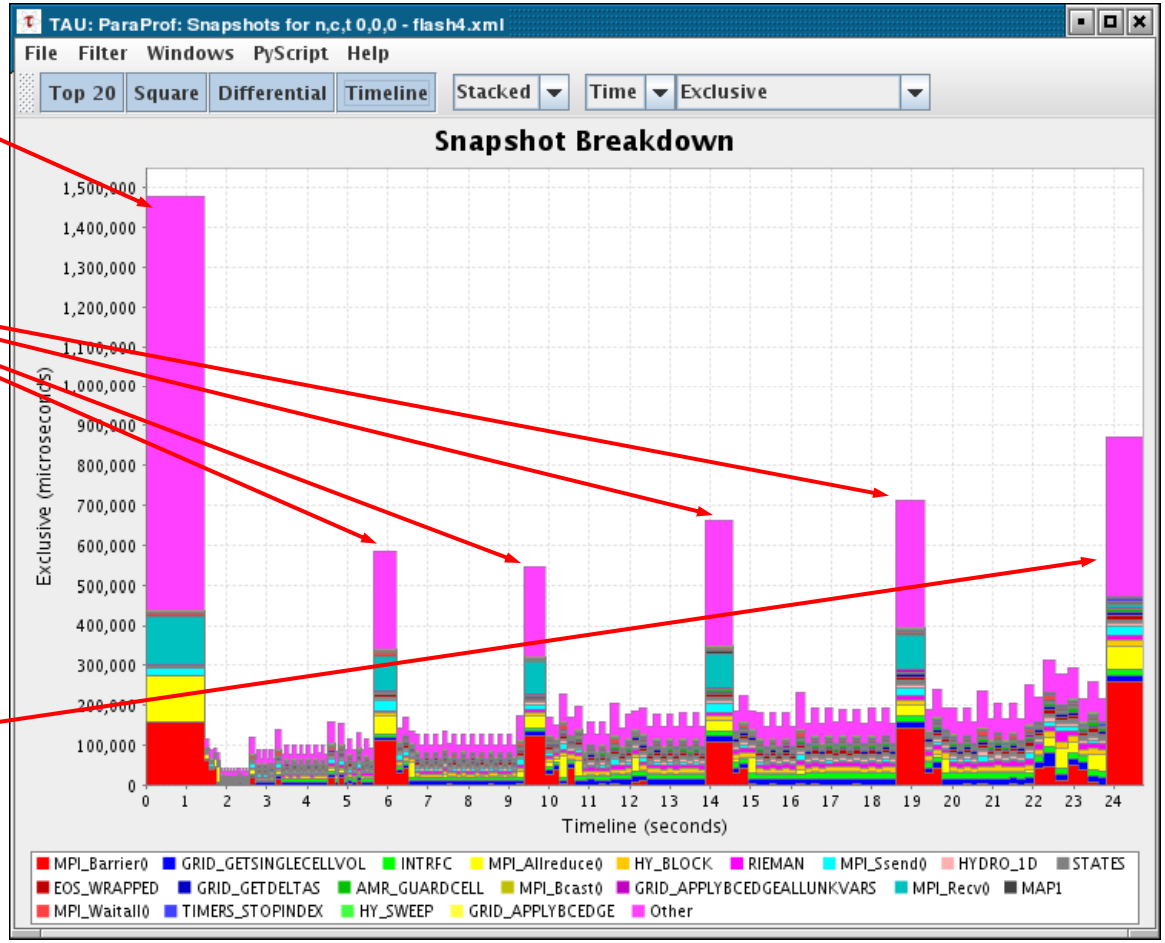
ParaProf – Profile Snapshots (Flash)

- Profile snapshots are parallel profiles recorded at runtime
- Used to highlight profile changes during execution

Initialization

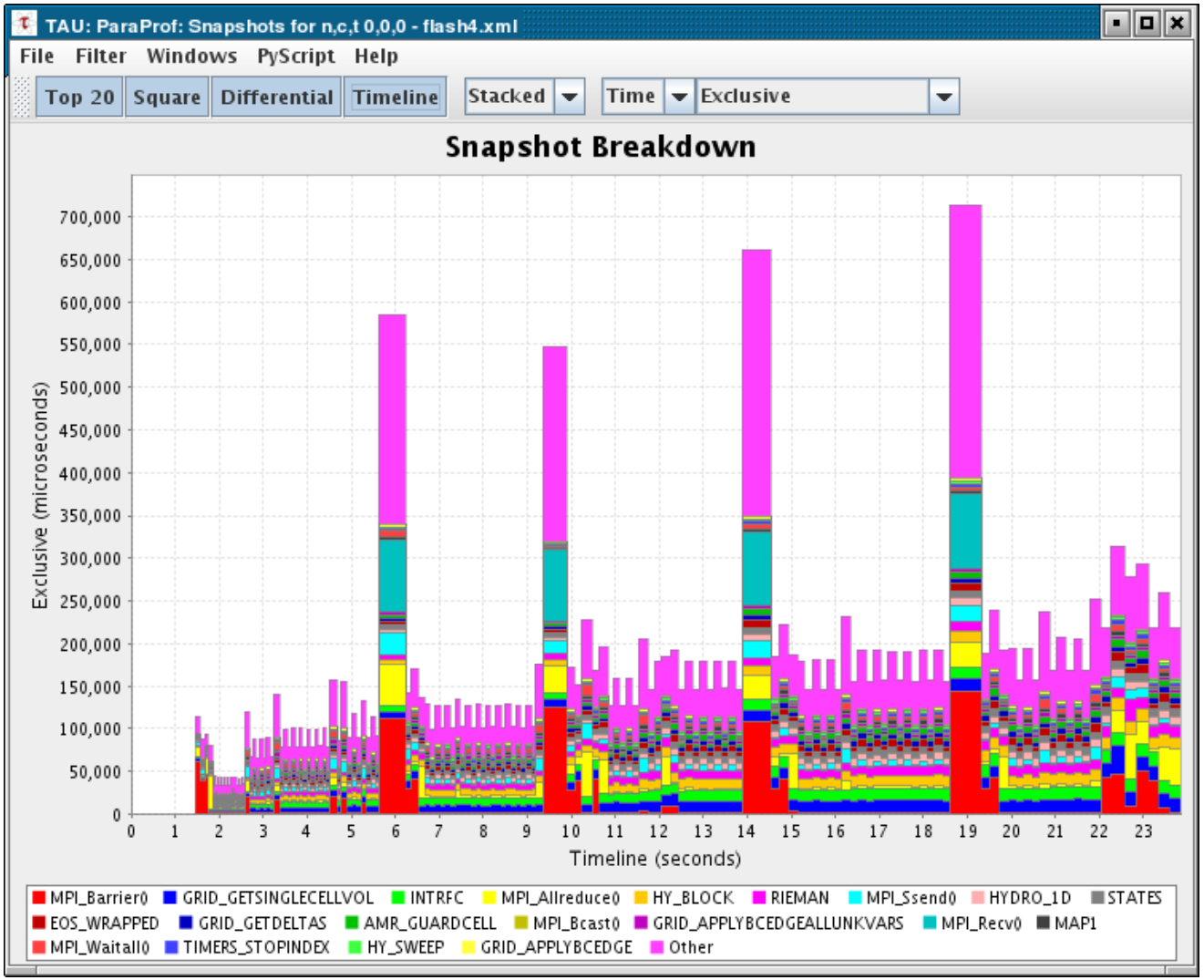
Checkpointing

Finalization



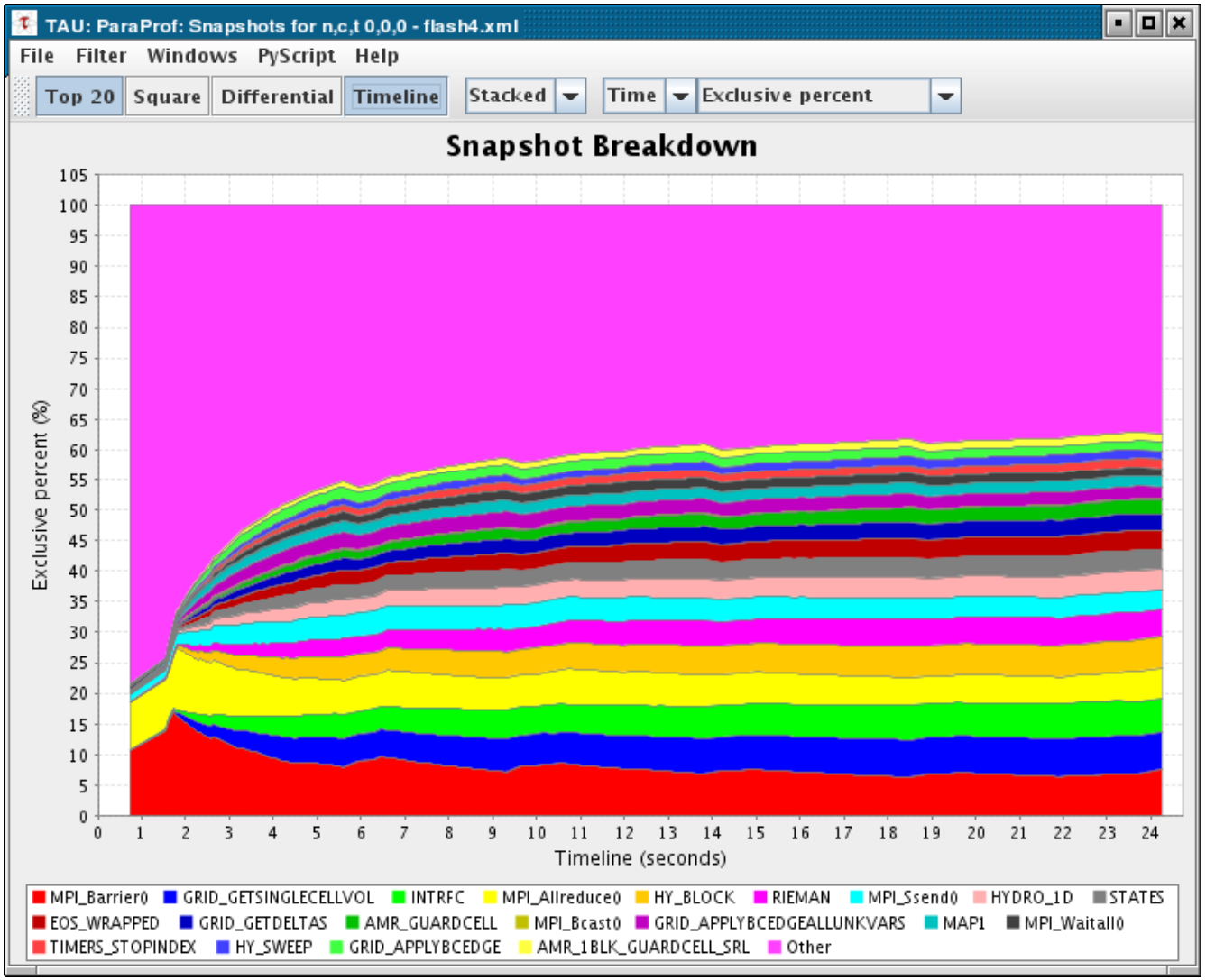
Filtered Profile Snapshots (Flash)

- Only show main loop iterations

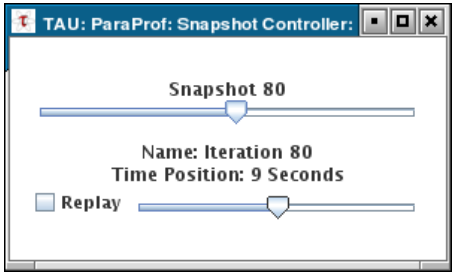


Profile Snapshots with Breakdown (Flash)

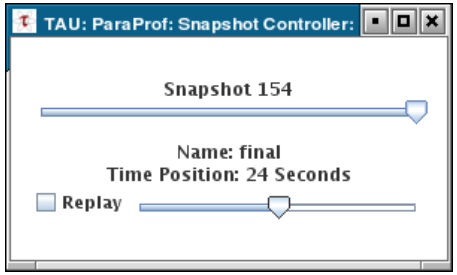
- Breakdown as a percentage

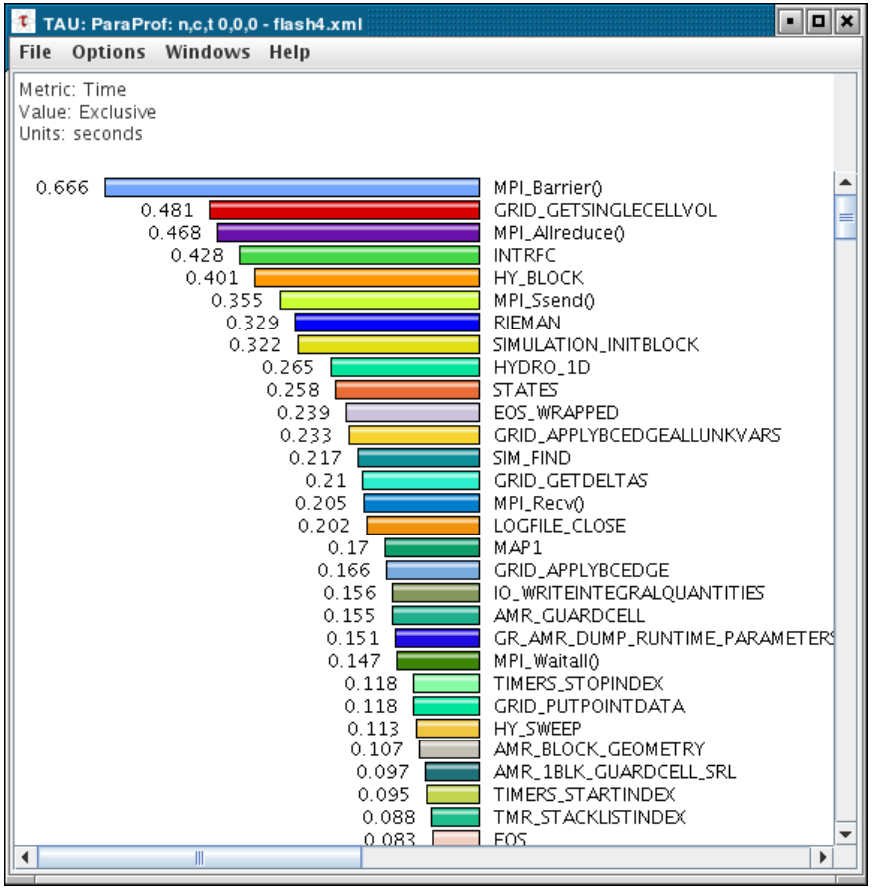


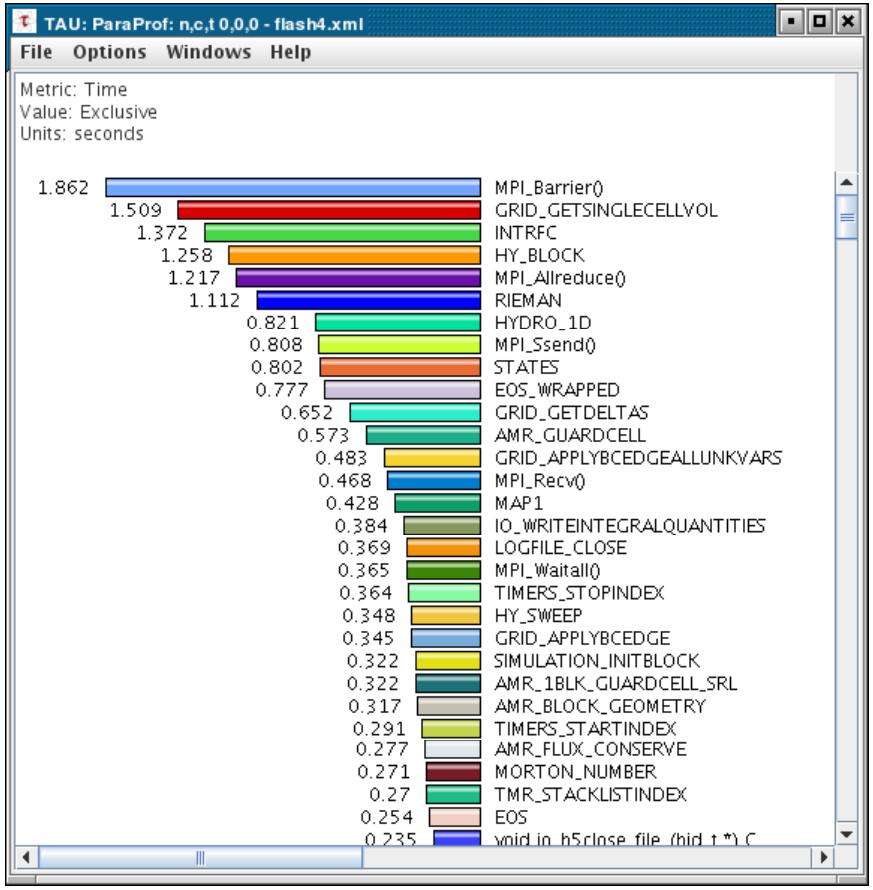
Profile Snapshot Replay (Flash)



All windows dynamically update

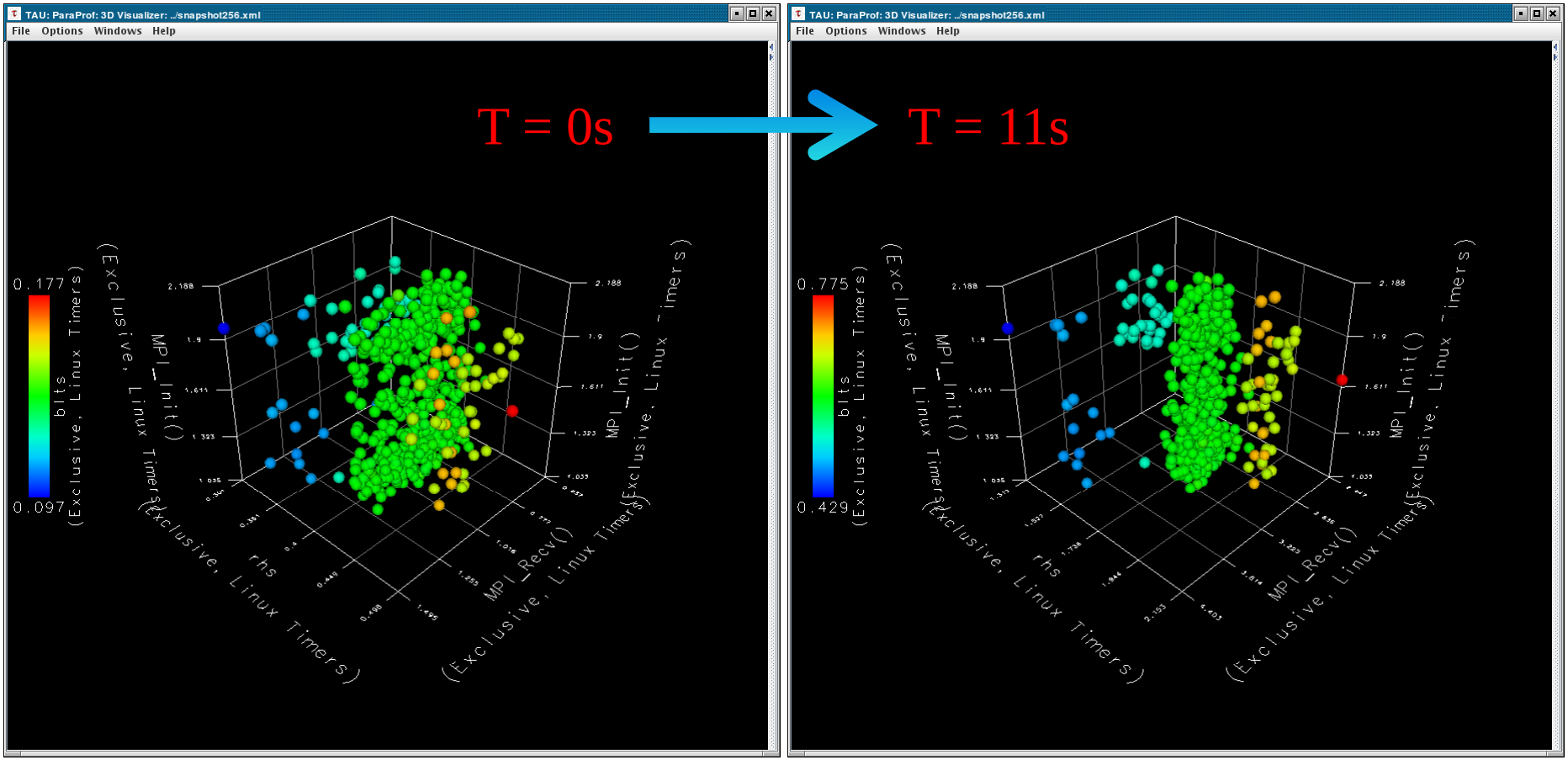






Snapshot Dynamics of Event Relations (Flash)

- Follow progression of various displays through time
- 3D scatter plot shown below



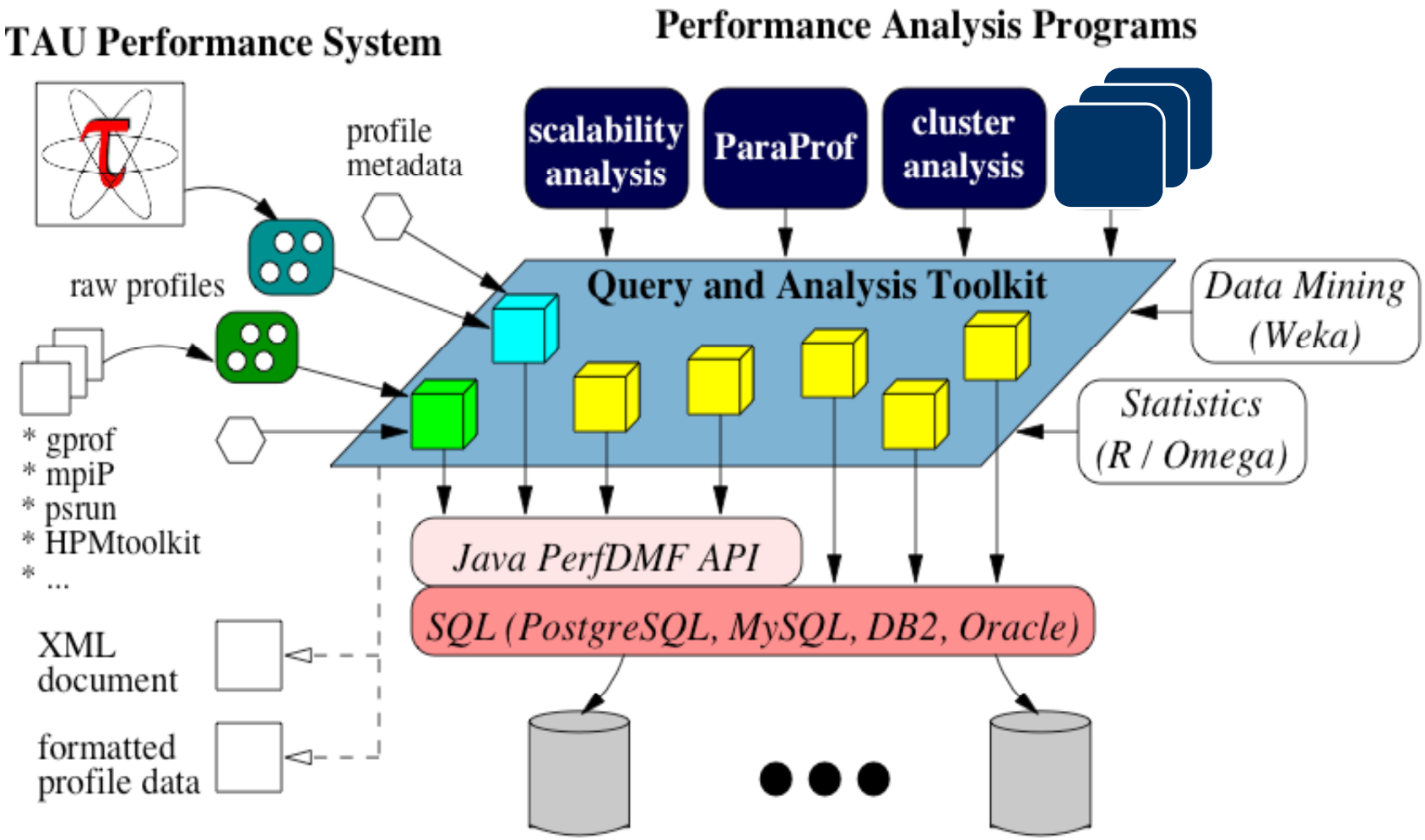
Performance Data Management

- Need for robust processing and storage of multiple profile performance data sets
- Avoid developing independent data management solutions
  - Waste of resources
  - Incompatibility among analysis tools
- Goals
  - Foster multi-experiment performance evaluation
  - Develop a common, reusable foundation of performance data storage, access and sharing
  - A core module in an analysis system, and/or as a central repository of performance data

PerfDMF Approach

- Performance Data Management Framework
- Originally designed to address critical TAU requirements
- Broader goal is to provide an open, flexible framework to support common data management tasks
- Extensible toolkit to promote integration and reuse across available performance tools
  - Supported profile formats: TAU, CUBE 2 & 3 (Kojak), Dynaprof, HPC Toolkit (Rice), HPM Toolkit (IBM), gprof, mpiP, psrun (PerfSuite), Open|SpeedShop, …
  - Supported DBMS: PostgreSQL, MySQL, Oracle, DB2, Derby/Cloudscape
  - Profile query and analysis API

PerfDMF Architecture

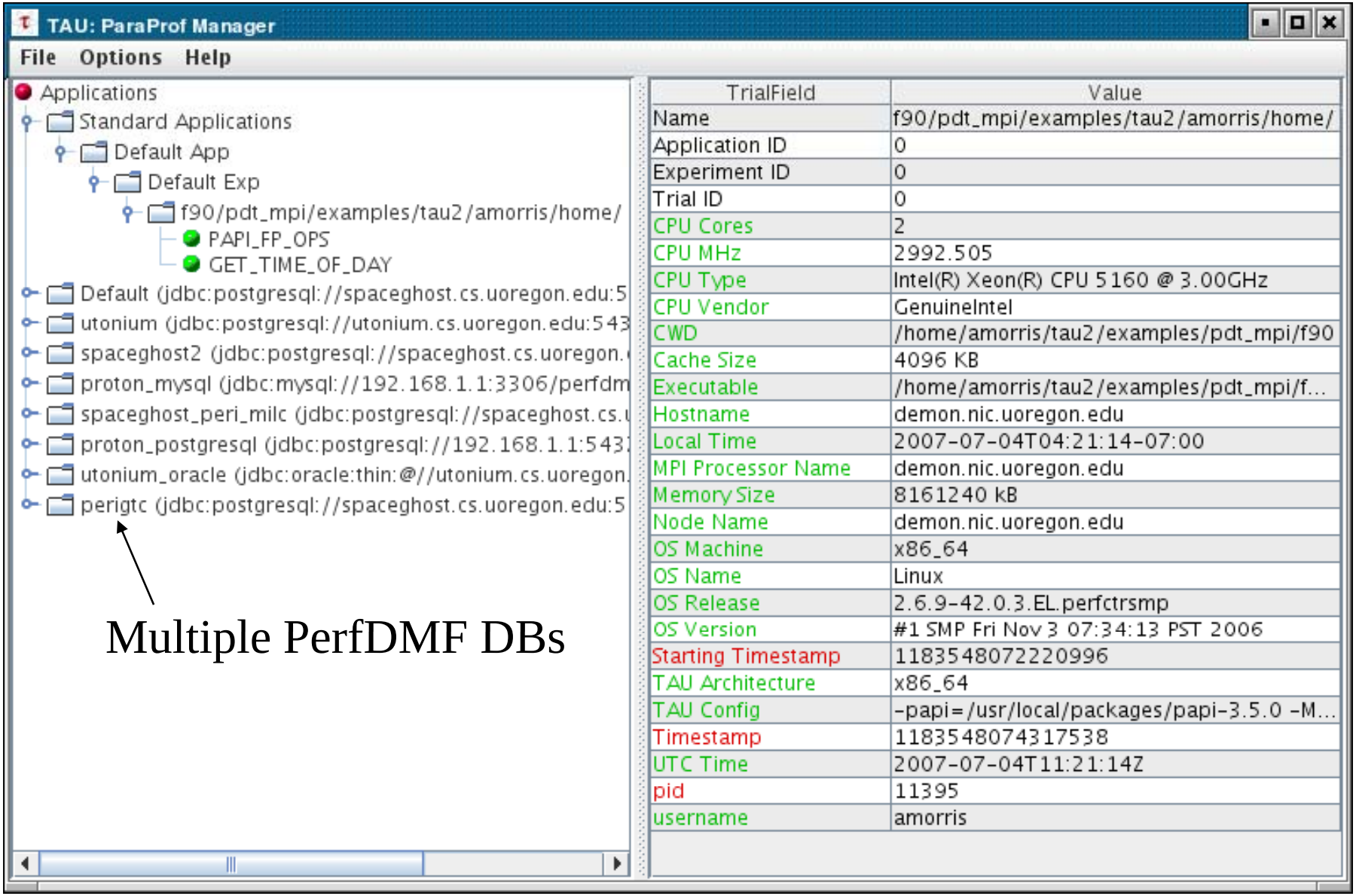


K. Huck, A. Malony, R. Bell, A. Morris, "**Design and Implementation of a Parallel Performance Data Management Framework**," ICPP 2005.

