TAU Performance Toolkit

(WOMPAT OpenMP Lab Sessions)

Sameer Shende, Allen D. Malony, Robert Bell
University of Oregon
{sameer, malony, bertie}@cs.uoregon.edu

Tuning and Analysis Utilities

UNIVERSITY OF OREGON

Los Alamos NATIONAL LABORATORY

John von Neumann - Institut für Computing NIC
Zentralinstitut für Angewandte Mathematik

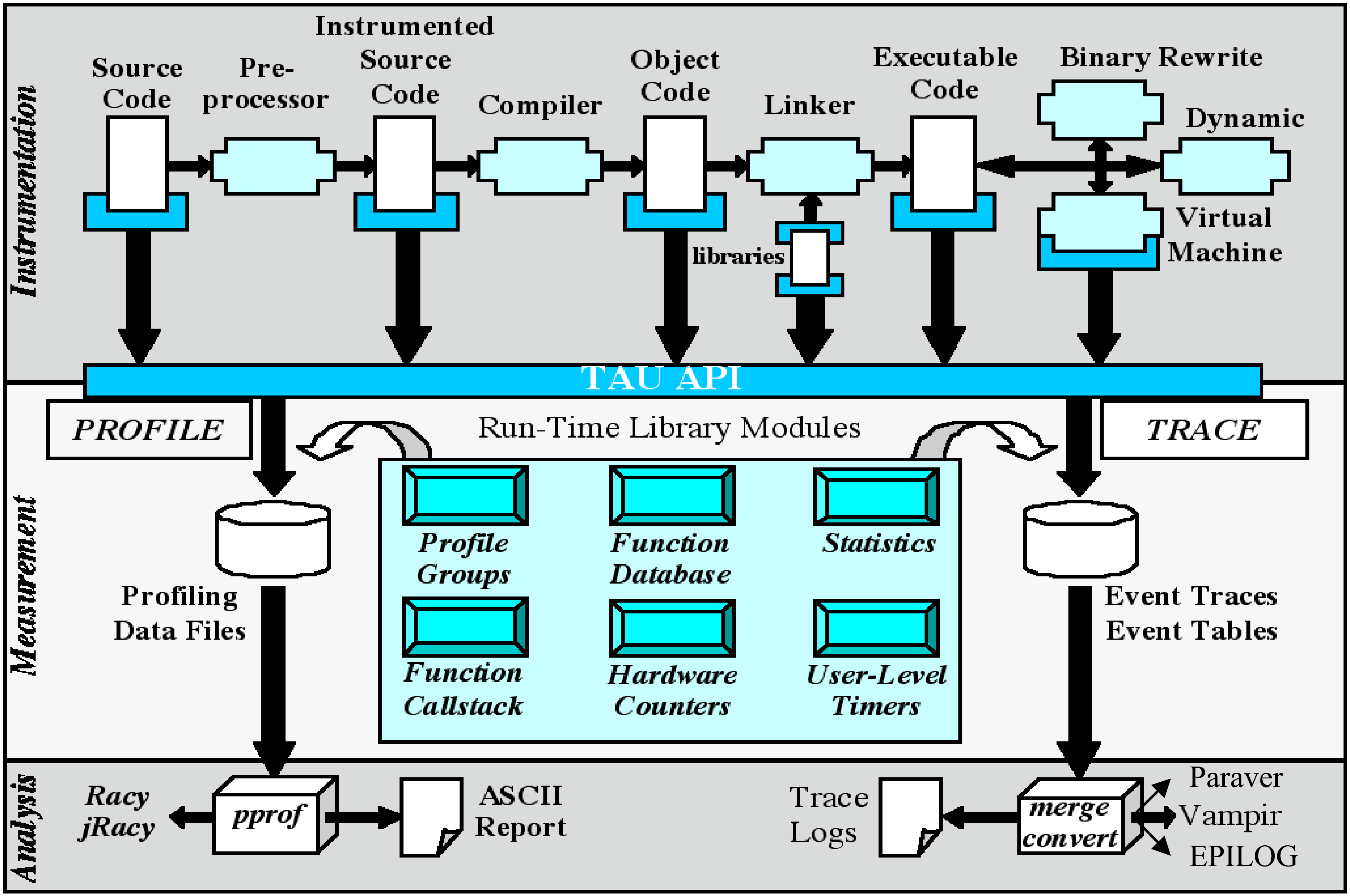
Outline

- Motivation
- Part I: Overview of TAU and PDT
- Performance Analysis and Visualization with TAU
  - Pprof
  - Paraprof
  - Performance Database
- Part II: Using TAU – a tutorial
- Conclusion

TAU Performance System

- Tuning and Analysis Utilities (11+ year project effort)
- *Performance system framework* for scalable parallel and distributed high-performance computing
- Targets a general complex system computation model
  - nodes / contexts / threads
  - Multi-level: system / software / parallelism
  - Measurement and analysis abstraction
- *Integrated toolkit* for performance instrumentation, measurement, analysis, and visualization
  - Portable performance profiling and tracing facility
  - Open software approach with technology integration
- University of Oregon , Forschungszentrum Jülich, LANL

TAU Performance System Architecture



Strategies for Empirical Performance Evaluation

- Empirical performance evaluation as a series of performance experiments
  - Experiment trials describing instrumentation and measurement requirements
  - Where/When/How axes of empirical performance space
    - where are performance measurements made in program
      - routines, loops, statements…
    - when is performance instrumentation done
      - compile-time, while pre-processing, runtime…
    - how are performance measurement/instrumentation chosen
      - profiling with hw counters, tracing, callpath profiling…

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

TAU Instrumentation

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

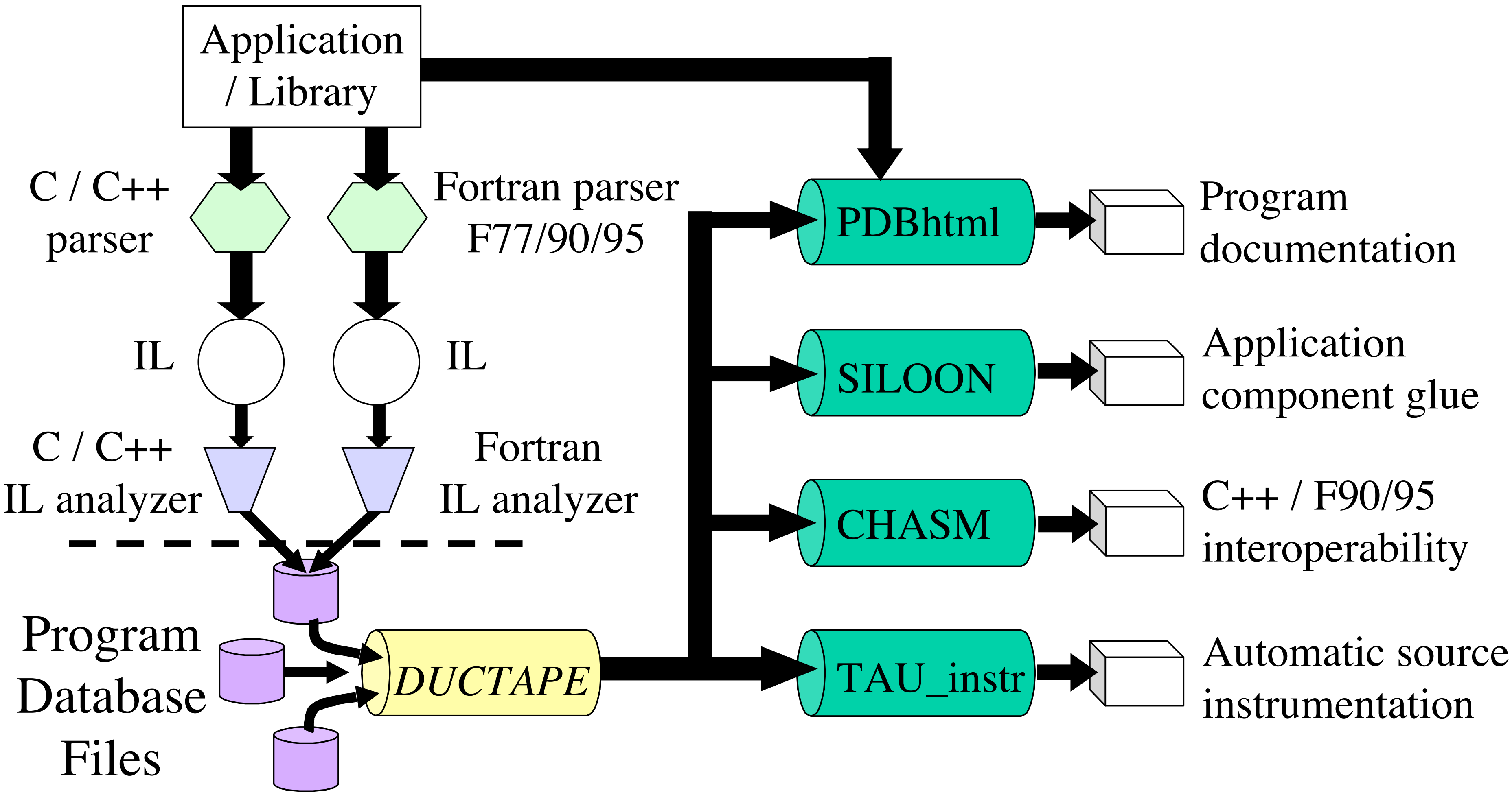
Multi-Level Instrumentation

- Targets common measurement interface
  - *TAU API*
- Multiple instrumentation interfaces
  - Simultaneously active
- Information sharing between interfaces
  - Utilizes instrumentation knowledge between levels
- Selective instrumentation
  - Available at each level
  - Cross-level selection
- Targets a common performance model
- Presents a unified view of execution
  - Consistent performance events

Program Database Toolkit (PDT)

- Program code analysis framework
  - develop source-based tools
- *High-level interface* to source code information
- *Integrated toolkit* for source code parsing, database creation, and database query
  - Commercial grade front-end parsers
  - Portable IL analyzer, database format, and access API
  - Open software approach for tool development
- Multiple source languages
- Implement automatic performance instrumentation tools
  - *tau_instrumentor*

Program Database Toolkit (PDT)



PDT 3.1 Functionality

- C++ statement-level information implementation
  - for, while loops, declarations, initialization, assignment…
  - PDB records defined for most constructs
- DUCTAPE
  - Processes PDB 1.x, 2.x, 3.x uniformly
- PDT applications
  - XMLgen
    - PDB to XML converter
    - Used for CHASM and CCA tools
  - PDBstmt
    - Statement callgraph display tool

PDT 3.1 Functionality (continued)

- Cleanscape Flint parser fully integrated for F90/95
  - Flint parser (f95parse) is very robust
  - Produces PDB records for TAU instrumentation (stage 1)
    - Linux (x86, IA-64, Opteron, Power4), HP Tru64, IBM AIX, Cray X1,T3E, Solaris, SGI, Apple, Windows, Power4 Linux (IBM Blue Gene/L compatible)
  - Full PDB 2.0 specification (stage 2) [SC'04]
  - Statement level support (stage 3) [SC'04]
- PDT 3.1 released in March 2004.
- URL: http://www.cs.uoregon.edu/research/paracomp/pdtoolkit

TAU Performance Measurement

- TAU supports profiling and tracing measurement
- Robust timing and hardware performance support using PAPI
- Support for online performance monitoring
  - Profile and trace performance data export to file system
  - Selective exporting
- Extension of TAU measurement for multiple counters
  - Creation of user-defined TAU counters
  - Access to system-level metrics
- Support for callpath measurement
- Integration with system-level performance data
  - Linux MAGNET/MUSE (Wu Feng, LANL)

TAU Measurement

- Performance information
  - Performance events
  - High-resolution timer library (real-time / virtual clocks)
  - General software counter library (user-defined events)
  - Hardware performance counters
    - *PAPI* (Performance API) (UTK, Ptools Consortium)
    - consistent, portable API
- Organization
  - Node, context, thread levels
  - Profile groups for collective events (runtime selective)
  - Performance data mapping between software levels

TAU Measurement Options

- Parallel profiling
  - Function-level, block-level, statement-level
  - Supports user-defined events
  - TAU parallel profile data stored during execution
  - Hardware counts values
  - Support for multiple counters
  - Support for callgraph and callpath profiling
- Tracing
  - All profile-level events
  - Inter-process communication events
  - Trace merging and format conversion

Grouping Performance Data in TAU

- Profile Groups
  - A group of related routines forms a profile group
  - Statically defined
    - TAU_DEFAULT, TAU_USER[1-5], TAU_MESSAGE, TAU_IO, …
  - Dynamically defined
    - group name based on string, such as "adlib" or "particles"
    - runtime lookup in a map to get unique group identifier
    - uses *tau_instrumentor* to instrument
  - Ability to change group names at runtime
  - Group-based instrumentation and measurement control

TAU Analysis

- Parallel profile analysis
  - *Pprof*
    - parallel profiler with text-based display
  - *ParaProf*
    - Graphical, scalable, parallel profile analysis and display
- Trace analysis and visualization
  - Trace merging and clock adjustment (if necessary)
  - Trace format conversion (ALOG, SDDF, VTF, Paraver)
  - Trace visualization using *Vampir* (Pallas/Intel)

Pprof Output (NAS Parallel Benchmark – LU)

- Intel Quad PIII Xeon
- F90 + MPICH
- Profile
  - Node
  - Context
  - Thread
- Events
  - code
  - MPI

emacs@neutron.cs.uoregon.edu

Buffers Files Tools Edit Search Mule Help

Reading Profile files in profile.*

NODE 0;CONTEXT 0;THREAD 0:

%Time	Exclusive msec	Inclusive total msec	#Call	#Subrs	Inclusive usec/call	Name
100.0	1	3:11.293	1	15	191293269	applu
99.6	3,667	3:10.463	3	37517	63487925	bcast_inputs
67.1	491	2:08.326	37200	37200	3450	exchange_1
44.5	6,461	1:25.159	9300	18600	9157	buts
41.0	1:18.436	1:18.436	18600	0	4217	MPI_Recv()
29.5	6,778	56,407	9300	18600	6065	blts
26.2	50,142	50,142	19204	0	2611	MPI_Send()
16.2	24,451	31,031	301	602	103096	rhs
3.9	7,501	7,501	9300	0	807	jacld
3.4	838	6,594	604	1812	10918	exchange_3
3.4	6,590	6,590	9300	0	709	jacu
2.6	4,989	4,989	608	0	8206	MPI_Wait()
0.2	0.44	400	1	4	400081	init_comm
0.2	398	399	1	39	399634	MPI_Init()
0.1	140	247	1	47616	247086	setiv
0.1	131	131	57252	0	2	exact
0.1	89	103	1	2	103168	erhs
0.1	0.966	96	1	2	96458	read_input
0.0	95	95	9	0	10603	MPI_Bcast()
0.0	26	44	1	7937	44878	error
0.0	24	24	608	0	40	MPI_Irecv()
0.0	15	15	1	5	15630	MPI_Finalize()
0.0	4	12	1	1700	12335	setbv
0.0	7	8	3	3	2893	l2norm
0.0	3	3	8	0	491	MPI_Allreduce()
0.0	1	3	1	6	3874	pintgr
0.0	1	1	1	0	1007	MPI_Barrier()
0.0	0.116	0.837	1	4	837	exchange_4
0.0	0.512	0.512	1	0	512	MPI_Keyval_create()
0.0	0.121	0.353	1	2	353	exchange_5
0.0	0.024	0.191	1	2	191	exchange_6
0.0	0.103	0.103	6	0	17	MPI_Type_contiguous()

--:-- NPB_LU.out (Fundamental)--L8--Top----

Terminology – Example

- For routine "int main( )":
- Exclusive time
  - 100-20-50-20=10 secs
- Inclusive time
  - 100 secs
- Calls
  - 1 call
- Subrs (no. of child routines called)
  - 3
- Inclusive time/call
  - 100secs

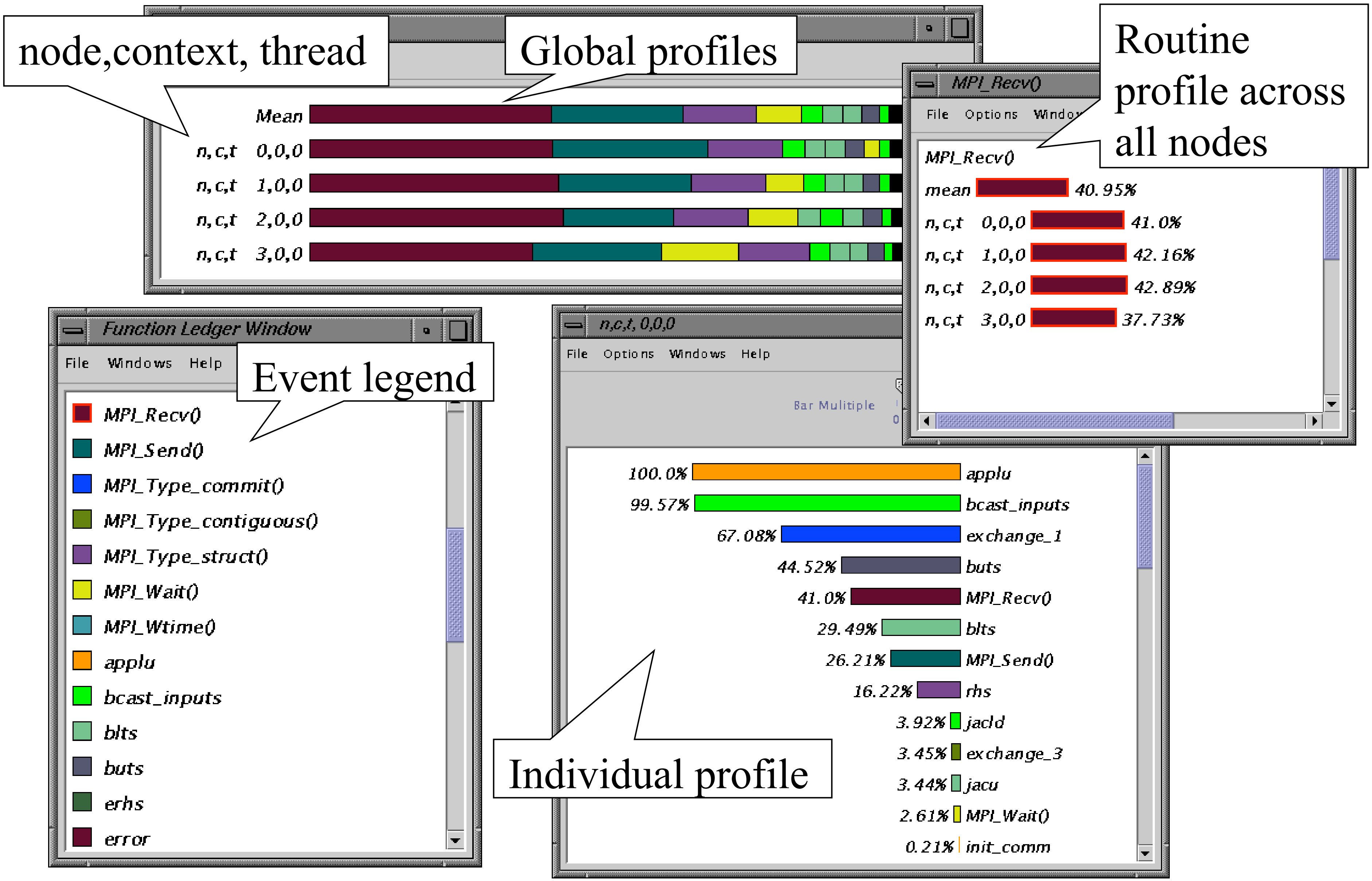
```
int main( )
{ /* takes 100 secs */

  f1(); /* takes 20 secs */
  f2(); /* takes 50 secs */
  f1(); /* takes 20 secs */

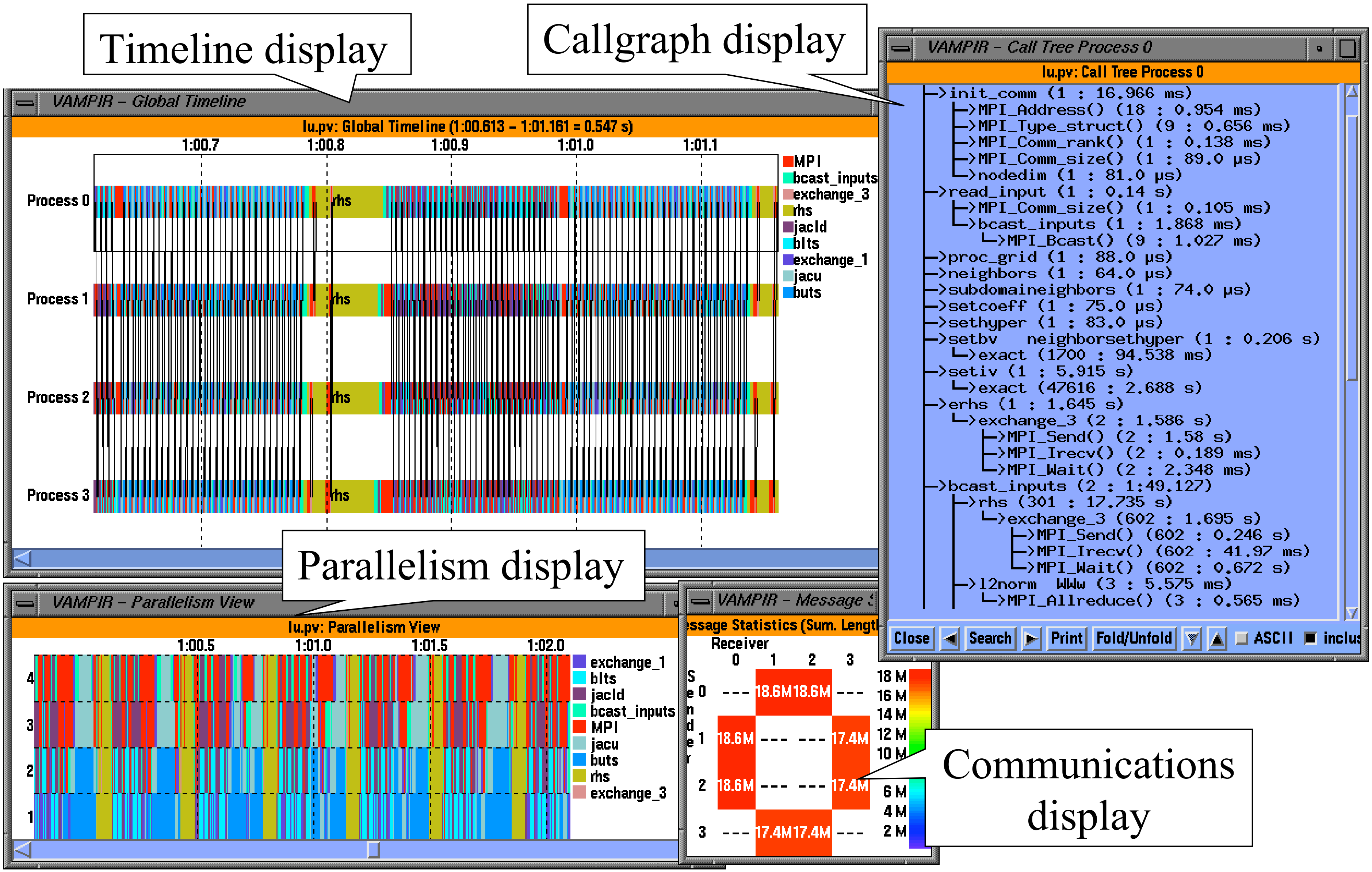
  /* other work */
}

/*
Time can be replaced by  counts
from PAPI e.g., PAPI_FP_INS. */
```

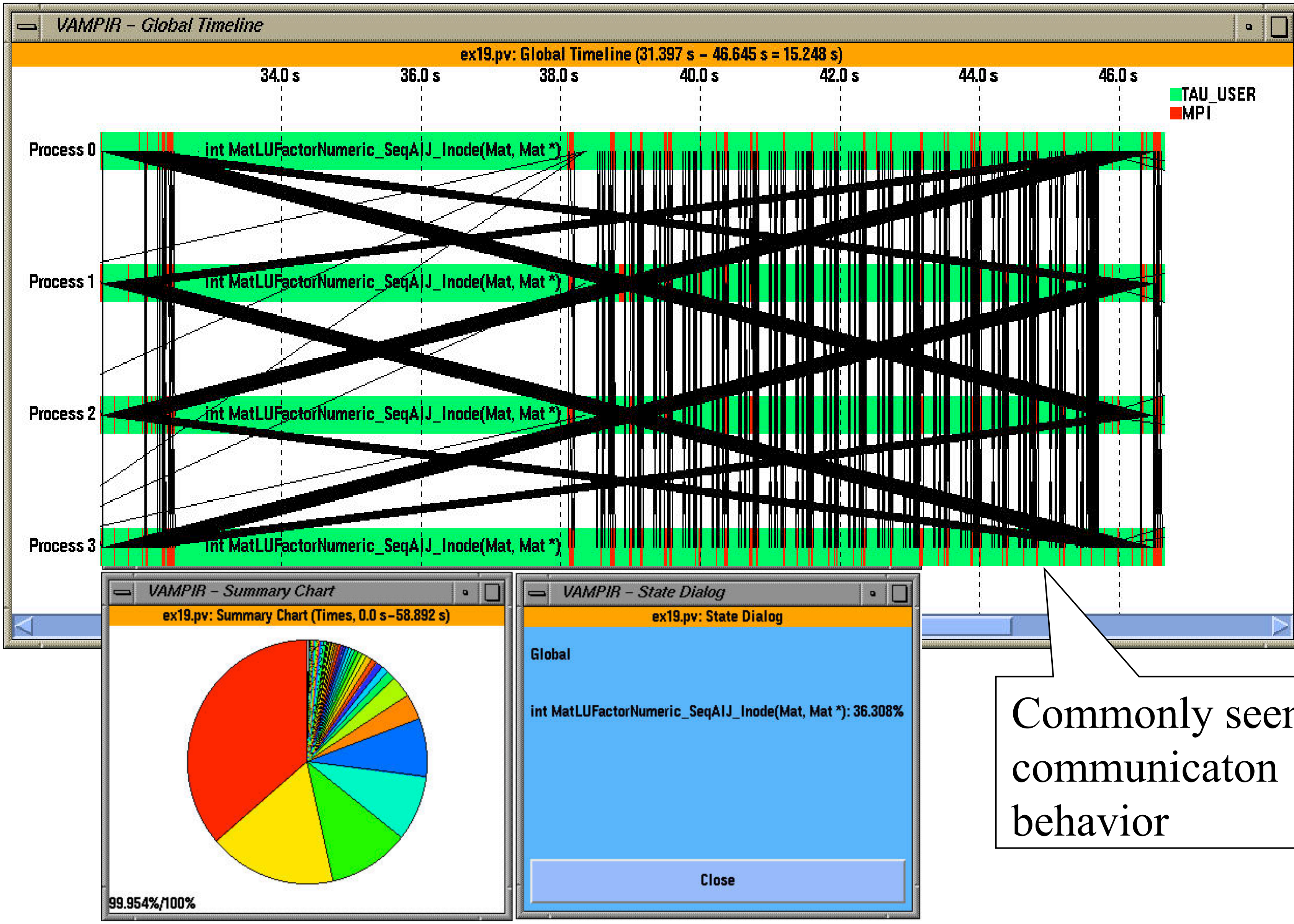
ParaProf (NAS Parallel Benchmark – LU)



