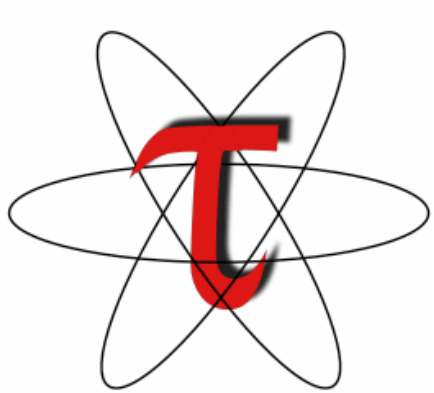
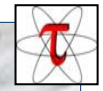
Performance and Memory Evaluation using TAU

Sameer Shende, Allen D. Malony, Alan Morris
University of Oregon
{sameer, malony, amorris}@cs.uoregon.edu

Peter H. Beckman
Argonne National Laboratory
beckman@mcs.anl.gov

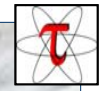


Outline of Talk



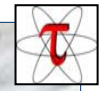
- Overview of TAU
- Instrumentation
- Measurement
- Analysis
- Conclusions

TAU Performance System



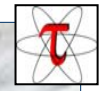
- *T*uning and *A*nalysis *U*tilities (14+ year project effort)
- Performance system framework for HPC systems
  - Integrated, scalable, flexible, and parallel
- Targets a general complex system computation model
  - *Entities*: nodes / contexts / threads
  - Multi-level: system / software / parallelism
  - Measurement and analysis abstraction
- Integrated toolkit for performance problem solving
  - Instrumentation, measurement, analysis, and visualization
  - Portable performance profiling and tracing facility
  - Performance data management and data mining
- **http://www.cs.uoregon.edu/research/tau**

Definitions – Profiling



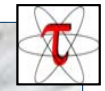
- Profiling
  - Recording of summary information during execution
    - inclusive, exclusive time, # calls, hardware statistics, …
  - Reflects performance behavior of program entities
    - functions, loops, basic blocks
    - user-defined “semantic” entities
  - Very good for low-cost performance assessment
  - 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



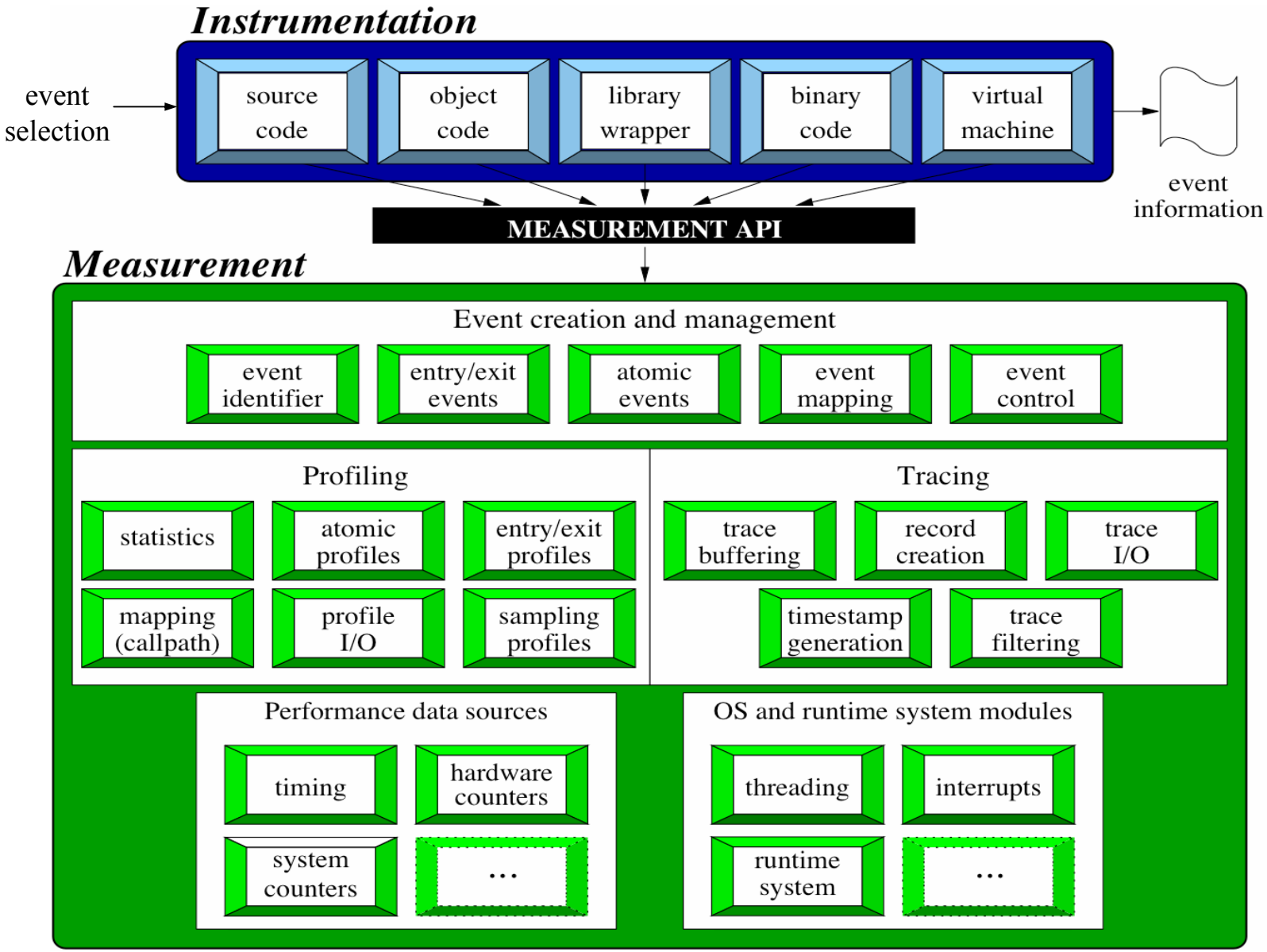
- Tracing
  - Recording of information about significant points (events) during program execution
    - entering/exiting code region (function, loop, block, …)
    - thread/process interactions (e.g., send/receive message)
  - Save information in event record
    - timestamp
    - CPU identifier, thread 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

TAU Parallel Performance System Goals

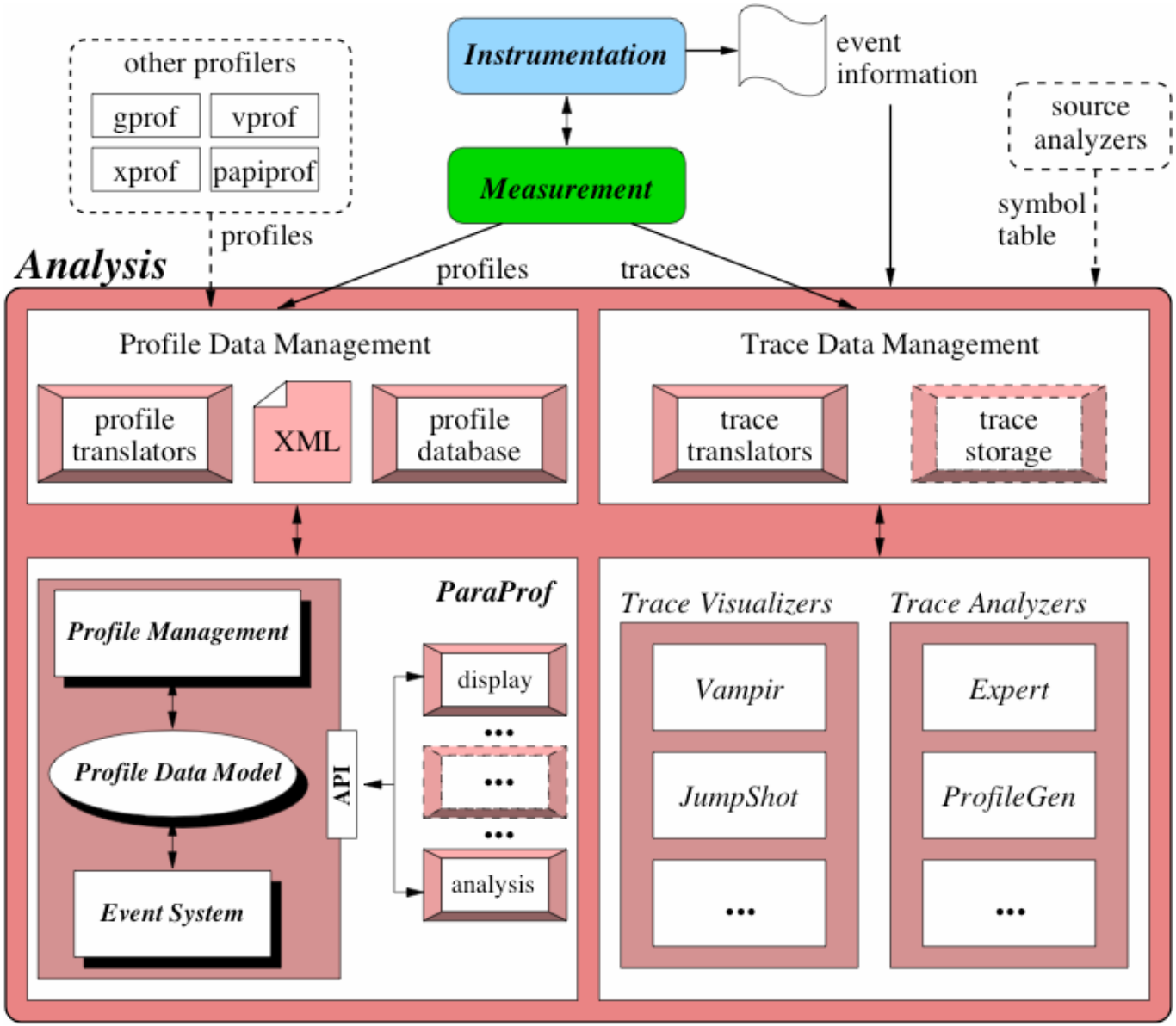


- Multi-level performance instrumentation
  - Multi-language automatic source instrumentation
- Flexible and configurable performance measurement
- Widely-ported parallel performance profiling system
  - Computer system architectures and operating systems
  - Different programming languages and compilers
- Support for multiple parallel programming paradigms
  - Multi-threading, message passing, mixed-mode, hybrid
- Support for performance mapping
- Support for object-oriented and generic programming
- Integration in complex software, systems, applications

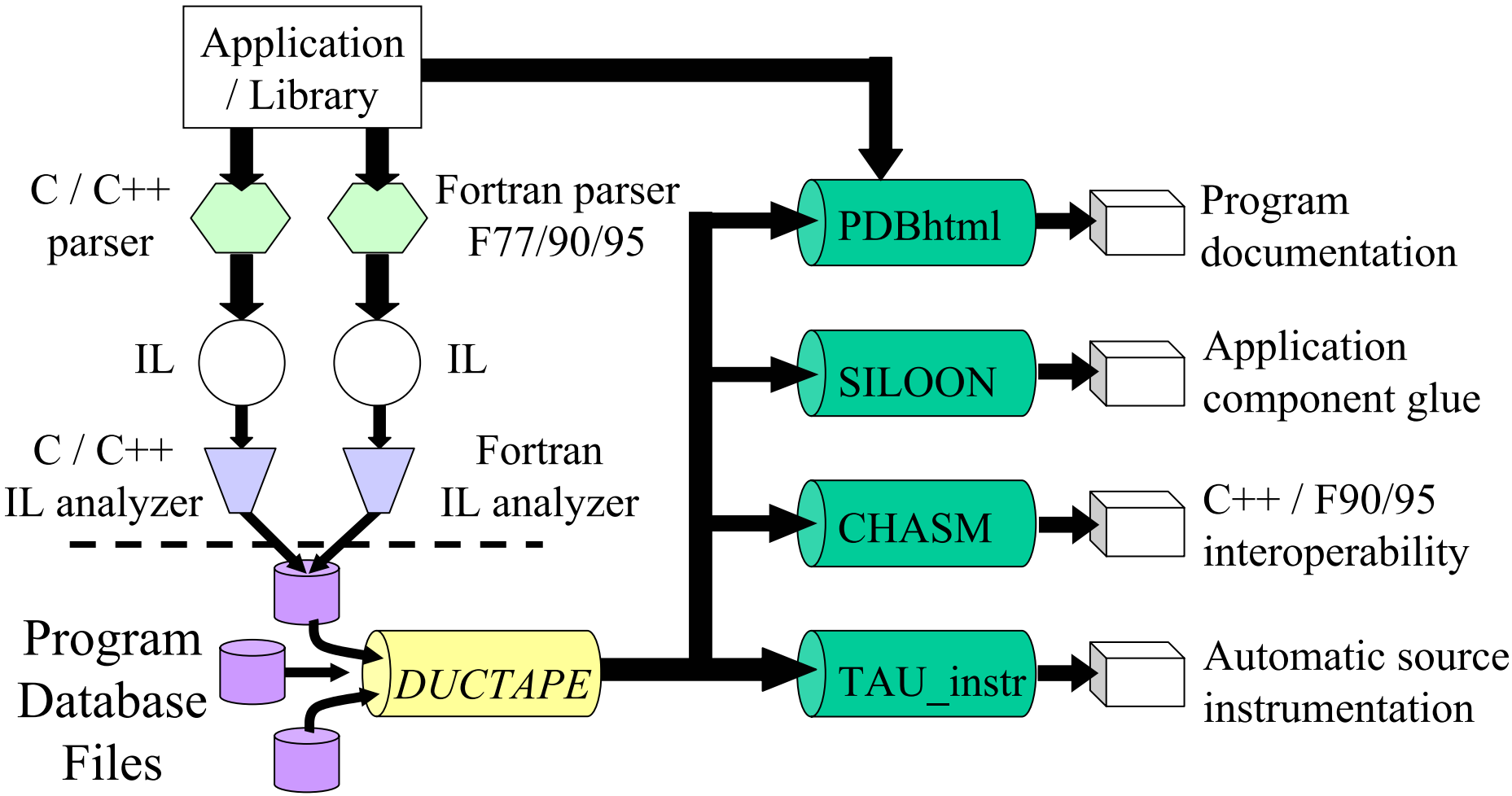
TAU Performance System Architecture



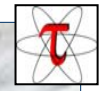
TAU Performance System Architecture



Program Database Toolkit (PDT)

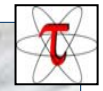


TAU Instrumentation Approach



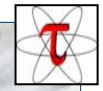
- Support for standard program events
  - Routines
  - Classes and templates
  - Statement-level blocks
- Support for user-defined events
  - Begin/End events (“user-defined timers”)
  - Atomic events (e.g., size of memory allocated/freed)
  - Selection of event statistics
- Support definition of “semantic” entities for mapping
- Support for event groups
- Instrumentation optimization (eliminate instrumentation in lightweight routines)

TAU Instrumentation



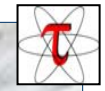
- Flexible instrumentation mechanisms at multiple levels
  - Source code
    - manual (TAU API, TAU Component API)
    - automatic
      - C, C++, F77/90/95 (Program Database Toolkit (*PDT*))
      - OpenMP (*tau_ompcheck*, directive rewriting (*Opari*), *POMP spec [KOJAK – UTK, FZJ Germany])*
  - Object code
    - pre-instrumented libraries (e.g., MPI using *PMPI*)
    - *LD_PRELOAD* tau_load.sh library preloading for MPI
  - Executable code
    - dynamic instrumentation (pre-execution, rewriting) (*DynInstAPI*) [U. Maryland, U. Wisconsin, Madison]
    - virtual machine instrumentation (e.g., Java using *JVMPI*)
  - Proxy Components

Using TAU – A tutorial



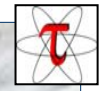
- Configuration
- Instrumentation
  - Manual
  - MPI – Wrapper interposition library
  - PDT- Source rewriting for C,C++, F77/90/95
  - OpenMP – Directive rewriting
  - Component based instrumentation – Proxy components
  - Binary Instrumentation
    - DyninstAPI – Runtime Instrumentation/Rewriting binary
    - Java – Runtime instrumentation
    - Python – Runtime instrumentation
- Measurement
- Performance Analysis

TAU Measurement System Configuration



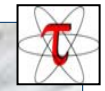
- configure [OPTIONS]
  - {-c++=<CC>, -cc=<cc>} Specify C++ and C compilers
  - {-pthread, -sproc} Use pthread or SGI sproc threads
  - -openmp Use OpenMP threads
  - -jdk=<dir> Specify Java instrumentation (JDK)
  - -opari=<dir> Specify location of Opari OpenMP tool
  - -papi=<dir> Specify location of PAPI
  - -pdt=<dir> Specify location of PDT
  - -dyninst=<dir> Specify location of DynInst Package
  - -mpi[inc/lib]=<dir> Specify MPI library instrumentation
  - -shmem[inc/lib]=<dir> Specify PSHMEM library instrumentation
  - -python[inc/lib]=<dir> Specify Python instrumentation
  - -epilog=<dir> Specify location of EPILOG
  - -slog2[=<dir>] Specify location of SLOG2/Jumpshot
  - -vtf=<dir> Specify location of VTF3 trace package
  - -arch=<architecture> Specify architecture explicitly (xt3,bgl…)