Metadata Collection

- Integration of XML metadata for each profile
- Three ways to incorporate metadata
  - Measured hardware/system information (TAU, PERI-DB)
    - CPU speed, memory in GB, MPI node IDs, …
  - Application instrumentation (application-specific)
    - TAU_METADATA() used to insert any name/value pair
    - Application parameters, input data, domain decomposition
  - PerfDMF data management tools can incorporate an XML file of additional metadata
    - Compiler flags, submission scripts, input files, …
- Metadata can be imported from / exported to PERI-DB
  - PERI SciDAC project (UTK, NERSC, UO, PSU, TAMU)

Metadata for Each Experiment



TrialField	Value
Name	f90/pdt_mpi/examples/tau2/amorris/home/
Application ID	0
Experiment ID	0
Trial ID	0
CPU Cores	2
CPU MHz	2992.505
CPU Type	Intel(R) Xeon(R) CPU 5160 @ 3.00GHz
CPU Vendor	GenuineIntel
CWD	/home/amorris/tau2/examples/pdt_mpi/f90
Cache Size	4096 KB
Executable	/home/amorris/tau2/examples/pdt_mpi/f...
Hostname	demon.nic.uoregon.edu
Local Time	2007-07-04T04:21:14-07:00
MPI Processor Name	demon.nic.uoregon.edu
Memory Size	8161240 kB
Node Name	demon.nic.uoregon.edu
OS Machine	x86_64
OS Name	Linux
OS Release	2.6.9-42.0.3.EL.perfctrsmp
OS Version	#1 SMP Fri Nov 3 07:34:13 PST 2006
Starting Timestamp	1183548072220996
TAU Architecture	x86_64
TAU Config	-papi=/usr/local/packages/papi-3.5.0 -M...
Timestamp	1183548074317538
UTC Time	2007-07-04T11:21:14Z
pid	11395
username	amorris

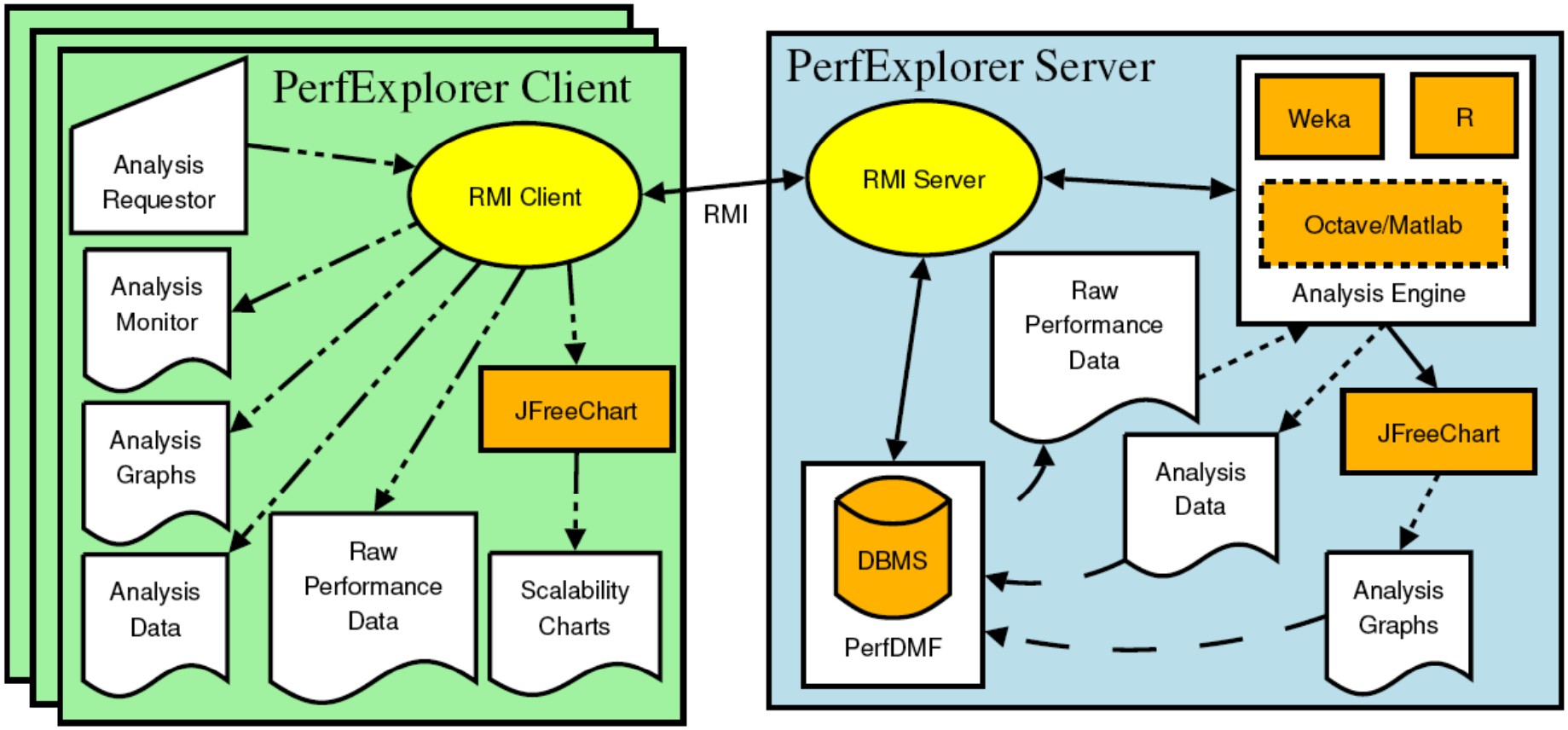
Performance Data Mining

- Conduct parallel performance analysis process
  - In a systematic, collaborative and reusable manner
  - Manage performance complexity
  - Discover performance relationship and properties
  - Automate process
- Multi-experiment performance analysis
- Large-scale performance data reduction
  - Summarize characteristics of large processor runs
- Implement extensible analysis framework
  - Abstraction / automation of data mining operations
  - Interface to existing analysis and data mining tools

Performance Data Mining (PerfExplorer)

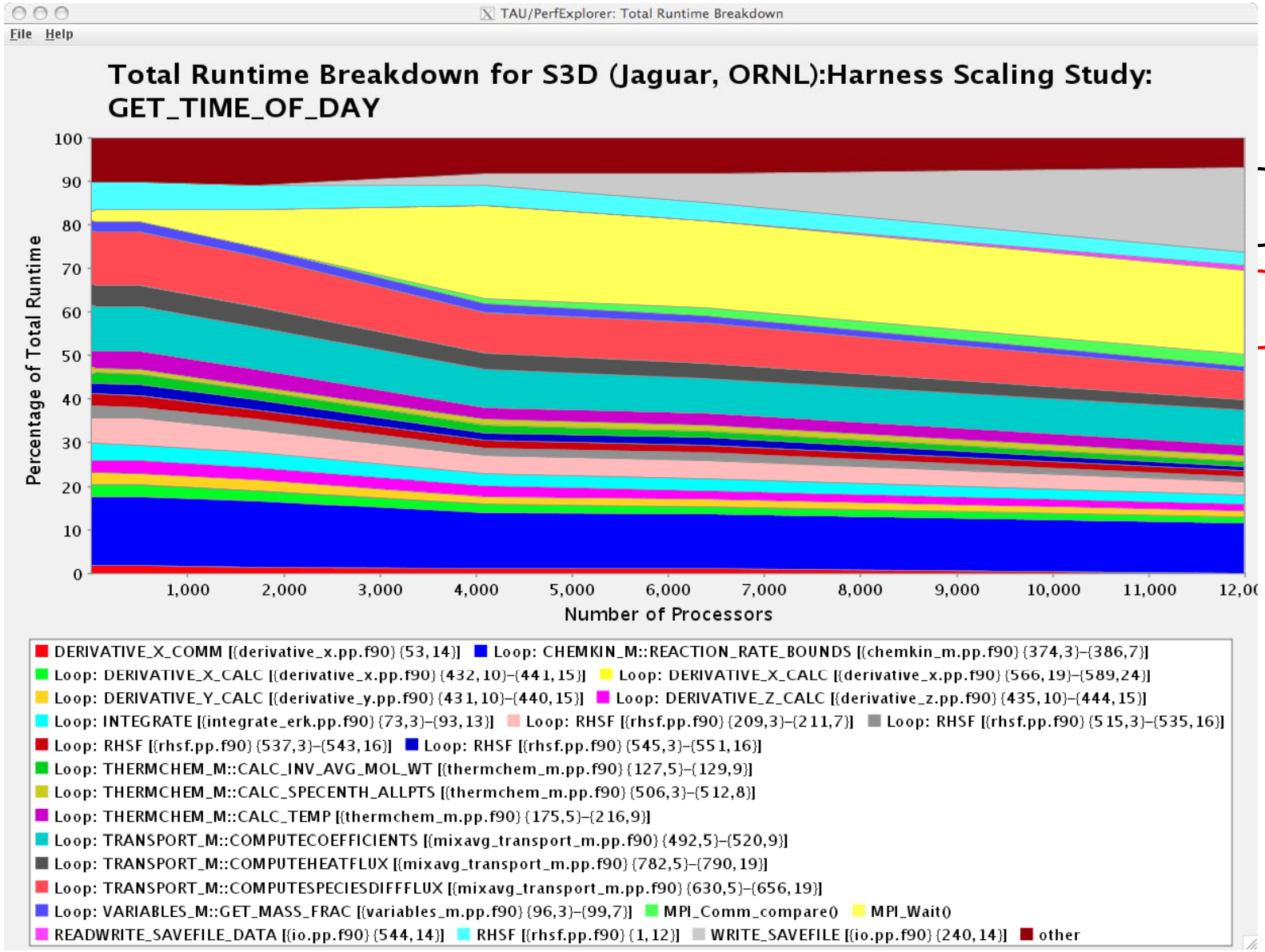
- Performance knowledge discovery framework
  - Data mining analysis applied to parallel performance data
    - comparative, clustering, correlation, dimension reduction, …
  - Use the existing TAU infrastructure
    - TAU performance profiles, PerfDMF
- Technology integration
  - Java API and toolkit for portability
  - Built on top of PerfDMF
  - R-project/Omegahat, Octave/Matlab statistical analysis
  - WEKA data mining package
  - JFreeChart for visualization, vector output (EPS, SVG)

Performance Data Mining (PerfExplorer v1)



K. Huck and A. Malony, "**PerfExplorer: A Performance Data Mining Framework For Large-Scale Parallel Computing**," SC 2005.

PerfExplorer: S3D Total Runtime Breakdown



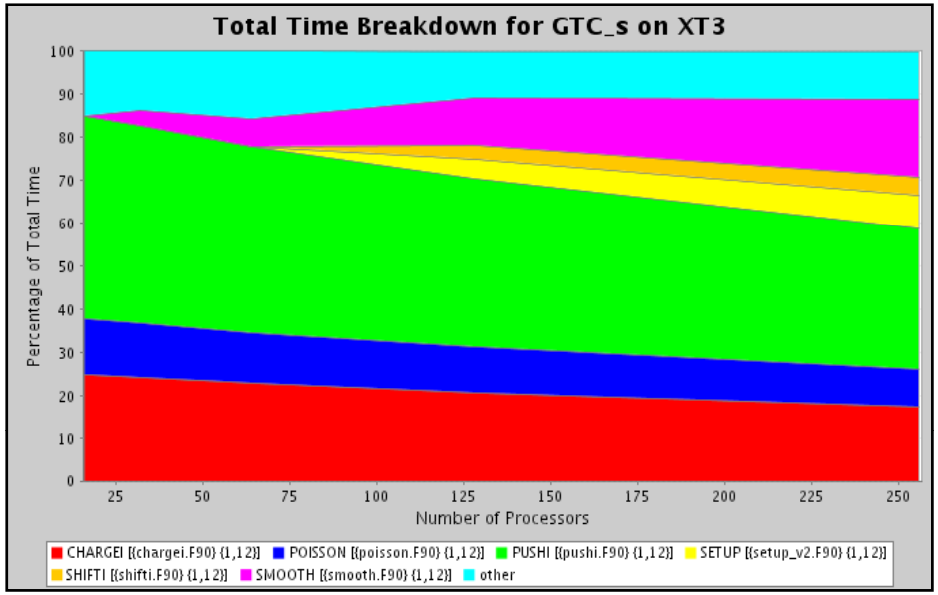
WRITE_ SAVEFILE

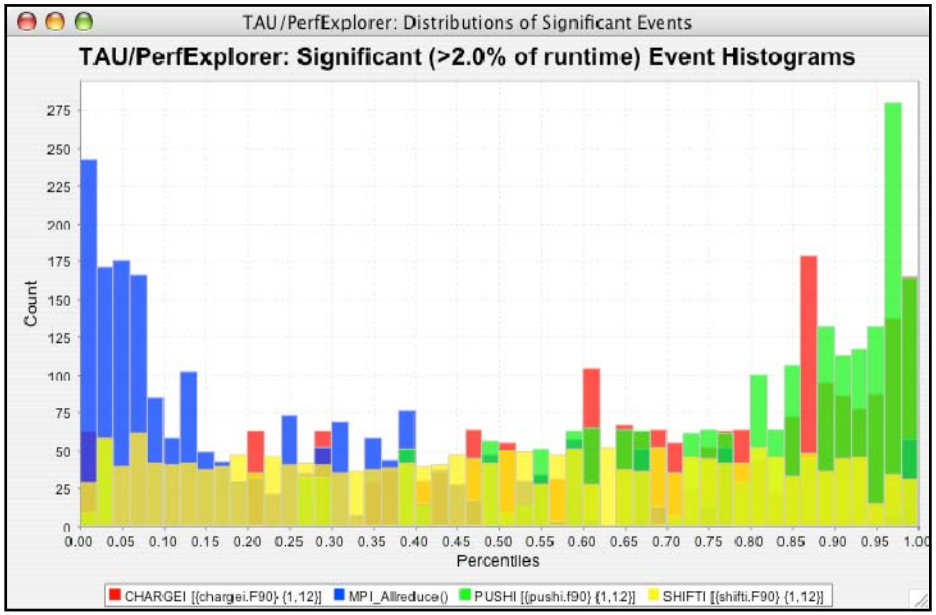
MPI_Wait

12,000 cores!

Relative Comparisons (GTC, XT3, DOE PERI)

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

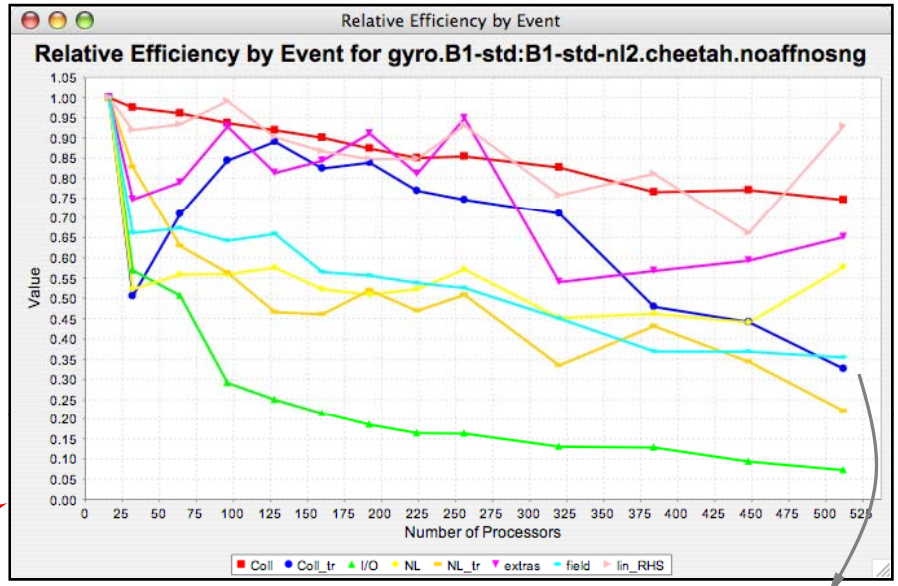


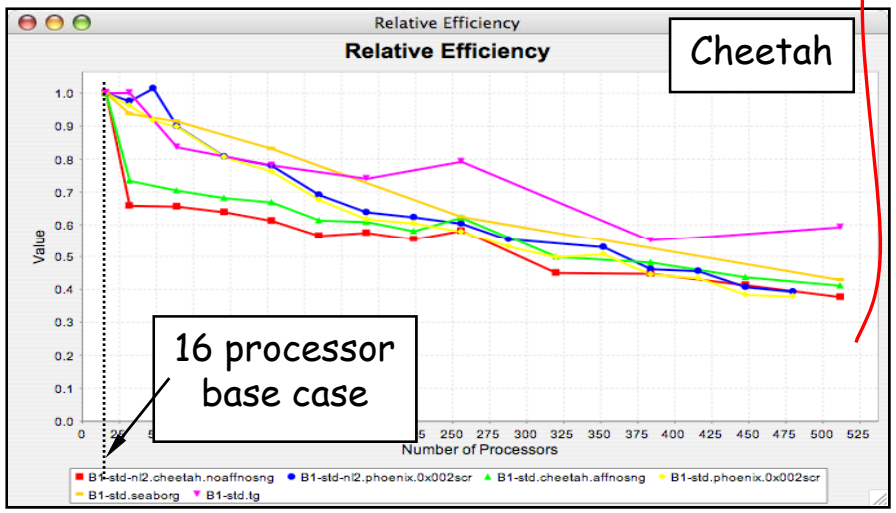


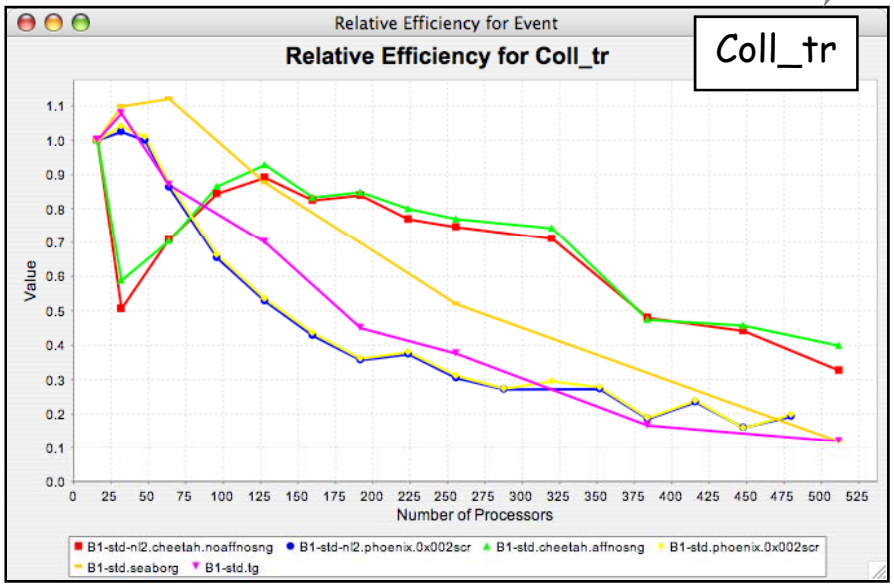
Data: GYRO on various architectures

PerfExplorer – GYRO Relative Efficiency

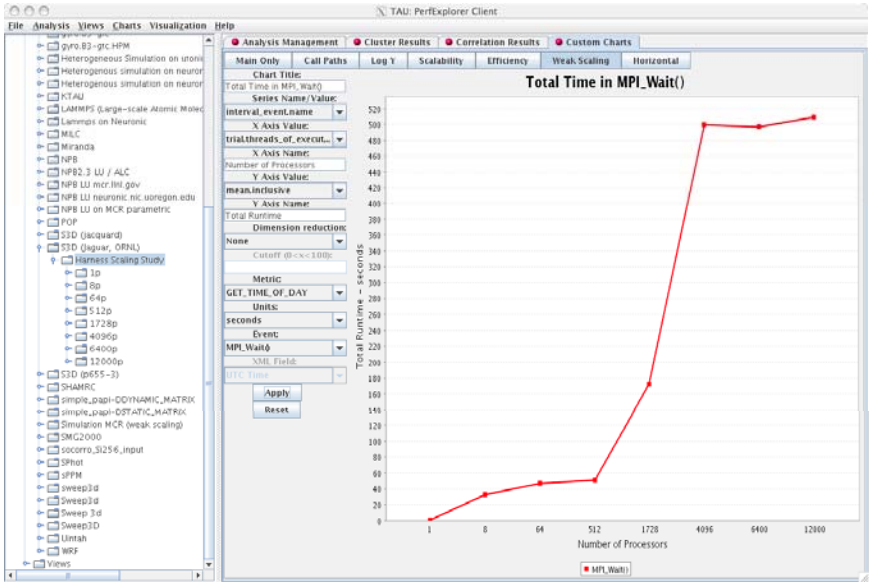
- By experiment (B1-std)
  - Total runtime (Cheetah (red))
- By event for one experiment
  - Coll_tr (blue) is significant
- By experiment for one event
  - Shows how Coll_tr behaves for all experiments
  - Data generated by Pat Worley, ORNL