TAU + Vampir (NAS Parallel Benchmark – LU)

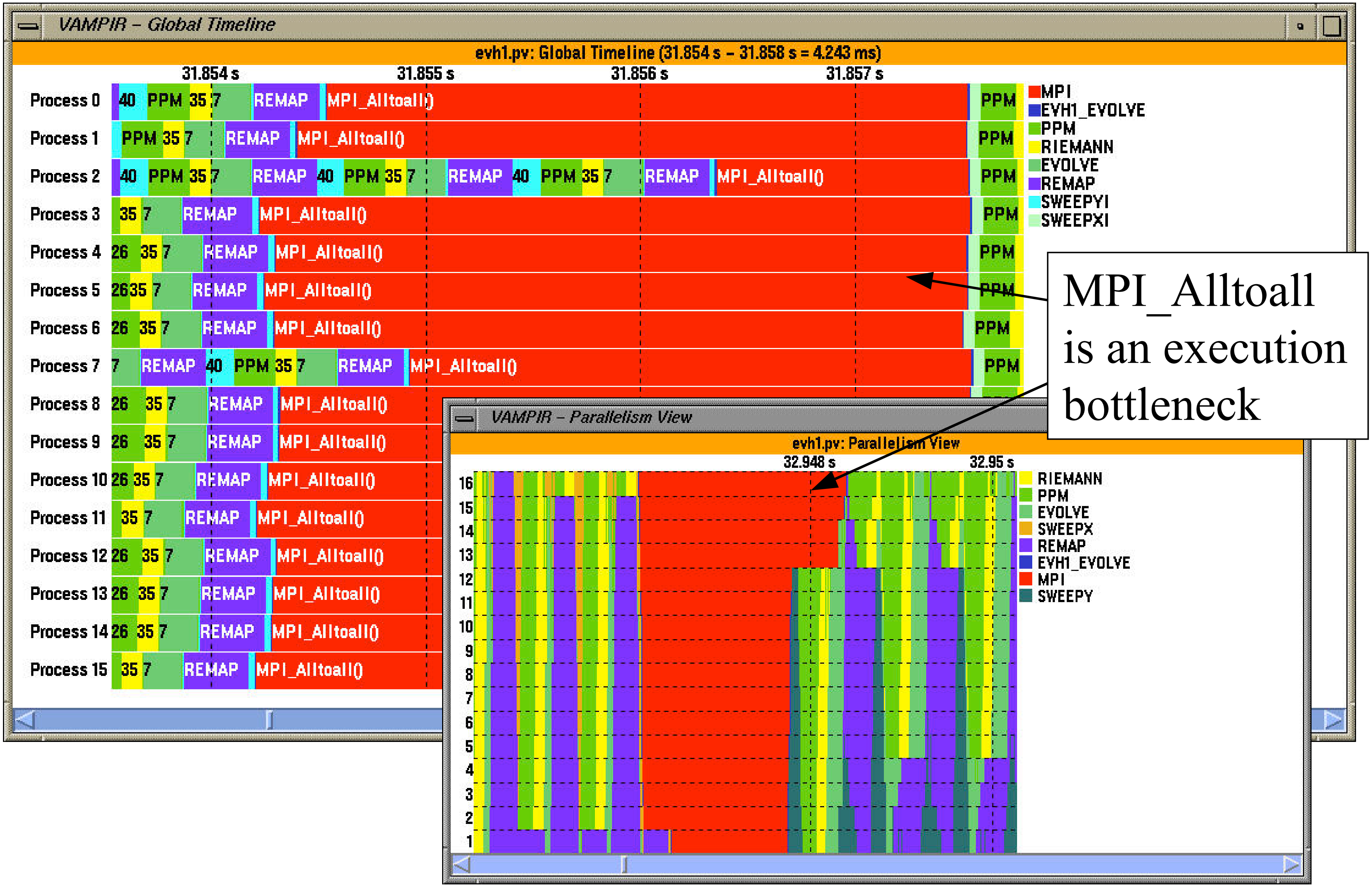


PETSc ex19 (Tracing)



Commonly seen communicaton behavior

TAU's EVH1 Execution Trace in Vampir

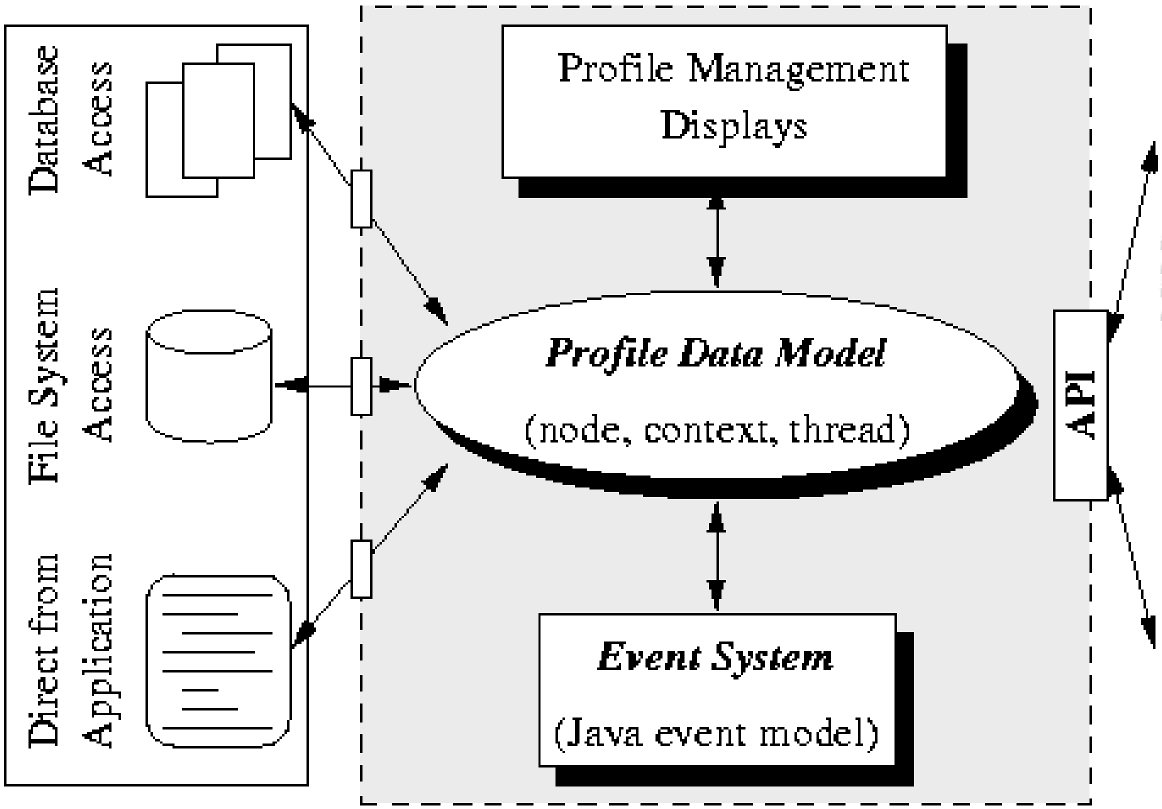


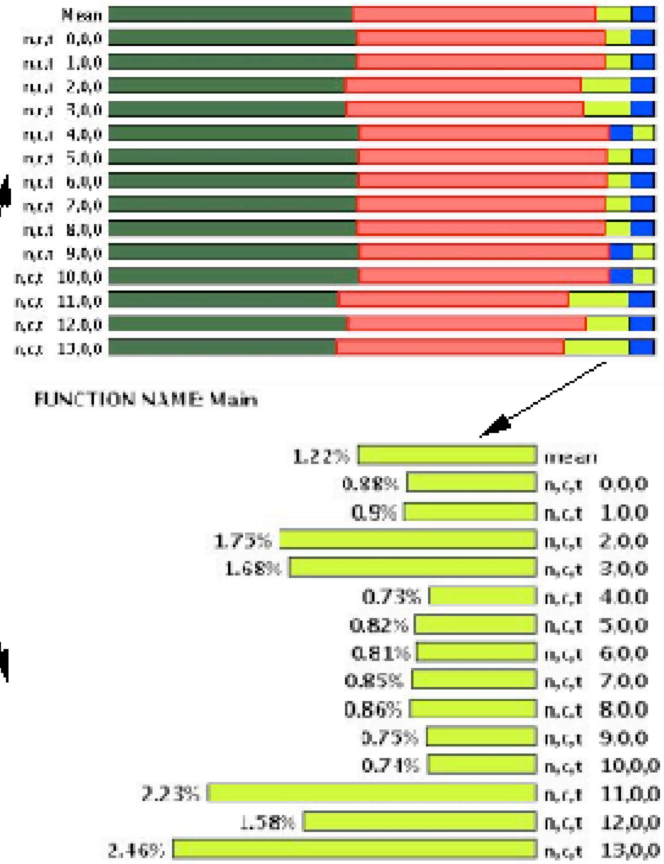
Performance Analysis and Visualization

- Analysis of parallel profile and trace measurement
- Parallel profile analysis
  - ParaProf
  - Profile generation from trace data
- Performance database framework (PerfDBF)
- Parallel trace analysis
  - Translation to VTF 3.0 and EPILOG
  - Integration with VNG (Technical University of Dresden)
- Online parallel analysis and visualization

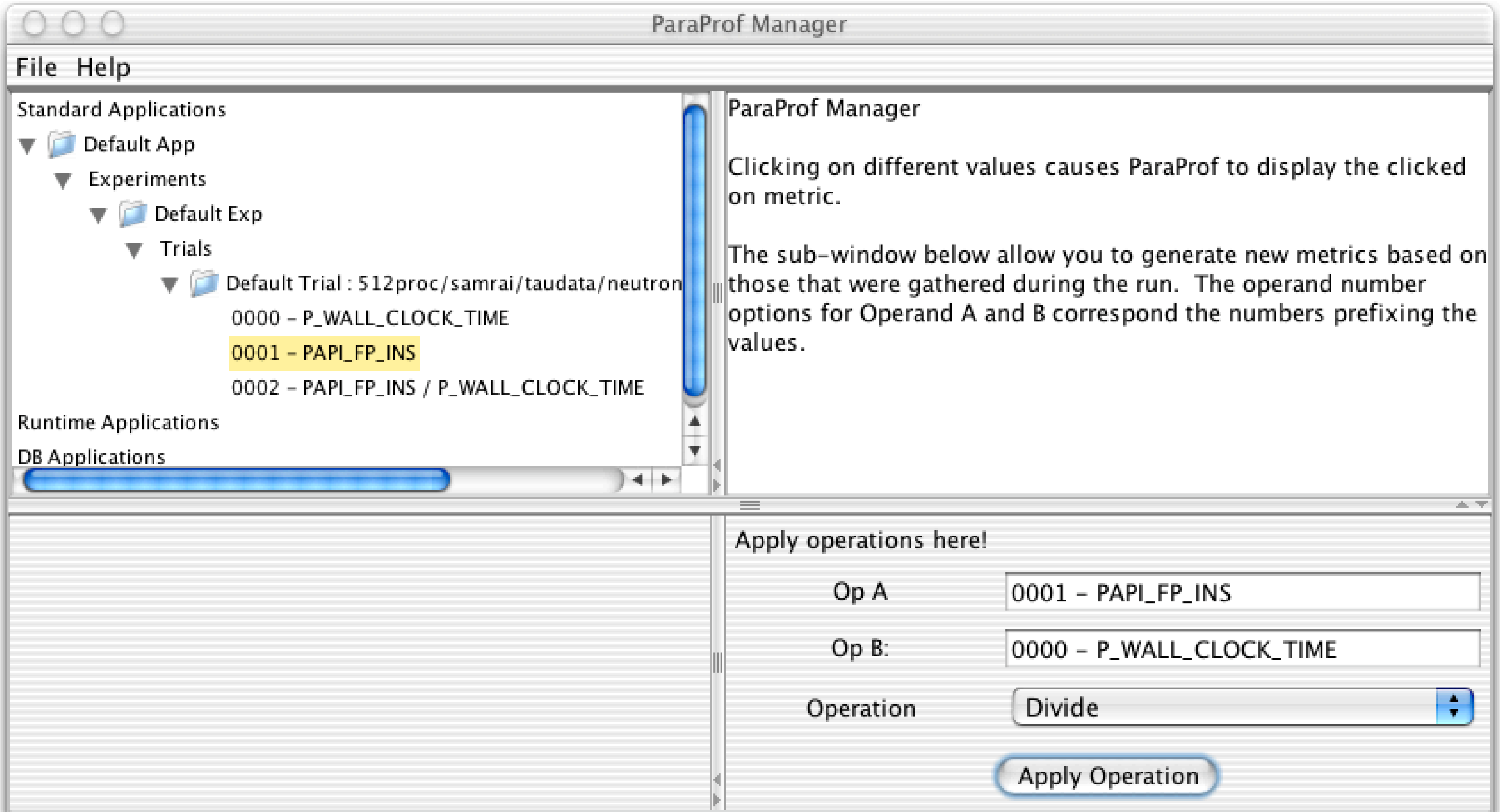
ParaProf Framework Architecture

- Portable, extensible, and scalable tool for profile analysis
- Try to offer "best of breed" capabilities to analysts
- Build as profile analysis framework for extensibility