TAU Measurement System Configuration



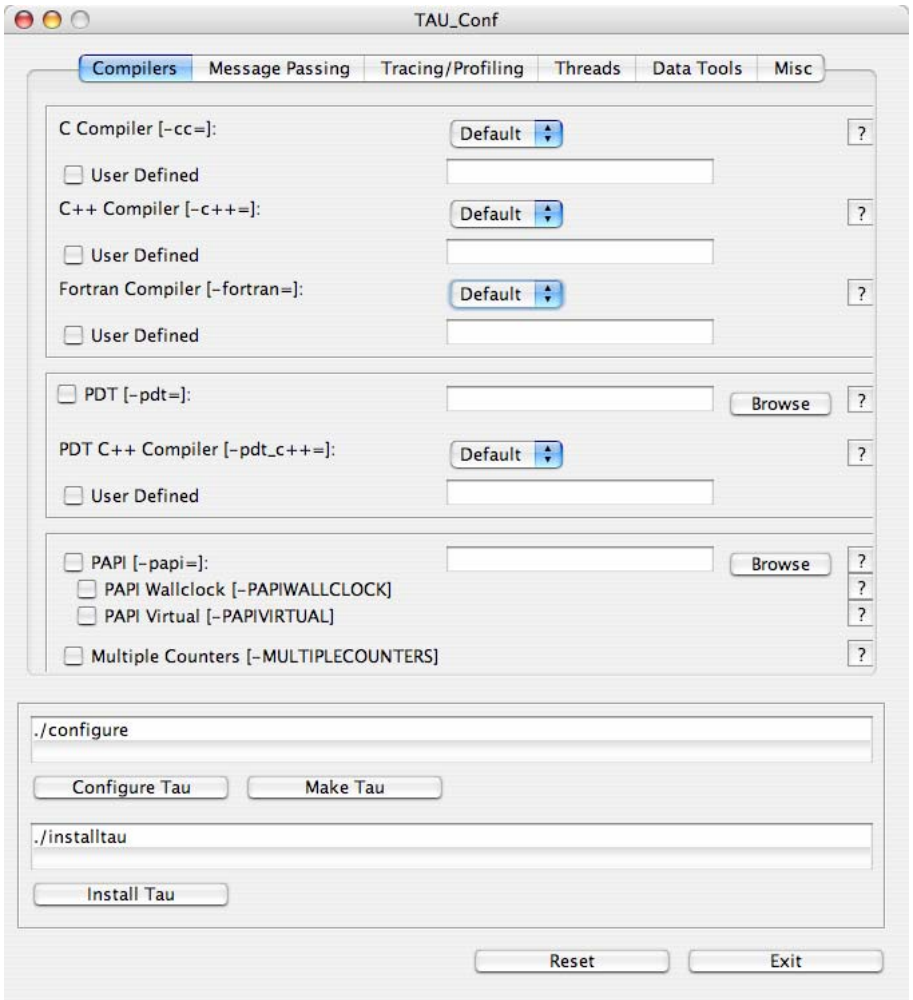
- configure [OPTIONS]
  - -TRACE — Generate binary TAU traces
  - -PROFILE (default) — Generate profiles (summary)
  - -PROFILECALLPATH — Generate call path profiles
  - -PROFILEPHASE — Generate phase based profiles
  - -PROFILEMEMORY — Track heap memory for each routine
  - -PROFILEHEADROOM — Track 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

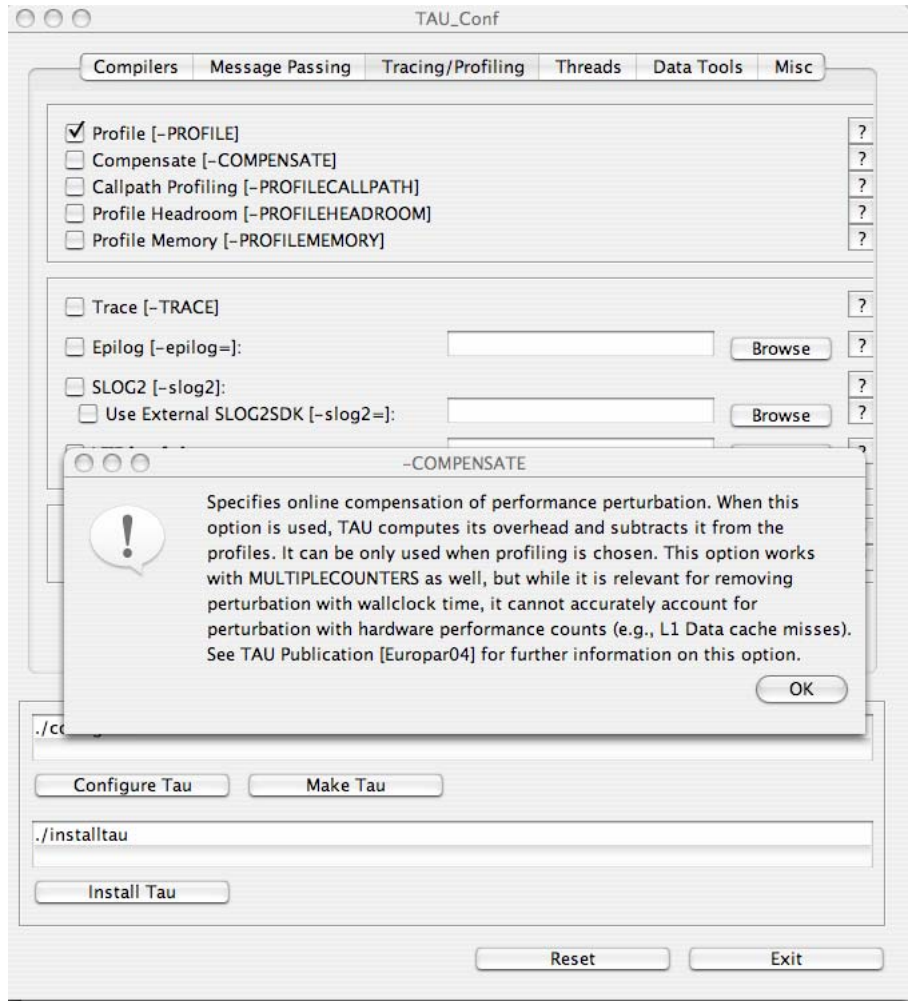
TAU Measurement Configuration – Examples



- ./configure –arch=xt3 –mpi –pdt=/usr/local/pdtoolkit-3.7 –pdt_c++=g++
  - Use TAU with MPI, PDT and use front-end compiler for compiling PDT based tau_instrumentor (that executes on the front-end)
  - Enable TAU profiling (default)
- ./configure
  - Build tools for the front-end (in x86_64/bin directory)
- ./configure –arch=xt3
  -papi=/usr/local/packages/papi-3.2.1
  -pdt=/usr/local/pdtoolkit-3.7 –arch=xt3
  -mpi -MULTIPLECOUNTERS
  - Use PAPI, PDT, MPI packages and multiple counters for measurements
- Typically configure multiple measurement libraries
- Each configuration creates a unique <arch>/lib/Makefile.tau-<options> stub makefile that corresponds to the configuration options specified. e.g.,
  - /usr/local/tau/tau-2.15.3/xt3/lib/Makefile.tau-mpi-pdt-pgi
  - /usr/local/tau/tau-2.15.3/xt3/lib/Makefile.tau-mpi-pdt-trace-pgi

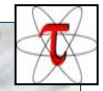
TAU_SETUP: A GUI for Installing TAU






```
tau-2.x>./tau_setup
```

Using TAU



- Install TAU

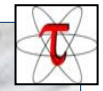
  % configure ; make clean install

- Typically modify application makefile
  - Change the name of compiler to tau_cxx.sh, tau_f90.sh
- Set environment variables
  - Name of the stub makefile: TAU_MAKEFILE
  - Options passed to tau_compiler.sh: TAU_OPTIONS
- Execute application

  % mpirun –np <procs> a.out;

- Analyze performance data
  - paraprof, vampir, vng, paraver, jumpshot …

Using Program Database Toolkit (PDT)



1. **Parse the Program to create foo.pdb:**

```
% cxxparse foo.cpp -I/usr/local/mydir -DMYFLAGS …
```

or

```
% cparse foo.c -I/usr/local/mydir -DMYFLAGS …
```

or

```
% f95parse foo.f90 -I/usr/local/mydir …
% f95parse *.f -omerged.pdb -I/usr/local/mydir -R free
```

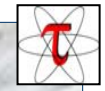
2. **Instrument the program:**

```
% tau_instrumentor foo.pdb   foo.f90 -o foo.inst.f90
    -f select.tau
```

3. **Compile the instrumented program:**

```
% ifort foo.inst.f90 -c -I/usr/local/mpi/include -o foo.o
```

Using TAU



Step 1: Configure and install TAU:

```
% configure -pdt=<dir> -mpiinc=<dir> -mpilib=<dir>
  -arch=xt3 -pdt_c++=g++
% make clean; make install
```

Builds <taudir>/<arch>/lib/Makefile.tau-<options>

```
% set path=($path <taudir>/<arch>/bin)
```

Step 2: Choose target stub Makefile

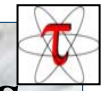
```
% setenv TAU_MAKEFILE
/san/cca/tau/tau-2.15.3/xt3/lib/Makefile.tau-mpi-pdt-pgi
% setenv TAU_OPTIONS '-optVerbose -optKeepFiles'
(see tau_compiler.sh for all options)
```

Step 3: Use tau_f90.sh, tau_cxx.sh and tau_cc.sh as the F90, C++ or C compilers respectively.

```
% tau_f90.sh -c app.f90
% tau_f90.sh app.o -o app -lm -lblas
```

Or use these in the application Makefile.

Tau_[cxx,cc,f90].sh – Improves Integration in Makefiles



```
# set TAU_MAKEFILE and TAU_OPTIONS env vars
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_[cxx,cc,f90].sh – Improves Integration in Makefiles

xterm

```
# Step 1: Set the compiler names
F90    = tau_f90.sh
CXX    = tau_cxx.sh
CC     = tau_cc.sh

# instead of mpif90, mpicxx, and mpicc

# Step 2: Set environment variables before invoking make:
# 2a) set TAU stub makefile name
# setenv TAU_MAKEFILE /usr/local/tau-2.15.3/xt3/lib/Makefile.tau-mpi-pdt-pgi
#
# 2b) and optional parameters to pass to tau_compiler.sh:
# setenv TAU_OPTIONS -optVerbose -optTauSelectFile=select.tau -optPdtGnuFortranParser
# See tau_compiler.sh  for a complete list of options!

# Use the same compilation rules
OBJS = f1.o f2.o c1.o c2.o
app: $(OBJS)
        $(F90) $(OBJS) -o app $(LIBS)
.f90.o:
        $(F90) -c $(INCLUDE) $< -o $@
.cpp.o:
        $(CXX) -c $(INCLUDE) $< -o $@
clean:
        /bin/rm -f $(OBJS) app
"Makefile" 171L, 4335C written                                  25,6          0%
```

TAU_COMPILER Options

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

-optVerbose	Turn on verbose debugging messages
-optPreProcess	Invoke cpp for pre-processing Fortran sources prior to instrumentation
-optPdtF95Opts=""	Options for Fortran parser in PDT (f95parse)
-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)
-optPdtF90Parser=""	Specify a different Fortran parser. For e.g., f90parse instead of f95parse
-optPdtGnuFortranParser	Specify the use of the GNU gfortran parser (gfparse instead of f95parse)
-optPDBFile=""	Specify [merged] PDB file. Skips parsing phase.
-optTauInstr=""	Specify location of tau_instrumentor. Typically $(TAUROOT)/$(CONFIG_ARCH)/bin/tau_instrumentor
-optTauSelectFile=""	Specify selective instrumentation file for tau_instrumentor
-optTau=""	Specify options for tau_instrumentor
-optCompile=""	Options passed to the compiler. Typically $(TAU_MPI_INCLUDE) $(TAU_INCLUDE) $(TAU_DEFS)
-optLinking=""	Options passed to the linker. Typically $(TAU_MPI_FLIBS) $(TAU_LIBS) $(TAU_CXXLIBS)
-optNoMpi	Removes -l*mpi* libraries during linking (default)
-optKeepFiles	Does not remove intermediate .pdb and .inst.* files

e.g.,

```
% setenv TAU_OPTIONS '-optTauSelectFile=select.tau
   -optVerbose -optPdtGnuFortranParser '
% tau_f90.sh matrix.f90 -o matrix -lm
```

Using Stub Makefile and TAU_COMPILER

```
include /usr/common/acts/TAU/tau-2.15.3/xt3/lib/
                                        Makefile.tau-mpi-pdt-pgi

MYOPTIONS= -optVerbose -optKeepFiles -optTauSelectFile=s.tau

F90 = $(TAU_COMPILER) $(MYOPTIONS) mpif90
OBJS = f1.o f2.o f3.o …
LIBS = -Lappdir -lapplib1 -lapplib2 …

app: $(OBJS)
    $(F90) $(OBJS) -o app $(LIBS)

.f90.o:
    $(F90) -c $<
```

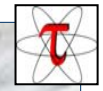
Instrumentation Specification

```
% tau_instrumentor
Usage : tau_instrumentor <pdbfile> <sourcefile> [-o <outputfile>] [-noinline]
[-g groupname] [-i headerfile] [-c|-c++|-fortran] [-f <instr_req_file> ]
For selective instrumentation, use -f option
% tau_instrumentor foo.pdb foo.cpp -o foo.inst.cpp -f selective.dat
% cat selective.dat
# Selective instrumentation: Specify an exclude/include list of routines/files.

BEGIN_EXCLUDE_LIST
void quicksort(int *, int, int)
void sort_5elements(int *)
void interchange(int *, int *)
END_EXCLUDE_LIST

BEGIN_FILE_INCLUDE_LIST
Main.cpp
Foo?.c
*.C
END_FILE_INCLUDE_LIST
# Instruments routines in Main.cpp, Foo?.c and *.C files only
# Use BEGIN_[FILE]_INCLUDE_LIST with END_[FILE]_INCLUDE_LIST
```

Automatic Outer Loop Level Instrumentation

```
BEGIN_INSTRUMENT_SECTION
loops file="loop_test.cpp" routine="multiply"
# comment. TAU accepts # as a wildcard in routine name
# and * and ? wildcards in file name.
# You can also specify the full
# name of the routine as is found in profile files.
#loops file="loop_test.cpp" routine="double multiply#"
#loops file="*.f90" routine = "#"
END_INSTRUMENT_SECTION