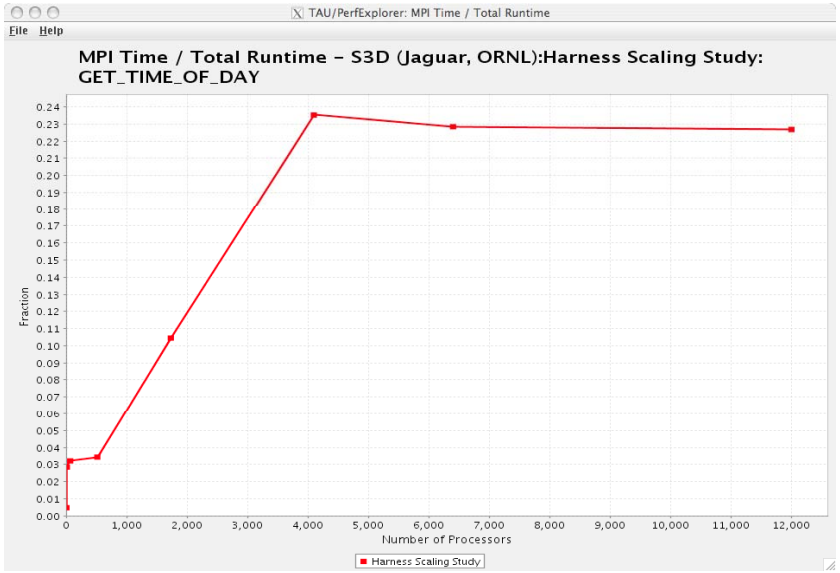


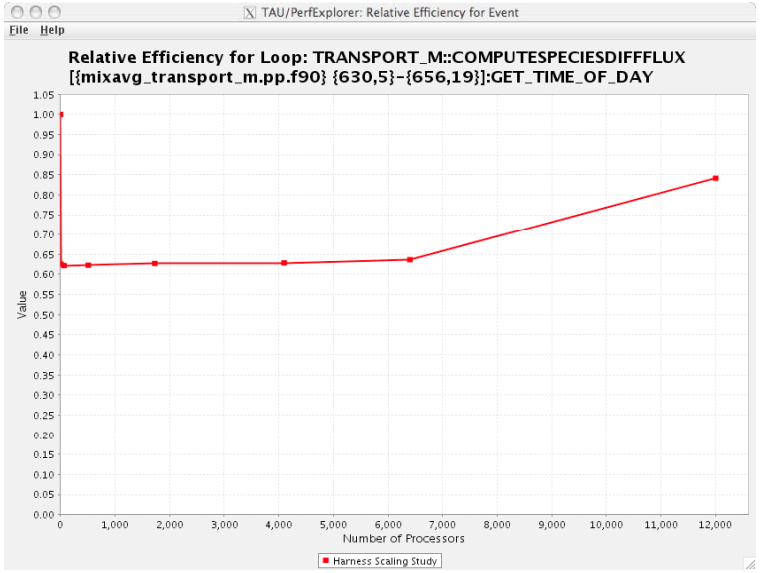


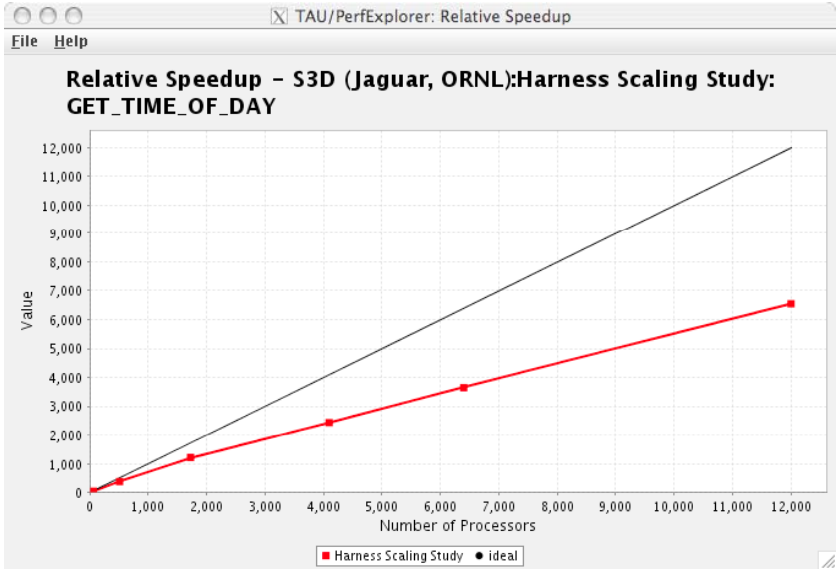


PerfExplorer: Cross Experiment Analysis for S3D

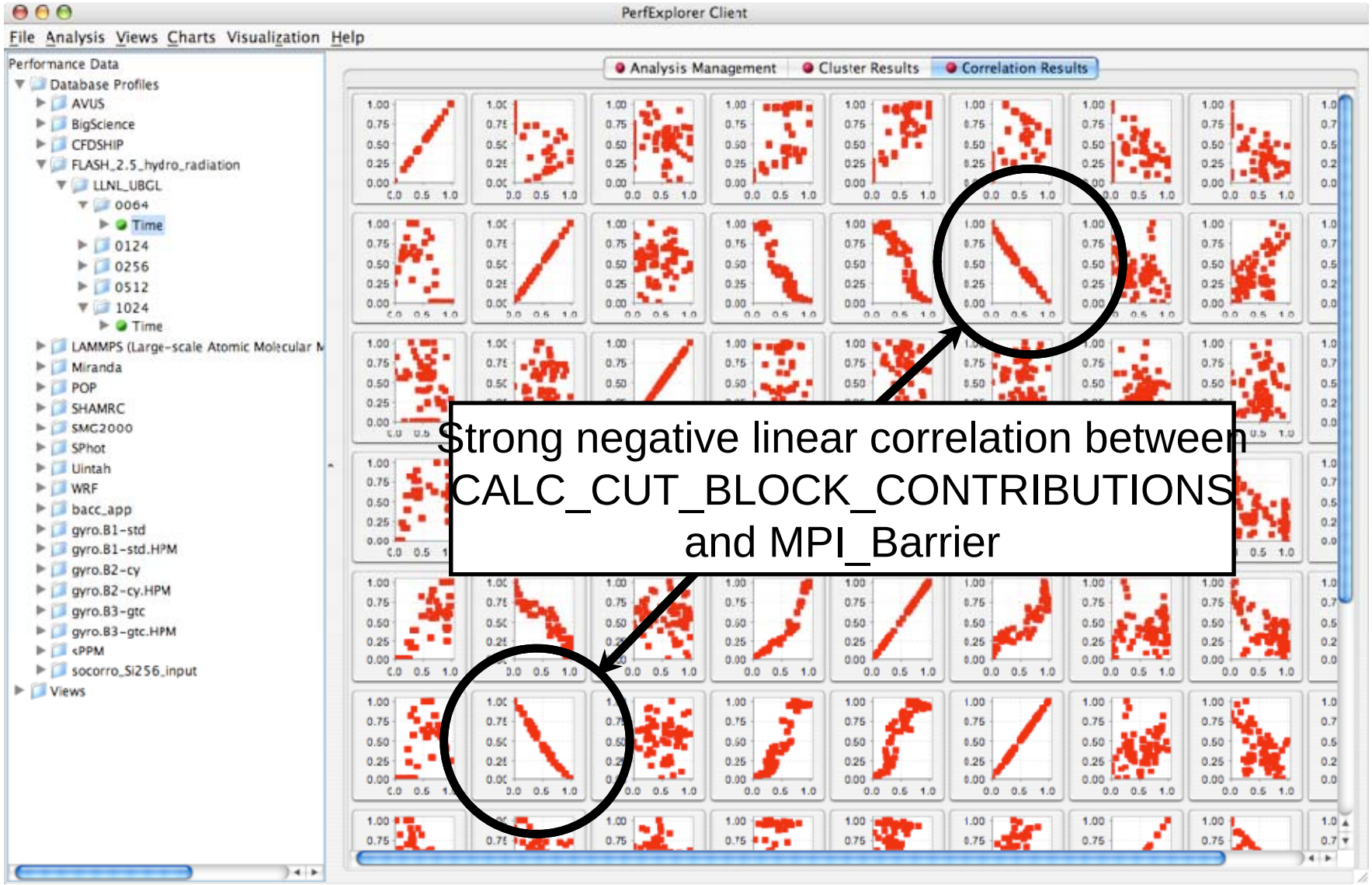








Correlation Analysis

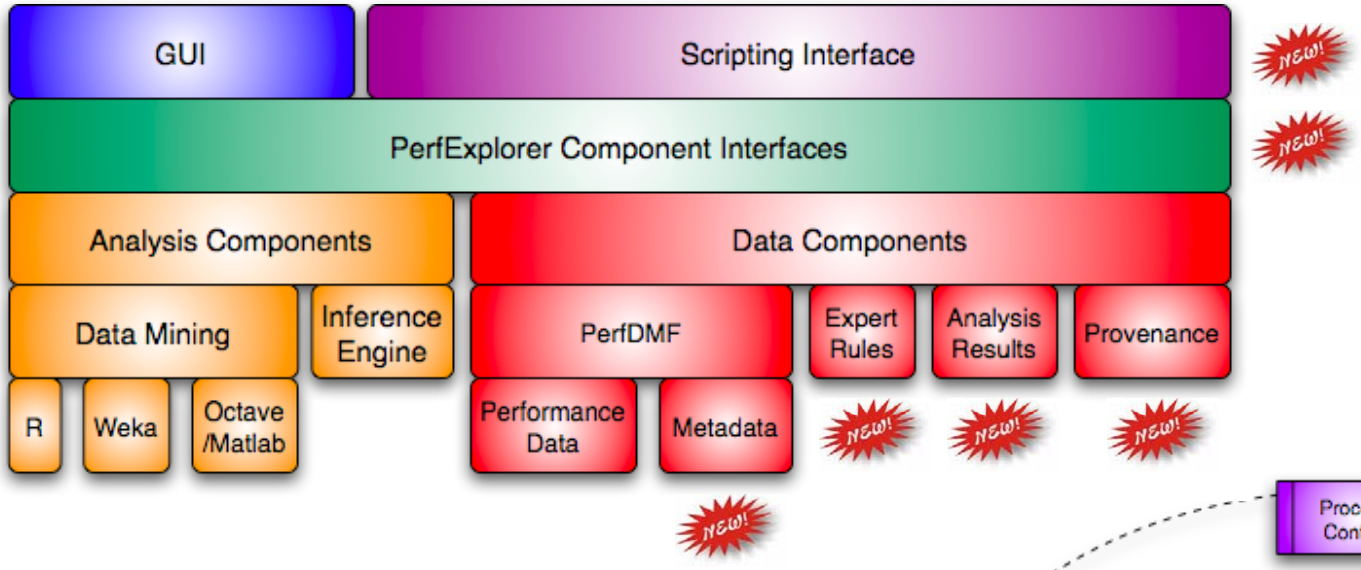


Data: FLASH on BGL(LLNL), 64 nodes

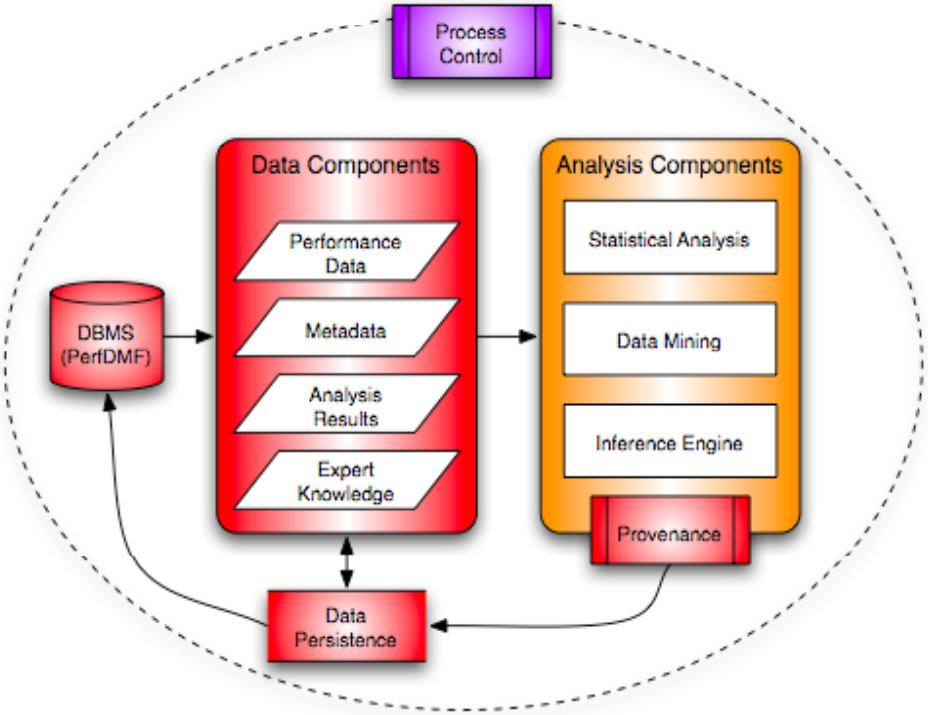
PerfExplorer v2 – Requirements and Features

- Component-based analysis process
  - Analysis operations implemented as modules
  - Linked together in analysis process and workflow
- Scripting
  - Provides process/workflow development and automation
- Metadata input, management, and access
- Inference engine
  - Reasoning about causes of performance phenomena
  - Analysis knowledge captured in expert rules
- Persistence of intermediate results
- Provenance
  - Provides historical record of analysis results

PerfExplorer v2: Architecture and Interaction



Interaction workflow

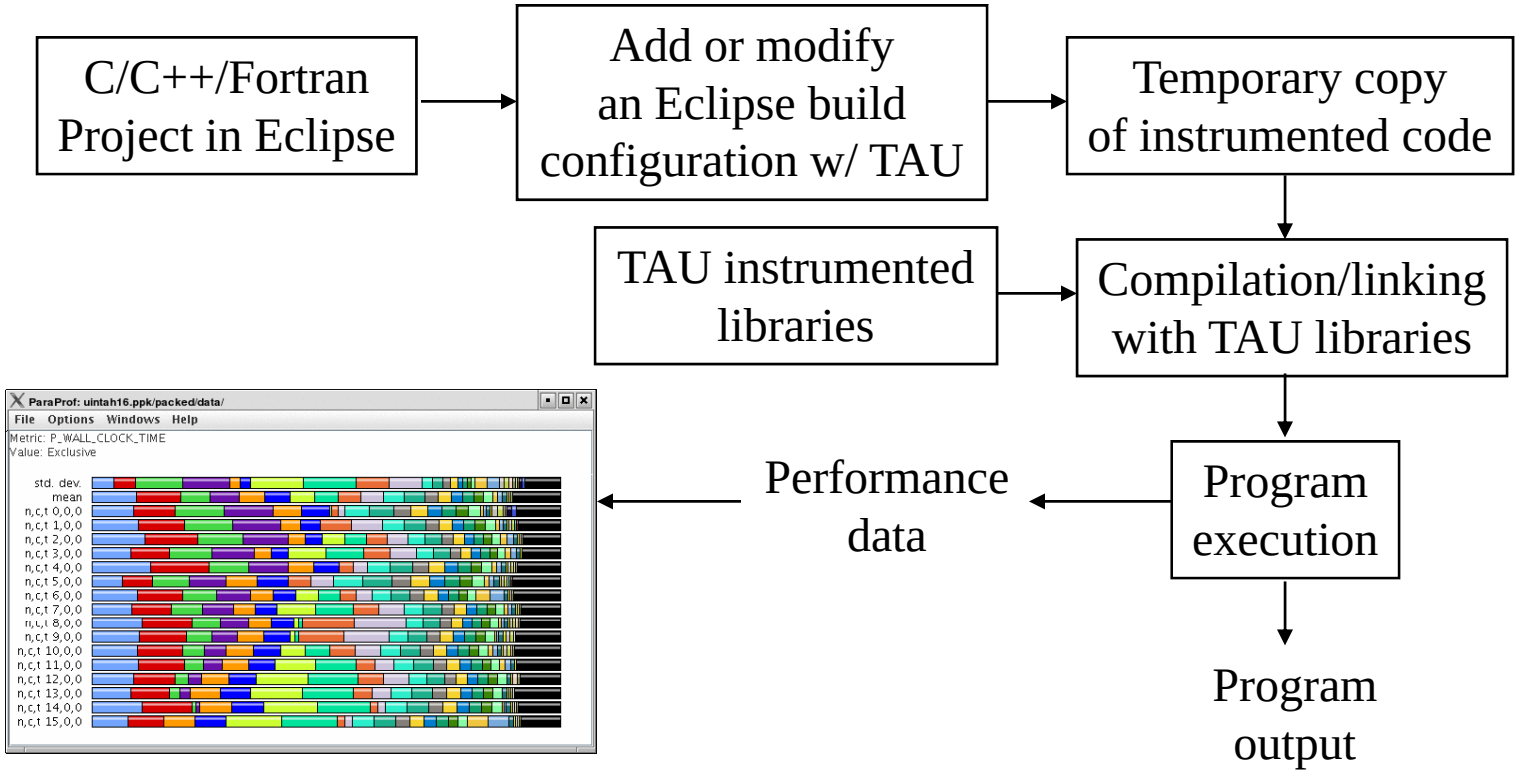


TAU Integration with IDEs

- 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

Fortran – matmult.f90 – Eclipse SDK

File Edit Refactor Navigate Search Project Run Window Help

Fortran Projects | Navigator

- matmultiply
  - Binaries
  - Includes
  - .settings
  - Debug_ia32
  - Debug_ia32(tau-icpc-mpi-pdt)
  - matmult.f90
  - .cdtbuild
  - .cdtproject
  - .project
  - profile.0.0.0
  - profile.1.0.0
  - profile.2.0.0
  - profile.3.0.0

matmult.f90

```
!*********************************************************************
!     matmult.f90 - simple matrix multiply implementation
!*********************************************************************
      subroutine initialize(a, b, n)
        double precision a(n,n)
        double precision b(n,n)
        integer n

! first initialize the A matrix
        do i = 1,n
          do j = 1,n
            a(j,i) = i
          end do
        end do

! then initialize the B matrix
        do i = 1,n
          do j = 1,n
            b(j,i) = i
          end do
        end do

      end subroutine initialize

      subroutine multiply_matrices(answer, buffer, b, matsize)
        double precision buffer(matsize), answer(matsize)
        double precision b(matsize, matsize)
        integer i, j
! multiply the row with the column
```

Outline | Make ...

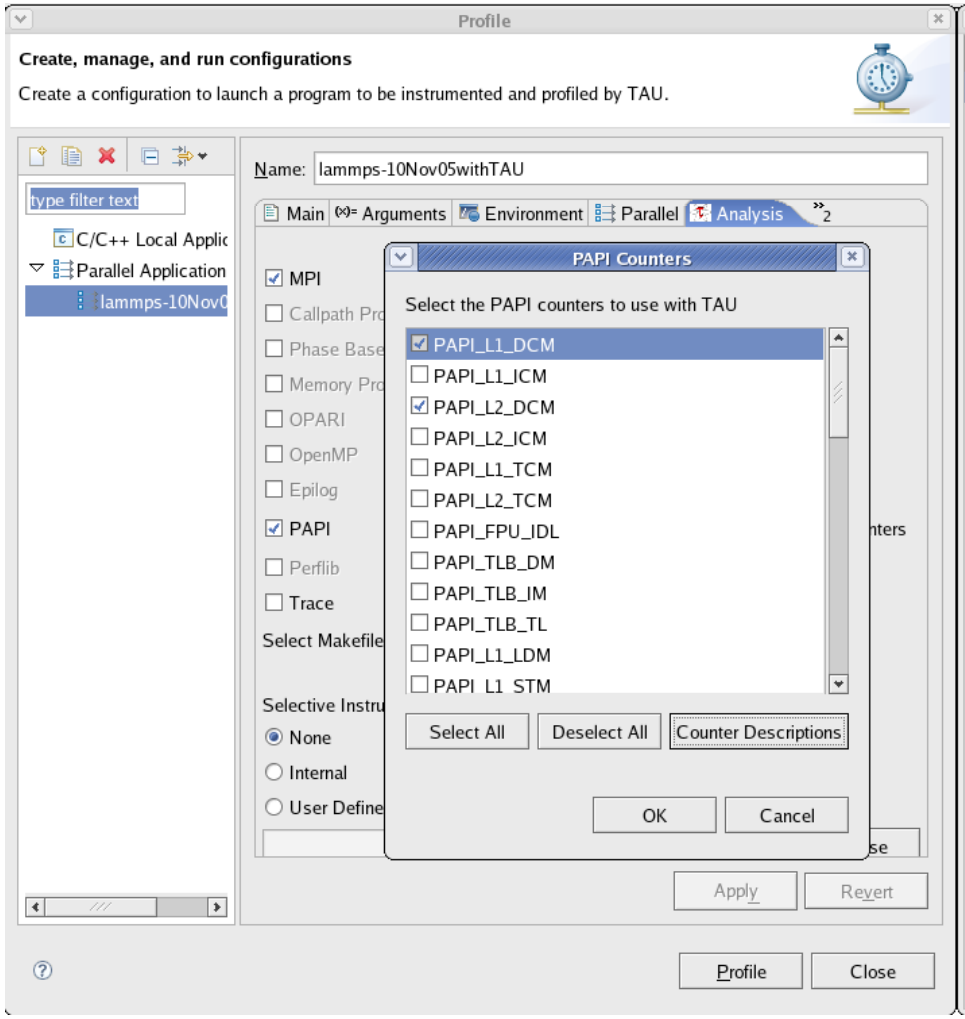
- initialize
- multiply_matrices
- main

Problems | Console | Properties | Performance Data Manager | Launch ParaProf

- AORSA2D
- matmultiply
  - Experiment
    - The New Trial: 2006-12-02 20:36:59
- mm
- ring

PerfDMF

Choosing PAPI Counters with TAU's 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

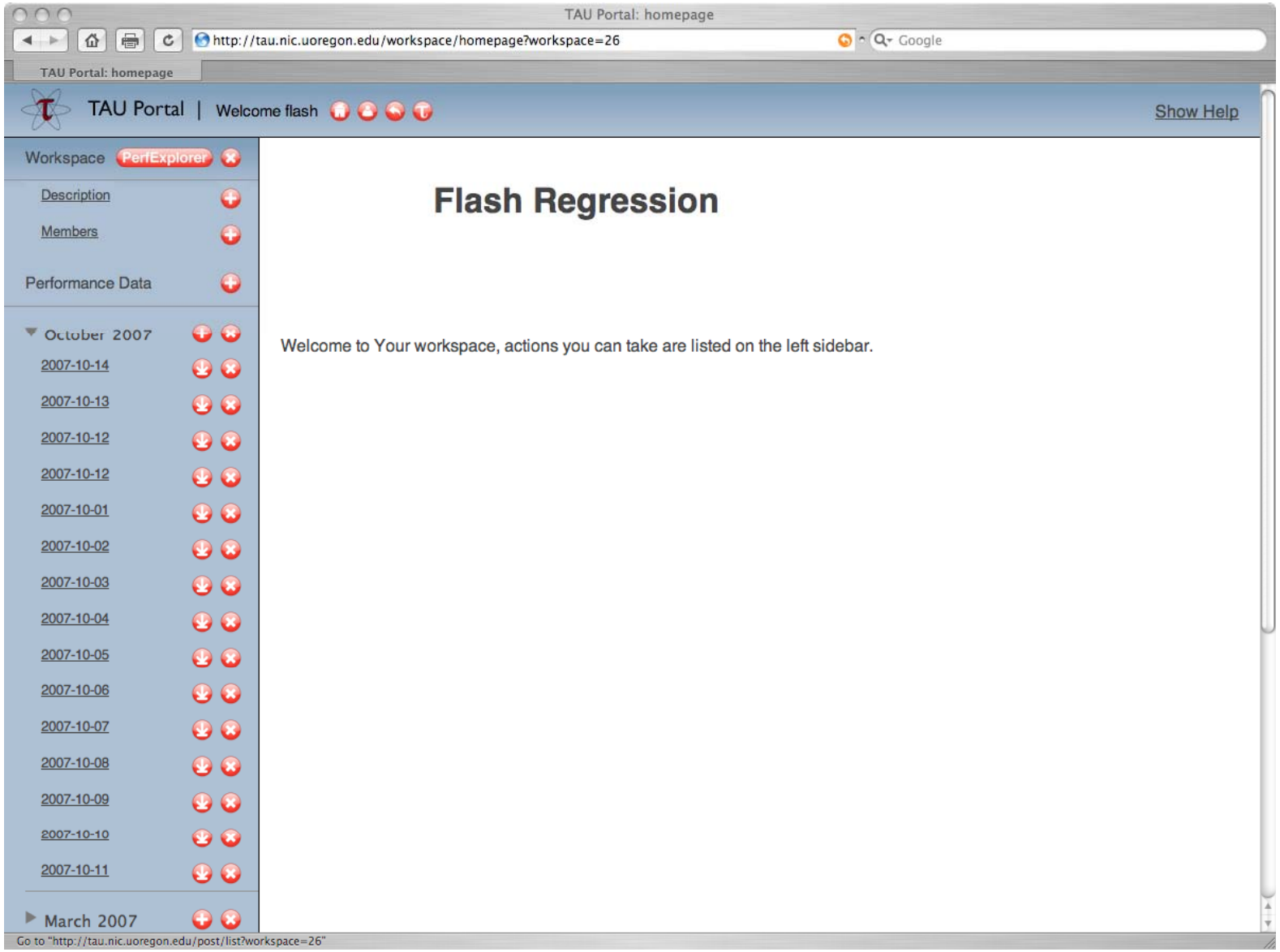
TAU Eclipse Plug-In Features

- Performance data collection
  - Graphical selection of TAU stub makefiles and compiler options
  - Automatic instrumentation, compilation and execution of target C, C++ or Fortran projects
  - Selective instrumentation via source editor and source outline views
  - Full integration with the Parallel Tools Platform (PTP) parallel launch system for performance data collection from parallel jobs launched within Eclipse
- Performance data management
  - Automatically place profile output in a PerfDMF database or upload to TAU-Portal
  - Launch ParaProf on profile data collected in Eclipse, with performance counters linked back to the Eclipse source editor

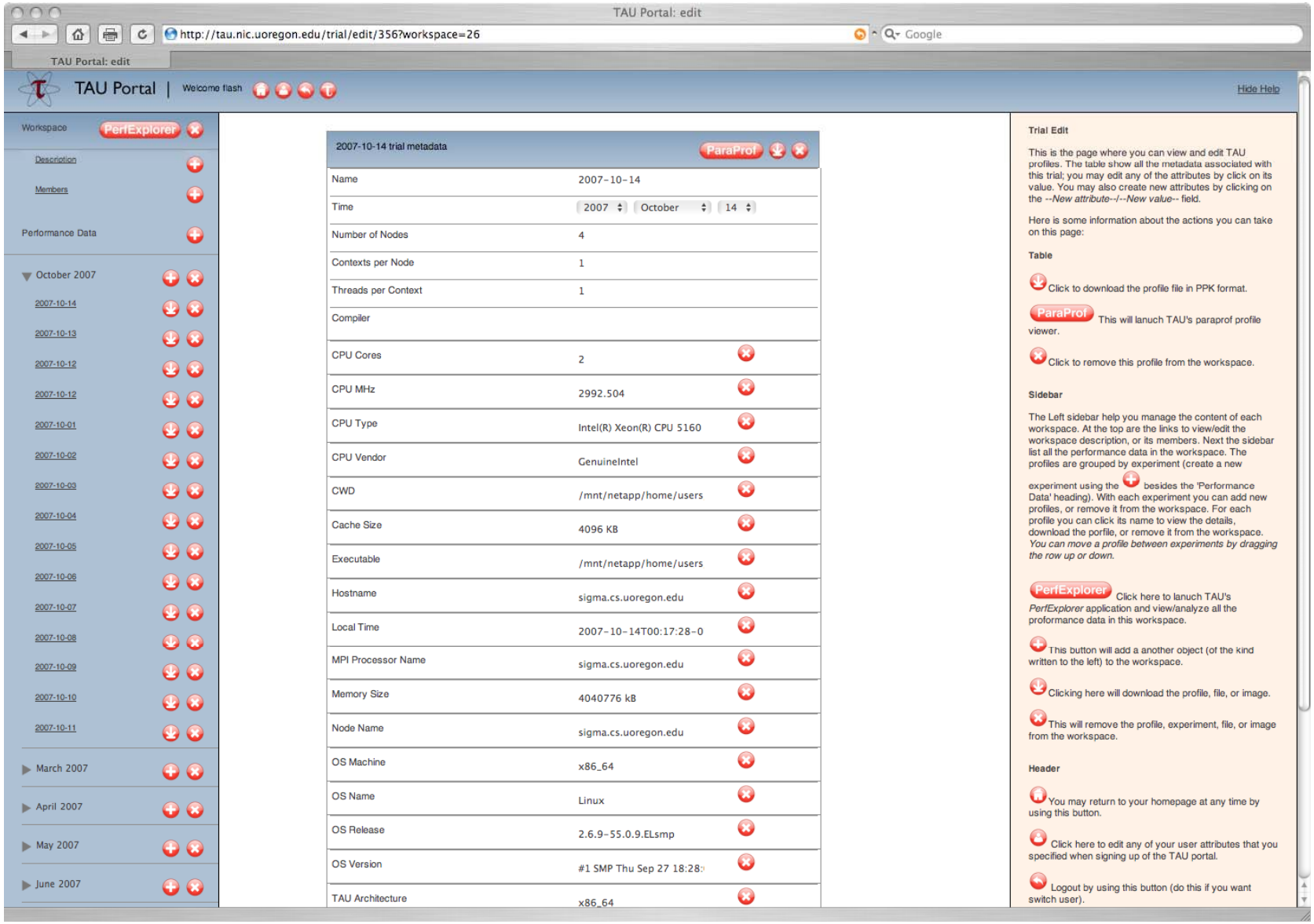
TAU Portal

- Web-based access to TAU
- Support collaborative performance study
  - Secure performance data sharing
  - Does not require TAU installation
  - Launch TAU performance tools with Java WebStart
    - ParaProf, PerfExplorer
- FLASH regression testing
  - Nightly regression testcases
  - Uploaded to the database automatically
  - Interactive review of performance through TAU portal
  - Multi-experiment analysis