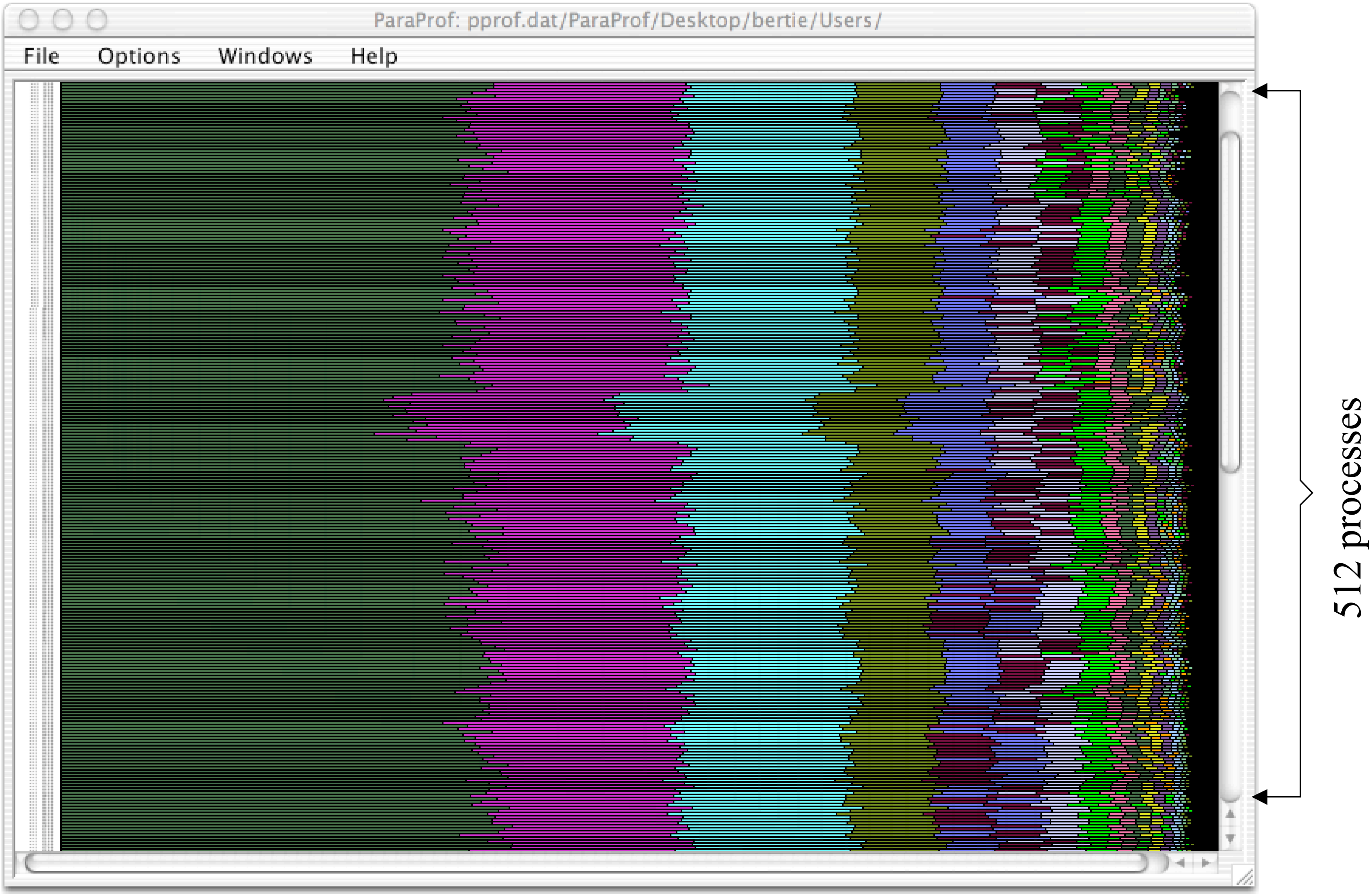


Profile Manager Window

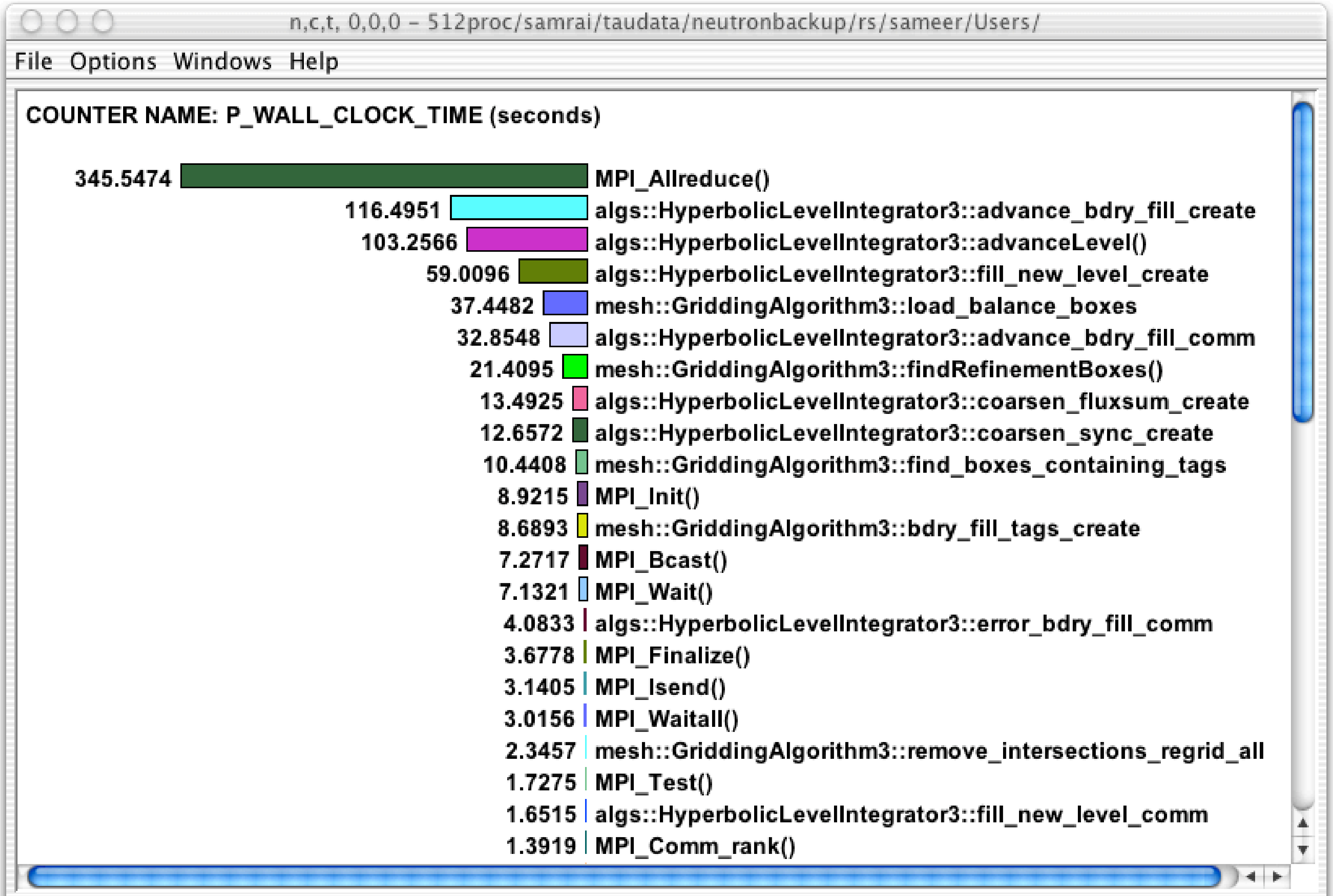


- Structured AMR toolkit (SAMRAI++), LLNL

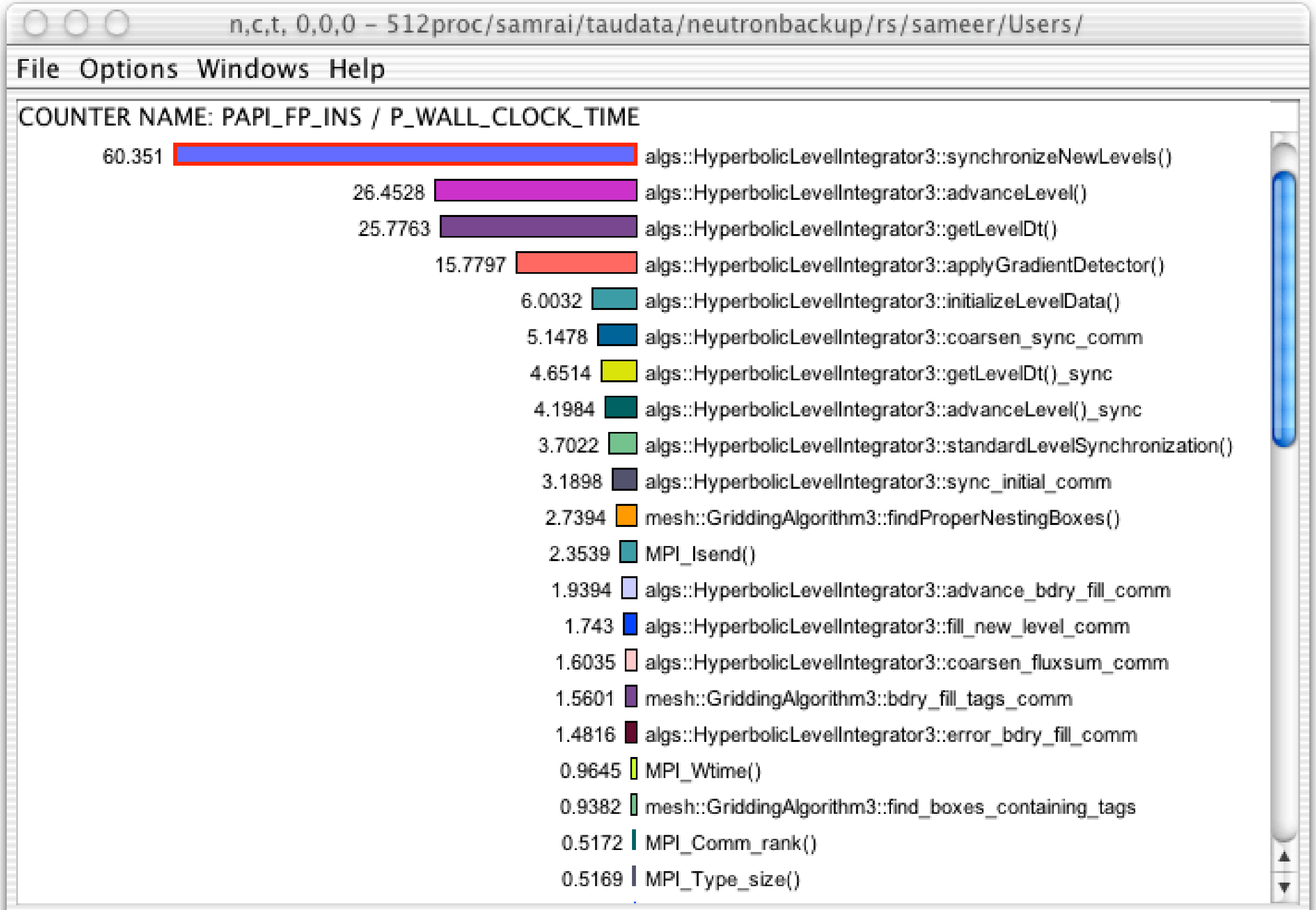
Full Profile Window (Exclusive Time)