% pprof
NODE 0;CONTEXT 0;THREAD 0:
---------------------------------------------------------------------------------------
%Time    Exclusive    Inclusive       #Call      #Subrs  Inclusive Name
              msec   total msec                          usec/call
---------------------------------------------------------------------------------------
100.0         0.12       25,162           1           1   25162827 int main(int, char **)
100.0        0.175       25,162           1           4   25162707 double multiply()
 90.5       22,778       22,778           1           0   22778959 Loop: double multiply()[
file = <loop_test.cpp> line,col = <23,3> to <30,3> ]
  9.3        2,345        2,345           1           0    2345823 Loop: double multiply()[
file = <loop_test.cpp> line,col = <38,3> to <46,7> ]
  0.1           33           33           1           0      33964 Loop: double
multiply()[ file = <loop_test.cpp> line,col = <16,10> to <21,12> ]
```

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 10000 times (TAU_THROTTLE_NUMCALLS) and take less than 10 microseconds of inclusive time per call (TAU_THROTTLE_PERCALL)

- Selective instrumentation file to filter events

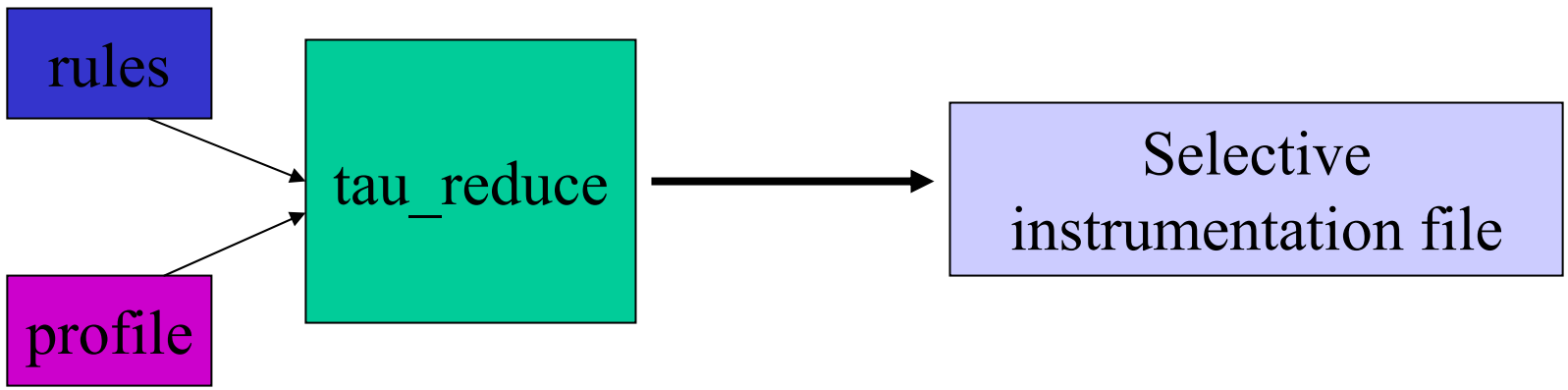
```
% tau_instrumentor [options] -f <file>
```

- Compensation of local instrumentation overhead

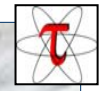
```
% configure -COMPENSATE
```

TAU_REDUCE

- Reads profile files and rules
- Creates selective instrumentation file
  - Specifies which routines should be excluded from instrumentation

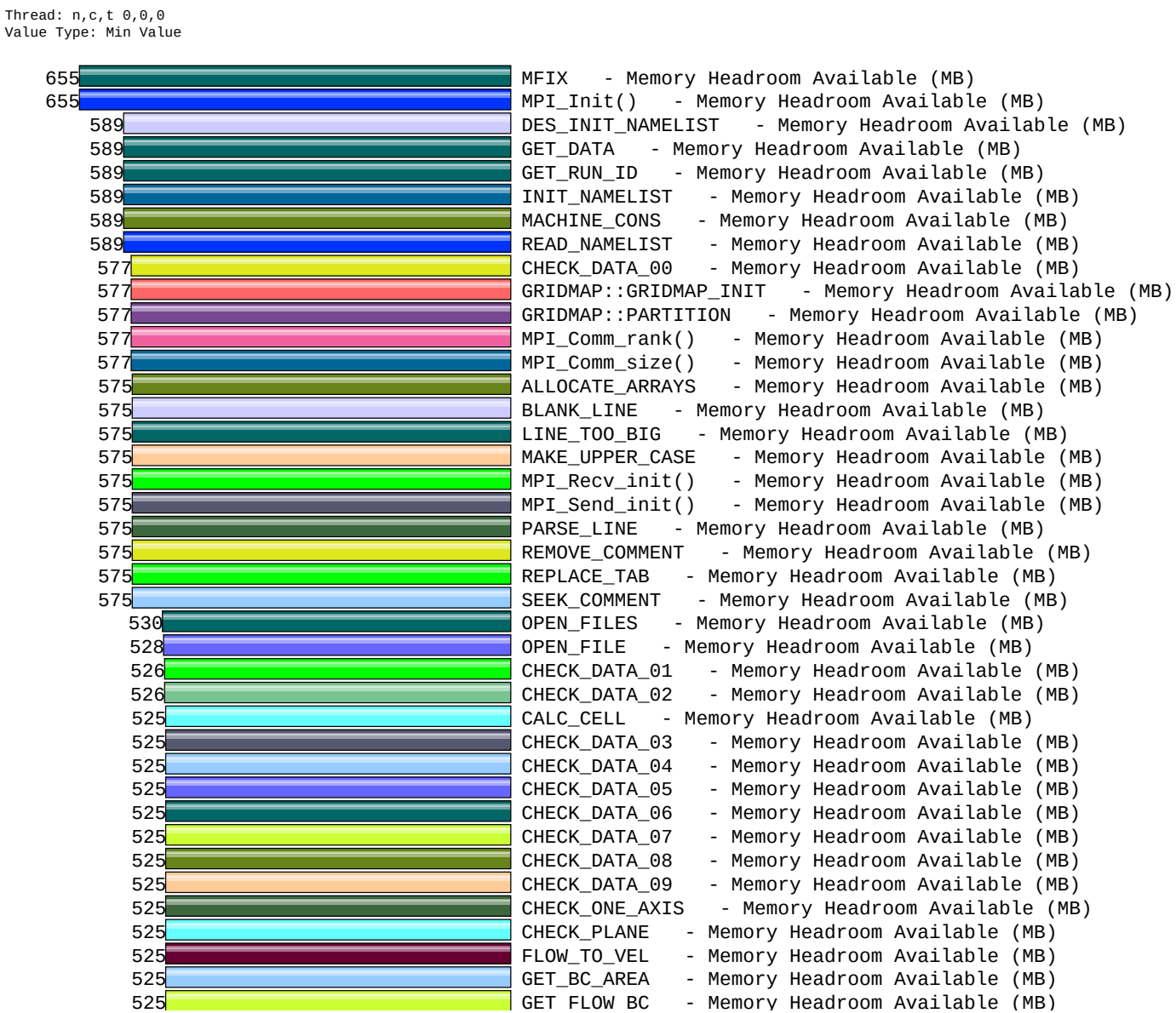


Memory Profiling in TAU



- Configuration option –PROFILEMEMORY
  - Records global heap memory utilization for each function
  - Takes one sample at beginning of each function and associates the sample with the 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 the sample with the function name
  - Useful for debugging memory usage on Cray XT3
- On Cray XT3, TAU uses the heap_info( ) call for memory headroom and heap memory utilization information
- On other platforms, TAU uses mallinfo for heap memory utilization, and
- TAU allocates a series of memory blocks (1, 2, 4, 8…512 MB) and when there is no memory left, it starts again (1, 2, 4…) until it can no longer allocate the smallest block. It records memory available and frees the blocks to compute memory headroom
- No need for modifications to the source code for using these options

ParaProf– Flat Profile (MFIX)



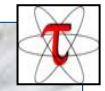
pprof – Memory Events on Cray XT3 (MFIX)

xterm

USER EVENTS Profile :NODE 0, CONTEXT 0, THREAD 0

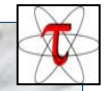
NumSamples	MaxValue	MinValue	MeanValue	Std. Dev.	Event Name
1	655	655	655	0	MFIX - Memory Headroom Available (MB)
486	579	575	576.9	1.04	BLANK_LINE - Memory Headroom Available (MB)
1	577	577	577	0	CHECK_DATA_00 - Memory Headroom Available (MB)
1405	503	502	503	0.1398	ACCUMULATION - Memory Headroom Available (MB)
300	503	502	503	0.14	ADJUST_A_U_G - Memory Headroom Available (MB)
300	503	502	503	0.14	ADJUST_A_U_S - Memory Headroom Available (MB)
2400	503	502	503	0.14	BOUND_X - Memory Headroom Available (MB)
174	525	525	525	0	CALC_CELL - Memory Headroom Available (MB)
1.408E+05	525	468	500.1	1.034	MPI_Startall() - Memory Headroom Available (MB)
21	525	491	523.4	7.241	MPI_UTILITY::BCAST_0C - Memory Headroom Available (MB)
42	525	489	523.3	7.457	MPI_UTILITY::BCAST_0I - Memory Headroom Available (MB)
1	491	491	491	0	MPI_UTILITY::BCAST_0R - Memory Headroom Available (MB)
2	509	496	502.5	6.5	MPI_UTILITY::GATHER_1C - Memory Headroom Available (MB)
128	491	441	459.3	12.39	MPI_UTILITY::GATHER_1D - Memory Headroom Available (MB)
11	508	461	479.5	14.33	MPI_UTILITY::GATHER_1I - Memory Headroom Available (MB)
1	526	526	526	0	CHECK_DATA_01 - Memory Headroom Available (MB)
1	526	526	526	0	CHECK_DATA_02 - Memory Headroom Available (MB)
1	502	502	502	0	ITERATE 1 - Memory Headroom Available (MB)
1	503	503	503	0	ITERATE 2 - Memory Headroom Available (MB)
1	503	503	503	0	ITERATE 3 - Memory Headroom Available (MB)
1	503	503	503	0	ITERATE 4 - Memory Headroom Available (MB)

TAU's High Resolution Timers -CRAYTIMERS



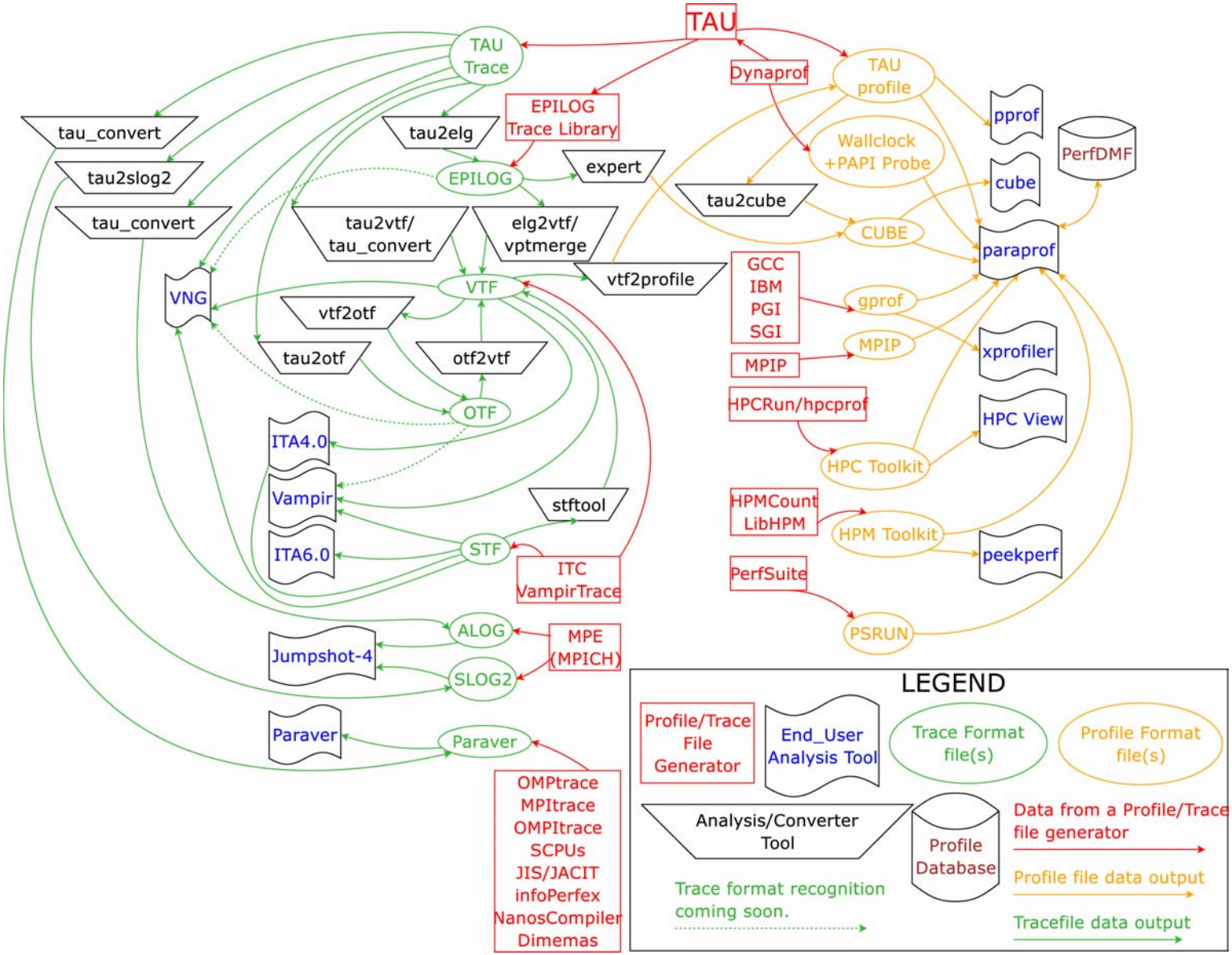
- Instead of the default gettimeofday(), it uses low-overhead timers
- On Cray XD1 (Linux x86_64), TAU uses TSC (time stamp counter) registers to access wallclock time
- On Cray X1E, _rtc( ) call is used
- On Cray XT3, dclock( ) call is used
  - **setenv TAU_MAKEFILE <dir>/xt3/lib/Makefile.tau-craytimers-mpi-pdt-pgi**
  - Use tau_cxx.sh, tau_f90.sh and tau_cc.sh as compilers

TAU's MPI Wrapper Interposition Library

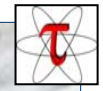


- Uses standard MPI Profiling Interface
  - Provides name shifted interface
    - MPI_Send = PMPI_Send
    - Weak bindings
- Interpose TAU's MPI wrapper library between MPI and TAU
  - -lmpi replaced by –lTauMpi –lpmpi –lmpi
- No change to the source code! Just re-link the application to generate performance data. Similar to SHMEM/PSHMEM
  - `setenv TAU_MAKEFILE <dir>/<arch>/lib/Makefile.tau-mpi-[options]`
  - Use tau_cxx.sh, tau_f90.sh and tau_cc.sh as compilers

Building Bridges to Other Tools: TAU

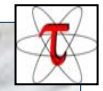


TAU Performance System Interfaces



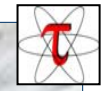
- PDT [U. Oregon, LANL, FZJ] for instrumentation of C++, C99, F95 source code
- PAPI [UTK] & PCL[FZJ] for accessing hardware performance counters data
- DyninstAPI [U. Maryland, U. Wisconsin] for runtime instrumentation
- KOJAK [FZJ, UTK]
  - Epilog trace generation library
  - CUBE callgraph visualizer
  - Opari OpenMP directive rewriting tool
- Vampir/Intel® Trace Analyzer [Pallas/Intel]
- VTF3 trace generation library for Vampir [TU Dresden] (available from TAU website)
- Paraver trace visualizer [CEPBA]
- Jumpshot-4 trace visualizer [MPICH, ANL]
- JVMPI from JDK for Java program instrumentation [Sun]
- Paraprof profile browser/PerfDMF database supports:
  - TAU format
  - Gprof [GNU]
  - HPM Toolkit [IBM]
  - MpiP [ORNL, LLNL]
  - Dynaprof [UTK]
  - PSRun [NCSA]
- PerfDMF database can use Oracle, MySQL or PostgreSQL (IBM DB2 support planned)

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

TAU Timers and Phases



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

Static Timers in TAU

```
SUBROUTINE SUM_OF_CUBES
 integer profiler(2)
 save profiler
INTEGER :: H, T, U

  call TAU_PROFILE_TIMER(profiler, 'SUM_OF_CUBES')
  call TAU_PROFILE_START(profiler)

! This program prints all 3-digit numbers that
! equal the sum of the cubes of their digits.
DO H = 1, 9
  DO T = 0, 9
    DO U = 0, 9
    IF (100*H + 10*T + U == H**3 + T**3 + U**3) THEN
       PRINT "(3I1)", H, T, U
    ENDIF
    END DO
  END DO
END DO
call TAU_PROFILE_STOP(profiler)
END SUBROUTINE SUM_OF_CUBES
```

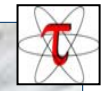
Static Phases and Timers

```
      SUBROUTINE FOO
       integer profiler(2)
       save profiler

        call TAU_PHASE_CREATE_STATIC(profiler, 'foo')
        call TAU_PHASE_START(profiler)
           call bar()
! Here bar calls MPI_Barrier and we evaluate foo=>MPI_Barrier and foo=>bar
        call TAU_PHASE_STOP(profiler)
      END SUBROUTINE SUM_OF_CUBES

      SUBROUTINE BAR
       integer profiler(2)
       save profiler
       call TAU_PROFILE_TIMER(profiler, 'bar')
       call TAU_PROFILE_START(profiler)
         call MPI_Barrier()
       call TAU_PROFILE_STOP(profiler)
      END SUBROUTINE BAR
```

Dynamic Phases

```
      SUBROUTINE ITERATE(IER, NIT)
      IMPLICIT NONE
      INTEGER IER, NIT
    character(11) taucharary
      integer tauiteration / 0 /
      integer profiler(2) / 0, 0 /
      save profiler, tauiteration

      write (taucharary, '(a8,i3)') 'ITERATE ', tauiteration
! Taucharary is the name of the phase e.g.,'ITERATION 23'
      tauiteration = tauiteration + 1

    call TAU_PHASE_CREATE_DYNAMIC(profiler,taucharary)
      call TAU_PHASE_START(profiler)