Portal: Nightly Performance Regression Testing

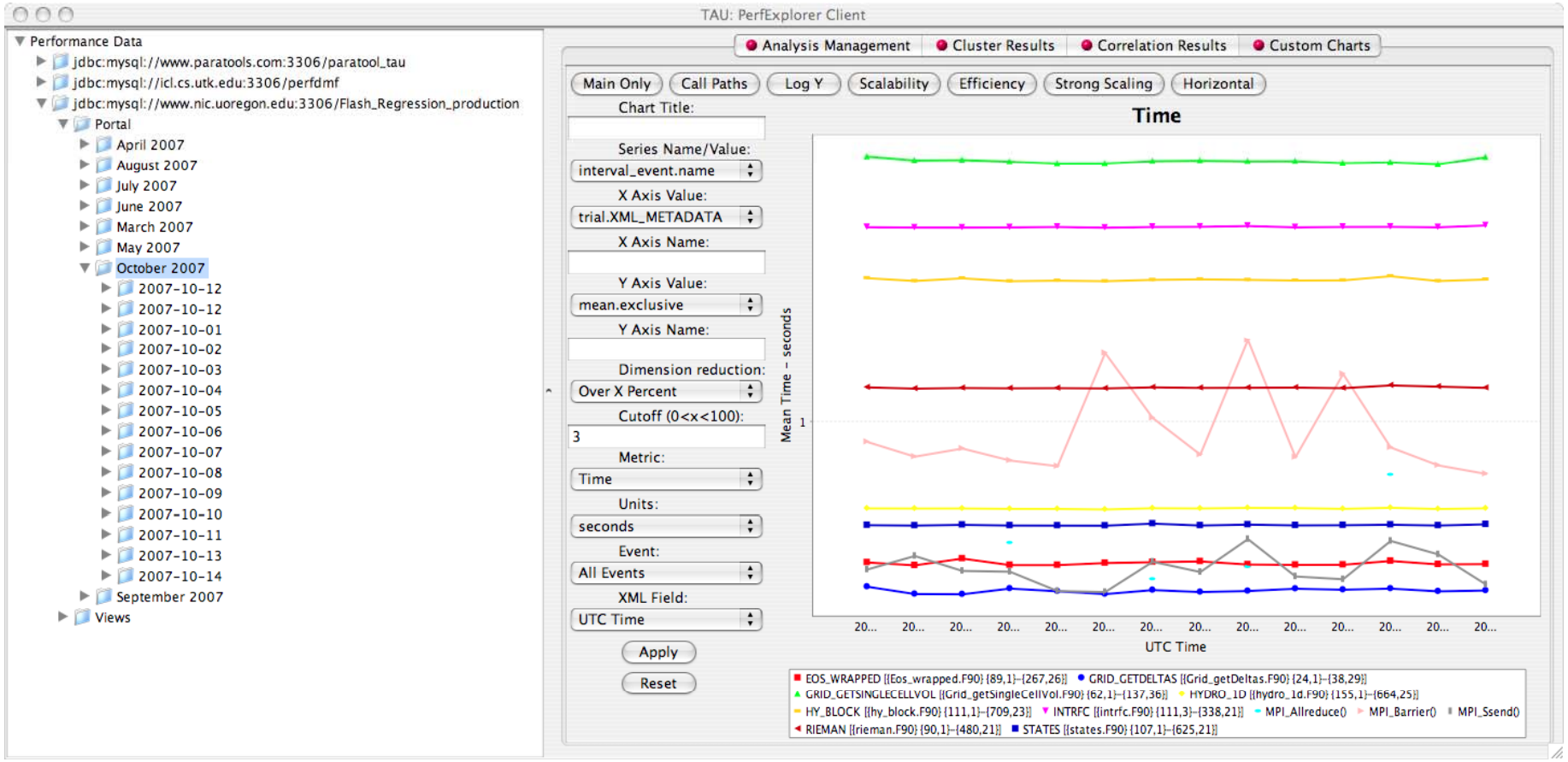


TAU Portal: Launch ParaProf/PerfExplorer

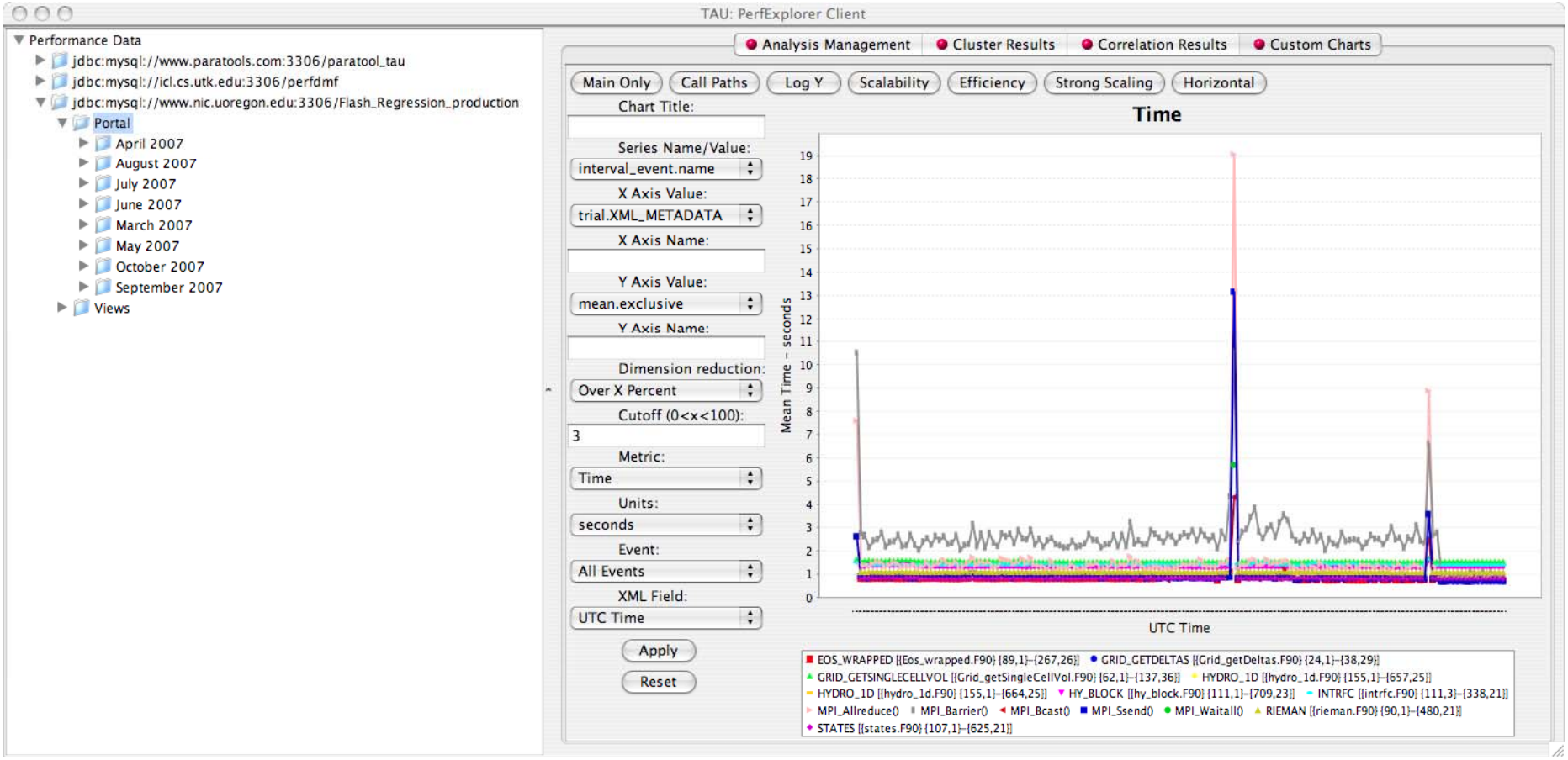


PerfExplorer: Regression Testing

PerfExplorer: Limiting Events (> 3%), Oct 2007



PerfExplorer: Exclusive Time for Events (2007)

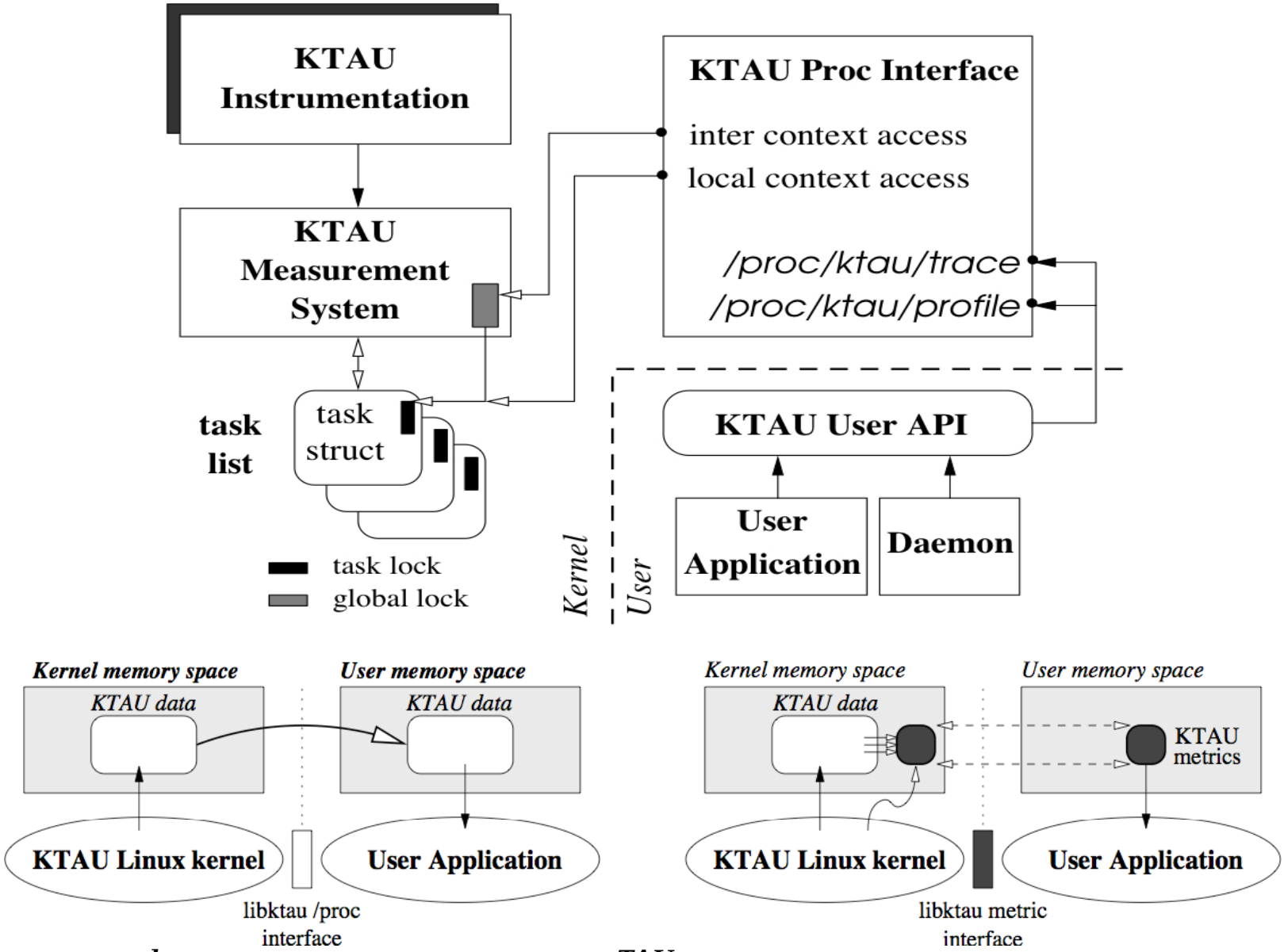


Full System Performance: the KTAU Project

- Trend toward extremely large scales
  - System-level influences are increasingly dominant performance bottleneck contributors
  - Application sensitivity at scale to the system
  - Complex I/O path and subsystems another example
  - Isolating system-level factors non-trivial
- OS Kernel instrumentation and measurement is important to understanding system-level influences
- How to correlate application and OS performance?
- KTAU / TAU (Part of the ANL/UO ZeptoOS Project)

A. Nataraj, A. Malony, S. Shende, and A. Morris, "**Kernel-level Measurement for Integrated Performance Views: the KTAU Project**," *Cluster 2006*.

KTAU System Architecture



Applying KTAU+TAU

- How does real OS-noise affect real applications?
  - Requires OS + application performance measurement
  - Estimate application slowdown due to noise components
    - interrupts and scheduling are significant

A. Nataraj, A. Morris, A. Malony, M. Sottile, and P. Beckman, "**The Ghost in the Machine: Observing the Effects of Kernel Operation on Parallel Application Performance**," *SC'07*. Wednesday, 10:30-12:00.

- Performance of multi-layered I/O systems
  - Requires measurement and analysis of multi-component I/O subsystems in system
  - Tracking of I/O "long path" and assignment to application
  - Working with Argonne on PVFS2

TAU Monitoring

- Runtime access to parallel performance data
- Monitoring modes
  - Offline / Post-mortem observation and analysis
    - least requirements for a specialized transport
  - Online observation
    - long running applications, especially at scale
    - Dumping snapshots to file-system can be suboptimal
  - Online observation with feedback into application
- TAUoverSupermon (Sottile and Minnich, LANL)

A. Nataraj, M. Sottile, A. Morris, A. Malony, and S. Shende, "**TAUoverSupermon: Low-overhead Online Parallel Performance Monitoring**," *Euro-Par 2007*.

- TAUoverMRNET (Arnold and Miller, UWisconsin)

Project Affiliations (selected)

- Lawrence Livermore National Lab
  - Hydrodynamics (*Miranda*), radiation diffusion (*KULL*)
  - Open Trace Format (*OTF*) implementation on BG/L
- Lawrence Berkeley National Lab
  - ACTS Toolkit
- Argonne National Lab
  - ZeptoOS project and KTAU
  - Astrophysical thermonuclear flashes (*Flash*)
- Center for Simulation of Accidental Fires and Explosion
  - University of Utah, ASCI ASAP Center, C-SAFE
  - Uintah Computational Framework (*UCF*)
- Oak Ridge National Lab
  - Contribution to the Joule Report/PERI for *S3D*, *GYRO*, *AORSA3D*
- NASA Goddard Space Flight Center, NASA Ames
  - GEOS/GCM, modelE, ...

Project Affiliations (continued)

- Sandia National Lab
  - Simulation of turbulent reactive flows (*S3D*)
  - Combustion code (*CFRFS*)
- Los Alamos National Lab
  - Monte Carlo transport (*MCNP*)
  - SAIC's Adaptive Grid Eulerian (*SAGE*, *RAGE*)
  - perflib integration (Jeff Brown)
- *CCSM* / *ESMF* / *WRF* climate/earth/weather simulation
  - NSF, NOAA, DOE, NASA, …
- Common component architecture (*CCA*) integration
- Performance Engineering Research Institute (*PERI*)

Concluding Discussion

- Performance tools must be used effectively
- More intelligent performance systems for productive use
  - Evolve to application-specific performance technology
  - Deal with scale by “full range” performance exploration
  - Autonomic and integrated tools
  - Knowledge-based and knowledge-driven process
- Performance observation methods do not necessarily need to change in a fundamental sense
  - More automatically controlled and efficiently use
- Develop next-generation tools and deliver to community
- Open source with support by ParaTools, Inc.
- http://tau.uoregon.edu

Support Acknowledgements

- Department of Energy (DOE)
  - Office of Science
    - MICS, Argonne National Lab
  - ASC/NNSA
    - University of Utah ASC/NNSA Level 1
    - ASC/NNSA, LLNL
- Department of Defense (DoD)
  - HPC Modernization Office (HPCMO)
- NSF SDCI
- Research Centre Juelich
- LBL, ORNL, ANL, LANL, LLNL
- TU Dresden
- ParaTools, Inc.

Even more details ...

TAU Measurement System Configuration

- configure [OPTIONS]

{-c++=<CC>, -cc=<cc>}	Specify C++ and C compilers
-pdt=<dir>	Specify location of PDT
-opari=<dir>	Specify location of Opari OpenMP tool
-papi=<dir>	Specify location of PAPI
-vampirtrace=<dir>	Specify location of VampirTrace
-mpi[inc/lib]=<dir>	Specify MPI library instrumentation
-dyninst=<dir>	Specify location of DynInst Package
-shmem[inc/lib]=<dir>	Specify PSHMEM library instrumentation
-python[inc/lib]=<dir>	Specify Python instrumentation
-tag=<name>	Specify a unique configuration name
-epilog=<dir>	Specify location of EPILOG
-slog2	Build SLOG2/Jumpshot tracing package
-otf=<dir>	Specify location of OTF trace package
-arch=<architecture>	Specify architecture explicitly (bgl, bgp, craycnl, xt3,ibm64, …)
{-pthread, -sproc}	Use pthread or SGI sproc threads
-openmp	Use OpenMP threads
-jdk=<dir>	Specify Java instrumentation (JDK)
-fortran=[vendor]	Specify Fortran compiler

TAU Measurement System Configuration

- configure [OPTIONS]

-TRACE	Generate binary TAU traces
-PROFILE (default)	Generate profiles (summary)
-PROFILECALLPATH	Generate call path profiles
-PROFILEPHASE	Generate phase based profiles
-PROFILEPARAM	Generate parameter based profiles
-PROFILEMEMORY	Track heap memory for each routine
-PROFILEHEADROOM	Track memory headroom to grow
-MULTIPLECOUNTERS	Use hardware counters + time
-COMPENSATE	Compensate timer overhead
-CPUTIME	Use usertime+system time
-PAPIWALLCLOCK	Use PAPI's wallclock time
-PAPIVIRTUAL	Use PAPI's process virtual time
-SGITIMERS	Use fast IRIX timers
-LINUXTIMERS	Use fast x86 Linux timers

TAU Configuration – Examples

- ./configure -pdt=<dir> -c++=pgCC -cc=pgcc -fortran=pgi –mpi
  - Under Linux with PDT, MPI for PGI compilers
- ./configure –papi=<dir> -MULTIPLECOUNTERS –pdt=<dir> -mpi –arch=x86_64; make clean install
  - Use PAPI counters (one or more) with C/C++/F90 automatic instrumentation for AIX. Also instrument the MPI library.
- Typically configure multiple measurement libraries
  - .all_configs, .last_config files contain all and last configuration
  - tau_validate --html --build x86_64 >& results.html
  - ./upgradetau /path/to/old/tau-2.17
- Each configuration creates a unique <arch>/lib/Makefile.tau<options> stub makefile. It corresponds to the configuration options used. e.g.,
  - /usr/common/acts/TAU/tau-2.17.2/x86_64/lib/Makefile.tau-mpi-pdt
  - /usr/common/acts/TAU/tau-2.17.2/x86_64/lib/Makefile.tau-multiplecounters-mpi-papi-pdt

TAU Measurement Configuration – Examples

% cd /usr/common/acts/TAU/tau-2.17.2/x86_64/lib; ls Makefile.*
Makefile.tau-pdt-pgi
Makefile.tau-mpi-pdt-pgi
Makefile.tau-callpath-mpi-pdt-pgi
Makefile.tau-mpi-pdt-trace-pgi
Makefile.tau-mpi-compensate-pdt-pgi
Makefile.tau-multiplecounters-mpi-papi-pdt-pgi
Makefile.tau-multiplecounters-mpi-papi-pdt-trace-pgi
Makefile.tau-mpi-papi-pdt-epilog-trace-pgi…

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

Makefile.tau-mpi-pdt-pgi

  - Supports MPI instrumentation & PDT for automatic source instrumentation for PGI compilers

Configuration Parameters in Stub Makefiles

- Each TAU stub Makefile resides in <tau>/<arch>/lib directory
- Variables:
  - TAU_CXX — Specify the C++ compiler used by TAU
  - TAU_CC, TAU_F90 — Specify the C, F90 compilers
  - TAU_DEFS — Defines used by TAU. Add to CFLAGS
  - TAU_LDFLAGS — Linker options. Add to LDFLAGS
  - TAU_INCLUDE — Header files include path. Add to CFLAGS
  - TAU_LIBS — Statically linked TAU library. Add to LIBS
  - TAU_SHLIBS — Dynamically linked TAU library
  - TAU_MPI_LIBS — TAU's MPI wrapper library for C/C++
  - TAU_MPI_FLIBS — TAU's MPI wrapper library for F90
  - TAU_FORTRANLIBS — Must be linked in with C++ linker for F90
  - TAU_CXXLIBS — Must be linked in with F90 linker
  - TAU_INCLUDE_MEMORY — Use TAU's malloc/free wrapper lib
  - TAU_DISABLE — TAU's dummy F90 stub library
  - TAU_COMPILER — Instrument using tau_compiler.sh script
- Each stub makefile encapsulates the parameters that TAU was configured with
- It represents a specific instance of the TAU libraries. TAU scripts use stub makefiles to identify what performance measurements are to be performed.

Automatic Instrumentation

- We now provide compiler wrapper scripts
  - Simply replace `mpxlf90_r` with `tau_f90.sh`
  - Automatically instruments Fortran source code, links with TAU MPI Wrapper libraries.
- Use `tau_cc.sh` and `tau_cxx.sh` for C/C++

Before

```
CXX = mpCC
F90 = mpxlf90_r
CFLAGS =
LIBS = -lm
OBJS = f1.o f2.o f3.o … fn.o

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

After

```
CXX = tau_cxx.sh
F90 = tau_f90.sh
CFLAGS =
LIBS = -lm
OBJS = f1.o f2.o f3.o … fn.o

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

TAU_COMPILER Commandline Options

- See `<taudir>/<arch>/bin/tau_compiler.sh –help`
- Compilation:

```
% cc -c foo.f90
```

Changes to

```
% f95parse foo.f90 $(OPT1)
% tau_instrumentor foo.pdb foo.f90 –o foo.inst.f90 $(OPT2)
% xlf90_r –c foo.f90 $(OPT3)
```

- Linking:

```
% ftn foo.o bar.o –o app
```

Changes to

```
% qk-pgf90 foo.o bar.o –o app $(OPT4)
```

- Where options OPT[1-4] default values may be overridden by the user:

```
F90 = $(TAU_COMPILER) $(MYOPTIONS) ftn
```

Overriding Default Options:TAU_COMPILER

```
% cat Makefile
F90 = tau_f90.sh
OBJS = f1.o f2.o f3.o …
LIBS = -Lappdir –lapplib1 –lapplib2 …

app: $(OBJS)
    $(F90) $(OBJS) –o app $(LIBS)
.f90.o:
    $(F90) –c $<
% setenv TAU_OPTIONS '-optVerbose -optTauSelectFile=select.tau
      -optKeepFiles'
% setenv TAU_MAKEFILE <taudir>/x86_64/lib/Makefile.tau-mpi-pdt
```

Optimization of Program Instrumentation

- Need to eliminate instrumentation in frequently executing lightweight routines
- Throttling of events at runtime:

```
% setenv TAU_THROTTLE 1
```

  Turns off instrumentation in routines that execute over 100000 times (TAU_THROTTLE_NUMCALLS) and take less than 10 microseconds of inclusive time per call (TAU_THROTTLE_PERCALL)

- Selective instrumentation file to filter events

```
% tau_instrumentor [options] –f <file>  OR
% setenv TAU_OPTIONS '-optTauSelectFile=tau.txt'
```

- Compensation of local instrumentation overhead

```
% configure -COMPENSATE
```

Selective Instrumentation File

- Specify a list of routines to exclude or include (case sensitive)
- # is a wildcard in a routine name. It cannot appear in the first column.

```
BEGIN_EXCLUDE_LIST
Foo
Bar
D#EMM
END_EXCLUDE_LIST
```

- Specify a list of routines to include for instrumentation

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

- Specify either an include list or an exclude list!