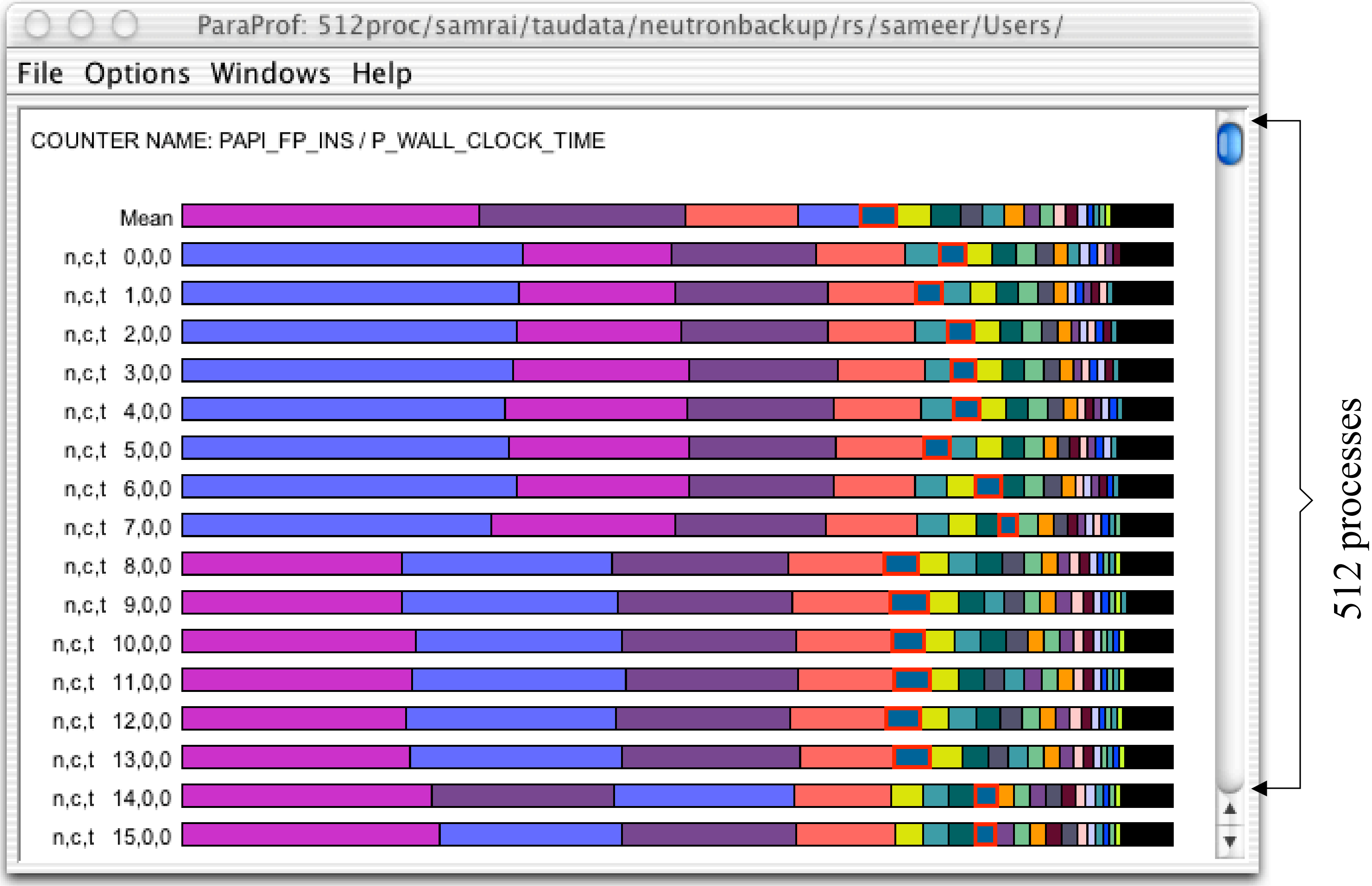
Node / Context / Thread Profile Window



Derived Metrics



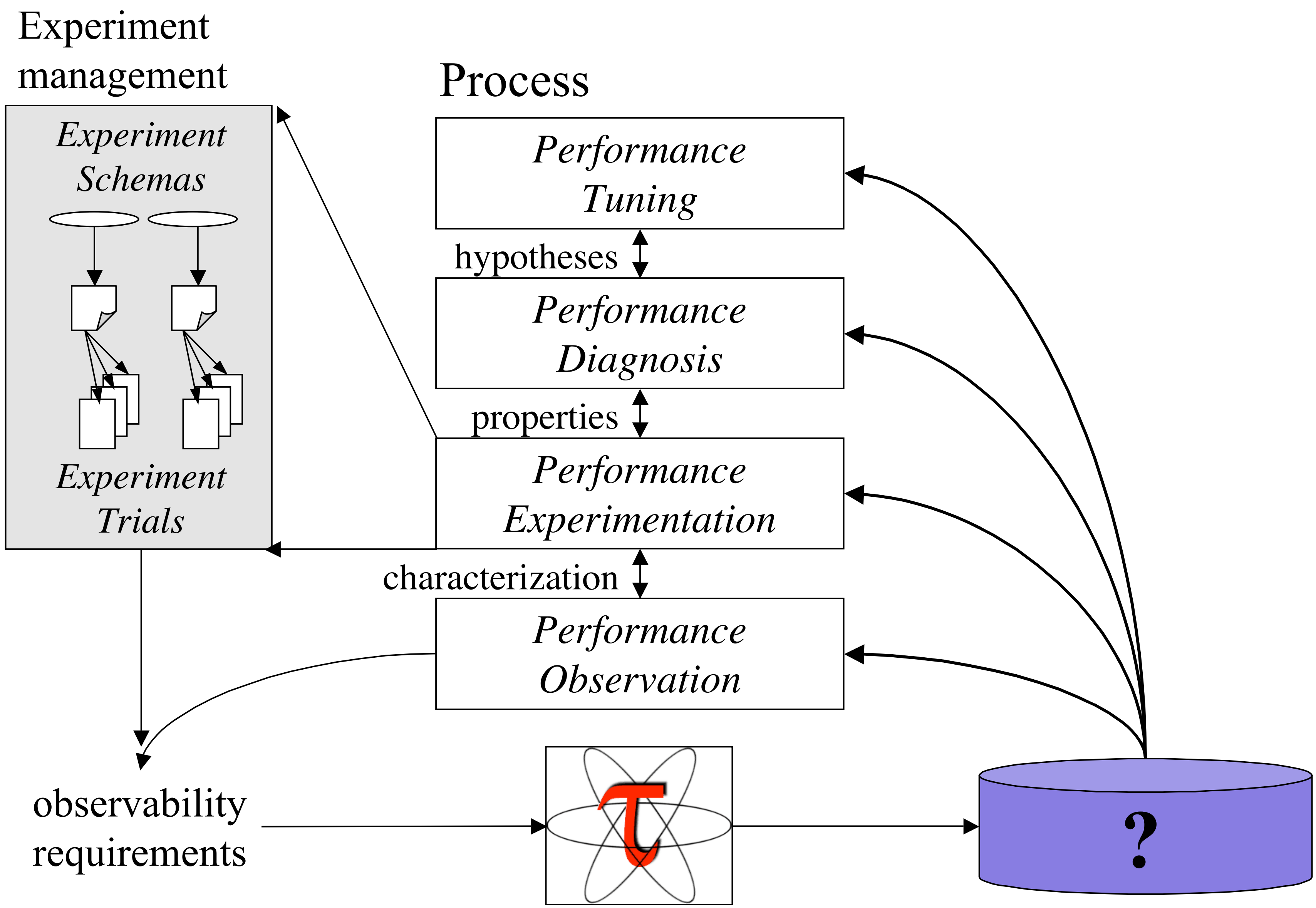
Full Profile Window (Metric-specific)



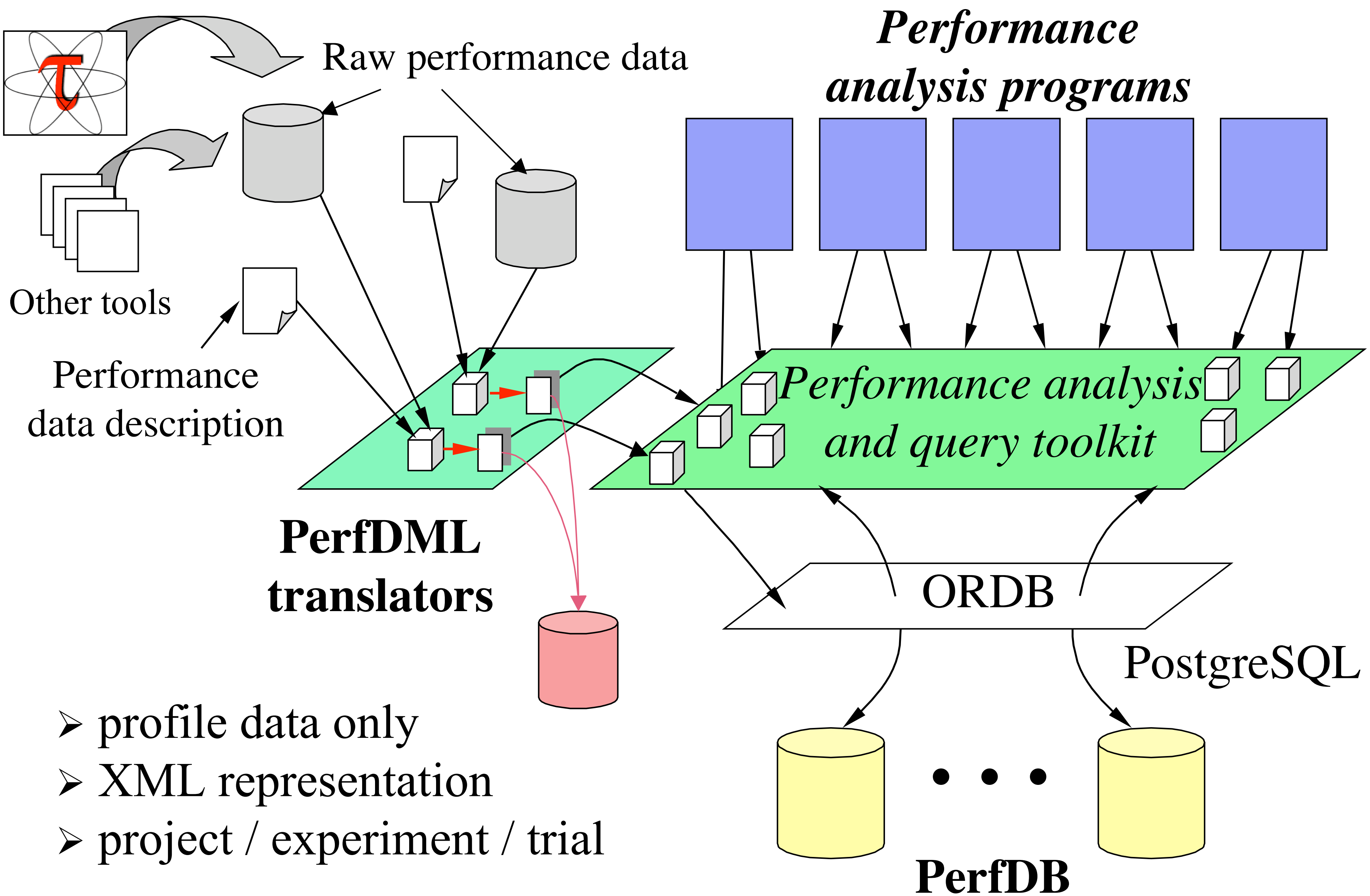
ParaProf Enhancements

- Readers completely separated from the GUI
- Access to performance profile database
- Profile translators
  - mpiP, papiprof, dynaprof
- Callgraph display
  - prof/gprof style with hyperlinks
- Integration of 3D performance plotting library
- Scalable profile analysis
  - Statistical histograms, cluster analysis, …
- Generalized programmable analysis engine
- Cross-experiment analysis