    IER = 0
    call SOLVE_K_EPSILON_EQ(IER)
! Other work

      call TAU_PHASE_STOP(profiler)
```

TAU's ParaProf Profile Browser: Static Timers



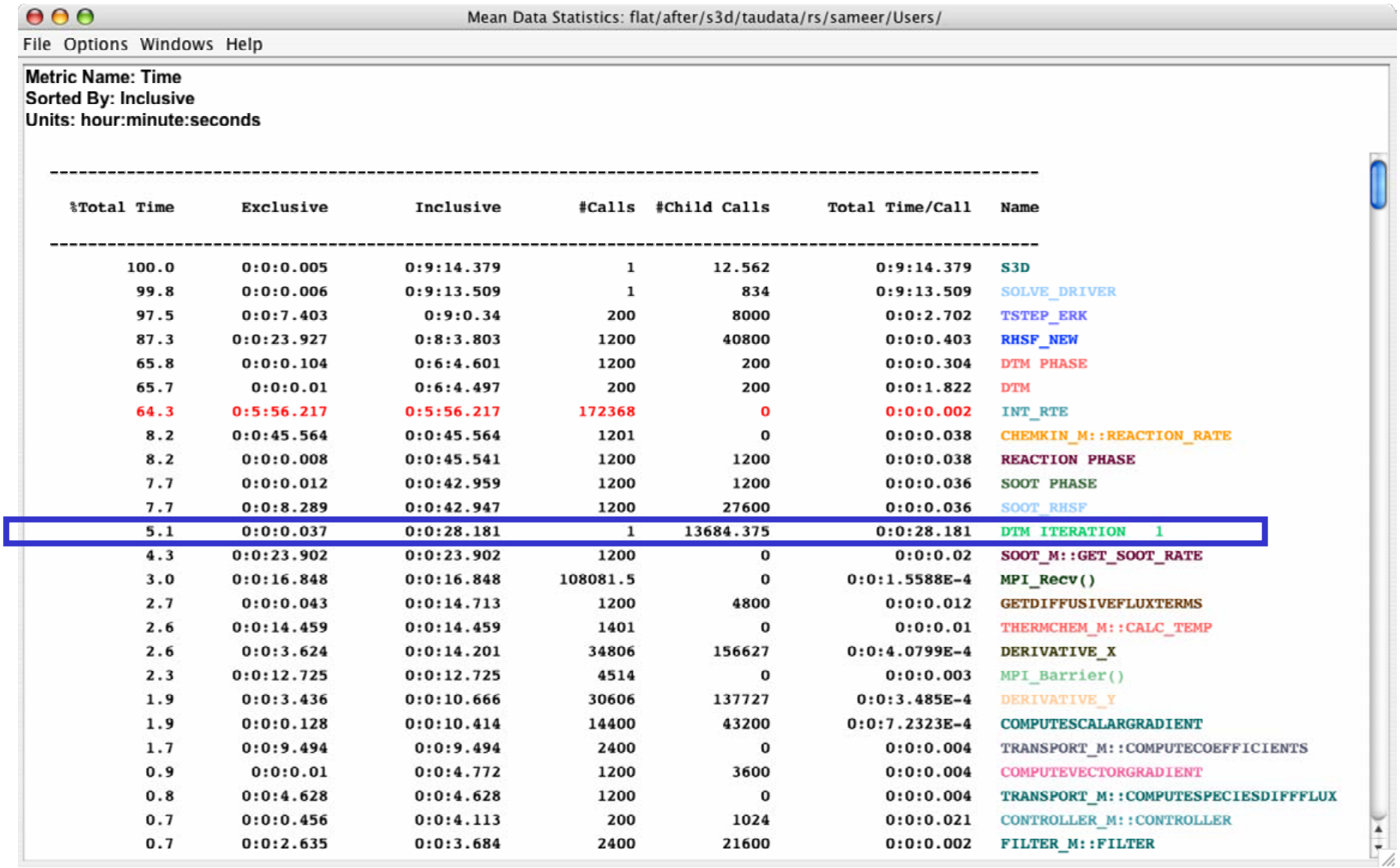
Mean Data Statistics: 16pAIX200iter/s3d/taudata/rs/sameer/Users/

File Options Windows Help

Metric Name: Time
Sorted By: Exclusive
Units: seconds

%Total Time	Exclusive	Inclusive	#Calls	#Child Calls	Total Time/Call	Name
81.5	2025.003	2025.003	969969	0	0.002	INT_RTE
6.0	148.335	148.335	11511	0	0.013	MPI_Barrier()
2.1	52.692	52.692	124265.5	0	4.2403E-4	MPI_Recv()
2.0	49.561	49.561	1201	0	0.041	CHEMKIN_M::REACTION_RATE
1.3	33.289	33.289	1200	0	0.028	SOOT_M::GET_SOOT_RATE
94.6	31.215	2349.911	1200	40800	1.958	RHSF_NEW
0.6	15.683	15.683	1401	0	0.011	THERMCHEM_M::CALC_TEMP
0.6	15.57	15.57	9884	0	0.002	MPI_Allreduce()
0.6	14.482	14.482	2	0	7.241	READWRITE_SAVEFILE_DATA
2.6	11.066	65.491	1200	27600	0.055	SOOT_RHSF
98.1	10.295	2436.084	200	8000	12.18	TSTEP_ERK
0.4	10.113	10.113	2400	0	0.004	TRANSPORT_M::COMPUTECOEFFICIENTS
0.3	7.163	7.163	124253	0	5.7647E-5	MPI_Wait()
88.2	5.442	2191.51	1200	995712	1.826	DTM
0.5	5.214	12.598	3400	30600	0.004	FILTER_M::FILTER
0.2	4.912	4.912	1200	0	0.004	TRANSPORT_M::COMPUTESPECIESDIFFFLUX
0.2	4.884	5.969	1200	7425	0.005	RADIATION_M::GET_ABSORPTION
1.2	4.635	29	34806	156627	8.3319E-4	DERIVATIVE_X
1.0	4.226	25.686	30606	137727	8.3923E-4	DERIVATIVE_Y
0.1	3.669	3.669	1200	0	0.003	TRANSPORT_M::COMPUTEHEATFLUX
0.1	3.382	3.382	124265.5	0	2.7216E-5	MPI_Isend()

Dynamic Timers



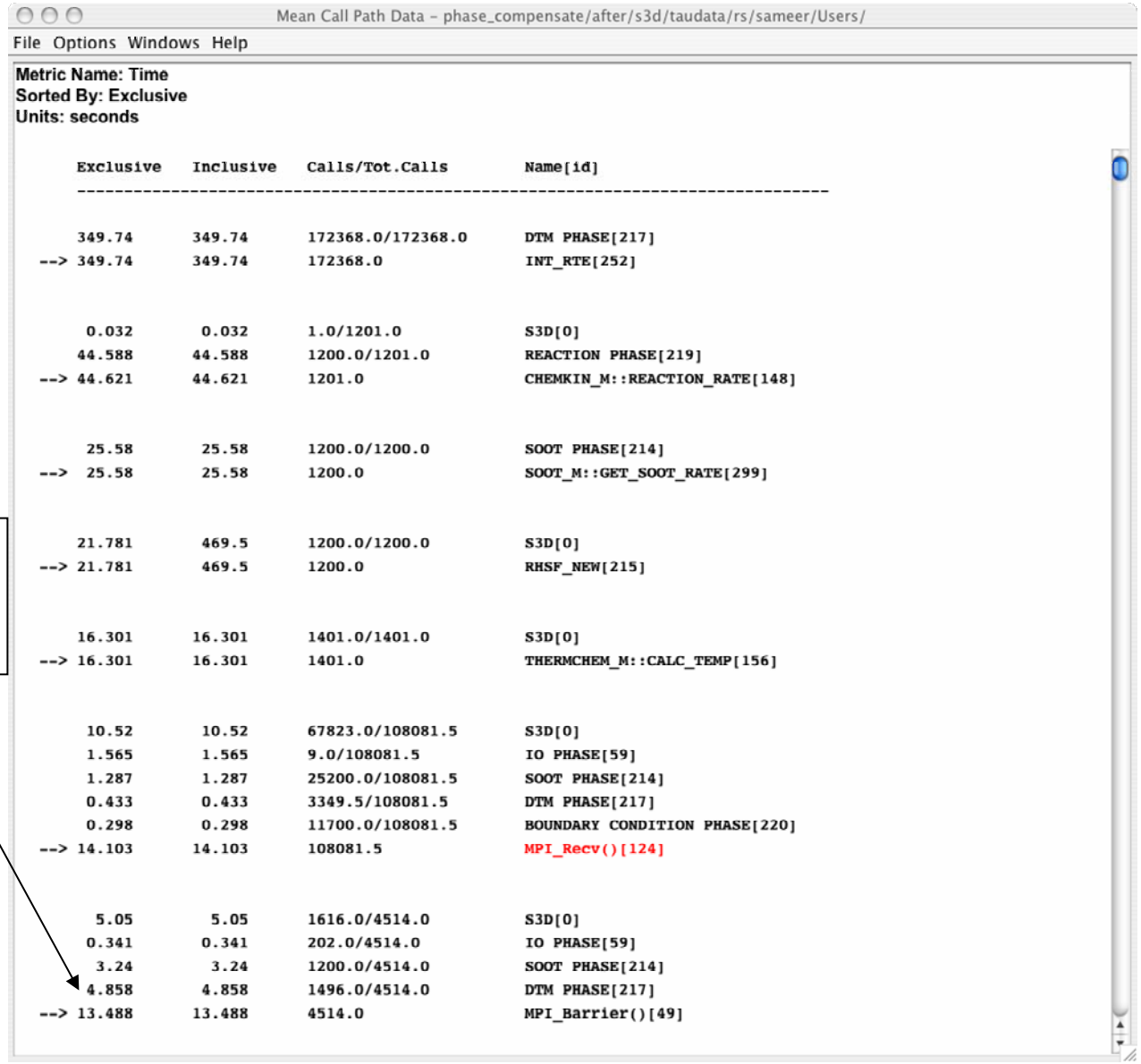
Mean Data Statistics: flat/after/s3d/taudata/rs/sameer/Users/

File Options Windows Help

Metric Name: Time
Sorted By: Inclusive
Units: hour:minute:seconds

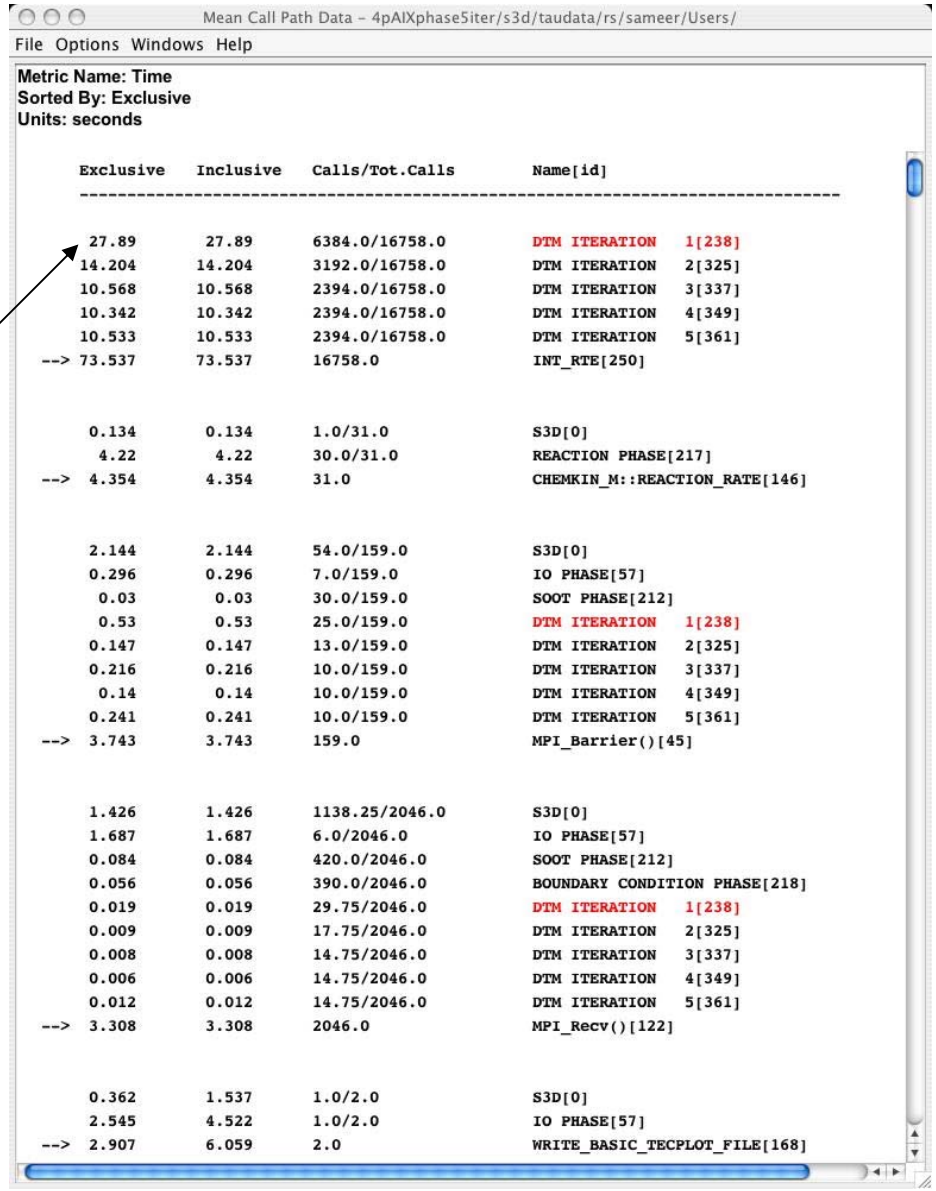
%Total Time	Exclusive	Inclusive	#Calls	#Child Calls	Total Time/Call	Name
100.0	0:0:0.005	0:9:14.379	1	12.562	0:9:14.379	S3D
99.8	0:0:0.006	0:9:13.509	1	834	0:9:13.509	SOLVE_DRIVER
97.5	0:0:7.403	0:9:0.34	200	8000	0:0:2.702	TSTEP_ERK
87.3	0:0:23.927	0:8:3.803	1200	40800	0:0:0.403	RHSF_NEW
65.8	0:0:0.104	0:6:4.601	1200	200	0:0:0.304	DTM PHASE
65.7	0:0:0.01	0:6:4.497	200	200	0:0:1.822	DTM
64.3	0:5:56.217	0:5:56.217	172368	0	0:0:0.002	INT_RTE
8.2	0:0:45.564	0:0:45.564	1201	0	0:0:0.038	CHEMKIN_M::REACTION_RATE
8.2	0:0:0.008	0:0:45.541	1200	1200	0:0:0.038	REACTION PHASE