Selective Instrumentation File

- Optionally specify a list of files to exclude or include (case sensitive)
- * and ? may be used as wildcard characters in a file name

```
BEGIN_FILE_EXCLUDE_LIST
f*.f90
Foo?.cpp
END_FILE_EXCLUDE_LIST
```

- Specify a list of routines to include for instrumentation

```
BEGIN_FILE_INCLUDE_LIST
main.cpp
foo.f90
END_FILE_INCLUDE_LIST
```

Selective Instrumentation File

- User instrumentation commands are placed in INSTRUMENT section
- ? and * used as wildcard characters for file name, # for routine name
- \ as escape character for quotes
- Routine entry/exit, arbitrary code insertion
- Outer-loop level instrumentation, static/dynamic phases, I/O, memory instrumentation

```
BEGIN_INSTRUMENT_SECTION
loops file="foo.f90" routine="matrix#"
memory file="foo.f90" routine="#"
io routine="MATRIX"
file="foo.f90" line = 123 code = "  print *, \" In foo\""
exit routine = "int f1()" code = "cout <<\"Out f1\"<<endl;"
dynamic timer name="foo" file="foo.f90" line=12 to line=22
static phase routine="bar"
END_INSTRUMENT_SECTION
```

Using TAU

- Install TAU

  % ./configure [options]; make clean install
- Replace the names of your compiler with tau_f90.sh, tau_cxx.sh and tau_cc.sh in your makefiles
- Set environment variables
  - Choose the measurement option and compile your code:
    - setenv TAU_MAKEFILE $TAU/Makefile.tau-icpc-mpi-pdt
    - setenv TAU_OPTIONS '-optVerbose -optKeepFiles -optPreProcess'
    - setenv TAU_THROTTLE 1
      - At runtime to keep instrumentation overhead in check
  - At runtime, if more than one metric is measured (-multiplecounters):
    - setenv COUNTER1 GET_TIME_OF_DAY
    - setenv COUNTER2 PAPI_FP_INS
    - setenv COUNTER3 PAPI_NATIVE_<native_name>
      - Use papi_native_avail, papi_avail, and papi_event_chooser to select these preset and native event names
- Build the application, run it, analyze performance data

Compiling Fortran Codes with TAU: Tips

- 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 '

- If it uses several module files, you may switch from the default Cleanscape Inc. parser in PDT to the GNU gfortran parser to generate PDB files:

  % setenv TAU_OPTIONS '-optPdtGnuFortranParser -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

Workload Characterization

- Simple example, send/receive squared message sizes (0-32MB)

```
#include <stdio.h>
#include <mpi.h>
int buffer[8*1024*1024];

int main(int argc, char **argv) {
  int rank, size, i, j;
  MPI_Init(&argc, &argv);
  MPI_Comm_size( MPI_COMM_WORLD, &size );
  MPI_Comm_rank( MPI_COMM_WORLD, &rank );
  for (i=0;i<1000;i++)
    for (j=1;j<=8*1024*1024;j*=2) {
      if (rank == 0) {
        MPI_Send(buffer,j,MPI_INT,1,42,MPI_COMM_WORLD);
      } else {
        MPI_Status status;
        MPI_Recv(buffer,j,MPI_INT,0,42,MPI_COMM_WORLD,&status);
      }
    }
  MPI_Finalize();
}
```

Instrumentation of OpenMP Constructs

- **O**penMP **P**ragma **A**nd **R**egion **I**nstrumentor [UTK, FZJ]
- Source-to-Source translator to insert POMP calls around OpenMP constructs and API functions
- Done: Supports
  - Fortran77 and Fortran90, OpenMP 2.0
  - C and C++, OpenMP 1.0
  - POMP Extensions
  - EPILOG and TAU POMP implementations
  - Preserves source code information (`#line line file`)
- tau_ompcheck
  - Balances OpenMP constructs (DO/END DO) and detects errors
  - Invoked by tau_compiler.sh prior to invoking Opari
- KOJAK Project website http://icl.cs.utk.edu/kojak

OpenMP API Instrumentation

- Transform
  - **`omp_#_lock()`** → **`pomp_#_lock()`**
  - **`omp_#_nest_lock()`** → **`pomp_#_nest_lock()`**

  `[# = init|destroy|set|unset|test]`

- POMP version
  - Calls omp version internally
  - Can do extra stuff before and after call

Example: !$OMP PARALLEL DO Instrumentation

```
call pomp_parallel_fork(d)
!$OMP PARALLEL other-clauses...
      call pomp_parallel_begin(d)
      call pomp_do_enter(d)
      !$OMP DO schedule-clauses, ordered-clauses,
               lastprivate-clauses
            do loop
      !$OMP END DO NOWAIT
      call pomp_barrier_enter(d)
      !$OMP BARRIER
      call pomp_barrier_exit(d)
      call pomp_do_exit(d)
      call pomp_parallel_end(d)
!$OMP END PARALLEL DO
call pomp_parallel_join(d)
```

Opari Instrumentation: Example

- OpenMP directive instrumentation

```
pomp_for_enter(&omp_rd_2);
#line 252 "stommel.c"
#pragma omp for schedule(static) reduction(+: diff) private(j)
  firstprivate (a1,a2,a3,a4,a5) nowait
for( i=i1;i<=i2;i++) {
  for(j=j1;j<=j2;j++){
    new_psi[i][j]=a1*psi[i+1][j] + a2*psi[i-1][j] + a3*psi[i][j+1]
      + a4*psi[i][j-1] - a5*the_for[i][j];
    diff=diff+fabs(new psi[i][j]-psi[i][j]);
  }
}
pomp_barrier_enter(&omp_rd_2);
#pragma omp barrier
pomp_barrier_exit(&omp_rd_2);
pomp_for_exit(&omp_rd_2);
```

Using Opari with TAU

Step I: Configure KOJAK/opari [Download from http://www.fz-juelich.de/zam/kojak/**]**

```
% cd kojak-2.1.1; cp mf/Makefile.defs.ibm Makefile.defs;
  edit Makefile
% make
```

Builds opari

Step II: Configure TAU with Opari (used here with MPI and PDT)

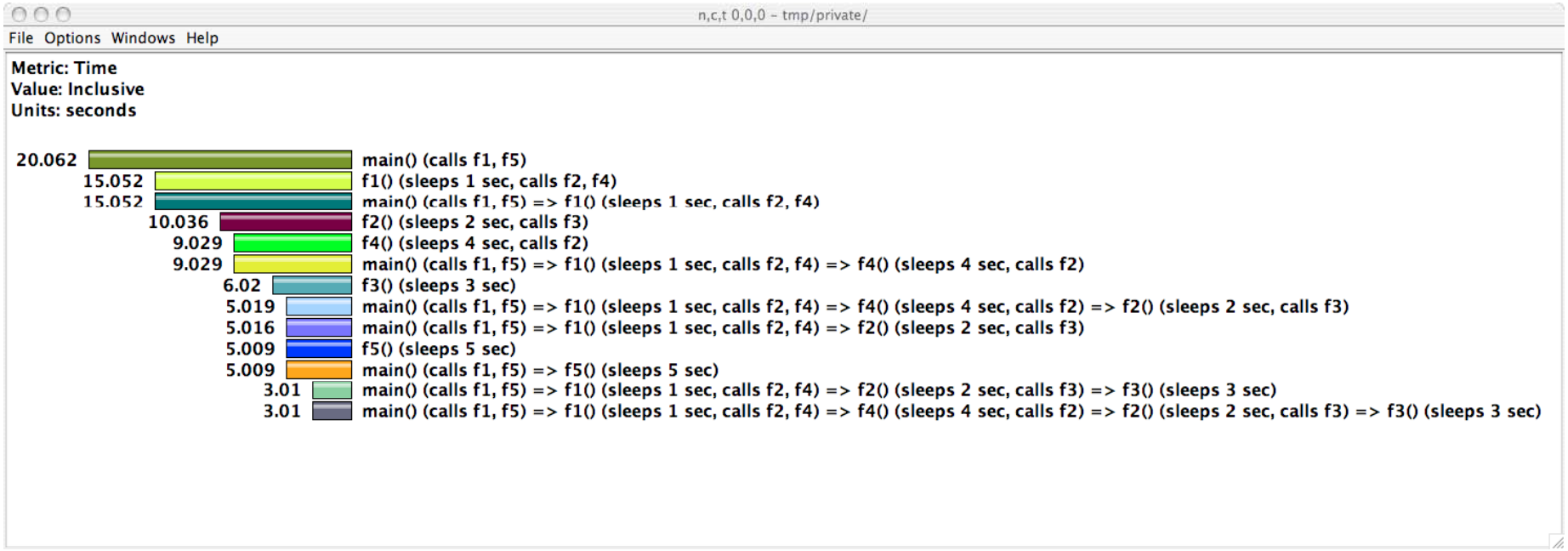
```
% configure –opari=/usr/contrib/TAU/kojak-2.1.1/opari
  -mpiinc=/usr/lpp/ppe.poe/include
  –mpilib=/usr/lpp/ppe.poe/lib
  –pdt=/usr/contrib/TAU/pdtoolkit-3.9
% make clean; make install
% setenv TAU_MAKEFILE /tau/<arch>/lib/Makefile.tau-…opari-…
% tau_cxx.sh -c foo.cpp
% tau_cxx.sh -c bar.f90
% tau_cxx.sh *.o -o app
```

-MULTIPLECOUNTERS Configuration Option

- Instead of one metric, profile or trace with more than one metric
  - Set environment variables COUNTER[1-25] to specify the metric
    - % setenv COUNTER1 GET_TIME_OF_DAY
    - % setenv COUNTER2 PAPI_L2_DCM
    - % setenv COUNTER3 PAPI_FP_OPS
    - % setenv COUNTER4 PAPI_NATIVE_<native_event>
    - % setenv COUNTER5 P_WALL_CLOCK_TIME …
- When used with –TRACE option, the first counter **must** be GET_TIME_OF_DAY
    - % setenv COUNTER1 GET_TIME_OF_DAY
    - Provides a globally synchronized real time clock for tracing
- -multiplecounters appears in the name of the stub Makefile
- Often used with –papi=<dir> to measure hardware performance counters and time
- papi_native_avail and papi_avail and papi_event_chooser are useful tools

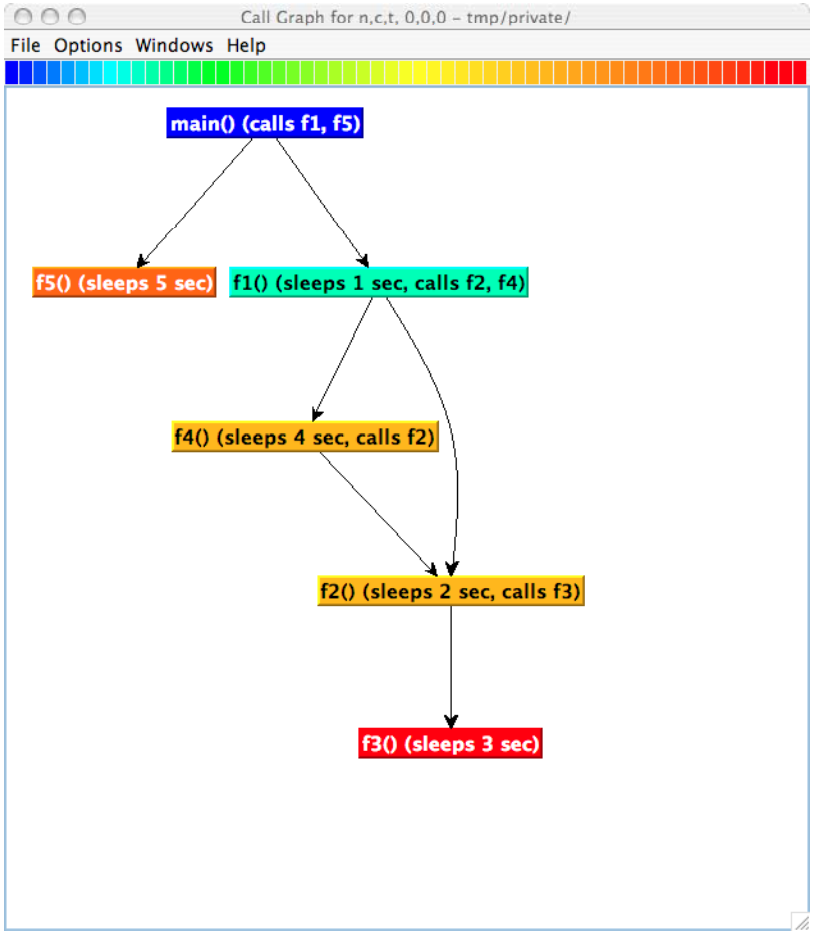
-PROFILECALLPATH Configuration Option

- Generates profiles that show the calling order (edges & nodes in callgraph)
  - A=>B=>C shows the time spent in C when it was called by B and B was called by A
  - Control the depth of callpath using TAU_CALLPATH_DEPTH env. Variable
  - -callpath in the name of the stub Makefile name



-PROFILECALLPATH Configuration Option

- Generates program callgraph



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

-DEPTHLIMIT Configuration Option

- Allows users to enable instrumentation at runtime based on the depth of a calling routine on a callstack.
  - Disables instrumentation in all routines a certain depth away from the root in a callgraph
- TAU_DEPTH_LIMIT environment variable specifies depth

  % setenv TAU_DEPTH_LIMIT 1

  enables instrumentation in only “main”

  % setenv TAU_DEPTH_LIMIT 2

  enables instrumentation in main and routines that are directly called by main

- Stub makefile has -depthlimit in its name:

  setenv TAU_MAKEFILE <taudir>/<arch>/lib/Makefile.tau-icpc-mpi-depthlimit-pdt

-COMPENSATE Configuration Option

- Specifies online compensation of performance perturbation
- TAU computes its timer overhead and subtracts it from the profiles
- Works well with time or instructions based metrics
- Does not work with level 1/2 data cache misses

-TRACE Configuration Option

- Generates event-trace logs, rather than summary profiles
- Traces show when and where an event occurred in terms of location and the process that executed it
- Traces from multiple processes are merged:

  % tau_treemerge.pl

  - generates tau.trc and tau.edf as merged trace and event definition file
- TAU traces can be converted to Vampir's OTF/VTF3, Jumpshot SLOG2, Paraver trace formats:

  % tau2otf tau.trc tau.edf app.otf

  % tau2vtf tau.trc tau.edf app.vpt.gz

  % tau2slog2 tau.trc tau.edf -o app.slog2

  % tau_convert -paraver tau.trc tau.edf app.prv
- Stub Makefile has -trace in its name

  % setenv TAU_MAKEFILE <taudir>/<arch>/lib/
  Makefile.tau-icpc-mpi-pdt-trace

-PROFILEPARAM Configuration Option

- Idea: partition performance data for individual functions based on runtime parameters
- Enable by configuring with –PROFILEPARAM
- TAU call: TAU_PROFILE_PARAM1L (value, “name”)
- Simple example:

```
void foo(long input) {
  TAU_PROFILE("foo", "", TAU_DEFAULT);
  TAU_PROFILE_PARAM1L(input, "input");
  ... }
```

Workload Characterization

- 5 seconds spent in function "`foo`" becomes
  - 2 seconds for "`foo [ <input> = <25> ]`"
  - 1 seconds for "`foo [ <input> = <5> ]`"
  - …
- Currently used in MPI wrapper library
  - Allows for partitioning of time spent in MPI routines based on parameters (message size, message tag, destination node)
  - Can be extrapolated to infer specifics about the MPI subsystem and system as a whole

Workload Characterization

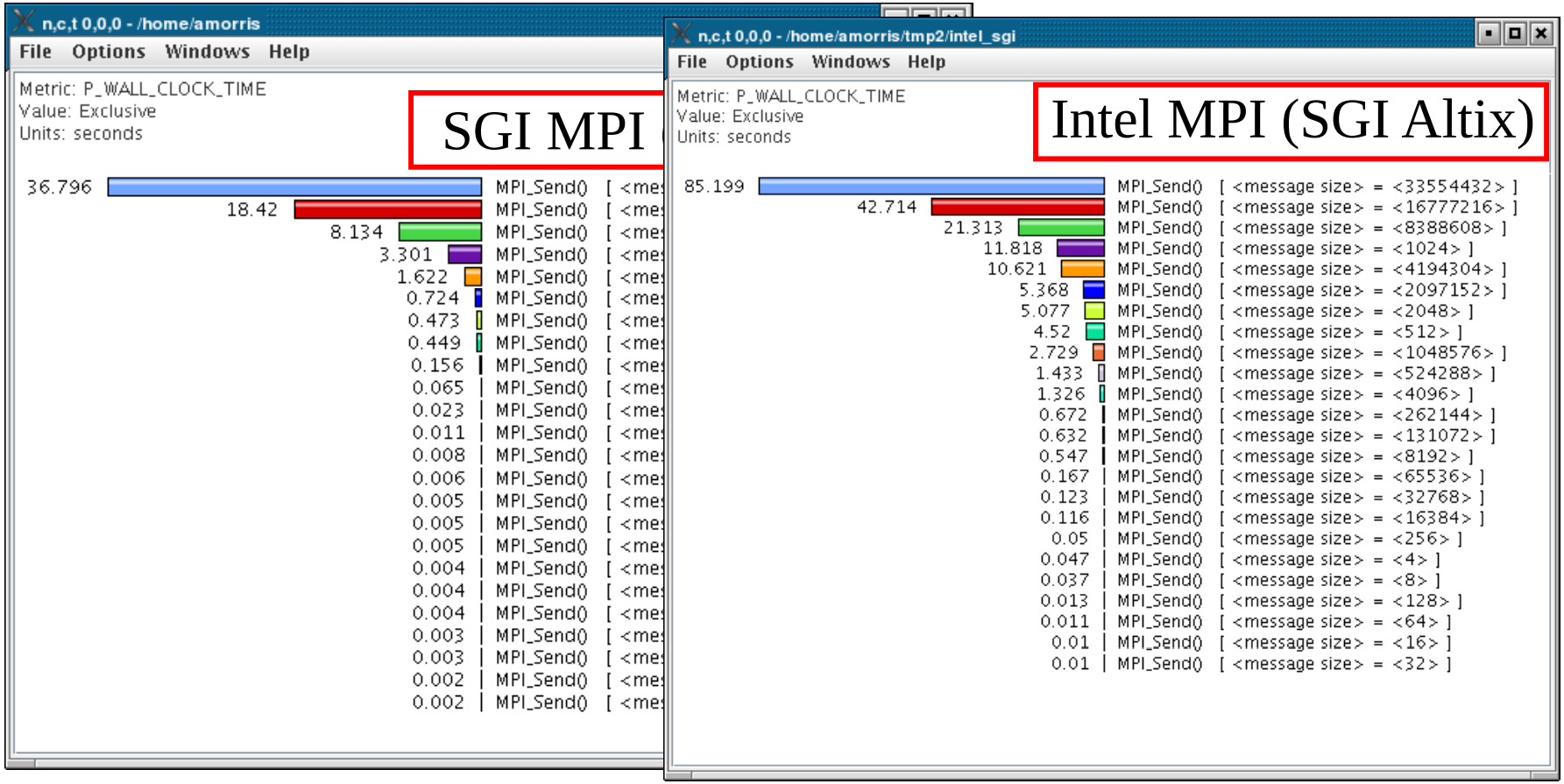
- Simple example, send/receive squared message sizes (0-32MB)

```
#include <stdio.h>
#include <mpi.h>
int buffer[8*1024*1024];

int main(int argc, char **argv) {
  int rank, size, i, j;
  MPI_Init(&argc, &argv);
  MPI_Comm_size( MPI_COMM_WORLD, &size );
  MPI_Comm_rank( MPI_COMM_WORLD, &rank );
  for (i=0;i<1000;i++)
    for (j=1;j<=8*1024*1024;j*=2) {
      if (rank == 0) {
        MPI_Send(buffer,j,MPI_INT,1,42,MPI_COMM_WORLD);
      } else {
        MPI_Status status;
        MPI_Recv(buffer,j,MPI_INT,0,42,MPI_COMM_WORLD,&status);
      }
    }
  MPI_Finalize();
}
```

Workload Characterization

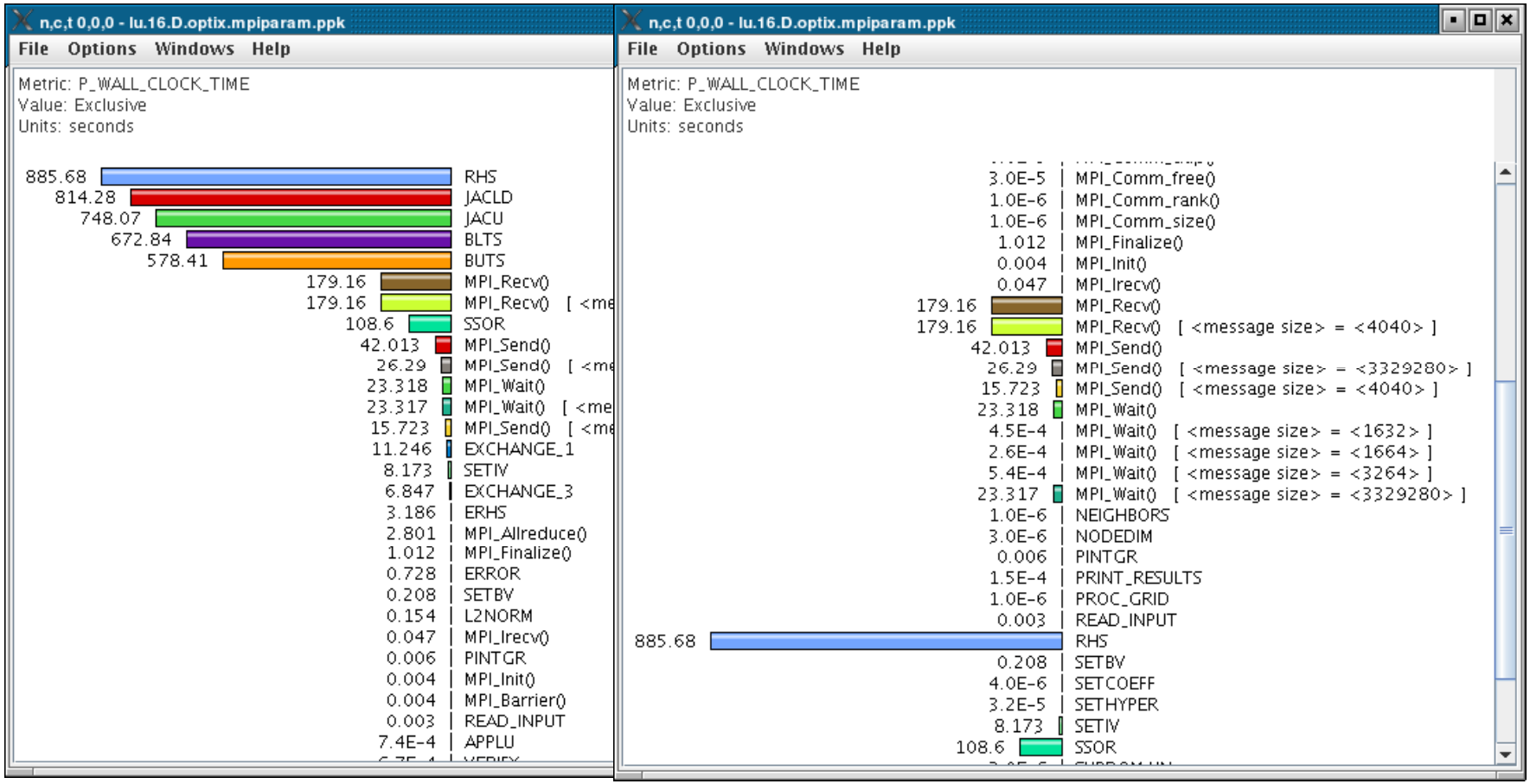
- Use `tau_load.sh` to instrument MPI routines (SGI Altix)

```
% icc mpi.c –lmpi
% mpirun –np 2 tau_load.sh –XrunTAU-icpc-mpi-pdt.so a.out
```



Workload Characterization

- MPI Results (NAS Parallel Benchmark 3.1, LU class D on 16 processors of SGI Altix)



Workload Characterization

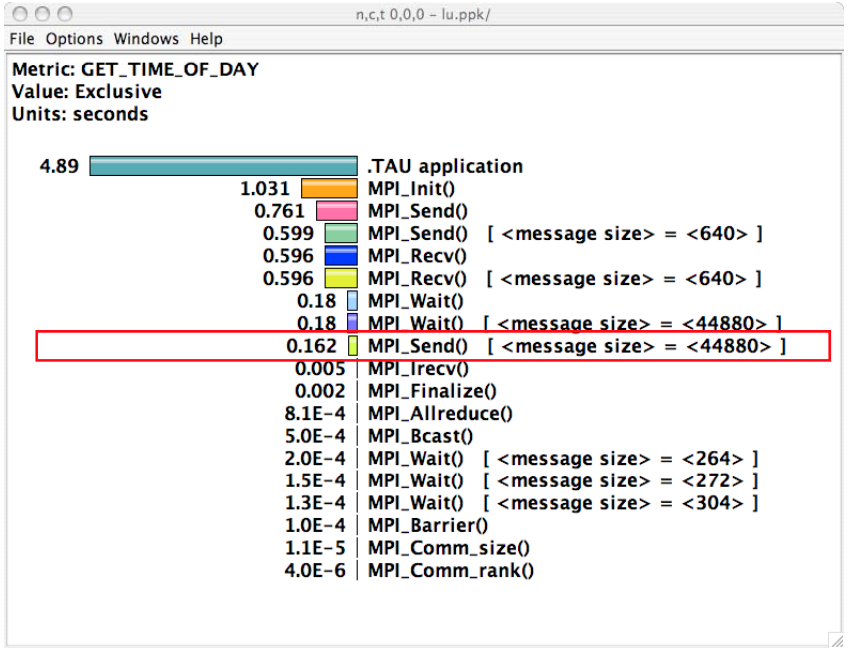
- Two different message sizes (~3.3MB and ~4K)

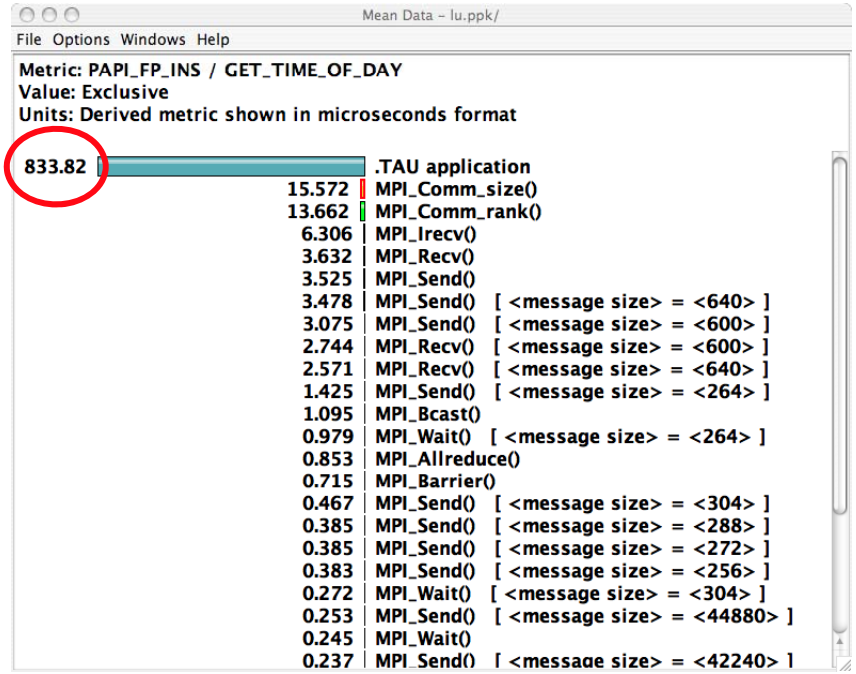
Thread Statistics: n,c,t, 0,0,0 - lu.16.D.optix.mpiparam.ppk

File Options Windows Help

Name	Inclusive ...	Exclusive...	Calls	Child ...
MPI_Comm_free()	0	0	1	0
MPI_Comm_rank()	0	0	1	0
MPI_Comm_size()	0	0	2	0
MPI_Finalize()	1.012	1.012	1	0
MPI_Init()	0.004	0.004	1	0
MPI_Irecv()	0.047	0.047	612	0
MPI_Recv()	179.165	179.165	244,412	0
MPI_Recv() [<message size> = <4040>]	179.165	179.165	244,412	0
MPI_Send()	42.013	42.013	245,020	0
MPI_Send() [<message size> = <3329280>]	26.29	26.29	608	0
MPI_Send() [<message size> = <4040>]	15.723	15.723	244,412	0
MPI_Wait()	23.318	23.318	612	0
MPI_Wait() [<message size> = <1632>]	0	0	1	0
MPI_Wait() [<message size> = <1664>]	0	0	1	0
MPI_Wait() [<message size> = <3264>]	0.001	0.001	2	0
MPI_Wait() [<message size> = <3329280>]	23.317	23.317	608	0
NEIGHBORS	0	0	1	0
NODEDIM	0	0	1	0
PINTGR	0.008	0.006	1	6
PRINT_RESULTS	0	0	1	0

Job Tracking: ParaProf profile browser





LU spent 0.162 seconds sending messages of size 44880

It got 833.82 Mflops!