Empirical-Based Performance Optimization

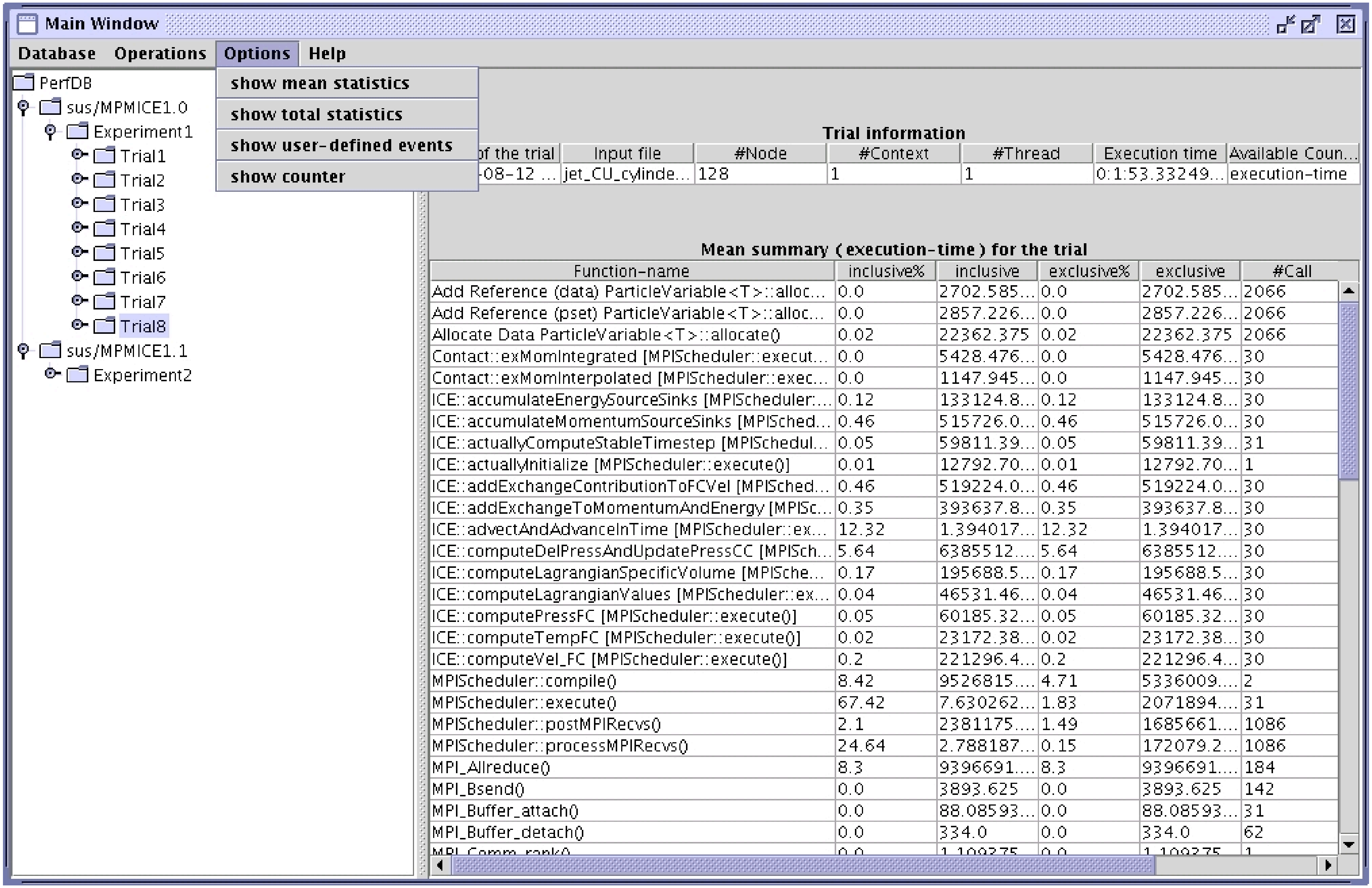


TAU Performance Database Framework

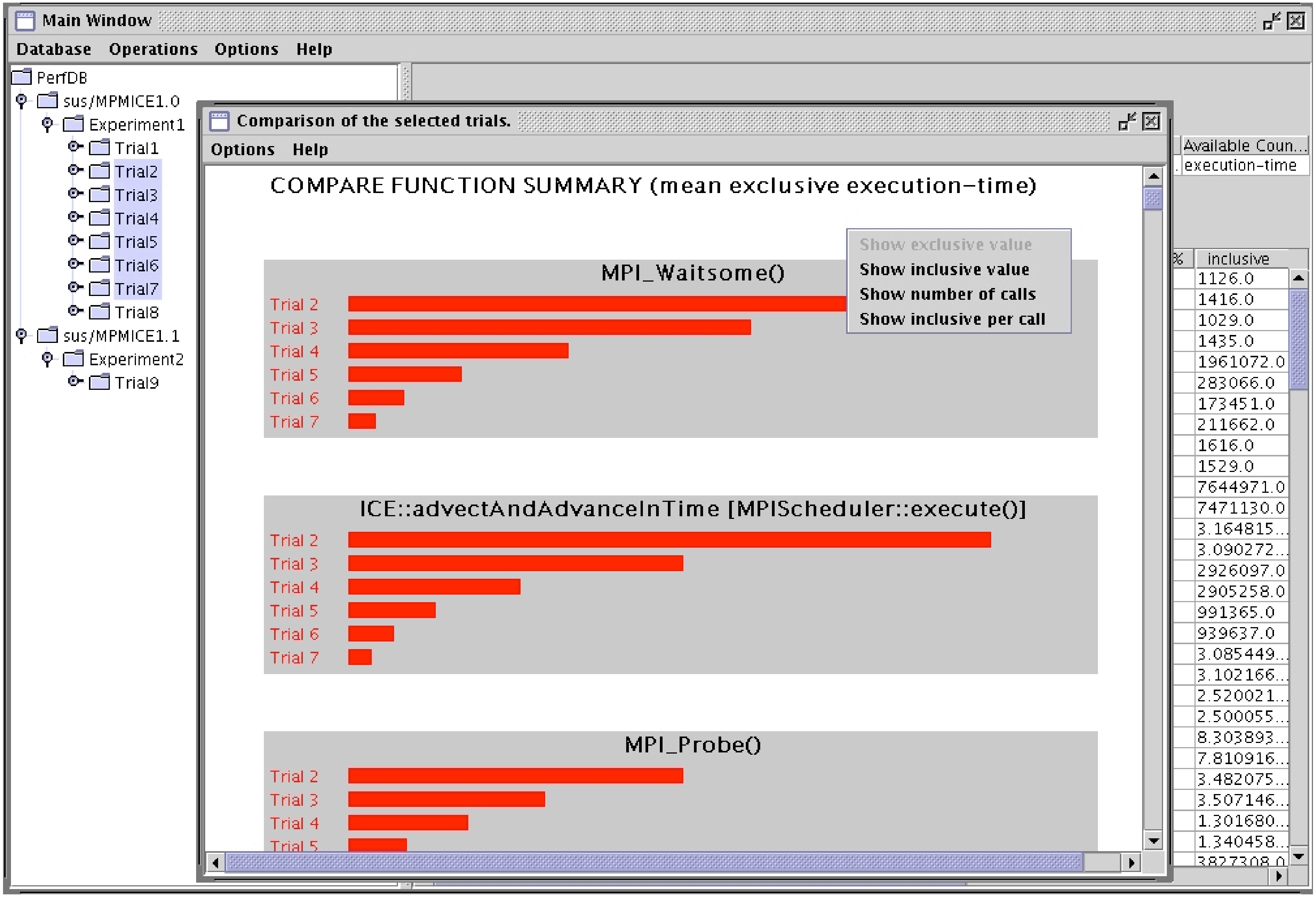


- profile data only
- XML representation
- project / experiment / trial

PerfDBF Browser



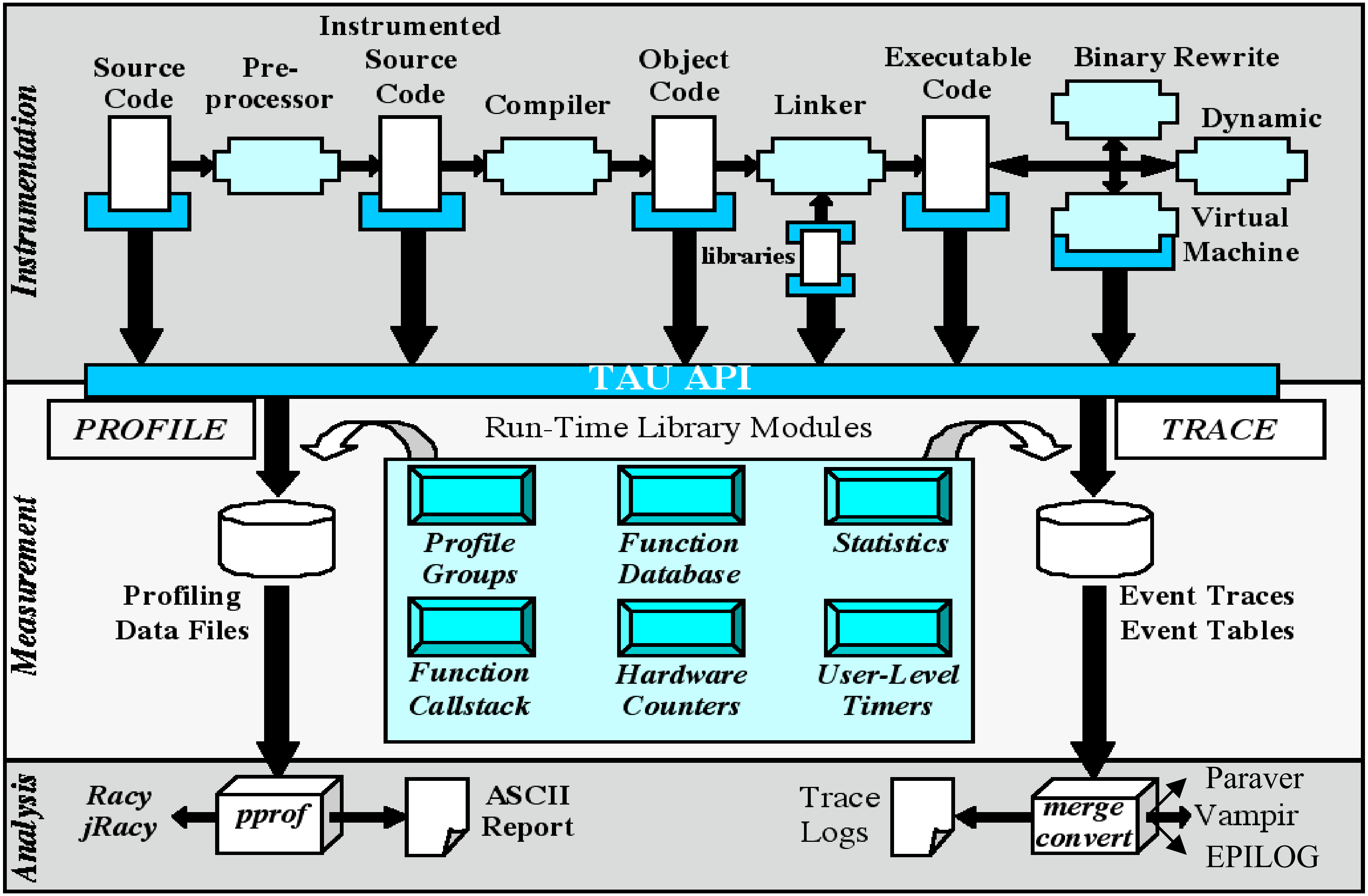
PerfDBF Cross-Trial Analysis



Using TAU – A tutorial

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

TAU Performance System Architecture



Using TAU

- Install TAU

  % configure ; make clean install
- Instrument application
  - TAU Profiling API
- Typically modify application makefile
  - include TAU's stub makefile, modify variables
- Set environment variables
  - directory where profiles/traces are to be stored
- Execute application

  % mpirun –np <procs> a.out;
- Analyze performance data
  - paraprof, vampir, pprof, paraver …

Using TAU with Vampir

- Configure TAU with -TRACE option

```
% configure -TRACE -SGITIMERS …
```

- Execute application

```
% mpirun -np 4 a.out
```

- This generates TAU traces and event descriptors
- Merge all traces using tau_merge

```
% tau_merge *.trc app.trc
```

- Convert traces to Vampir Trace format using tau_convert

```
% tau_convert -pv app.trc tau.edf app.pv
```

Note: Use -vampir instead of -pv for multi-threaded traces

- Load generated trace file in Vampir

```
% vampir app.pv
```

Description of Optional Packages

- PAPI – Measures hardware performance data e.g., floating point instructions, L1 data cache misses etc.
- DyninstAPI – Helps instrument an application binary at runtime or rewrites the binary
- EPILOG – Trace library. Epilog traces can be analyzed by EXPERT [FZJ], an automated bottleneck detection tool.
- Opari – Tool that instruments OpenMP programs
- Vampir – Commercial trace visualization tool [Pallas]
- Paraver – Trace visualization tool [CEPBA]

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
-python[inc/lib]=<dir>	Specify Python instrumentation
-epilog=<dir>	Specify location of EPILOG

TAU Measurement System Configuration

- configure [OPTIONS]
  - -TRACE — Generate binary TAU traces
  - -PROFILE (default) — Generate profiles (summary)
  - -PROFILECALLPATH — Generate call path profiles
  - -PROFILESTATS — Generate std. dev. statistics
  - -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 -c++=xlC_r –pthread
  - Use TAU with xlC_r and pthread library under AIX
  - Enable TAU profiling (default)
- ./configure -TRACE –PROFILE
  - Enable both TAU profiling and tracing
- ./configure -c++=xlC_r -cc=xlc_r
  -papi=/usr/local/packages/papi
  -pdt=/usr/local/pdtoolkit-3.1 –arch=ibm64
  -mpiinc=/usr/lpp/ppe.poe/include
  -mpilib=/usr/lpp/ppe.poe/lib -MULTIPLECOUNTERS
  - Use IBM's xlC_r and xlc_r compilers with PAPI, PDT, MPI packages and multiple counters for measurements
- Typically configure multiple measurement libraries

TAU Manual Instrumentation API for C/C++

- Initialization and runtime configuration
  - TAU_PROFILE_INIT(argc, argv);
    TAU_PROFILE_SET_NODE(myNode);
    TAU_PROFILE_SET_CONTEXT(myContext);
    TAU_PROFILE_EXIT(message);
    TAU_REGISTER_THREAD();
- Function and class methods for C++ only:
  - TAU_PROFILE(name, type, group);
- Template
  - TAU_TYPE_STRING(variable, type);
    TAU_PROFILE(name, type, group);
    CT(variable);
- User-defined timing
  - TAU_PROFILE_TIMER(timer, name, type, group);
    TAU_PROFILE_START(timer);
    TAU_PROFILE_STOP(timer);

TAU Measurement API (continued)

- User-defined events
  - TAU_REGISTER_EVENT(variable, event_name);
    TAU_EVENT(variable, value);
    TAU_PROFILE_STMT(statement);
- Heap Memory Tracking:
  - TAU_TRACK_MEMORY();
  - TAU_SET_INTERRUPT_INTERVAL(seconds);
  - TAU_DISABLE_TRACKING_MEMORY();
  - TAU_ENABLE_TRACKING_MEMORY();
- Reporting
  - TAU_REPORT_STATISTICS();
  - TAU_REPORT_THREAD_STATISTICS();

Manual Instrumentation – C++ Example

```
#include <TAU.h>
int main(int argc, char **argv)
{
  TAU_PROFILE("int main(int, char **)", " ", TAU_DEFAULT);
  TAU_PROFILE_INIT(argc, argv);
  TAU_PROFILE_SET_NODE(0); /* for sequential programs */
  foo();
  return 0;
}

int foo(void)
{
  TAU_PROFILE("int foo(void)", " ", TAU_DEFAULT); // measures entire foo()
  TAU_PROFILE_TIMER(t, "foo(): for loop", "[23:45 file.cpp]", TAU_USER);
  TAU_PROFILE_START(t);
  for(int i = 0; i < N ; i++){
    work(i);
  }
  TAU_PROFILE_STOP(t);
  // other statements in foo …
}
```

Manual Instrumentation – C Example

```
#include <TAU.h>
int main(int argc, char **argv)
{
  TAU_PROFILE_TIMER(tmain, "int main(int, char **)", " ", TAU_DEFAULT);
  TAU_PROFILE_INIT(argc, argv);
  TAU_PROFILE_SET_NODE(0); /* for sequential programs */
  TAU_PROFILE_START(tmain);
  foo();
  …
  TAU_PROFILE_STOP(tmain);
  return 0;
}
int foo(void)
{
  TAU_PROFILE_TIMER(t, "foo()", " ", TAU_USER);
  TAU_PROFILE_START(t);
  for(int i = 0; i < N ; i++){
    work(i);
  }
  TAU_PROFILE_STOP(t);
}
```

Manual Instrumentation – F90 Example

```
cc34567 Cubes program - comment line
      PROGRAM SUM_OF_CUBES
       integer profiler(2)
       save profiler
      INTEGER :: H, T, U
        call TAU_PROFILE_INIT()
        call TAU_PROFILE_TIMER(profiler, 'PROGRAM SUM_OF_CUBES')
        call TAU_PROFILE_START(profiler)
        call TAU_PROFILE_SET_NODE(0)
      ! 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 PROGRAM SUM_OF_CUBES
```

Compiling

```
% configure [options]
% make clean install
```

Creates <arch>/lib/Makefile.tau<options> stub Makefile

and <arch>/lib/libTau<options>.a [.so] libraries which defines a single configuration of TAU

Compiling: TAU Makefiles

- Include TAU Stub Makefile (<arch>/lib) in the user's Makefile.
- 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
- Note: Not including TAU_DEFS in CFLAGS disables instrumentation in C/C++ programs (TAU_DISABLE for f90).