7.7	0:0:0.012	0:0:42.959	1200	1200	0:0:0.036	SOOT PHASE
7.7	0:0:8.289	0:0:42.947	1200	27600	0:0:0.036	SOOT_RHSF
5.1	0:0:0.037	0:0:28.181	1	13684.375	0:0:28.181	DTM ITERATION 1
4.3	0:0:23.902	0:0:23.902	1200	0	0:0:0.02	SOOT_M::GET_SOOT_RATE
3.0	0:0:16.848	0:0:16.848	108081.5	0	0:0:1.5588E-4	MPI_Recv()
2.7	0:0:0.043	0:0:14.713	1200	4800	0:0:0.012	GETDIFFUSIVEFLUXTERMS
2.6	0:0:14.459	0:0:14.459	1401	0	0:0:0.01	THERMCHEM_M::CALC_TEMP
2.6	0:0:3.624	0:0:14.201	34806	156627	0:0:4.0799E-4	DERIVATIVE_X
2.3	0:0:12.725	0:0:12.725	4514	0	0:0:0.003	MPI_Barrier()
1.9	0:0:3.436	0:0:10.666	30606	137727	0:0:3.485E-4	DERIVATIVE_Y
1.9	0:0:0.128	0:0:10.414	14400	43200	0:0:7.2323E-4	COMPUTESCALARGRADIENT
1.7	0:0:9.494	0:0:9.494	2400	0	0:0:0.004	TRANSPORT_M::COMPUTECOEFFICIENTS
0.9	0:0:0.01	0:0:4.772	1200	3600	0:0:0.004	COMPUTEVECTORGRADIENT
0.8	0:0:4.628	0:0:4.628	1200	0	0:0:0.004	TRANSPORT_M::COMPUTESPECIESDIFFFLUX
0.7	0:0:0.456	0:0:4.113	200	1024	0:0:0.021	CONTROLLER_M::CONTROLLER
0.7	0:0:2.635	0:0:3.684	2400	21600	0:0:0.002	FILTER_M::FILTER

Static Phases



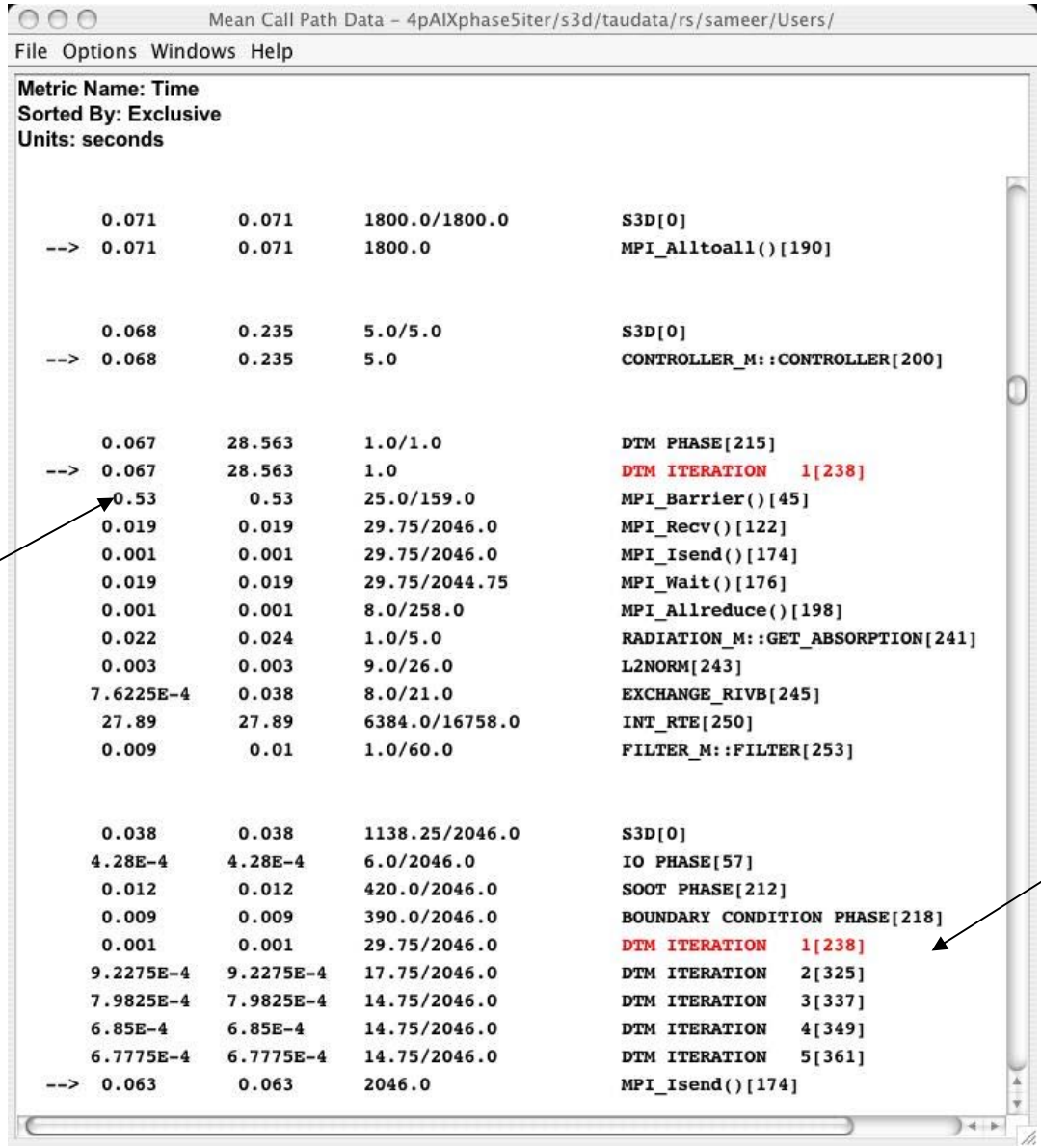
MPI_Barrier took 4.85 secs out of 13.48 secs in the DTM Phase

Dynamic Phases



The first iteration was expensive for INT_RTE. It took 27.89 secs. Other iterations took less time – 14.2, 10.5, 10.3, 10.5 seconds

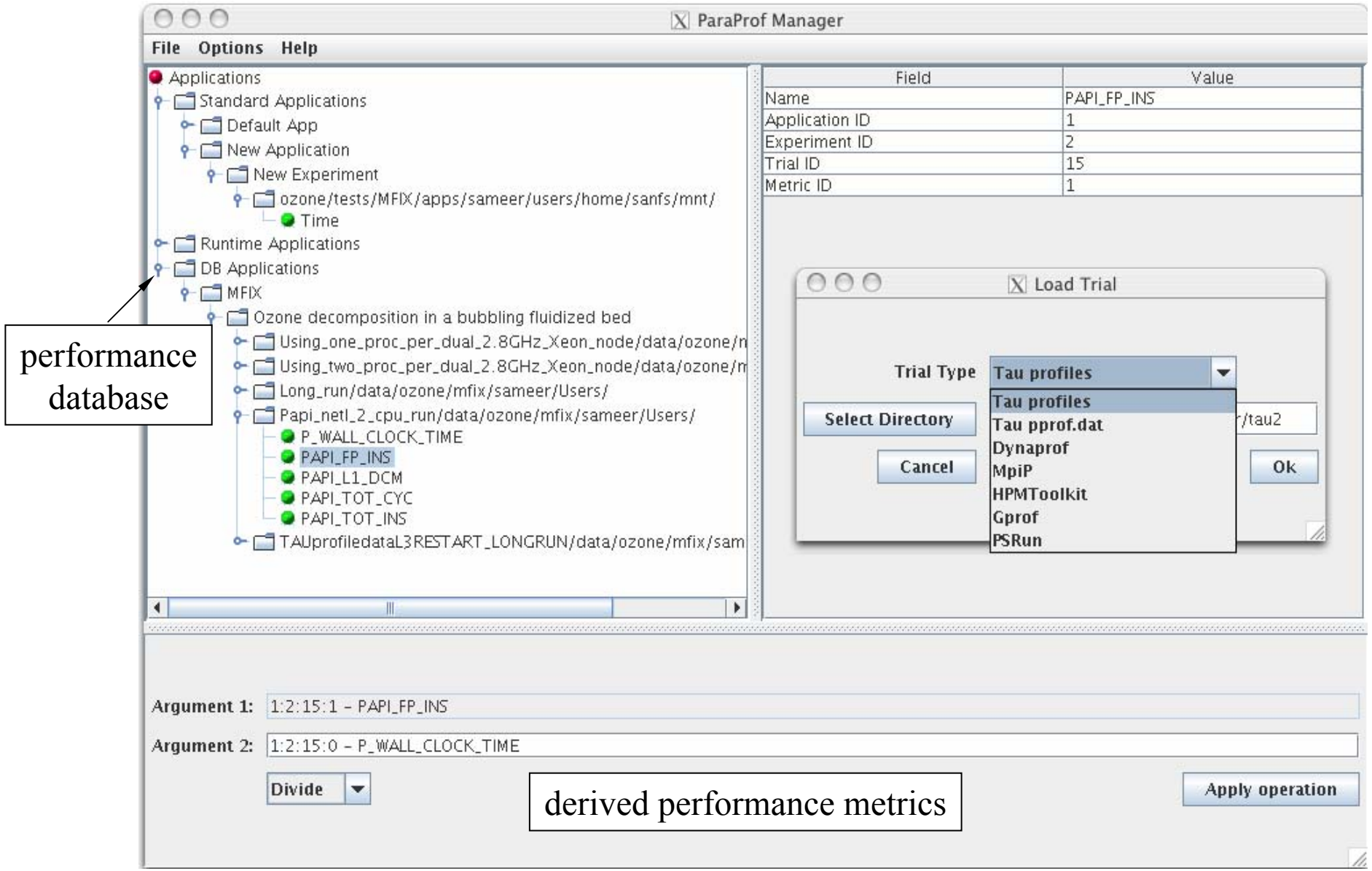
Dynamic Phases



Time spent in MPI_Barrier, MPI_Recv,… in DTM ITERATION 1

Breakdown of time spent in MPI_Isend based on its static and dynamic parent phases

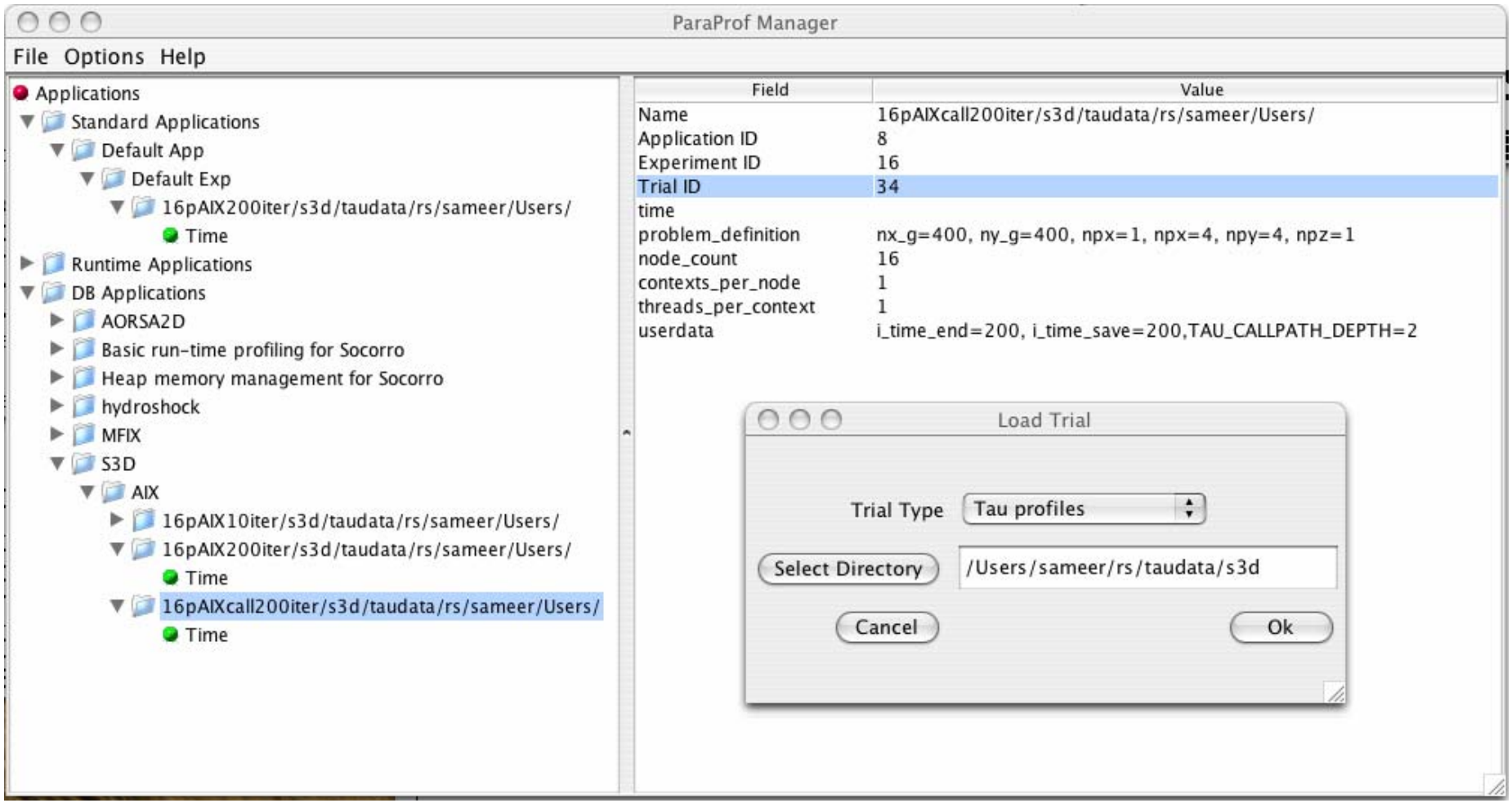
ParaProf – Manager Window



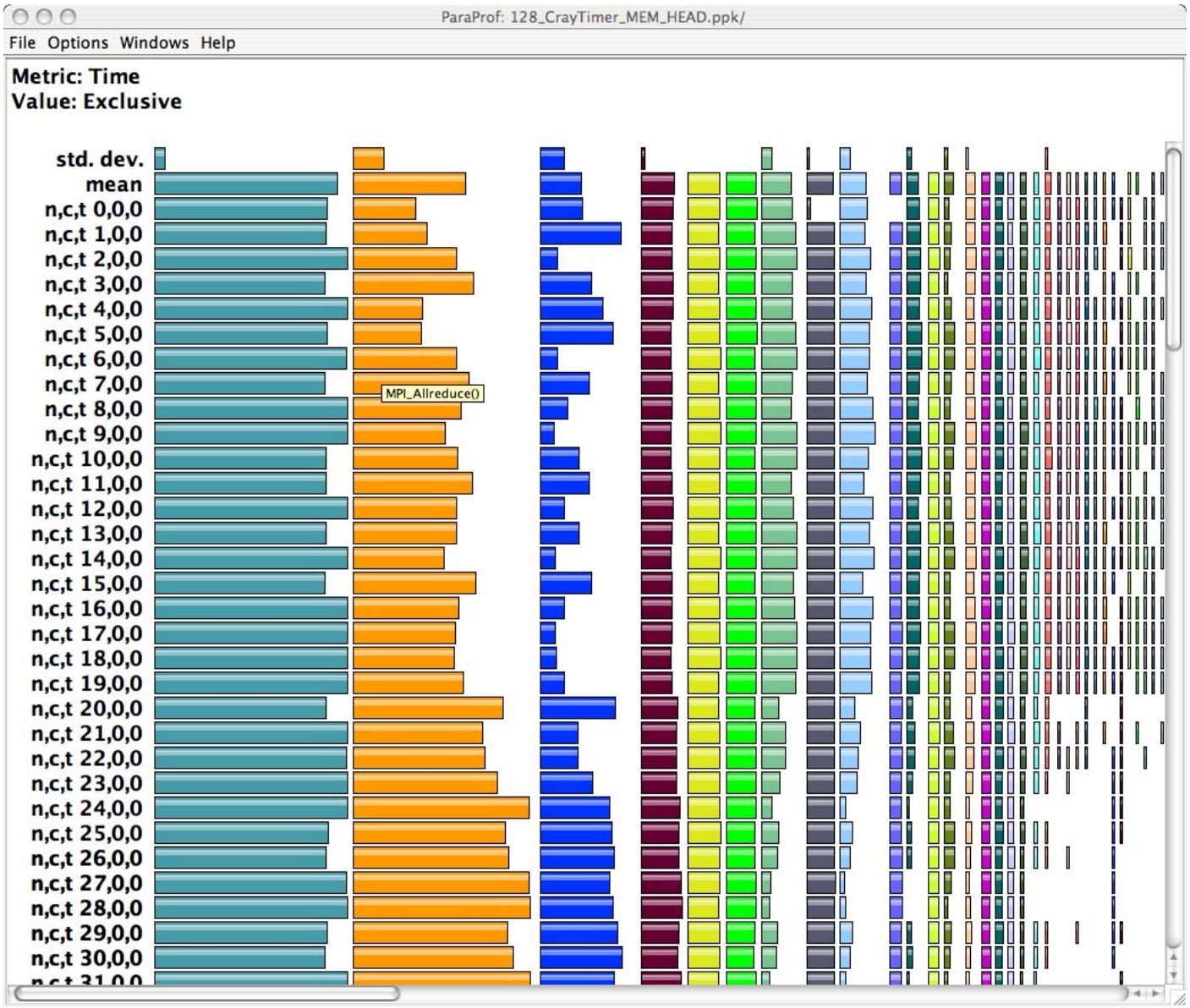
performance database

derived performance metrics

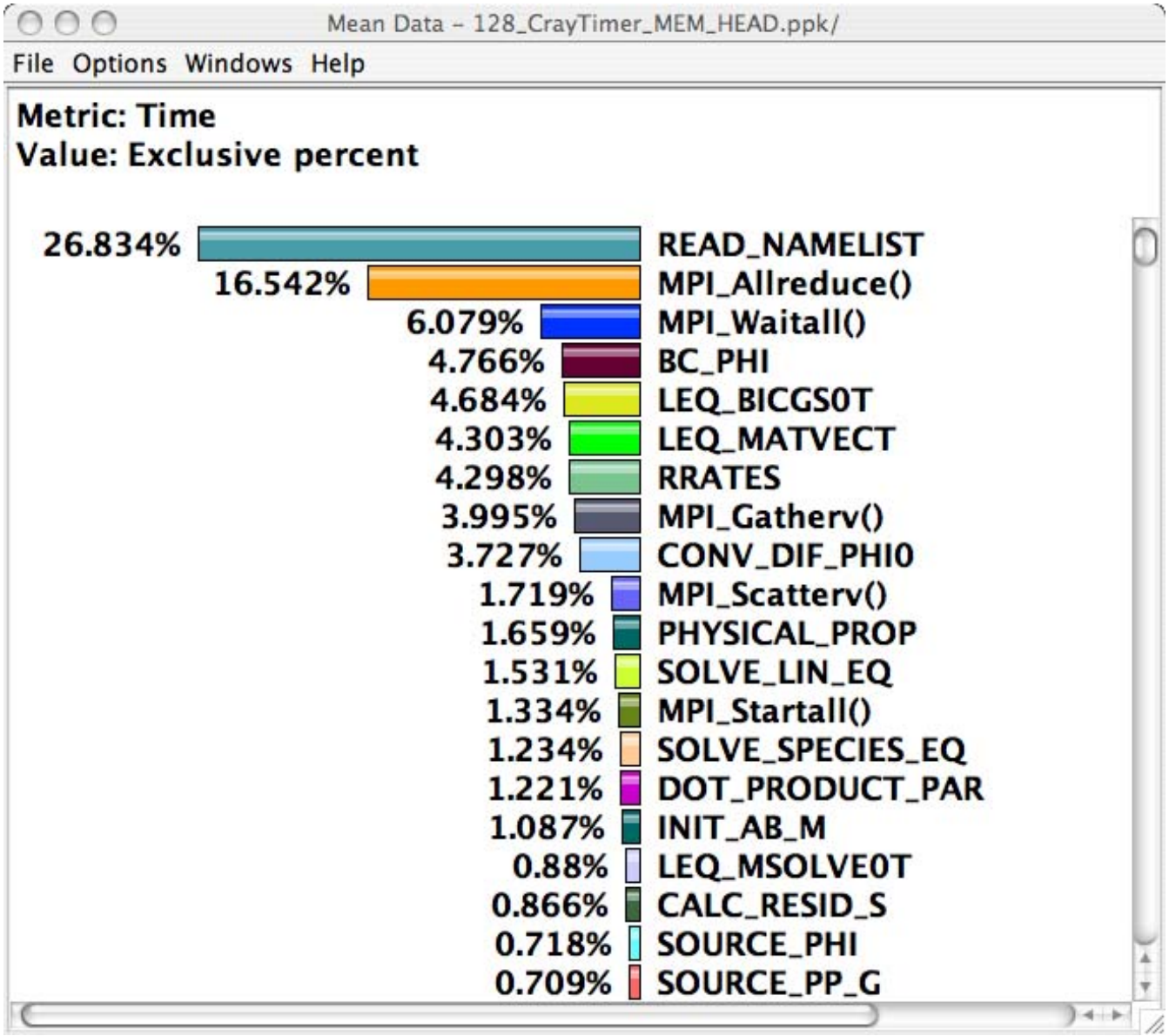
Performance Database: Storage of MetaData



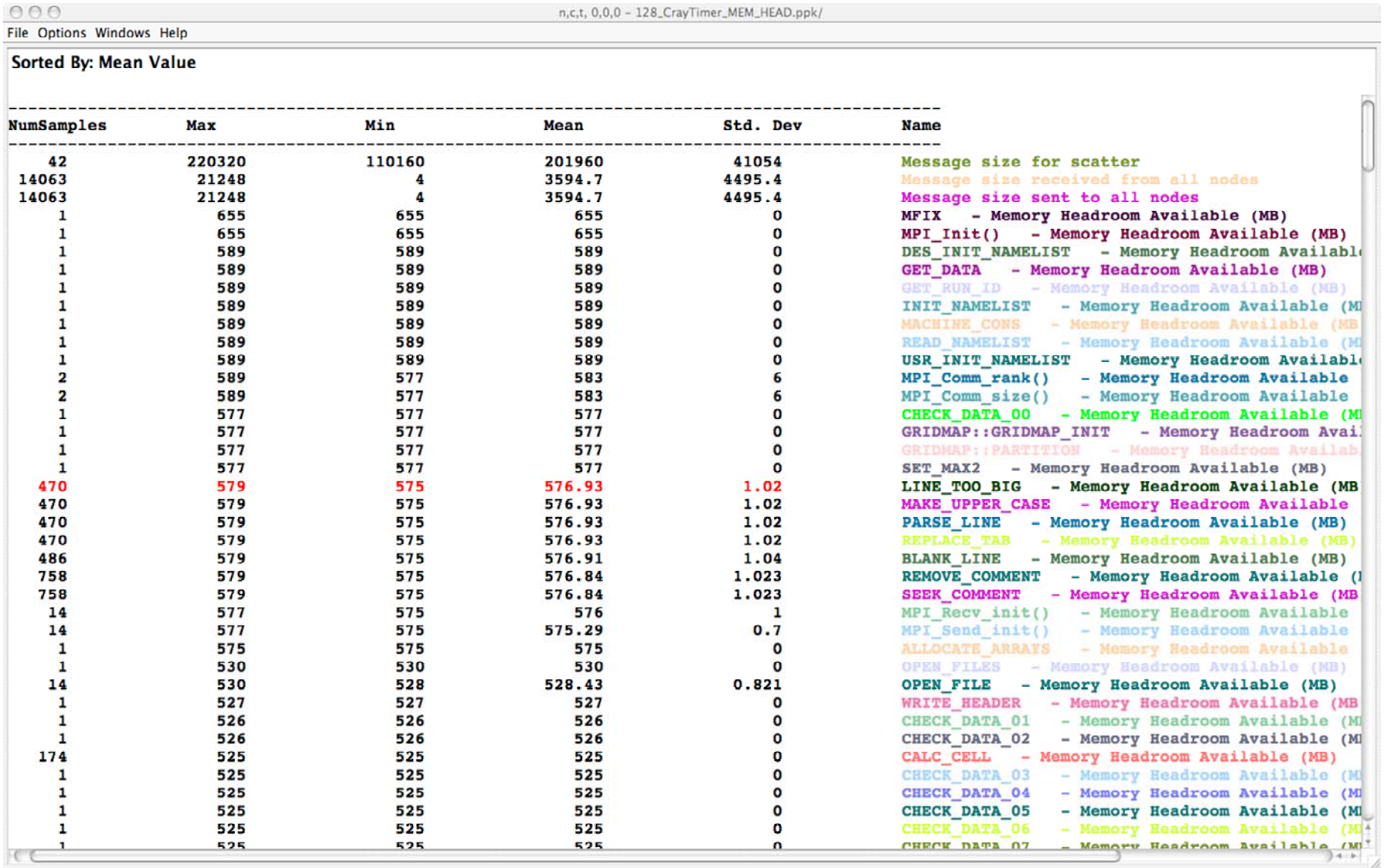
ParaProf – Full Profile (MFIX, NETL)



ParaProf – Mean Profile (MFIX)



ParaProf – Memory Profiling (MFIX)



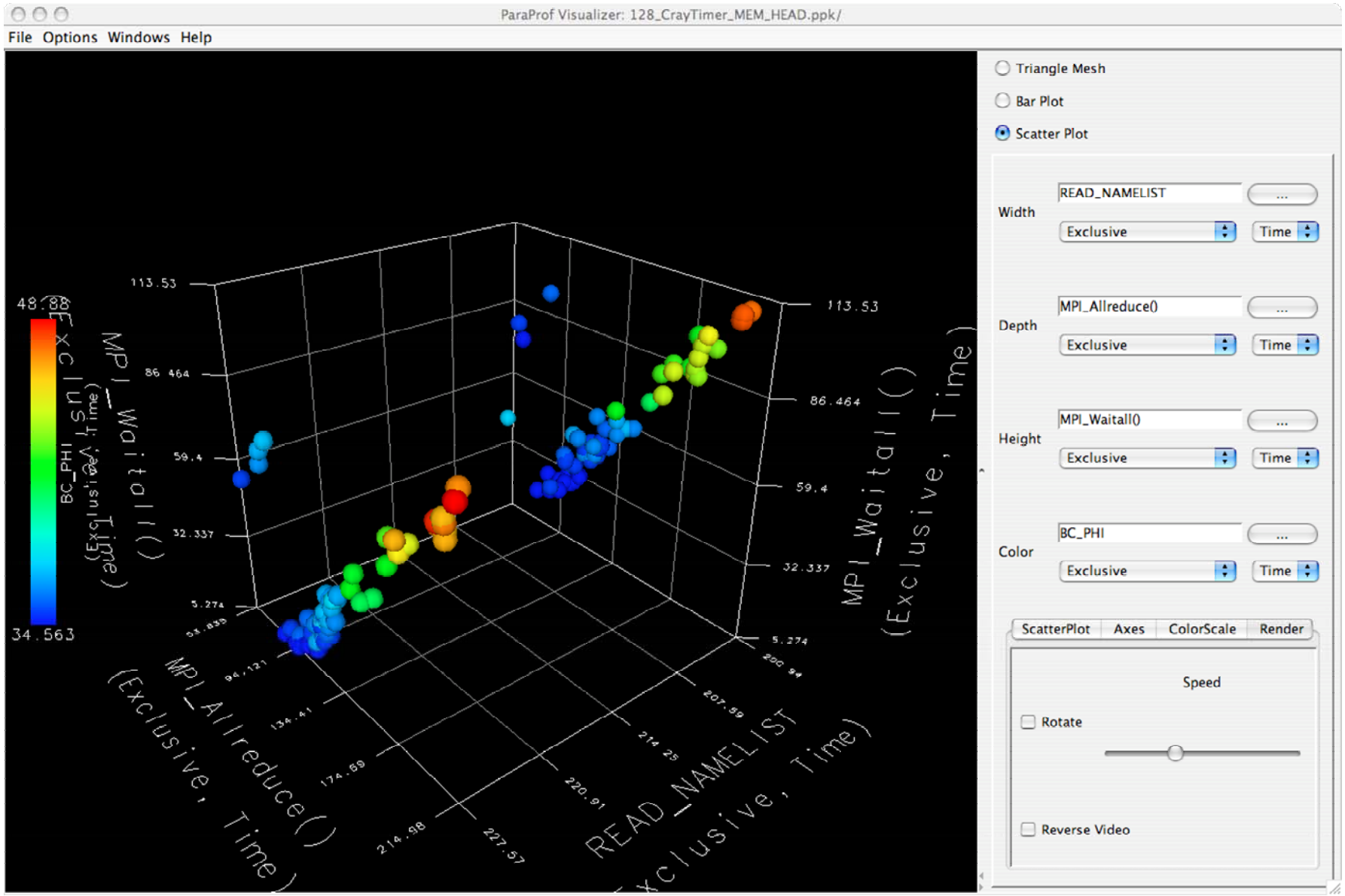
n,c,t, 0,0,0 - 128_CrayTimer_MEM_HEAD.ppk/

File Options Windows Help

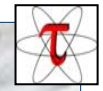
Sorted By: Mean Value

NumSamples	Max	Min	Mean	Std. Dev	Name
42	220320	110160	201960	41054	Message size for scatter
14063	21248	4	3594.7	4495.4	Message size received from all nodes
14063	21248	4	3594.7	4495.4	Message size sent to all nodes
1	655	655	655	0	MFIX - Memory Headroom Available (MB)