Memory Profiling in TAU

- Configuration option –PROFILEMEMORY
  - Records global heap memory utilization for each function
  - Takes one sample at beginning of each function and associates the sample with function name
- Configuration option -PROFILEHEADROOM
  - Records headroom (amount of free memory to grow) for each function
  - Takes one sample at beginning of each function and associates it with the callstack [TAU_CALLPATH_DEPTH env variable]
  - Useful for debugging memory usage on IBM BG/L, BG/P.
- Independent of instrumentation/measurement options selected
- No need to insert macros/calls in the source code
- User defined atomic events appear in profiles/traces

Memory Profiling in TAU (Atomic events)

Sorted By: number of userEvents

NumSamples	Max	Min	Mean	Std. Dev	Name
252032	2022.7	1181.2	1534.3	410.04	MODULEHYDRO_1D::HYDRO_1D - Heap Memory (KB)
252032	2022.8	1181.7	1534.3	410.04	MODULEINTRFC::INTRFC - Heap Memory (KB)
104559	2023.2	331.13	1526.6	409.54	MODULEEOS3D::EOS3D - Heap Memory (KB)
63008	2022.7	1182	1534.3	410.01	MODULEUPDATE_SOLN::UPDATE_SOLN - Heap Memory (KB)
55545	2023.3	333.07	1514.2	408.31	DBASETREE::DBASENEIGHBORBLOCKLIST - Heap Memory (KB)
51374	2023	1179.4	1497.7	402.53	AMR_PROLONG_GEN_UNK_FUN - Heap Memory (KB)
42120	2022.7	1187.5	1533.5	409.83	ABUNDANCE_RESTRICT - Heap Memory (KB)
41958	2023	346.12	1514.9	408.39	AMR_RESTRICT_UNK_FUN - Heap Memory (KB)
31832	2022.8	1187.4	1534.1	409.91	AMR_RESTRICT_RED - Heap Memory (KB)
31504	2022.7	1181.8	1534.3	410.04	DIFFUSE - Heap Memory (KB)
26042	2023	1179.2	1501.9	403.61	AMR_PROLONG_UNK_FUN - Heap Memory (KB)

Flash2 code profile (-PROFILEMEMORY) on IBM BlueGene/L [MPI rank 0]

Memory Profiling in TAU

- Instrumentation based observation of global heap memory (not per function)
  - call TAU_TRACK_MEMORY()
  - call TAU_TRACK_MEMORY_HEADROOM()
    - Triggers one sample every 10 secs
  - call TAU_TRACK_MEMORY_HERE()
  - call TAU_TRACK_MEMORY_HEADROOM_HERE()
    - Triggers sample at a specific location in source code
  - call TAU_SET_INTERRUPT_INTERVAL(seconds)
    - To set inter-interrupt interval for sampling
  - call TAU_DISABLE_TRACKING_MEMORY()
  - call TAU_DISABLE_TRACKING_MEMORY_HEADROOM()
    - To turn off recording memory utilization
  - call TAU_ENABLE_TRACKING_MEMORY()
  - call TAU_ENABLE_TRACKING_MEMORY_HEADROOM()
    - To re-enable tracking memory utilization

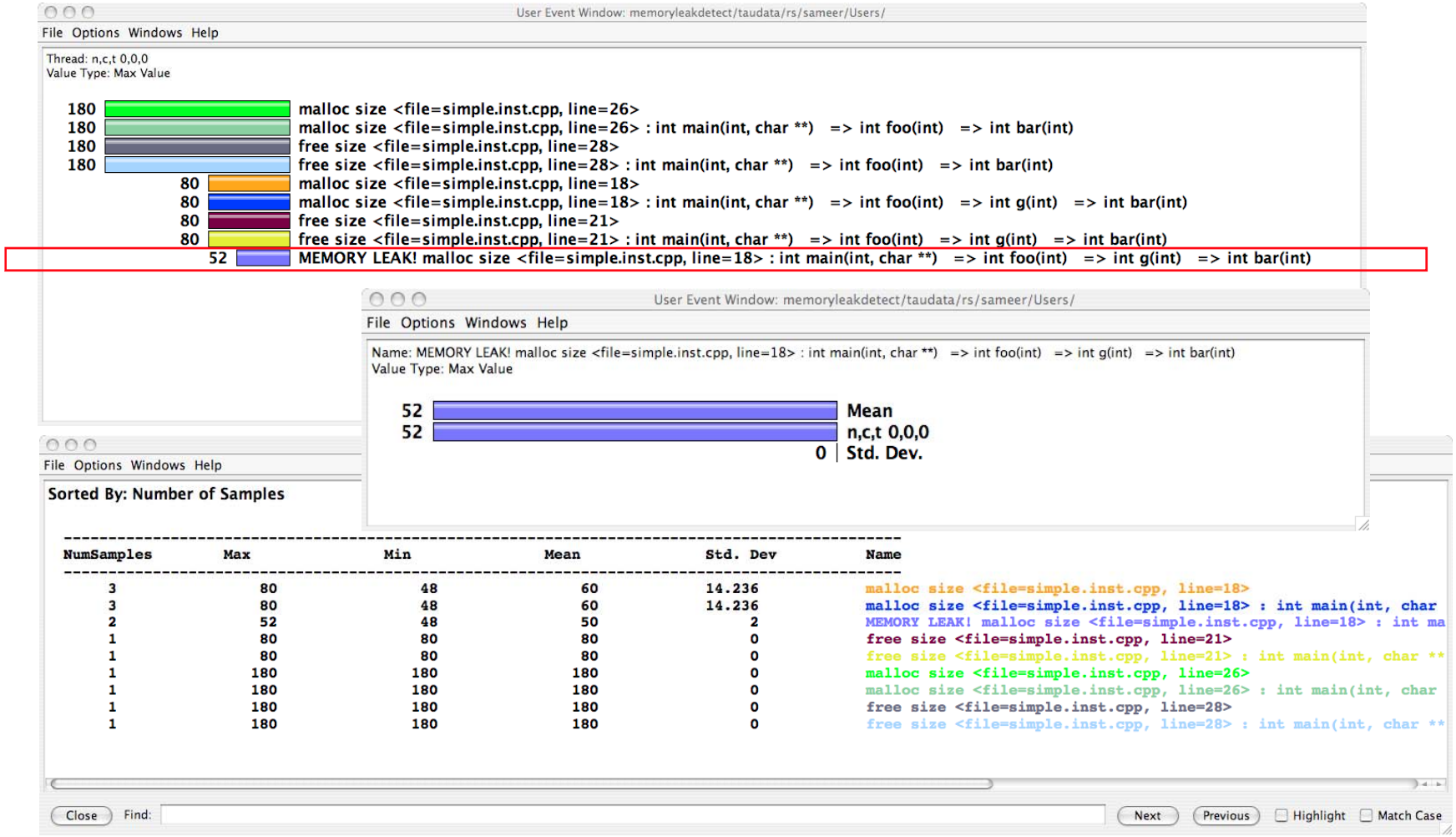
Detecting Memory Leaks in C/C++

- TAU wrapper library for malloc/realloc/free
- During instrumentation, specify

  -optDetectMemoryLeaks option to TAU_COMPILER

  ```
  % setenv TAU_OPTIONS '-optVerbose -optDetectMemoryLeaks'
  % setenv TAU_MAKEFILE <taudir>/<arch>/lib/Makefile.tau-icpc-mpi-pdt...
  % tau_cxx.sh foo.cpp ...
  ```
- Tracks each memory allocation/de-allocation in parsed files
- Correlates each memory event with the executing callstack
- At the end of execution, TAU detects memory leaks
- TAU reports leaks based on allocations and the executing callstack
- Set **TAU_CALLPATH_DEPTH** environment variable to limit callpath data
  - default is 2
- Future work
  - Support for C++ new/delete planned
  - Support for Fortran 90/95 allocate/deallocate planned

Memory Leak Detection



Detecting Memory Leaks in Fortran

```
subroutine foo(x)
integer:: x
integer, allocatable :: A(:), B(:), C(:)

print *, "inside foo"
allocate(A(x), B(x), C(x))
deallocate(A, C)
print *, "exiting foo"

end subroutine foo

program main
call foo(5)
end program main
```

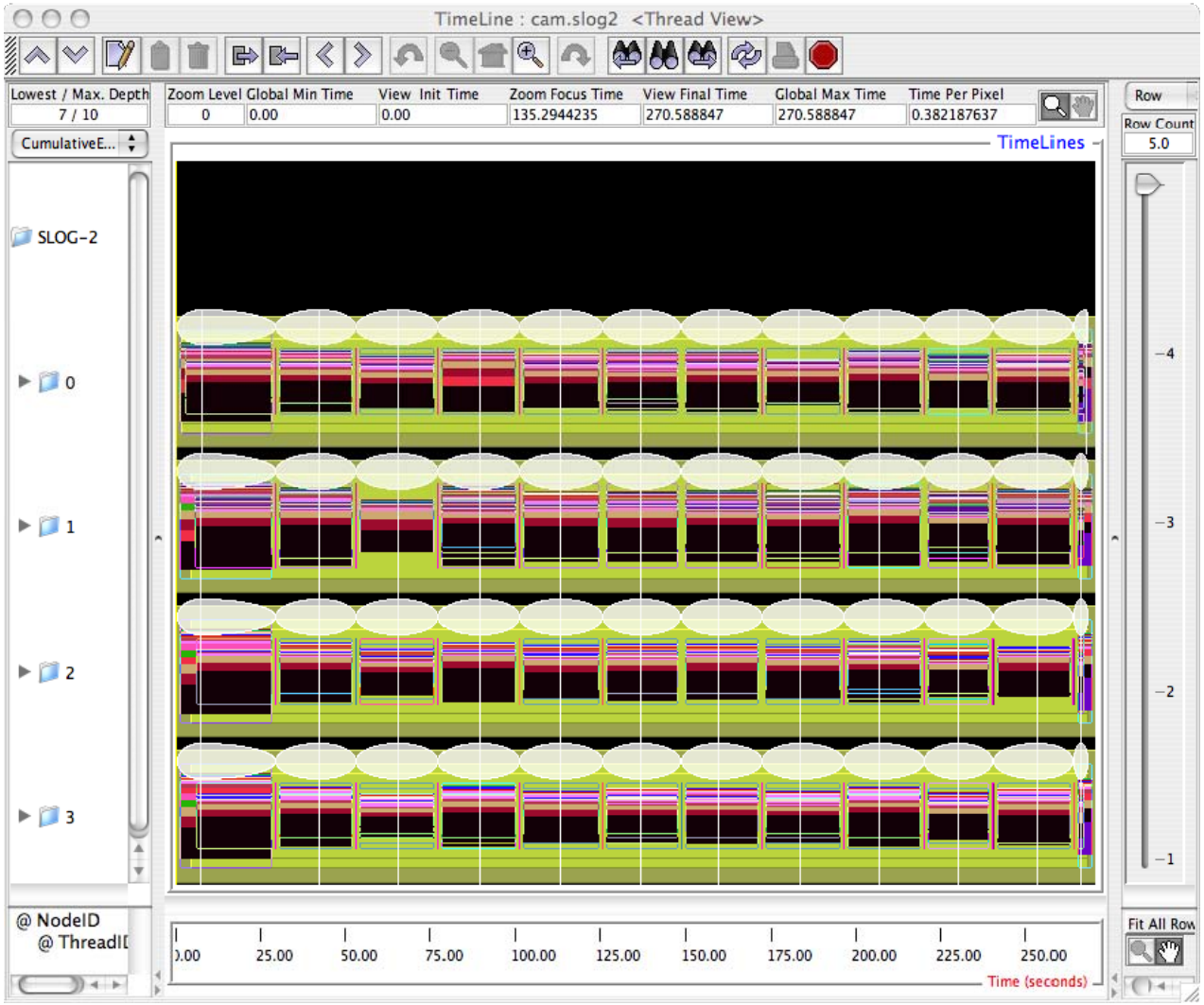
Detecting Memory Leaks in Fortran

```
USER EVENTS Profile :NODE 0, CONTEXT 0, THREAD 0
---------------------------------------------------------------------------------------
NumSamples   MaxValue   MinValue  MeanValue  Std. Dev.  Event Name
---------------------------------------------------------------------------------------
        1          5          5          5          0  MEMORY LEAK! malloc size <file=simple.f, variable=B, line=6> : MAIN =>
FOO
        1          5          5          5          0  free size <file=simple.f, variable=A, line=7>
        1          5          5          5          0  free size <file=simple.f, variable=A, line=7> : MAIN => FOO
        1          5          5          5          0  free size <file=simple.f, variable=C, line=7>
        1          5          5          5          0  free size <file=simple.f, variable=C, line=7> : MAIN => FOO
        1          5          5          5          0  malloc size <file=simple.f, variable=A, line=6>
        1          5          5          5          0  malloc size <file=simple.f, variable=A, line=6> : MAIN => FOO
        1          5          5          5          0  malloc size <file=simple.f, variable=B, line=6>
        1          5          5          5          0  malloc size <file=simple.f, variable=B, line=6> : MAIN => FOO
        1          5          5          5          0  malloc size <file=simple.f, variable=C, line=6>
        1          5          5          5          0  malloc size <file=simple.f, variable=C, line=6> : MAIN => FOO
---------------------------------------------------------------------------------------
```

Jumpshot

- http://www-unix.mcs.anl.gov/perfvis/software/viewers/index.htm
- Developed at Argonne National Laboratory as part of the MPICH project
- Rusty Lusk, PI
  - Also works with other MPI implementations
  - 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

Tracing: Using TAU and Jumpshot

- Configure TAU with -TRACE option:

```
% configure –TRACE –otf=<dir>
  -MULTIPLECOUNTERS –papi=<dir> -mpi
  –pdt=dir …
```

- Set environment variables:

```
% setenv TRACEDIR /p/gm1/<login>/traces
% setenv COUNTER1 GET_TIME_OF_DAY (reqd)
% setenv COUNTER2 PAPI_FP_INS
% setenv COUNTER3 PAPI_TOT_CYC …
```

- Execute application and analyze the traces:

```
% mpirun -np 32 ./a.out [args]

% tau_treemerge.pl
% tau2slog2 tau.trc tau.edf –o app.slog2
% jumpshot app.slog2
```

Performance Analysis and Visualization

- Analysis of parallel profile and trace measurement
- **Parallel profile analysis**
  - ***ParaProf***: parallel profile analysis and presentation
  - ***ParaVis***: parallel performance visualization package
  - Profile generation from trace data (***tau2profile***)
- Performance data management framework (***PerfDMF***)
- **Parallel trace analysis**
  - Translation to *VTF* (V3.0), *EPILOG*, *OTF* formats
  - Integration with *VNG* (Technical University of Dresden)
- Online parallel analysis and visualization
- Integration with *CUBE* browser (KOJAK, UTK, FZJ)

Using Performance Database (PerfDMF)

- Configure PerfDMF (Done by each user)

  % perfdmf_configure

  - Choose derby, PostgreSQL, MySQL, Oracle or DB2
  - Hostname
  - Username
  - Password
  - Say yes to downloading required drivers (we are not allowed to distribute these)
  - Stores parameters in your ~/.ParaProf/perfdmf.cfg file
- Configure PerfExplorer (Done by each user)

  % perfexplorer_configure
- Execute PerfExplorer

  % perfexplorer

PerfExplorer Analysis Methods

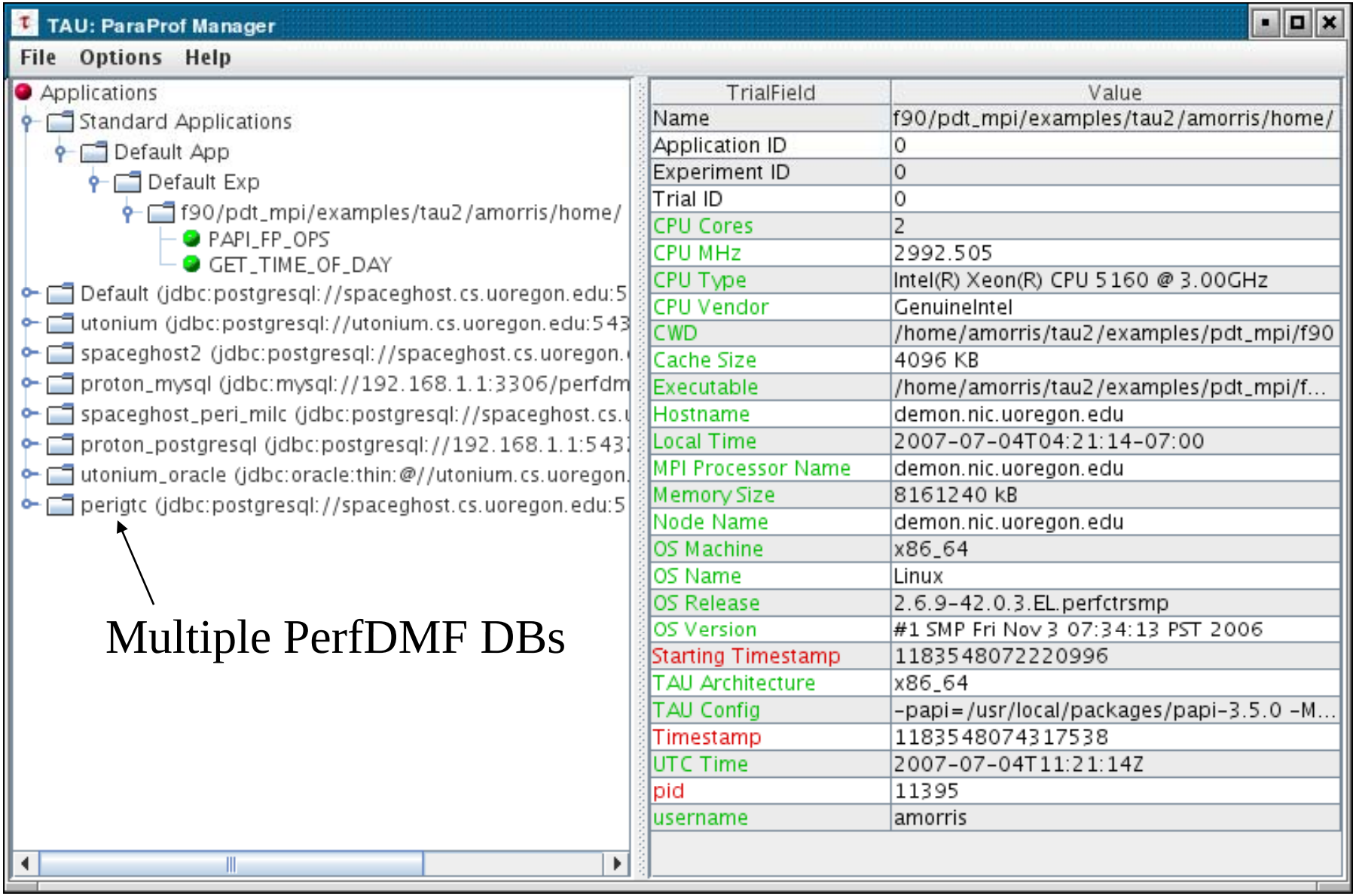
- Data summaries, distributions, scatterplots
- Clustering
  - *k*-means
  - Hierarchical
- Correlation analysis
- Dimension reduction
  - PCA
  - Random linear projection
  - Thresholds
- Comparative analysis
- Data management views

PerfDMF and the TAU Portal

- Development of the TAU portal
  - Common repository for collaborative data sharing
  - Profile uploading, downloading, user management
  - Paraprof, PerfExplorer can be launched from the portal using Java Web Start (no TAU installation required)
- Portal URL

  http://tau.nic.uoregon.edu

New automated metadata collection



TAU Performance System Status

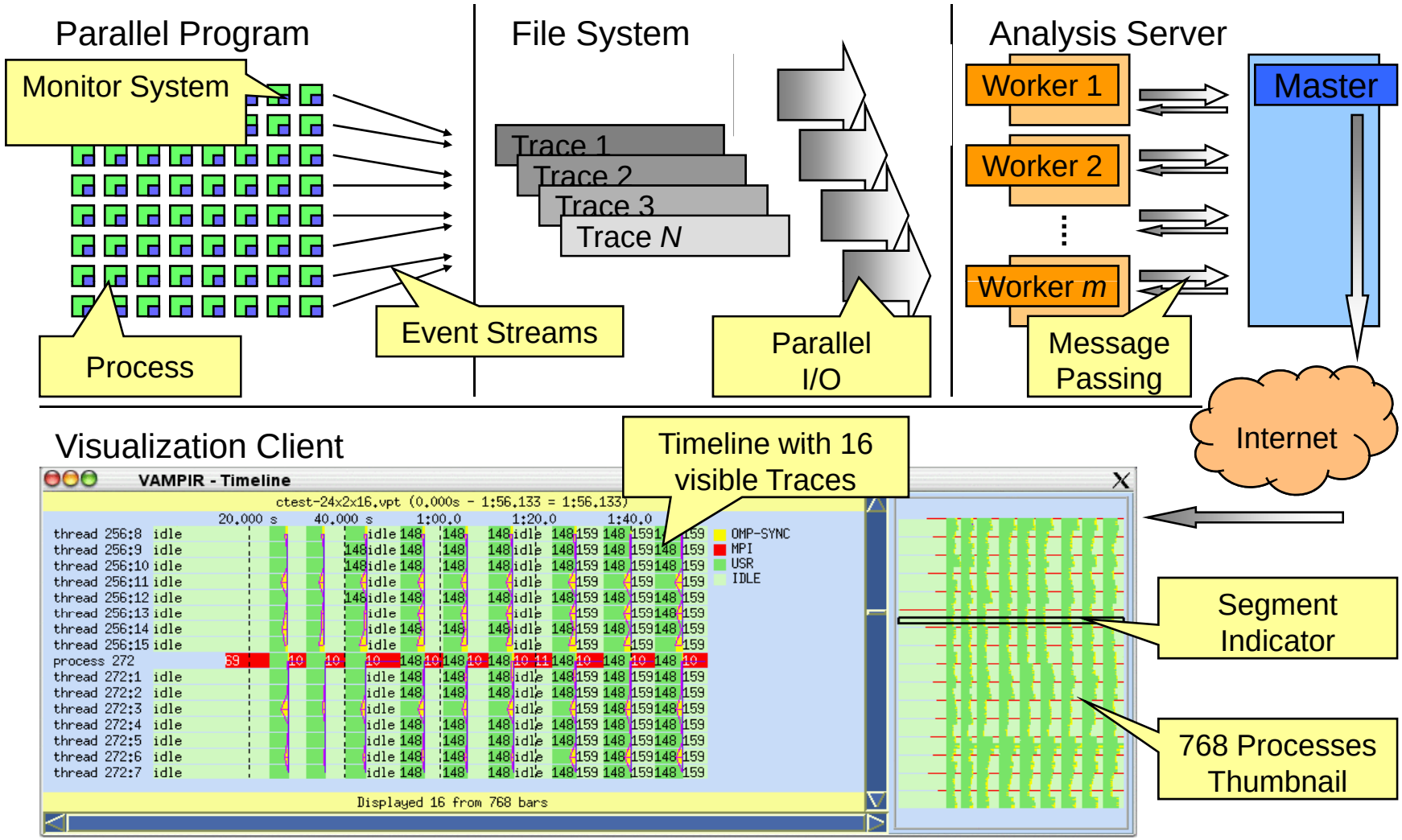
- Computing platforms (selected)
  - IBM SP/pSeries/BGL/BGP/Cell PPE, SGI Altix/Origin, Cray T3E/SV-1/X1/XT3/XT4 (CNL), HP (Compaq) SC (Tru64), Sun, Linux clusters (IA-32/64, Alpha, PPC, PA-RISC, Power, Opteron), Apple (G4/5, Intel OS X), Hitachi SR8000, NEC SX Series, Windows XP/Vista …
- Programming languages
  - C, C++, Fortran 77/90/95, HPF, Java, Python
- Thread libraries (selected)
  - pthreads, OpenMP, SGI sproc, Java,Windows, Charm++
- Compilers (selected)
  - Intel, PGI, GNU, Fujitsu, Sun, PathScale, SGI, Cray, IBM, HP, NEC, Absoft, Lahey, Nagware, ...