Including TAU Makefile - C++ Example

```
include /galaxy/wompat/sameer/tau-2.13.5/sgi64/lib/Makefile.tau-pdt
F90 = $(TAU_CXX)
CC  = $(TAU_CC)
CFLAGS = $(TAU_DEFS) $(TAU_INCLUDE)
LIBS = $(TAU_LIBS)
OBJS = ...
TARGET= a.out
TARGET: $(OBJS)
	$(CXX) $(LDFLAGS) $(OBJS) -o $@ $(LIBS)
.cpp.o:
	$(CC) $(CFLAGS) -c $< -o $@
```

Including TAU Makefile - F90 Example

```
include /galaxy/wompat/sameer/tau-2.13.5/solaris2/lib/Makefile.tau-pdt
F90 = $(TAU_F90)
FFLAGS = -I<dir>
LIBS = $(TAU_LIBS) $(TAU_CXXLIBS)
OBJS = ...
TARGET= a.out
TARGET: $(OBJS)
	$(F90) $(LDFLAGS) $(OBJS) -o $@ $(LIBS)
.f.o:
	$(F90) $(FFLAGS) -c $< -o $@
```

Including TAU Makefile - F90 Example

```
include /galaxy/wompat/sameer/tau-2.13.5/sgi64/lib/Makefile.tau-pdt
F90 = $(TAU_F90)
FFLAGS = -I<dir>
LIBS = $(TAU_LIBS) $(TAU_CXXLIBS)
OBJS = ...
TARGET= a.out
TARGET: $(OBJS)
	$(F90) $(LDFLAGS) $(OBJS) -o $@ $(LIBS)
.f.o:
	$(F90) $(FFLAGS) -c $< -o $@
```

Using TAU's Malloc Wrapper Library for C/C++

```
include /galaxy/wompat/sameer/tau-2.13.5/sgi64/lib/Makefile.tau-pdt
CC=$(TAU_CC)
CFLAGS=$(TAU_DEFS) $(TAU_INCLUDE) $(TAU_MEMORY_INCLUDE)
LIBS = $(TAU_LIBS)
OBJS = f1.o f2.o ...
TARGET= a.out
TARGET: $(OBJS)
	$(F90) $(LDFLAGS) $(OBJS) -o $@ $(LIBS)
.c.o:
	$(CC) $(CFLAGS) -c $< -o $@
```

TAU's malloc/free wrapper

```
#include <TAU.h>
#include <malloc.h>
int main(int argc, char **argv)
{
  TAU_PROFILE("int main(int, char **)", " ", TAU_DEFAULT);

  int *ary = (int *) malloc(sizeof(int) * 4096);

  // TAU's malloc wrapper library replaces this call automatically
  // when $(TAU_MEMORY_INCLUDE) is used in the Makefile.
  …
  free(ary);
  // other statements in foo …
}
```

Using TAU's Malloc Wrapper Library for C/C++

NumSamples	MaxValue	MinValue	MeanValue	name
1	40016.0	40016.0	40016.0	malloc size <file=main.cpp, line=252>
1	40016.0	40016.0	40016.0	free size <file=main.cpp, line=298>
12	30000.0	240.0	5590.0	malloc size <file=select.cpp, line=80>
12	30000.0	240.0	5590.0	malloc size <file=select.cpp, line=81>
3	30000.0	6000.0	17000.0	free size <file=select.cpp, line=107>
3	30000.0	6000.0	17000.0	free size <file=select.cpp, line=109>
1	8000.0	8000.0	8000.0	malloc size <file=main.cpp, line=258>
1	8000.0	8000.0	8000.0	free size <file=main.cpp, line=299>
7	6000.0	600.0	2228.5714	free size <file=select.cpp, line=118>
7	6000.0	600.0	2228.5714	free size <file=select.cpp, line=119>
2	240.0	240.0	240.0	free size <file=select.cpp, line=126>
2	240.0	240.0	240.0	free size <file=select.cpp, line=128>

Using TAU – A tutorial

- Configuration
- Instrumentation
  - Manual
  - PDT- Source rewriting for C,C++, F77/90/95
  - MPI – Wrapper interposition library
  - OpenMP – Directive rewriting
- Measurement
- Performance Analysis

Using Program Database Toolkit (PDT)

Step I: Configure PDT:

```
% configure -arch=ibm64 -XLC
% make clean; make install
```

Builds <pdtdir>/<arch>/bin/cxxparse, cparse, f90parse and f95parse

Builds <pdtdir>/<arch>/lib/libpdb.a. See <pdtdir>/README file.

Step II: Configure TAU with PDT for auto-instrumentation of source code:

```
% configure -arch=ibm64 -c++=xlC -cc=xlc
  -pdt=/usr/contrib/TAU/pdtoolkit-3.1
% make clean; make install
```

Builds <taudir>/<arch>/bin/tau_instrumentor,

<taudir>/<arch>/lib/Makefile.tau<options> and libTau<options>.a

See <taudir>/INSTALL file.

TAU Makefile for PDT (C++)

```
include /usr/tau/include/Makefile
CXX = $(TAU_CXX)
CC  = $(TAU_CC)
PDTPARSE = $(PDTDIR)/$(PDTARCHDIR)/bin/cxxparse
TAUINSTR = $(TAUROOT)/$(CONFIG_ARCH)/bin/tau_instrumentor
CFLAGS = $(TAU_DEFS) $(TAU_INCLUDE)
LIBS = $(TAU_LIBS)
OBJS = ...
TARGET= a.out
TARGET: $(OBJS)
      $(CXX) $(LDFLAGS) $(OBJS) -o $@ $(LIBS)
.cpp.o:
      $(PDTPARSE) $<
      $(TAUINSTR) $*.pdb $< -o $*.inst.cpp -f select.dat
      $(CC) $(CFLAGS) -c $*.inst.cpp -o $@
```

TAU Makefile for PDT (F90)

```
include /wompat/sameer/tau 2.13.5/solaris2/lib/Makefile.tau-pdt
F90 = $(TAU_F90)
CC  = $(TAU_CC)
PDTPARSE = $(PDTDIR)/$(PDTARCHDIR)/bin/f95parse
TAUINSTR = $(TAUROOT)/$(CONFIG_ARCH)/bin/tau_instrumentor
LIBS = $(TAU_LIBS) $(TAU_CXXLIBS)
OBJS = ...
TARGET= f1.o f2.o f3.o
PDB=merged.pdb
TARGET:$(PDB) $(OBJS)
      $(F90) $(LDFLAGS) $(OBJS) -o $@ $(LIBS)
$(PDB): $(OBJS:.o=.f)
      $(PDTF95PARSE) $(OBJS:.o=.f) -o$(PDB) -R free
# This expands to f95parse *.f -omerged.pdb -R free
.f.o:
      $(TAU_INSTR) $(PDB) $< -o $*.inst.f -f sel.dat;\
      $(FCOMPILE) $*.inst.f -o $@;
```

Using PDT: tau_instrumentor

```
% 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
```

Using TAU – A tutorial

- Configuration
- Instrumentation
  - Manual
  - PDT- Source rewriting for C,C++, F77/90/95
  - MPI – Wrapper interposition library
  - OpenMP – Directive rewriting
- Measurement
- Performance Analysis

Using MPI Wrapper Interposition Library

Step I: Configure TAU with MPI:

```
% configure -mpiinc=/usr/lpp/ppe.poe/include
  -mpilib=/usr/lpp/ppe.poe/lib -arch=ibm64 -c++=CC -cc=cc
  -pdt=$PET_HOME/PTOOLS/pdtoolkit-3.1
% make clean; make install
```

Builds <taudir>/<arch>/lib/libTauMpi<options>,

<taudir>/<arch>/lib/Makefile.tau<options> and libTau<options>.a

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

Including TAU's stub Makefile

```
include /galaxy/wompat/tau-2.13.5/sgi64/lib/Makefile.tau-mpi-pdt
F90 = $(TAU_F90)
CC  = $(TAU_CC)
LIBS = $(TAU_MPI_LIBS) $(TAU_LIBS) $(TAU_CXXLIBS)
LD_FLAGS = $(TAU_LDFLAGS)
OBJS = ...
TARGET= a.out
TARGET: $(OBJS)
     $(CXX) $(LDFLAGS) $(OBJS) -o $@ $(LIBS)
.f.o:
     $(F90) $(FFLAGS) -c $< -o $@
```

Including TAU's stub Makefile with PAPI

```
include /galaxy/wompat/sameer/tau-2.13.5/sgi64/lib/Makefile.tau-papiwallclock-multiplecounters-papivirtual-mpi-papi-pdt
CC   = $(TAU_CC)
LIBS = $(TAU_MPI_LIBS) $(TAU_LIBS) $(TAU_CXXLIBS)
LD_FLAGS = $(TAU_LDFLAGS)
OBJS = ...
TARGET= a.out
TARGET: $(OBJS)
     $(CXX) $(LDFLAGS) $(OBJS) -o $@ $(LIBS)
.f.o:
     $(F90) $(FFLAGS) -c $< -o $@
```

Setup: Running Applications

```
% set path=($path <taudir>/<arch>/bin)
% set path=($path $PET_HOME/PTOOLS/tau-2.13.5/src/rs6000/bin)
% setenv LD_LIBRARY_PATH
$LD_LIBRARY_PATH\:<taudir>/<arch>/lib

For PAPI (1 counter, if multiplecounters is not used):
% setenv PAPI_EVENT PAPI_L1_DCM (PAPI's Level 1 Data cache
misses)
For PAPI (multiplecounters):
% setenv COUNTER1 PAPI_FP_INS     (PAPI's Floating point ins)
% setenv COUNTER2 PAPI_TOT_CYC    (PAPI's Total cycles)
% setenv COUNTER3 P_VIRTUAL_TIME    (PAPI's virtual time)
% setenv COUNTER4 LINUX_TIMERS     (Wallclock time)
% mpirun -np <n> <application>
% paraprof   (for performance analysis)
```

Using TAU with Vampir

```
include /galaxy/wompat/sameer/tau-
2.13.5/rs6000/lib/Makefile.tau-mpi-pdt-trace
F90 = $(TAU_F90)
LIBS = $(TAU_MPI_LIBS) $(TAU_LIBS) $(TAU_CXXLIBS)
OBJS = ...
TARGET= a.out
TARGET: $(OBJS)
	$(CXX) $(LDFLAGS) $(OBJS) -o $@ $(LIBS)
.f.o:
	$(F90) $(FFLAGS) -c $< -o $@
```

Using TAU with Vampir

```
% llsubmit job.sh
% ls *.trc *.edf
```

Merging Trace Files

```
% tau_merge tau*.trc app.trc
```

Converting TAU Trace Files to Vampir and Paraver Trace formats

```
% tau_convert -pv app.trc tau.edf app.pv
  (use -vampir if application is multi-threaded)
% vampir app.pv
% tau_convert -paraver app.trc tau.edf app.par
  (use -paraver -t if application is multi-threaded)
% paraver app.par
```

TAU Makefile for PDT with MPI and F90

```
include /wompat/tau-2.13.5/rs6000/lib/Makefile.tau-mpi-pdt
FCOMPILE = $(TAU_F90) $(TAU_MPI_INCLUDE)
PDTF95PARSE = $(PDTDIR)/$(PDTARCHDIR)/bin/f95parse
TAUINSTR = $(TAUROOT)/$(CONFIG_ARCH)/bin/tau_instrumentor
PDB=merged.pdb
COMPILE_RULE= $(TAU_INSTR) $(PDB) $< -o $*.inst.f -f sel.dat;\
       $(FCOMPILE) $*.inst.f -o $@;
LIBS = $(TAU_MPI_FLIBS) $(TAU_LIBS) $(TAU_CXXLIBS)
OBJS = f1.o f2.o f3.o …
TARGET= a.out
TARGET: $(PDB) $(OBJS)
       $(TAU_F90) $(LDFLAGS) $(OBJS) -o $@ $(LIBS)
$(PDB): $(OBJS:.o=.f)
       $(PDTF95PARSE) $(OBJS:.o=.f) $(TAU_MPI_INCLUDE) -o$(PDB)
# This expands to f95parse *.f -I…/mpi/include -omerged.pdb
.f.o:
       $(COMPILE_RULE)
```

Using TAU – A tutorial

- Configuration
- Instrumentation
  - Manual
  - PDT- Source rewriting for C,C++, F77/90/95
  - MPI – Wrapper interposition library
  - OpenMP – Directive rewriting
- Measurement
- Performance Analysis

Using Opari with TAU

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

```
% cd kojak-1.0; 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=/galaxy/wompat/sameer/kojak/sun/kojak-1.0/opari
  -mpiinc=/usr/include
  -mpilib=/usr/lib
  -pdt=/galaxy/wompat/sameer/pdtoolkit-3.1
% make clean; make install
```

Instrumentation of OpenMP Constructs

- **O**penMP **P**ragma **A**nd **R**egion **I**nstrumentor
- 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`***)
- Work in Progress:
  Investigating standardization through OpenMP Forum