1	655	655	655	0	MPI_Init() - Memory Headroom Available (MB)
1	589	589	589	0	DES_INIT_NAMELIST - Memory Headroom Available
1	589	589	589	0	GET_DATA - Memory Headroom Available (MB)
1	589	589	589	0	GET_RUN_ID - Memory Headroom Available (MB)
1	589	589	589	0	INIT_NAMELIST - Memory Headroom Available (M
1	589	589	589	0	MACHINE_CONS - Memory Headroom Available (MB
1	589	589	589	0	READ_NAMELIST - Memory Headroom Available (M
1	589	589	589	0	USR_INIT_NAMELIST - Memory Headroom Available
2	589	577	583	6	MPI_Comm_rank() - Memory Headroom Available
2	589	577	583	6	MPI_Comm_size() - Memory Headroom Available
1	577	577	577	0	CHECK_DATA_00 - Memory Headroom Available (M
1	577	577	577	0	GRIDMAP::GRIDMAP_INIT - Memory Headroom Avai
1	577	577	577	0	GRIDMAP::PARTITION - Memory Headroom Availab
1	577	577	577	0	SET_MAX2 - Memory Headroom Available (MB)
470	579	575	576.93	1.02	LINE_TOO_BIG - Memory Headroom Available (MB
470	579	575	576.93	1.02	MAKE_UPPER_CASE - Memory Headroom Available
470	579	575	576.93	1.02	PARSE_LINE - Memory Headroom Available (MB)
470	579	575	576.93	1.02	REPLACE_TAB - Memory Headroom Available (MB)
486	579	575	576.91	1.04	BLANK_LINE - Memory Headroom Available (MB)
758	579	575	576.84	1.023	REMOVE_COMMENT - Memory Headroom Available (
758	579	575	576.84	1.023	SEEK_COMMENT - Memory Headroom Available (MB
14	577	575	576	1	MPI_Recv_init() - Memory Headroom Available
14	577	575	575.29	0.7	MPI_Send_init() - Memory Headroom Available
1	575	575	575	0	ALLOCATE_ARRAYS - Memory Headroom Available
1	530	530	530	0	OPEN_FILES - Memory Headroom Available (MB)
14	530	528	528.43	0.821	OPEN_FILE - Memory Headroom Available (MB)
1	527	527	527	0	WRITE_HEADER - Memory Headroom Available (MB
1	526	526	526	0	CHECK_DATA_01 - Memory Headroom Available (M
1	526	526	526	0	CHECK_DATA_02 - Memory Headroom Available (M
174	525	525	525	0	CALC_CELL - Memory Headroom Available (MB)
1	525	525	525	0	CHECK_DATA_03 - Memory Headroom Available (M
1	525	525	525	0	CHECK_DATA_04 - Memory Headroom Available (M
1	525	525	525	0	CHECK_DATA_05 - Memory Headroom Available (M
1	525	525	525	0	CHECK_DATA_06 - Memory Headroom Available (M
1	525	525	525	0	CHECK_DATA_07 - Memory Headroom Available (M

ParaProf – 3D Scatter Plot (MFIX)



Gprof Style Callpath View in Paraprof (SAGE)

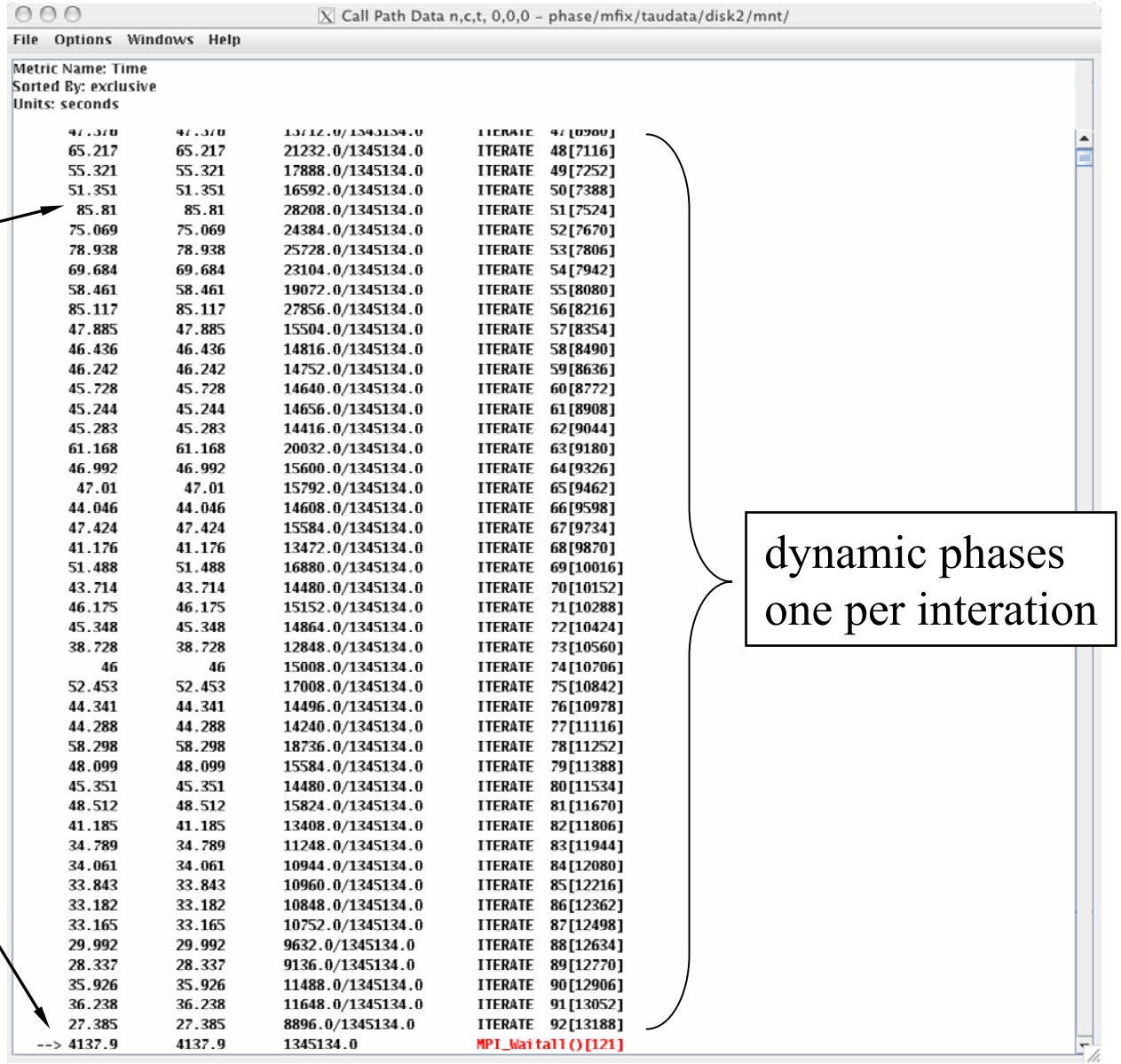