Vampir, VNG, and OTF

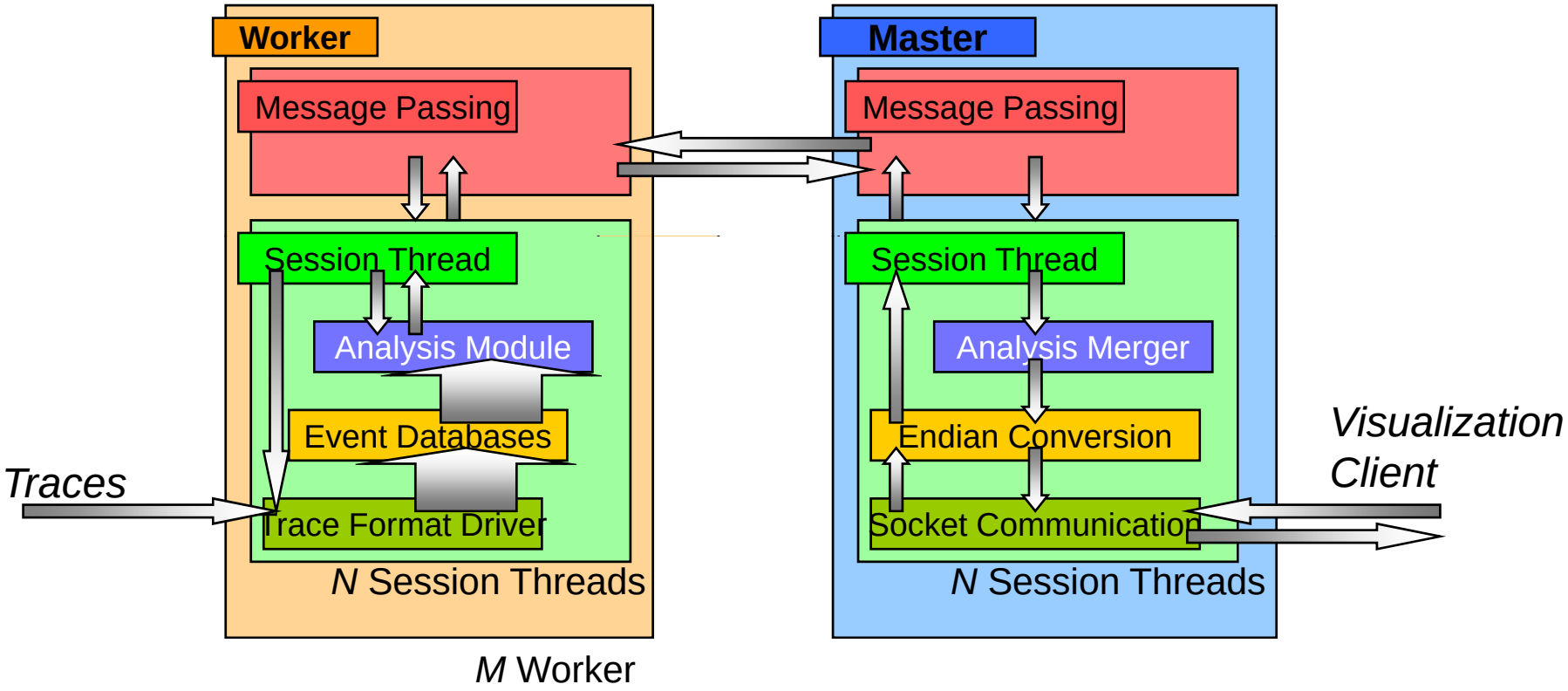
- Commercial trace based tools developed at ZiH, T.U. Dresden
  - Wolfgang Nagel, Holger Brunst and others…
- Vampir Trace Visualizer (aka Intel ® Trace Analyzer v4.0)
  - Sequential program
- Vampir Next Generation (VNG)
  - Client (vng) runs on a desktop, server (vngd) on a cluster
  - Parallel trace analysis
  - Orders of magnitude bigger traces (more memory)
  - State of the art in parallel trace visualization
- Open Trace Format (OTF)
  - Hierarchical trace format, efficient streams based parallel access with VNGD
  - Replacement for proprietary formats such as STF
  - Tracing library available with a evaluation license now. Open source package at SC'06.

http://www.vampir-ng.de

Vampir Next Generation (VNG) Architecture

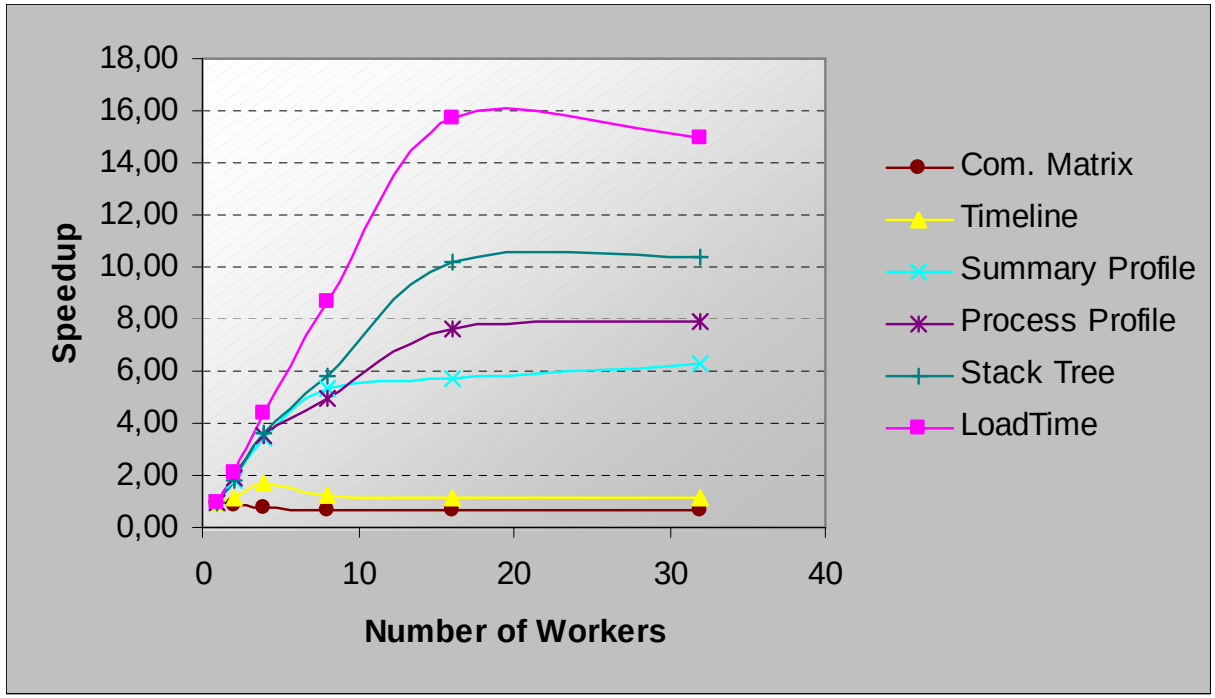


VNG Parallel Analysis Server



Scalability of VNG [Holger Brunst, WAPA 2005]

- sPPM
- 16 CPUs
- 200 MB



Number of Workers	1	2	4	8	16	32
Load Time	47,33	22,48	10,80	5,43	3,01	3,16
Timeline	0,10	0,09	0,06	0,08	0,09	0,09
Summary Profile	1,59	0,87	0,47	0,30	0,28	0,25
Process Profile	1,32	0,70	0,38	0,26	0,17	0,17
Com. Matrix	0,06	0,07	0,08	0,09	0,09	0,09
Stack Tree	2,57	1,39	0,70	0,44	0,25	0,25

VNG Analysis Server Architecture

- Implementation using MPI and Pthreads
- Client/server approach
- MPI and pthreads are available on most platforms
- Workload and data distribution among “physical” MPI processes
- Support of multiple visualization clients by using virtual sessions handled by individual threads
- Sessions are scheduled as threads

TAU Tracing Enhancements

- Configure TAU with -TRACE –otf=<dir> option

```
% configure –TRACE –otf=<dir> …
```

Generates tau_merge, tau2vtf, tau2otf tools in <tau>/<arch>/bin directory

```
% tau_f90.sh app.f90 –o app
```

- Instrument and execute application

```
% mpirun -np 4 app
```

- Merge and convert trace files to OTF format

```
% tau_treemerge.pl
% tau2otf tau.trc tau.edf app.otf [-z][–n <nstreams>]
% vampir app.otf
```

OR use VNG to analyze OTF/VTF trace files

Environment Variables

- Configure TAU with -TRACE –otf=<dir> option

```
% configure –TRACE –otf=<dir>
  -MULTIPLECOUNTERS –papi=<dir> -mpi
  –pdt=dir …
```

- Set environment variables

```
% setenv TRACEDIR /p/gm1/<login>/traces
% setenv COUNTER1 GET_TIME_OF_DAY (reqd)
% setenv COUNTER2 PAPI_FP_INS
% setenv COUNTER3 PAPI_TOT_CYC …
```

- Execute application

```
% mpirun -np 32 ./a.out [args]

% tau_treemerge.pl
% tau2otf tau.trc tau.edf app.otf -z
```

Using VampirTrace to generate OTF traces

- Configure TAU with -TRACE –vampirtrace=<dir> option

```
% configure –TRACE  -vampirtrace=<dir> –papi=<dir>
   -mpi
   –pdt=dir …
```

- Set environment variables

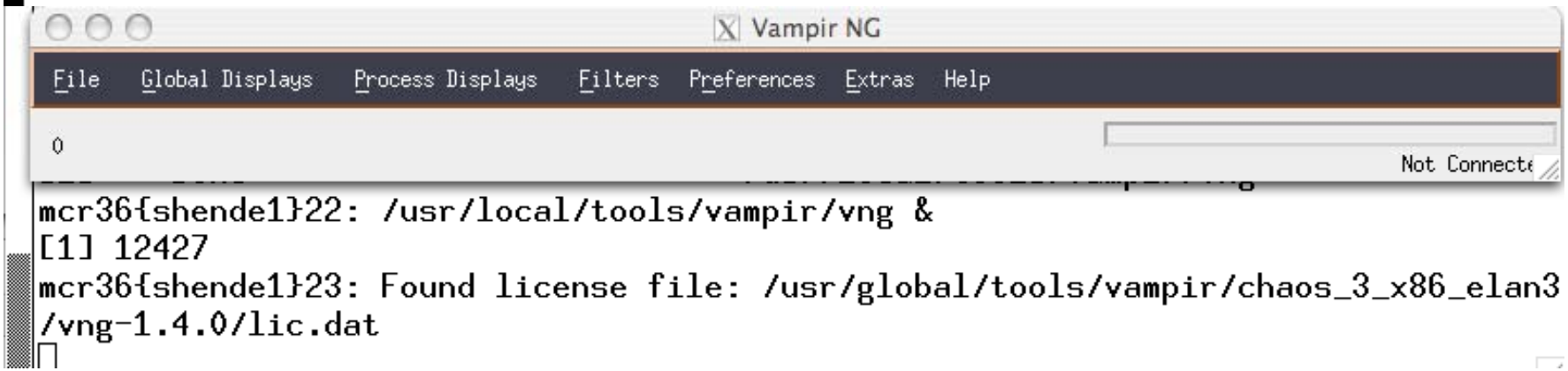
```
% setenv VT_METRICS PAPI_FP_OPS:PAPI_TOT_CYC
```

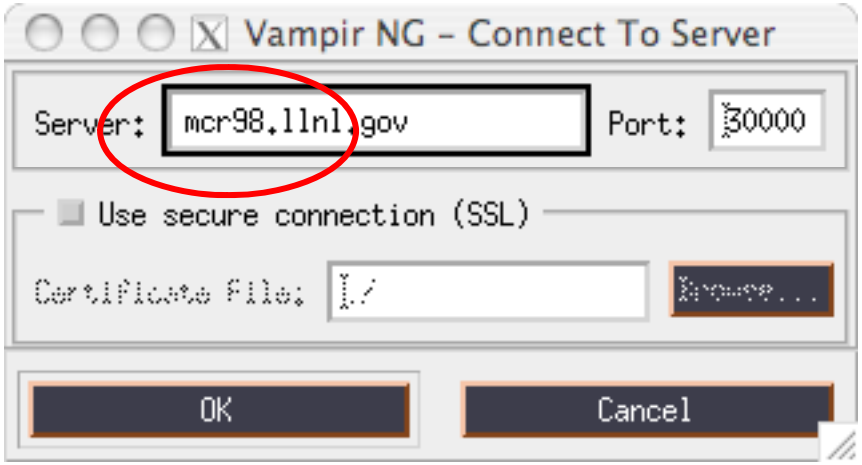
- Execute application

```
% mpirun -np 20 ./a.out [args]
On IBM AIX, running the application will create
   a.otf after unifying the events
Unifies the descriptors to generate a.otf
% vampir a.otf &
```

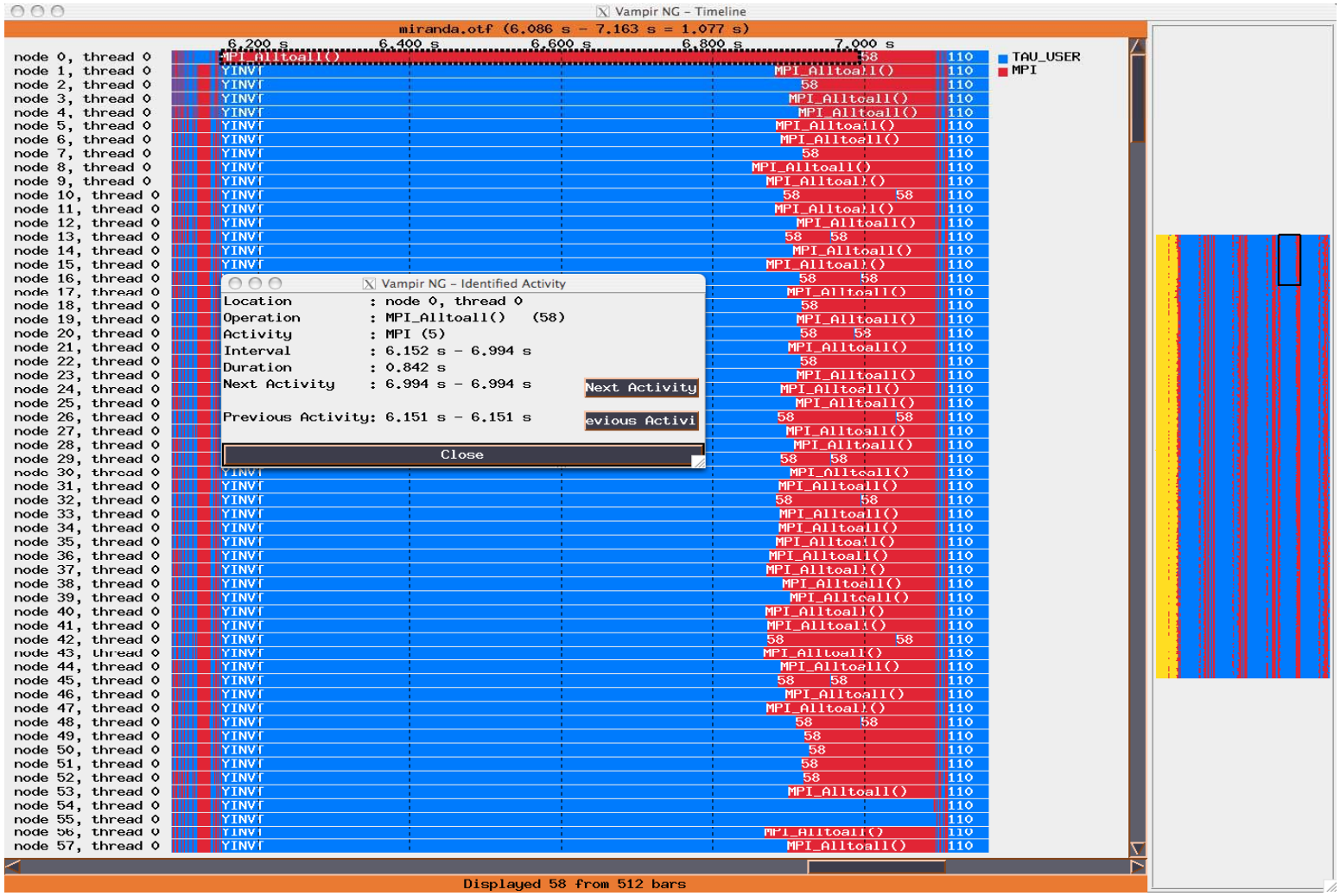
Using Vampir Next Generation (VNG v1.4)

```
mcr36{shende1}32: srun -N2 -n4 -p pdebug /usr/local/tools/vampir/vngd
Service process resides on "mcr98"
Found license file: /usr/global/tools/vampir/chaos_3_x86_elan3/vng-1.4.0/lic.dat
running...
```

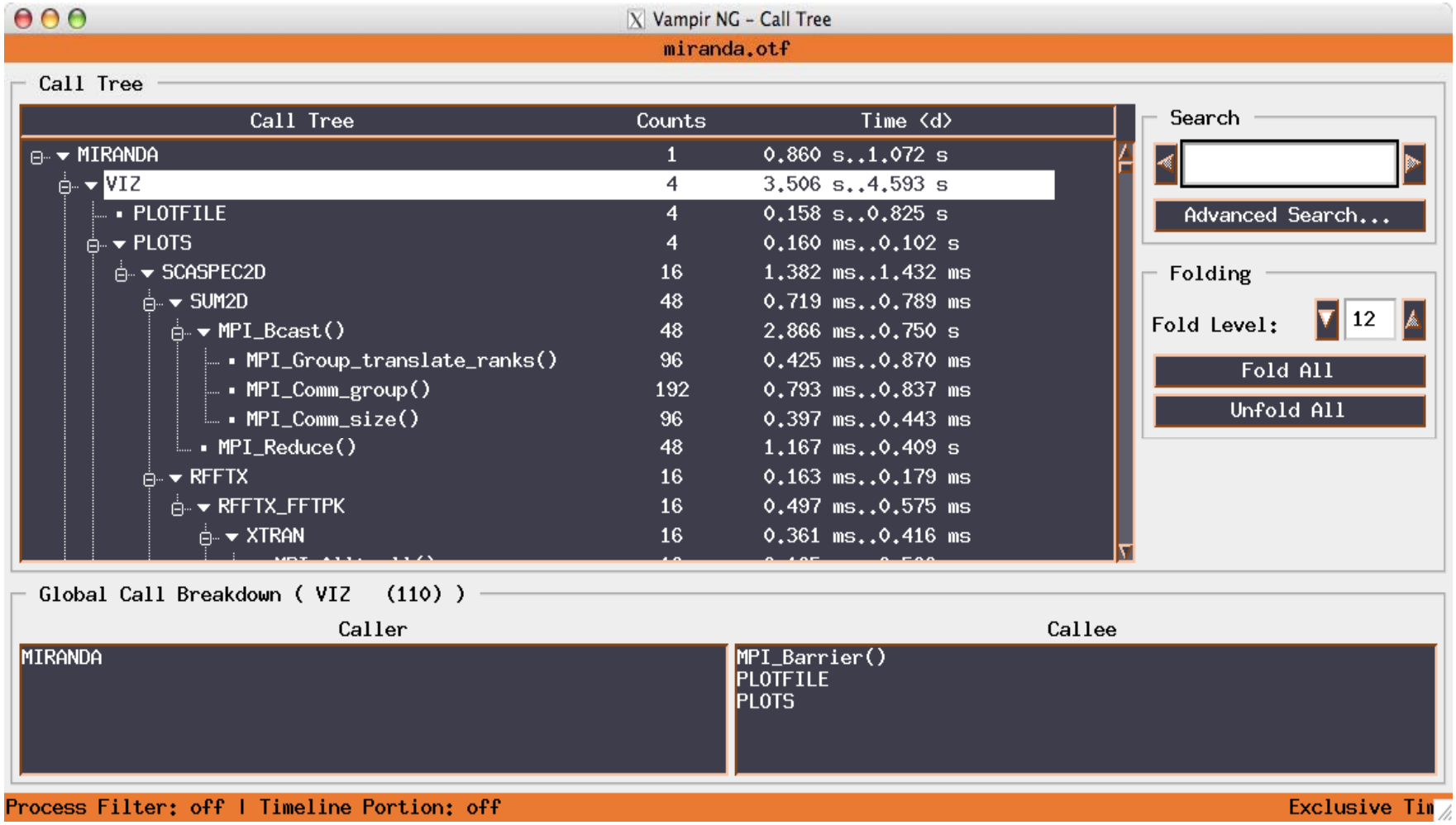




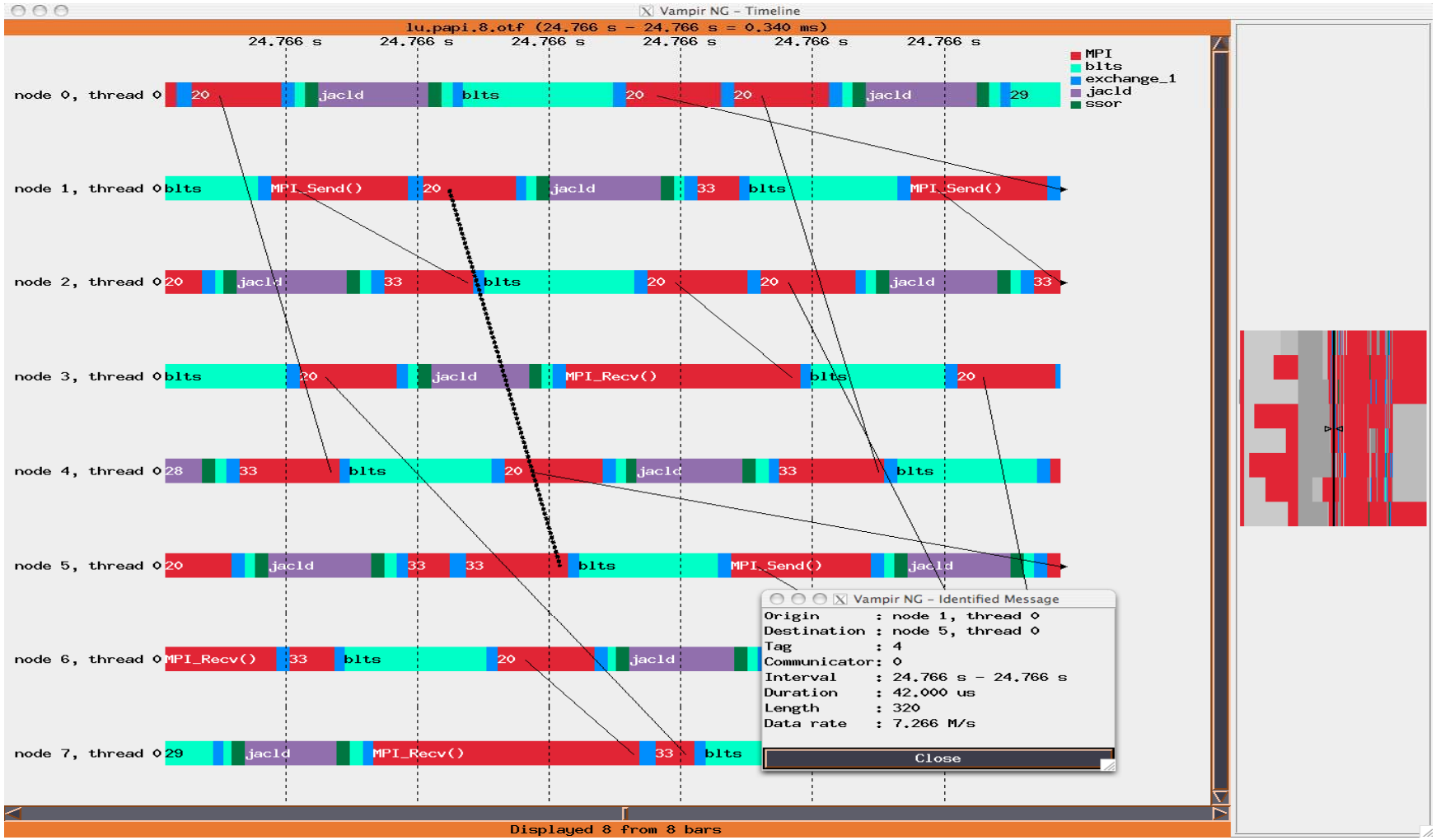
VNG Timeline Display



VNG Calltree Display

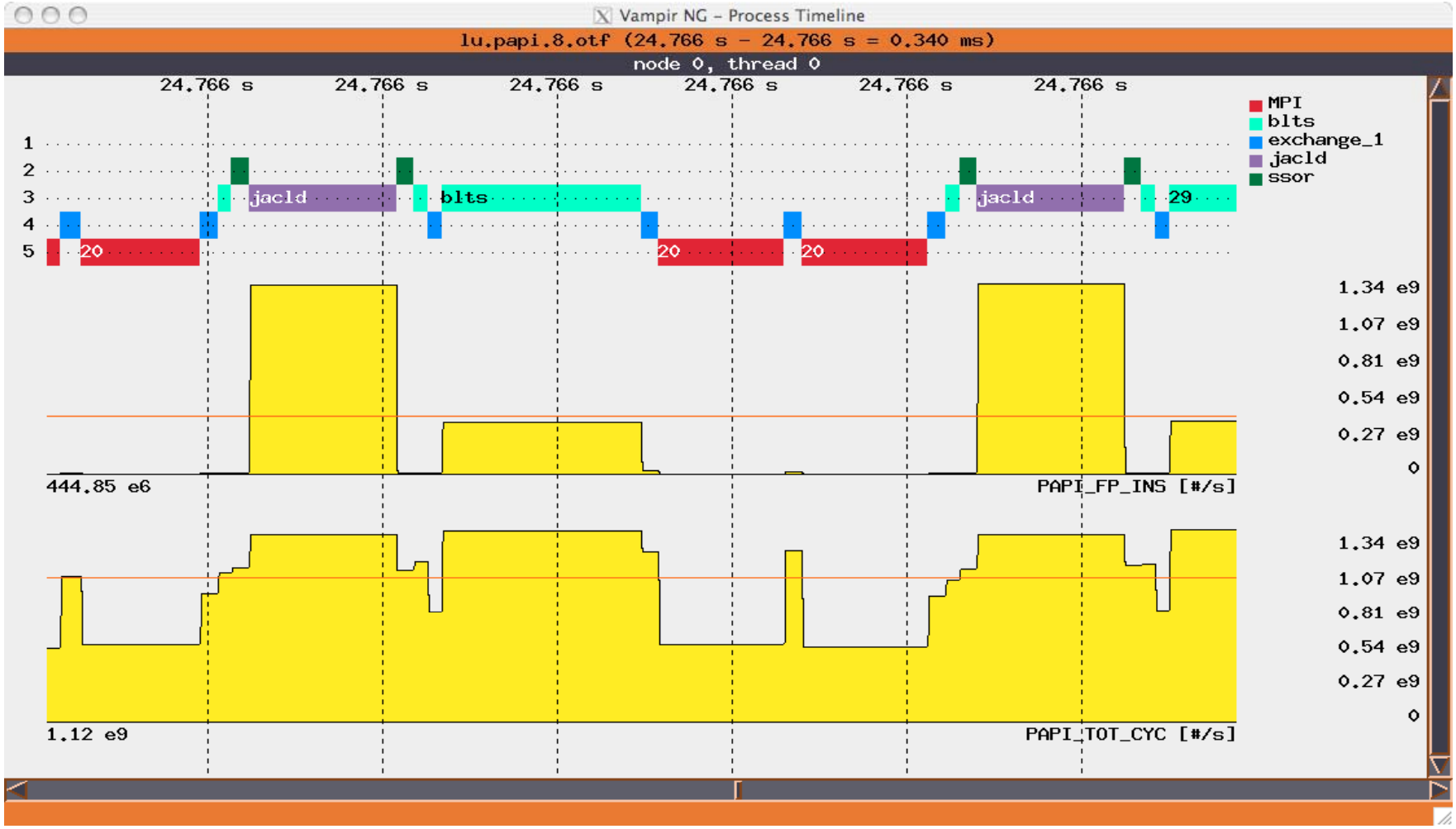


VNG Timeline Zoomed In

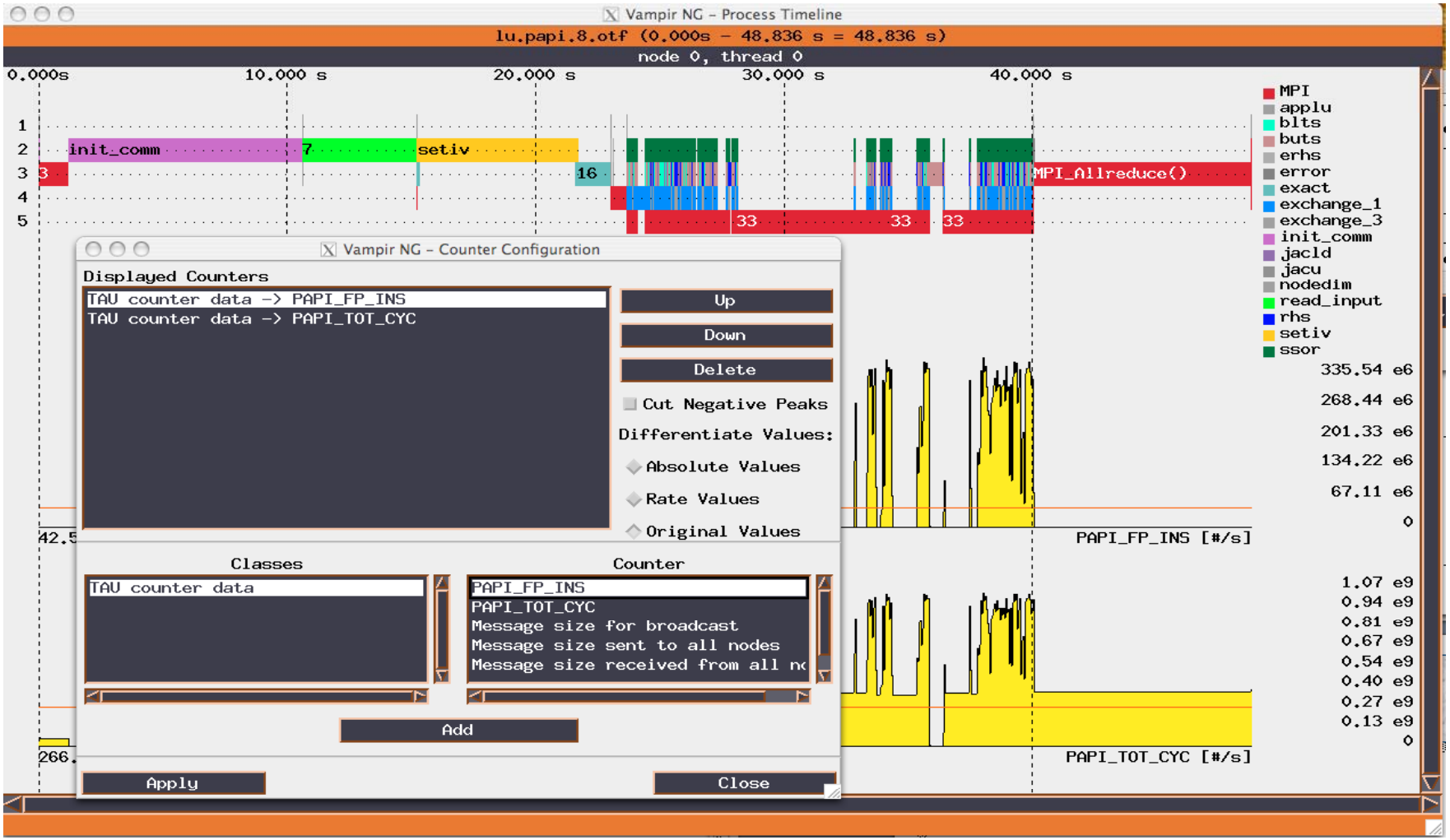


VNG Grouping of Interprocess Communications

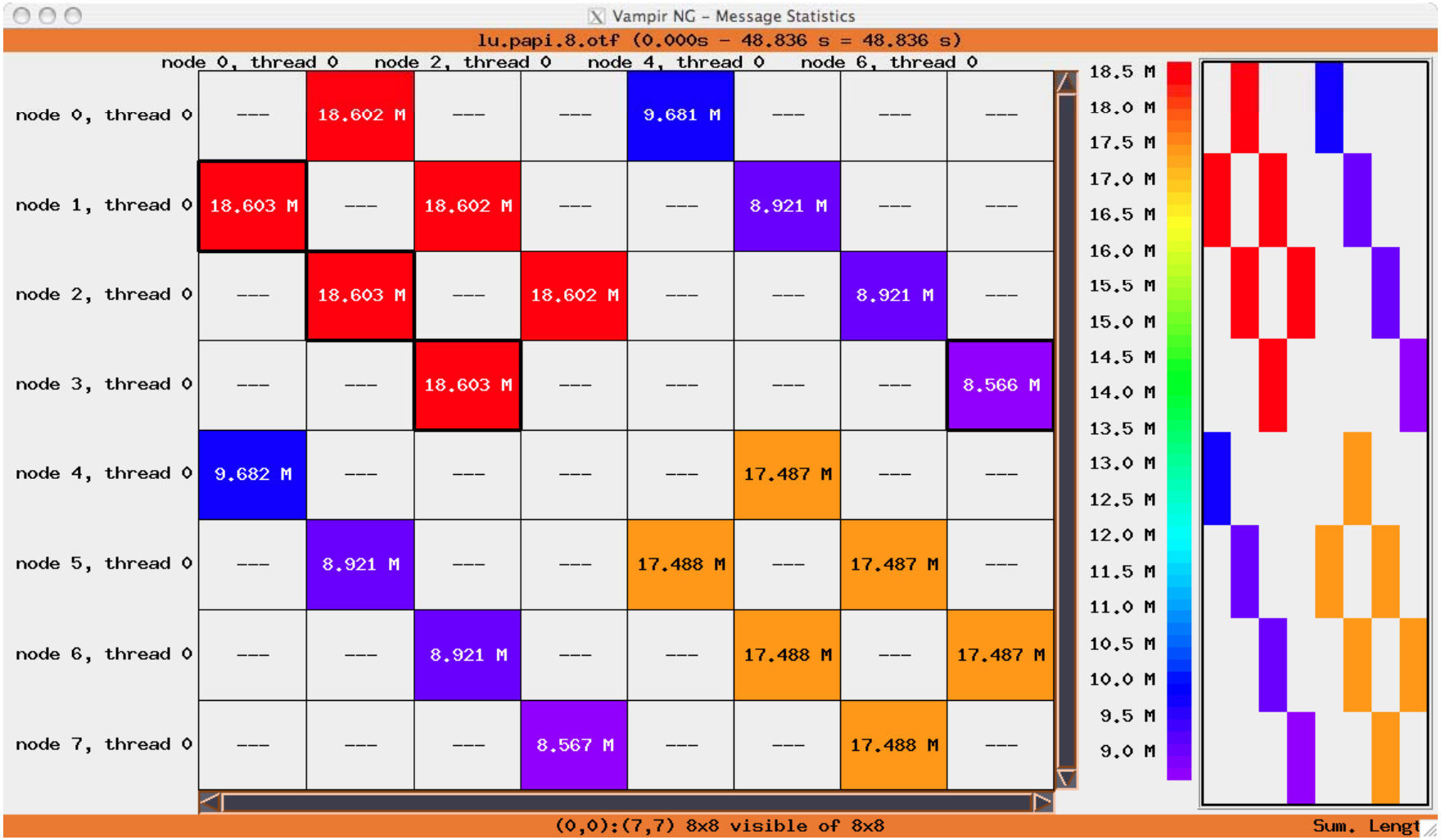
VNG Process Timeline with PAPI Counters



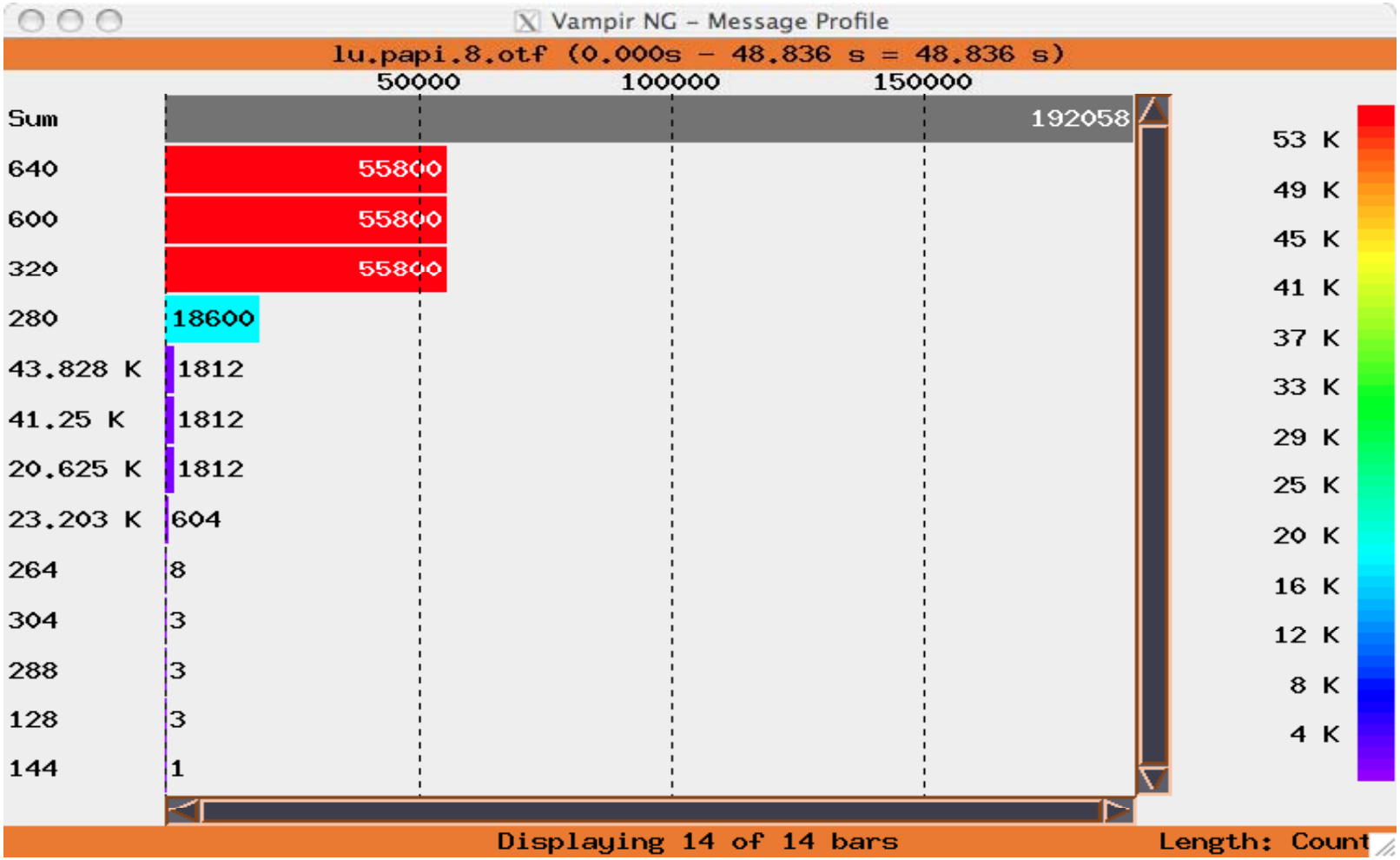
OTF/VNG Support for Counters



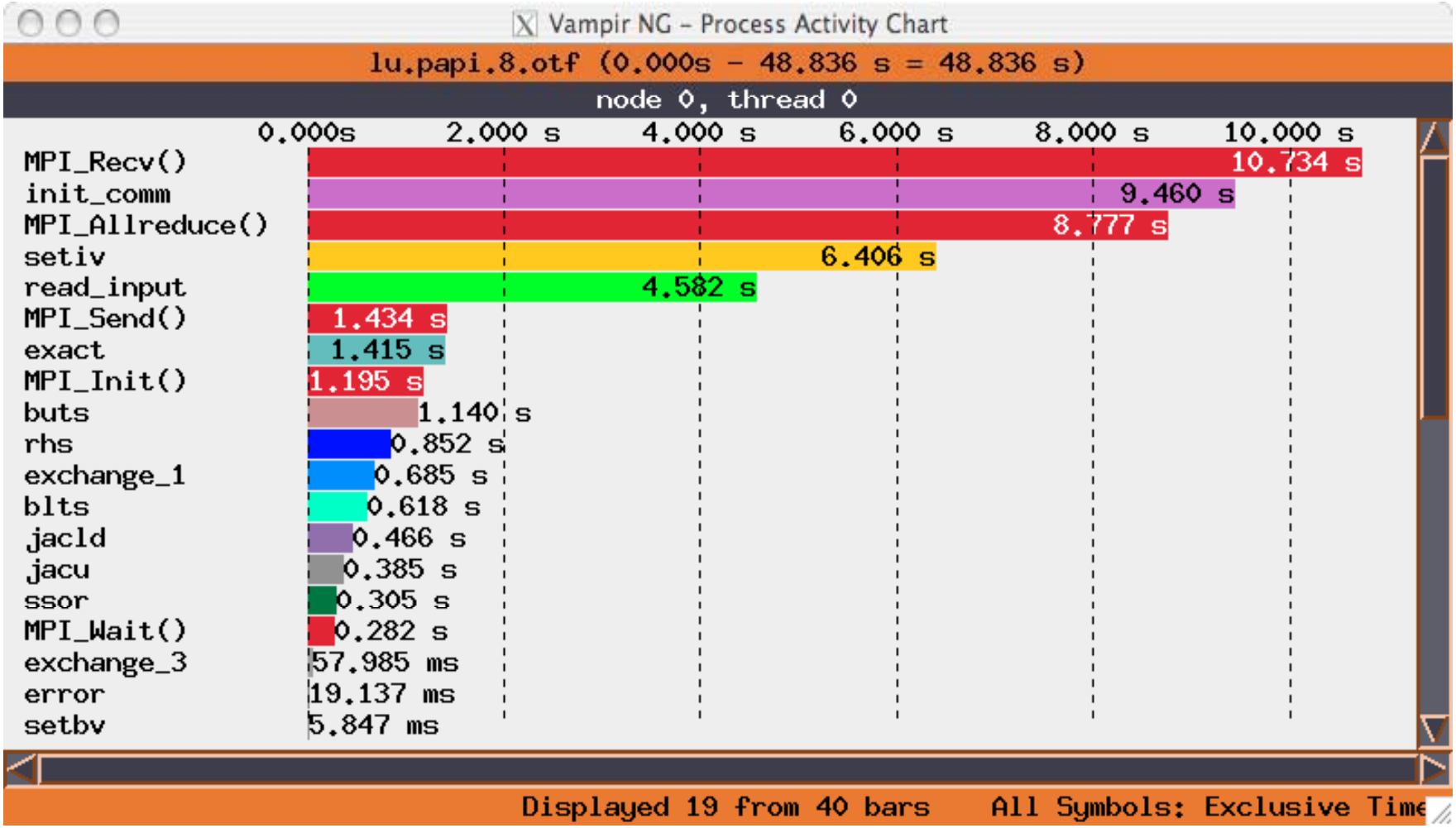
VNG Communication Matrix Display



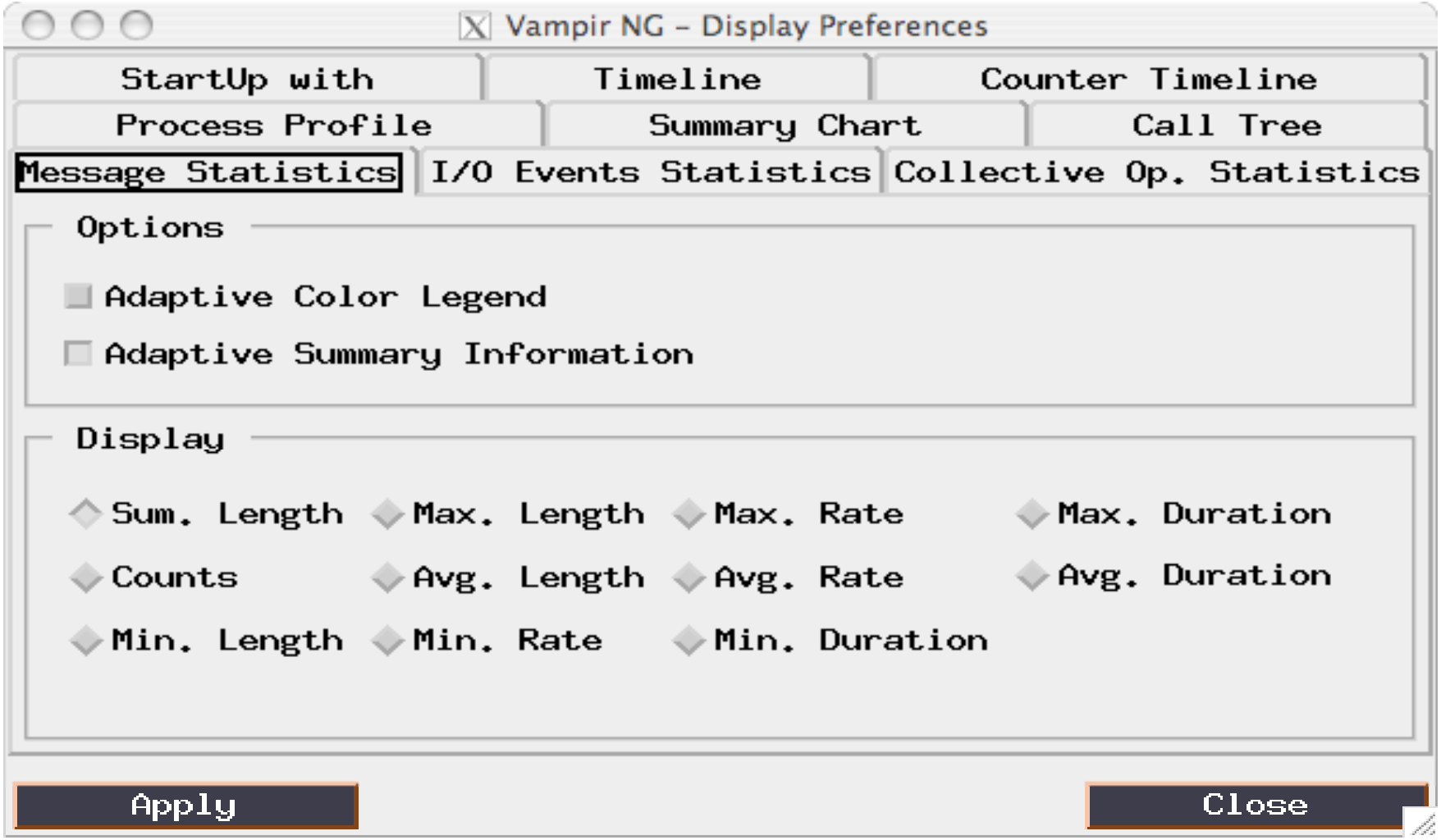
VNG Message Profile



VNG Process Activity Chart



VNG Preferences



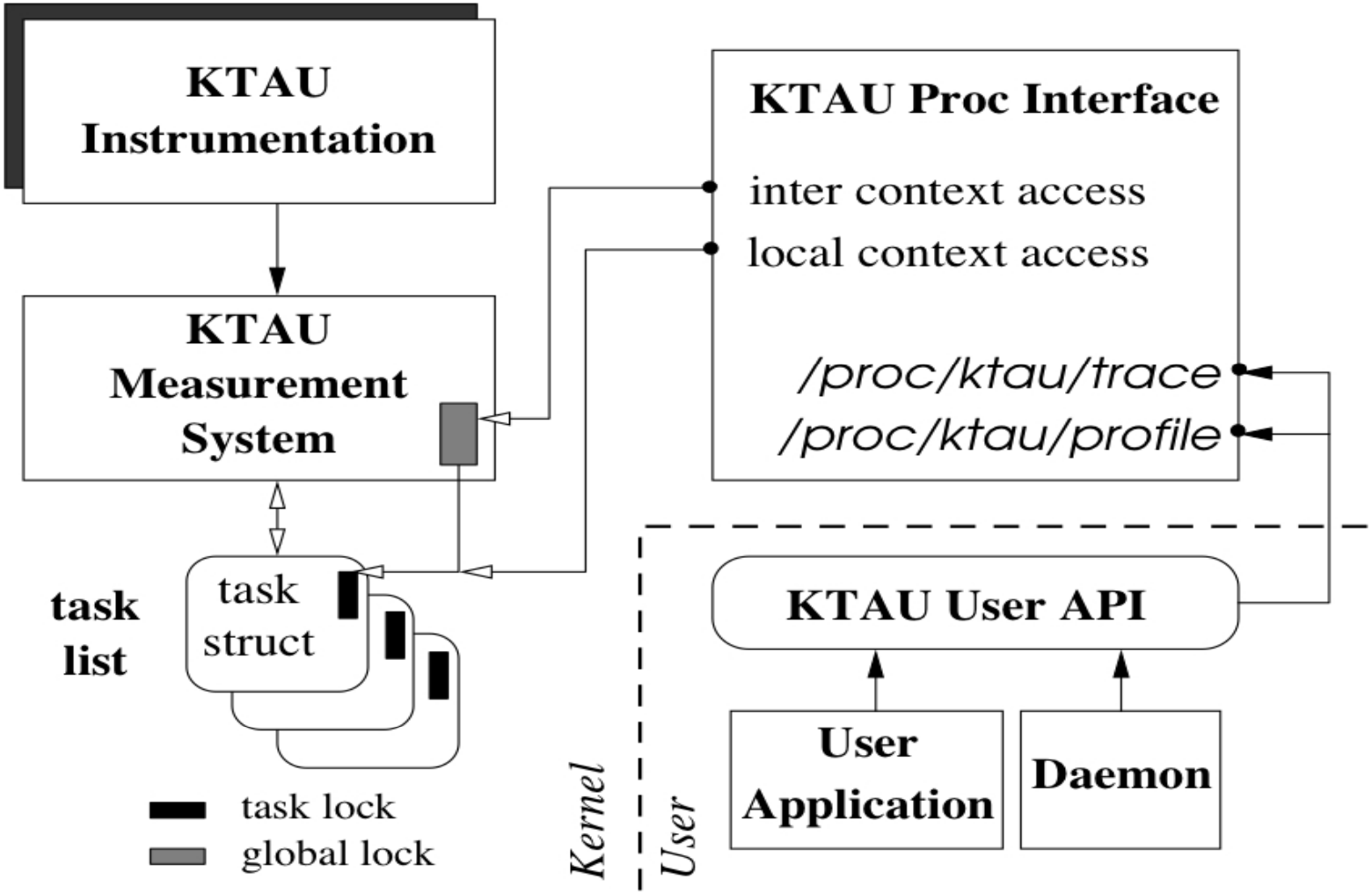
KTAU Project

- Trend toward Extremely Large Scales
  - System-level influences are increasingly dominant performance bottleneck contributors
  - Application sensitivity at scale to the system (e.g., OS noise)
  - Complex I/O path and subsystems another example
  - Isolating system-level factors non-trivial
- OS Kernel instrumentation and measurement is important to understanding system-level influences
- But can we closely correlate observed application and OS performance?
- KTAU / TAU (Part of the ANL/UO ZeptoOS Project)
  - Integrated methodology and framework to measure whole-system performance

Applying KTAU+TAU

- How does ***real*** OS-noise affect ***real*** applications on target platforms?
  - Requires a tightly coupled performance measurement & analysis approach provided by KTAU+TAU
  - Provides an estimate of application slowdown due to Noise (and in particular, different noise-components - IRQ, scheduling, etc)
  - Can empower both application and the middleware and OS communities.
  - A. Nataraj, A. Morris, A. Malony, M. Sottile, P. Beckman, "The Ghost in the Machine : Observing the Effects of Kernel Operation on Parallel Application Performance", SC'07.
- Measuring and analyzing complex, multi-component I/O subsystems in systems like BG(L/P) (work in progress).

KTAU System Architecture



A. Nataraj, A. Malony, S. Shende, and A. Morris, "**Kernel-level Measurement for Integrated Performance Views: the KTAU Project**," *Cluster 2006*, distinguished paper.

TAU Performance System Status

- Computing platforms (selected)
  - IBM SP/pSeries/BGL, SGI Altix/Origin, Cray T3E/SV-1/X1/XT3, HP (Compaq) SC (Tru64), Sun, Linux clusters (IA-32/64, Alpha, PPC, PA-RISC, Power, Opteron), Apple (G4/5, OS X), Hitachi SR8000, NEC SX-5/6, SiCortex, Windows …
- Programming languages
  - C, C++, Fortran 77/90/95, HPF, Java, Python
- Thread libraries (selected)
  - pthreads, OpenMP, SGI sproc, Java,Windows, Charm++
- Compilers (selected)
  - Intel, , GNU, Fujitsu, Sun, PathScale, SGI, Cray, IBM, HP, NEC, Absoft, Lahey, Nagware

Labs!

Lab: TAU

Lab Instructions

Get workshop.tar.gz on Jacquard:

```
% wget http://tau.uoregon.edu/workshop.tar.gz
% gtar zxf workshop.tar.gz
```

and follow the instructions in the README file.

These slides are available at:
http://tau.uoregon.edu/tau.ppt

Lab Instructions

To profile a code:

1. Add TAU to your path:
   % module load tau
   or
   % set path=($path /usr/common/acts/TAU/tau-2.17.2/x86_64/bin)
2. Change the compiler name to tau_cxx.sh, tau_f90.sh, tau_cc.sh:
   F90 = tau_f90.sh
3. Choose TAU stub makefile
   % setenv TAU_MAKEFILE
   /opt/apps/tau/tau_latest/x86_64/lib/Makefile.tau-[options]
4. If stub makefile has –multiplecounters in its name, set COUNTER[1-<n>] environment variables:
   % setenv COUNTER1 GET_TIME_OF_DAY
   % setenv COUNTER2 PAPI_FP_INS
   % setenv COUNTER3 PAPI_TOT_CYC …
5. Set TAU_THROTTLE environment variable to throttle instrumentation:
   % setenv TAU_THROTTLE 1
6. Build and run workshop examples, then run pprof/paraprof

Support Acknowledgements

- Department of Energy (DOE)
  - Office of Science
    - MICS, Argonne National Lab
  - ASC/NNSA
    - University of Utah ASC/NNSA Level 1
    - ASC/NNSA, LLNL
- Department of Defense (DoD)
  - HPC Modernization Office (HPCMO)
- NSF SDCI
- Research Centre Juelich
- LBL, ORNL, ANL, LANL, LLNL
- TU Dresden
- ParaTools, Inc.