```
Metric Name: Time
Sorted By: exclusive
Units: seconds

    Exclusive    Inclusive    Calls/Tot.Calls              Name[id]
    ---------------------------------------------------------------------------

    1.8584       1.8584       1196/13188                   TOKEN_MODULE::TOKEN_GS_I  [521]
    0.584        0.584        234/13188                    TOKEN_MODULE::TOKEN_GS_L  [544]
    25.0819      25.0819      11758/13188                  TOKEN_MODULE::TOKEN_GS_R8  [734]
--> 27.5242      27.5242      13188                        MPI_Waitall()  [525]

    17.9579      39.1657      156/156                      DERIVATIVE_MODULE::DERIVATIVES_NOFACE  [841]
--> 17.9579      39.1657      156                          DERIVATIVE_MODULE::DERIVATIVES_FACE  [843]
    0.0156       0.0195       312/312                      TIMER_MODULE::TIMERSET  [77]
    0.1133       9.1269       2340/2340                    MESSAGE_MODULE::CLONE_GET_R8  [808]
    0.1602       11.4608      4056/4056                    MESSAGE_MODULE::CLONE_PUT_R8  [850]
    0.0059       0.6006       117/117                      MESSAGE_MODULE::CLONE_PUT_I  [856]

    14.1151      21.6209      5/5                          MATRIX_MODULE::MCGDS  [1443]
--> 14.1151      21.6209      5                            MATRIX_MODULE::CSR_CG_SOLVER  [1470]
    0.0654       1.2617       1005/1005                    TOKEN_MODULE::TOKEN_GET_R8  [769]
    0.0557       5.2714       1005/1005                    TOKEN_MODULE::TOKEN_REDUCTION_R8_S  [1475]
    0.0703       0.9726       1000/1000                    TOKEN_MODULE::TOKEN_REDUCTION_R8_V  [208]
```

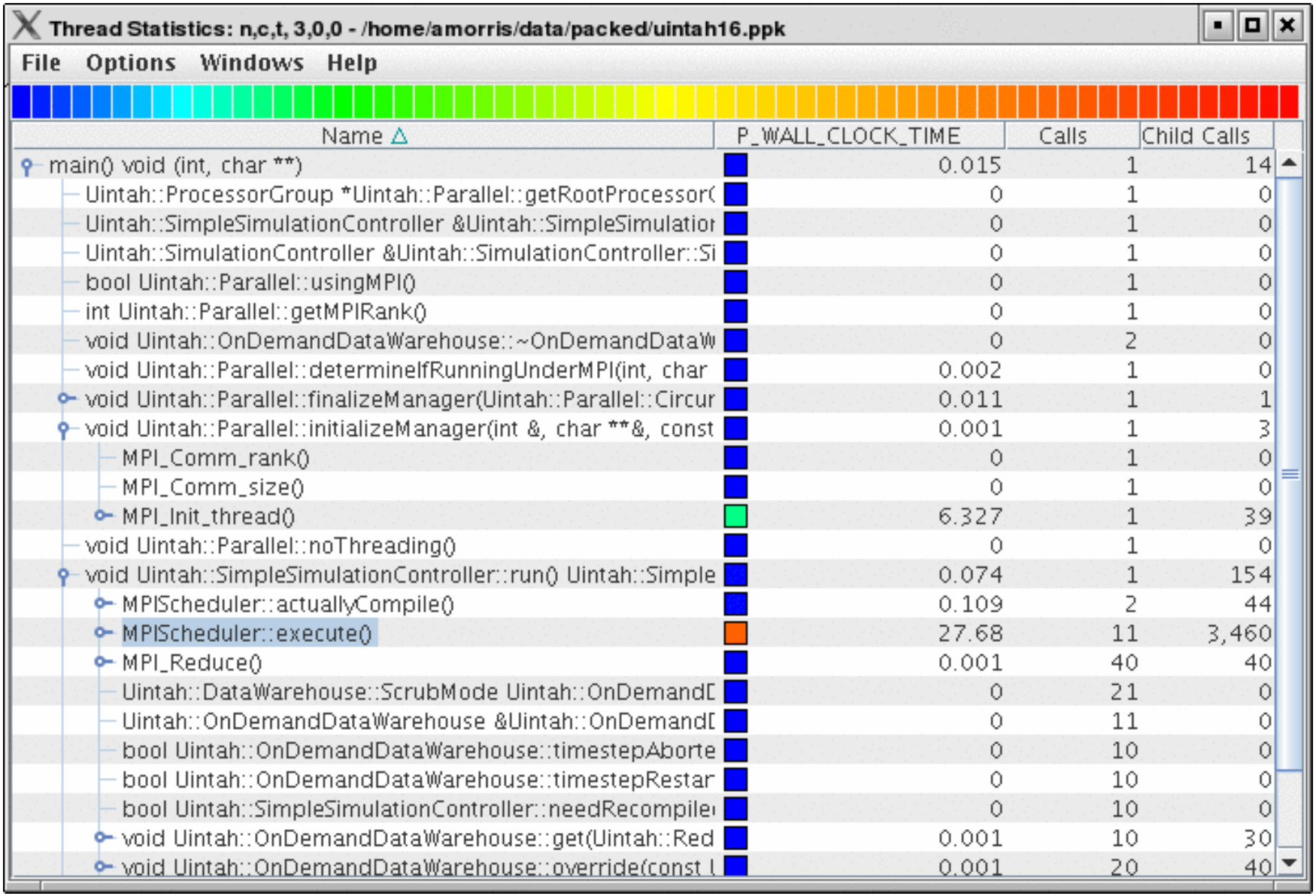
ParaProf – Phase Profile (MFIX)

In 51st iteration, time spent in MPI_Waitall was 85.81 secs

Total time spent in MPI_Waitall was 4137.9 secs across all 92 iterations



ParaProf - Statistics Table (Uintah)

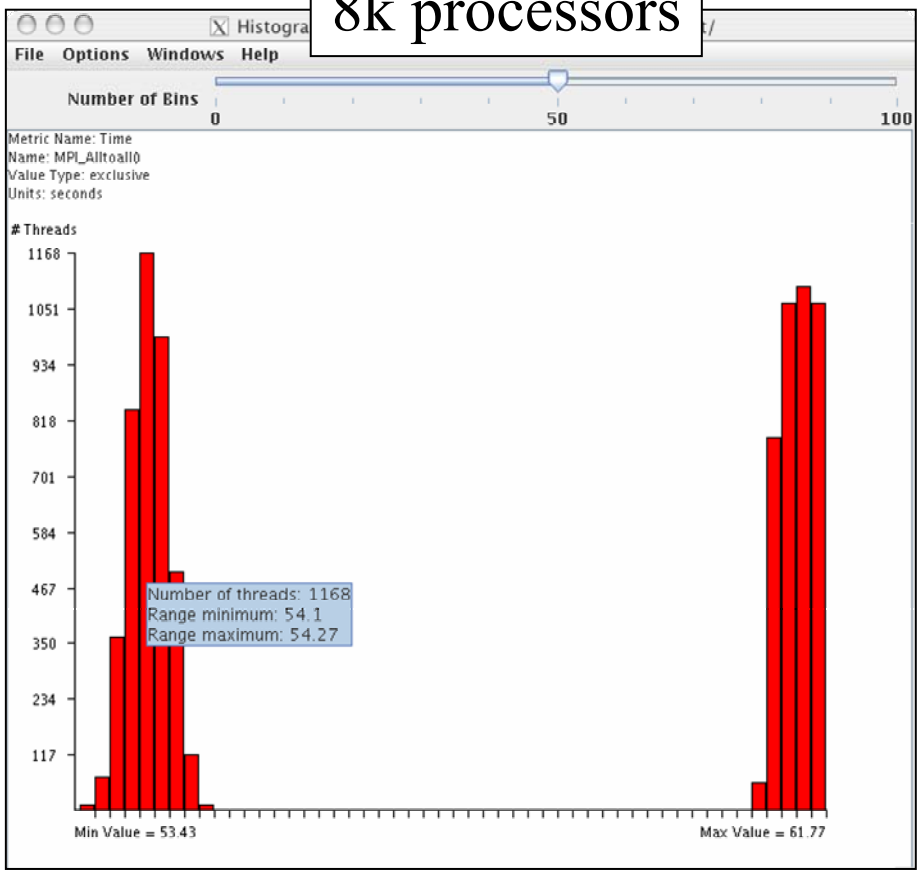


Name △	P_WALL_CLOCK_TIME	Calls	Child Calls
main() void (int, char **)	0.015	1	14
Uintah::ProcessorGroup *Uintah::Parallel::getRootProcessor(	0	1	0
Uintah::SimpleSimulationController &Uintah::SimpleSimulatior	0	1	0
Uintah::SimulationController &Uintah::SimulationController::Si	0	1	0
bool Uintah::Parallel::usingMPI()	0	1	0
int Uintah::Parallel::getMPIRank()	0	1	0
void Uintah::OnDemandDataWarehouse::~OnDemandDataW	0	2	0
void Uintah::Parallel::determineIfRunningUnderMPI(int, char	0.002	1	0
void Uintah::Parallel::finalizeManager(Uintah::Parallel::Circur	0.011	1	1
void Uintah::Parallel::initializeManager(int &, char **&, const	0.001	1	3
MPI_Comm_rank()	0	1	0
MPI_Comm_size()	0	1	0
MPI_Init_thread()	6.327	1	39
void Uintah::Parallel::noThreading()	0	1	0
void Uintah::SimpleSimulationController::run() Uintah::Simple	0.074	1	154
MPIScheduler::actuallyCompile()	0.109	2	44
MPIScheduler::execute()	27.68	11	3,460
MPI_Reduce()	0.001	40	40
Uintah::DataWarehouse::ScrubMode Uintah::OnDemandD	0	21	0
Uintah::OnDemandDataWarehouse &Uintah::OnDemandD	0	11	0
bool Uintah::OnDemandDataWarehouse::timestepAborte	0	10	0
bool Uintah::OnDemandDataWarehouse::timestepRestar	0	10	0
bool Uintah::SimpleSimulationController::needRecompile(	0	10	0
void Uintah::OnDemandDataWarehouse::get(Uintah::Red	0.001	10	30
void Uintah::OnDemandDataWarehouse::override(const I	0.001	20	40

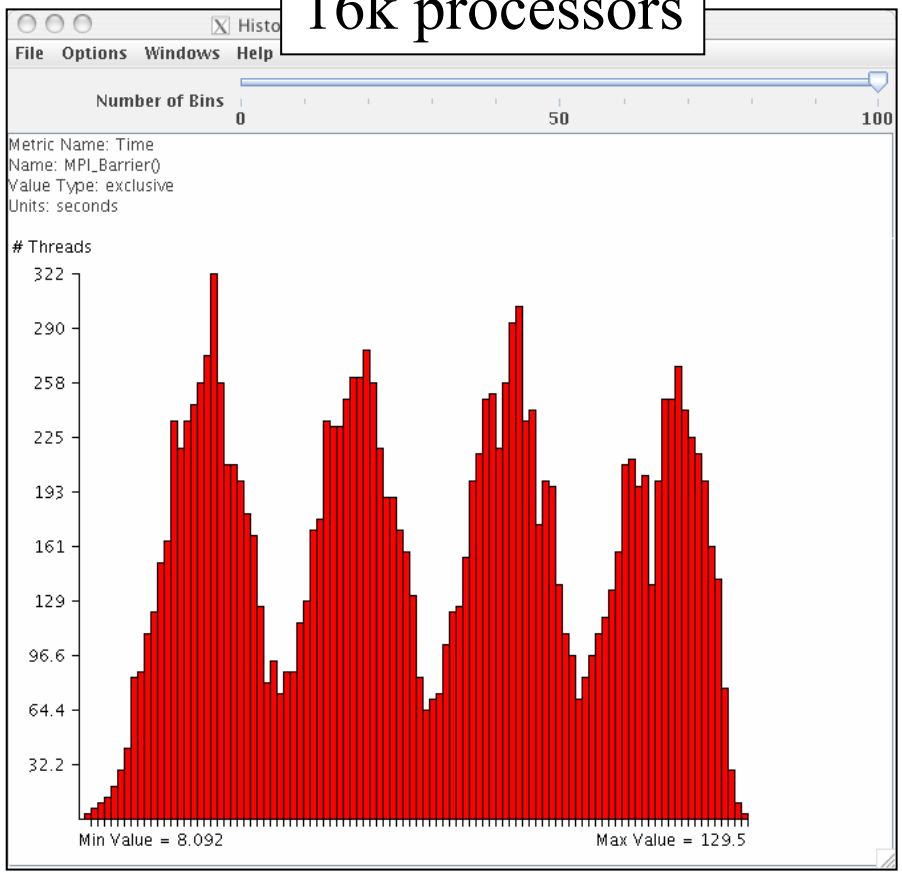
ParaProf – Histogram View (Miranda)

- Scalable 2D displays

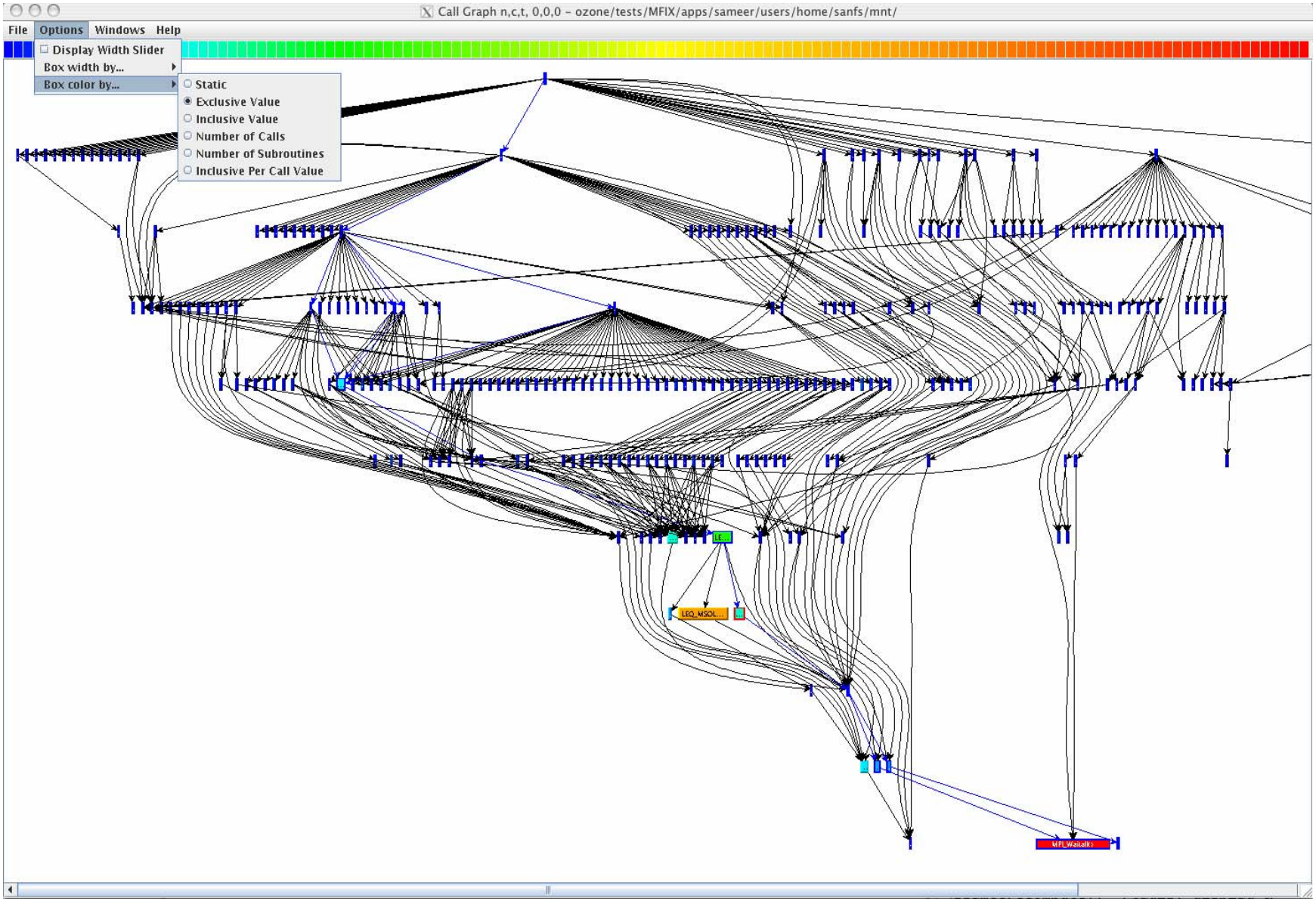
8k processors



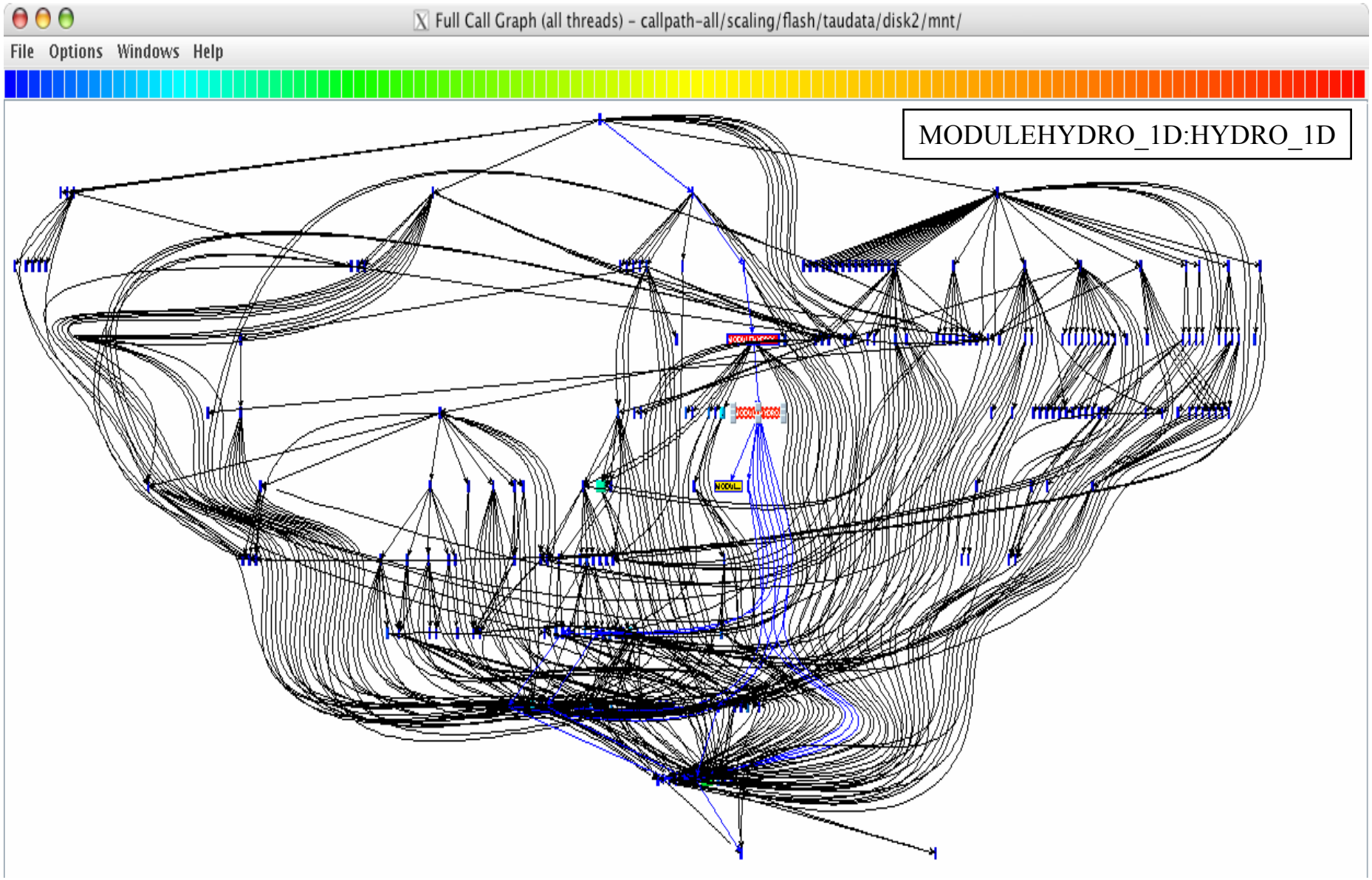
16k processors



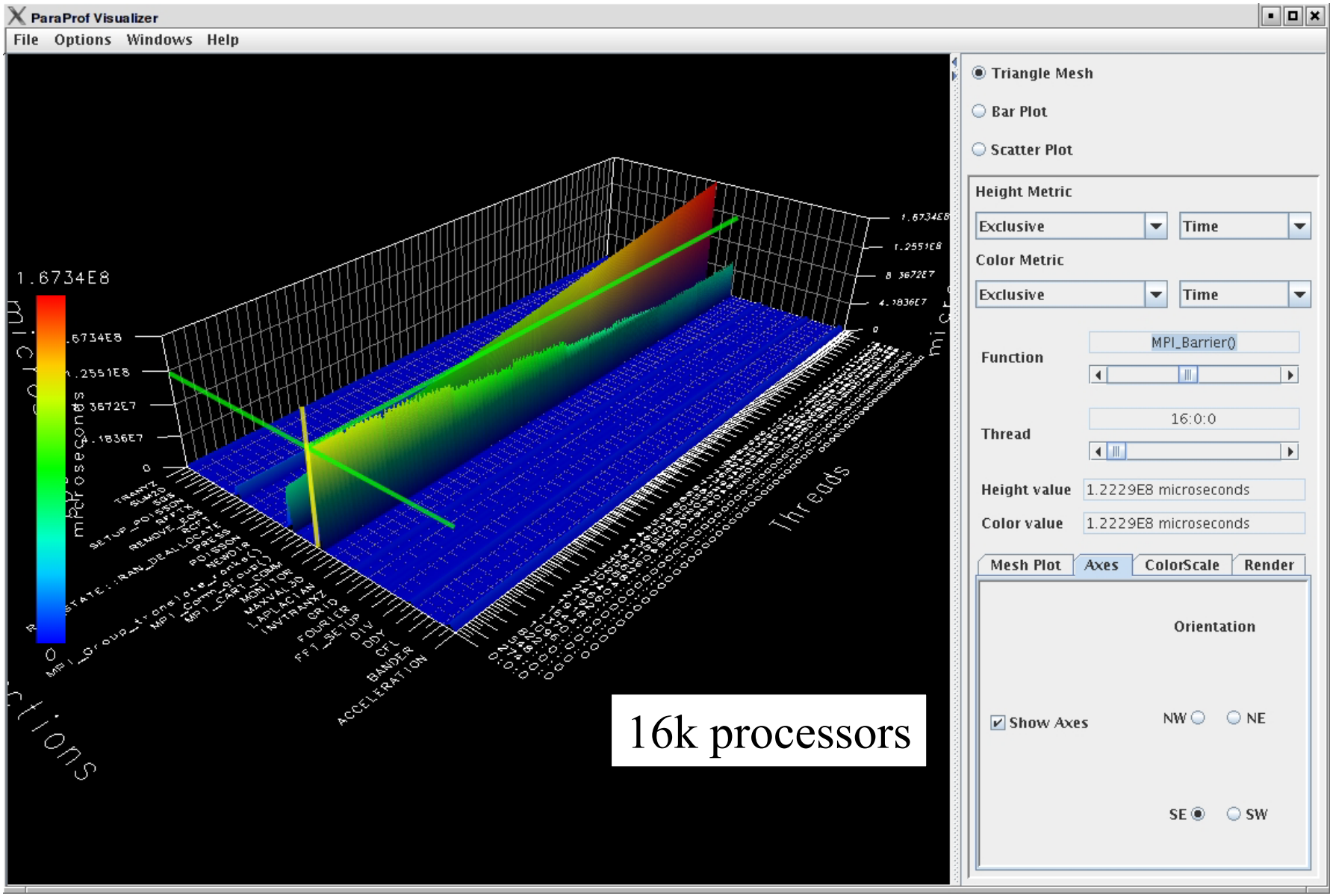
ParaProf –Callgraph View (MFIX)



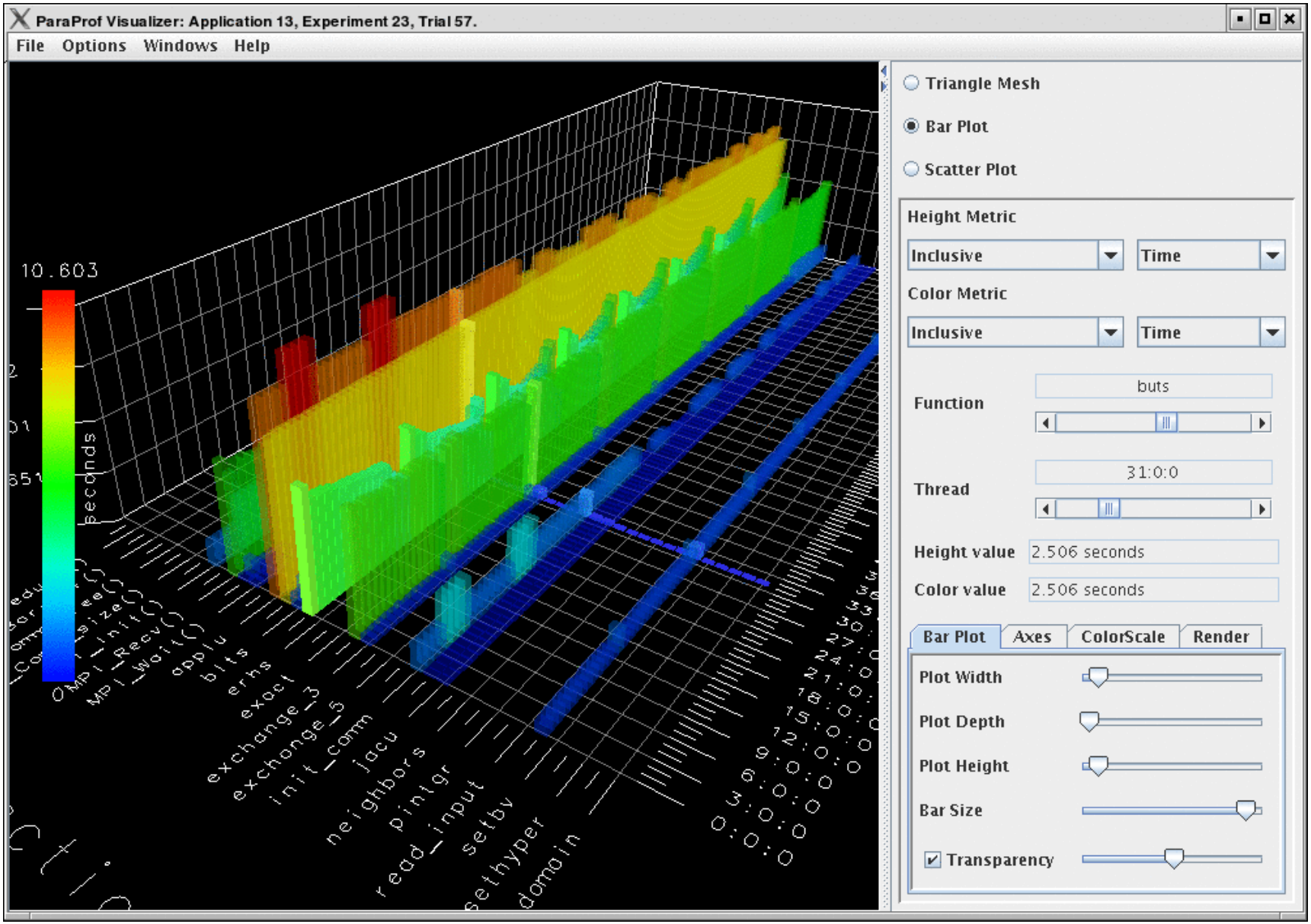
ParaProf – Callpath Highlighting (Flash)



ParaProf – 3D Full Profile (Miranda)

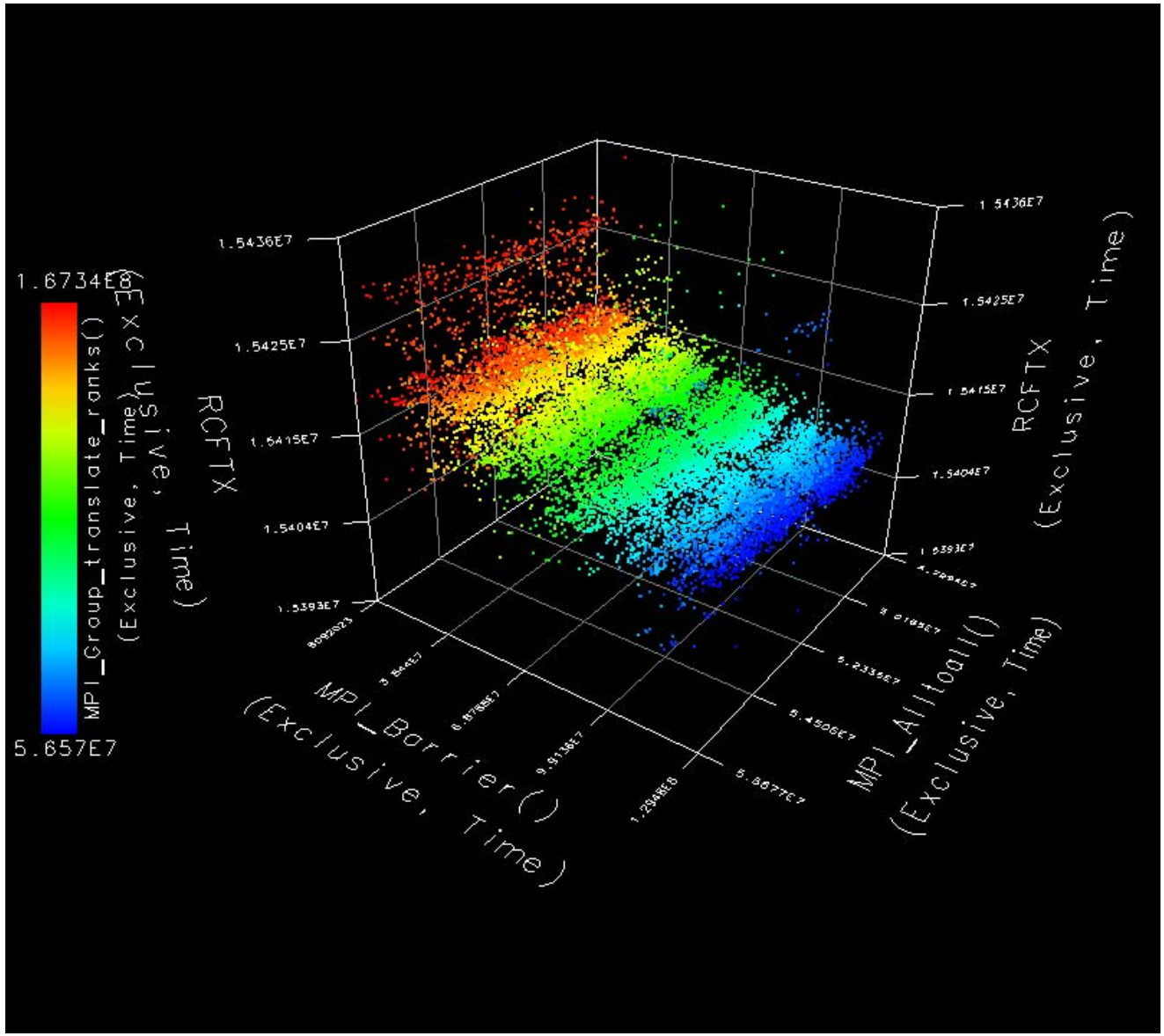


ParaProf Bar Plot (Zoom in/out +/-)

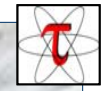


ParaProf – 3D Scatterplot (Miranda)

- Each point is a “thread” of execution (32k cpus)
- A total of four metrics shown in relation
- ParaVis 3D profile visualization library
  - JOGL

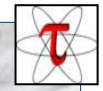


TAU Performance System Status



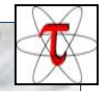
- Computing platforms (selected)
  - Cray XT3/X1E/XD1/T3E/SV1, IBM SP/pSeries/BGL, SGI Altix/Origin, HP (Compaq) SC (Tru64), Sun, Linux clusters (IA-32/64, Alpha, PPC, PA-RISC, Power, Opteron), Apple (G4/5, OS X), Hitachi SR8000, NEC SX-5/6, FreeBSD, Windows …
- Programming languages
  - C, C++, Fortran 77/90/95, HPF, Java, Python
- Thread libraries (selected)
  - pthreads, OpenMP, SGI sproc, Java,Windows, Charm++
- Compilers (selected)
  - Cray, Pathscale, Intel, PGI, GNU, Fujitsu, Sun, IBM, HP, NEC, Absoft, Lahey, Nagware

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.

Support Acknowledgements



- Department of Energy (DOE) contracts
  - Office of Science
  - University of Utah ASC
  - LLNL ASC/NNSA Level 3
  - Los Alamos National Laboratory
  - Argonne National Laboratory (ZeptoOS)
- DoD PET HPCMO
- Oak Ridge National Laboratory
- Pittsburgh Supercomputing Center
- NETL/Aeolus Research
  - Aytekin Gel
- **http://www.cs.uoregon.edu/research/tau**