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);
#line 261 "stommel.c"
```

OPARI: Basic Usage (f90)

- Reset OPARI state information
  - ```
    rm -f opari.rc
    ```
- Call OPARI for each input source file
  - ```
    opari file1.f90
    ...
    opari fileN.f90
    ```
- Generate OPARI runtime table, compile it with ANSI C
  - ```
    opari -table opari.tab.c
    cc -c opari.tab.c
    ```
- Compile modified files `*.mod.f90` using OpenMP
- Link the resulting object files, the OPARI runtime table `opari.tab.o` and the TAU POMP RTL
    ```

OPARI: Makefile Template (C/C++)

```
OMPCC  = ...          # insert C OpenMP compiler here
OMPCXX = ...          # insert C++ OpenMP compiler here

.c.o:
        opari $<
        $(OMPCC) $(CFLAGS) -c $*.mod.c
.cc.o:
        opari $<
        $(OMPCXX) $(CXXFLAGS) -c $*.mod.cc

opari.init:
        rm -rf opari.rc

opari.tab.o:
        opari -table opari.tab.c
        $(CC) -c opari.tab.c

myprog: opari.init myfile*.o ... opari.tab.o
        $(OMPCC) -o myprog myfile*.o opari.tab.o -lpomp

myfile1.o: myfile1.c myheader.h
myfile2.o: ...
```

OPARI: Makefile Template (Fortran)

```
OMPF77 = ...           # insert f77 OpenMP compiler here
OMPF90 = ...           # insert f90 OpenMP compiler here

.f.o:
        opari $<
        $(OMPF77) $(CFLAGS) -c $*.mod.F
.f90.o:
        opari $<
        $(OMPF90) $(CXXFLAGS) -c $*.mod.F90

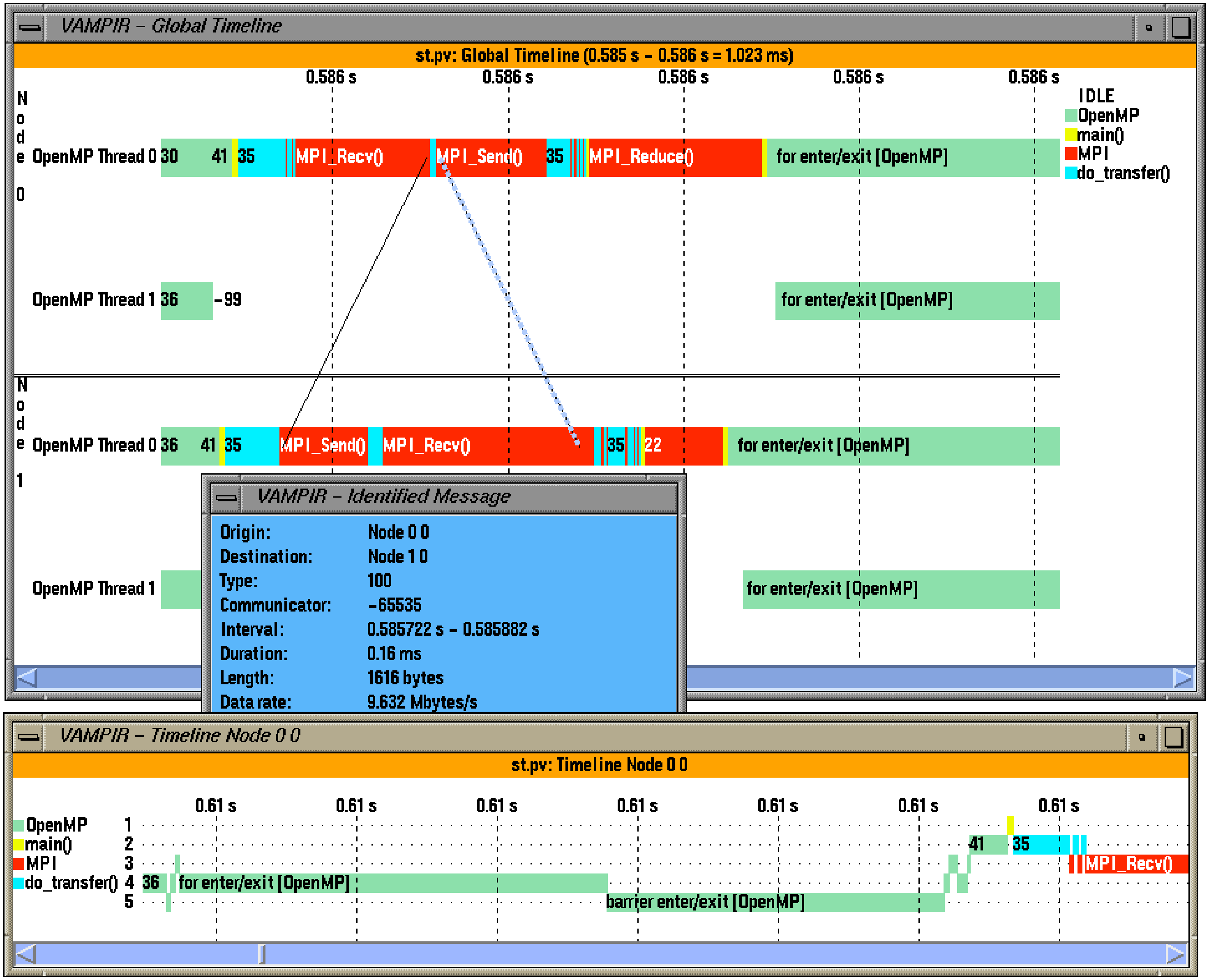
opari.init:
        rm -rf opari.rc

opari.tab.o:
        opari -table opari.tab.c
        $(CC) -c opari.tab.c

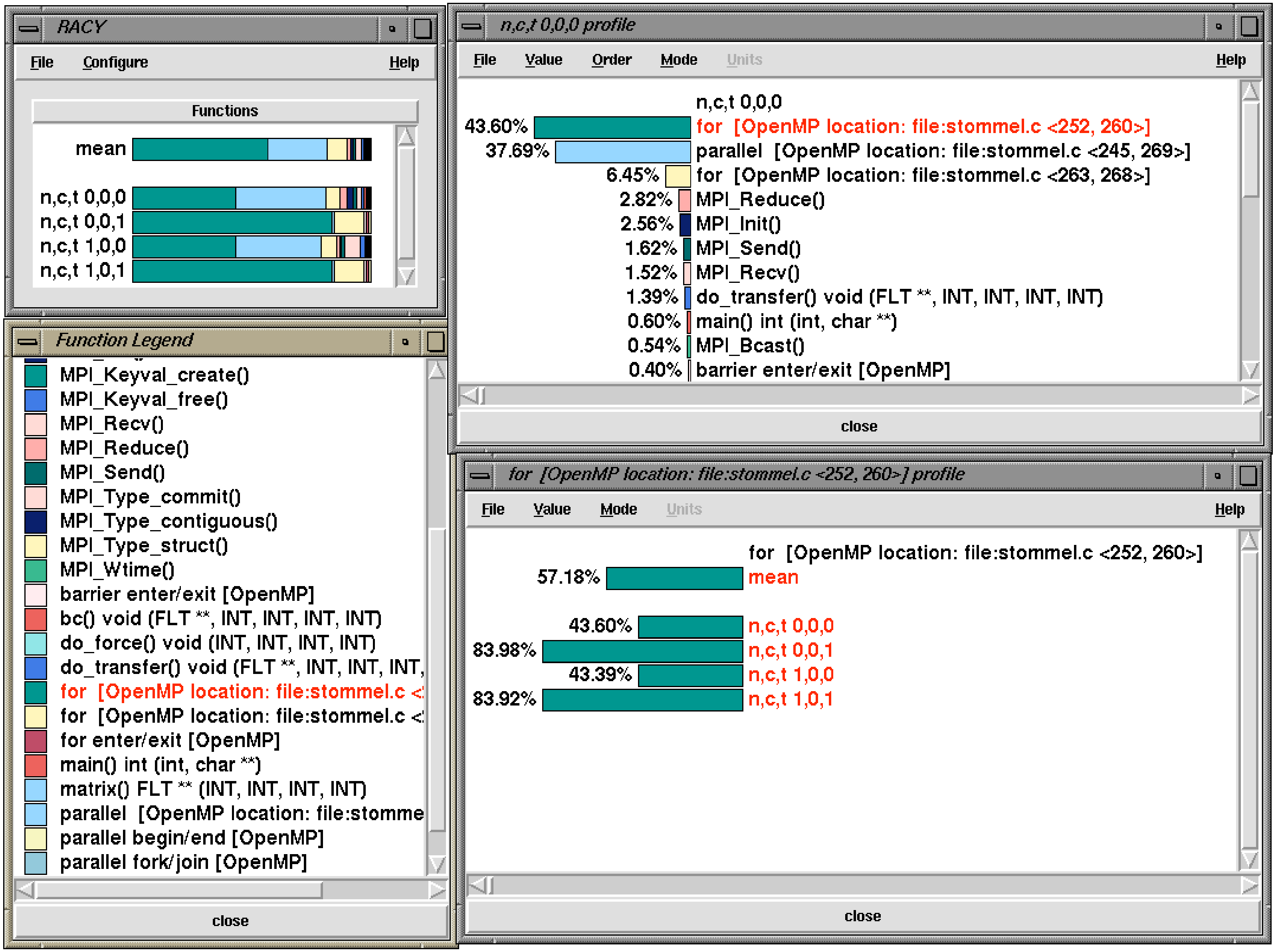
myprog: opari.init myfile*.o ... opari.tab.o
        $(OMPF90) -o myprog myfile*.o opari.tab.o -lpomp

myfile1.o: myfile1.f90
myfile2.o: ...
```

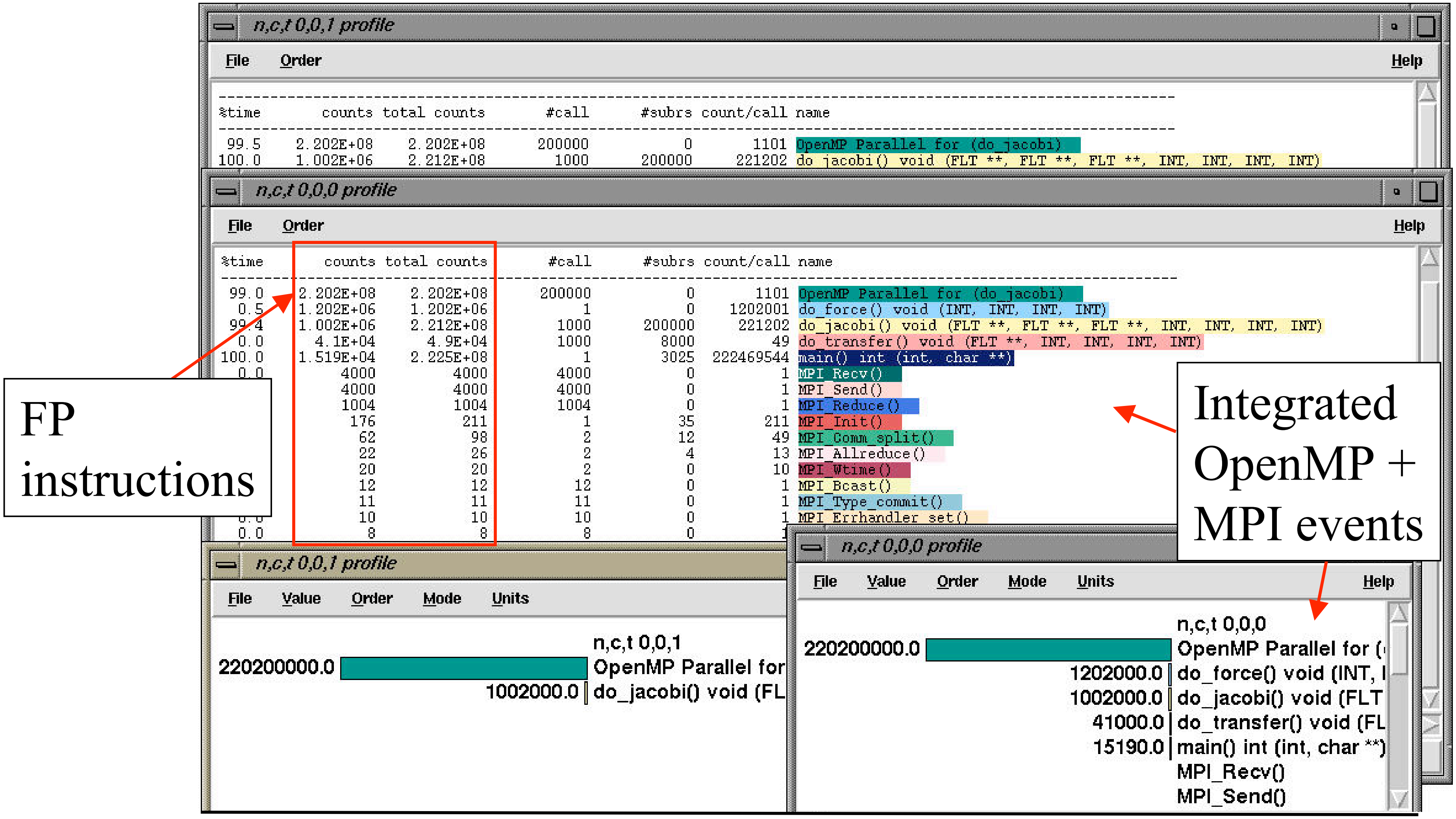
Tracing Hybrid Executions – TAU and Vampir



Profiling Hybrid Executions



OpenMP + MPI Ocean Modeling (HW Profile)



% configure -papi=../packages/papi -openmp -c++=pgCC -cc=pgcc
-mpiinc=../packages/mpich/include -mpilib=../packages/mpich/lib

TAU Performance System Status

- Computing platforms (selected)
  - IBM SP / pSeries, SGI Origin 2K/3K, Cray T3E / SV-1 / X1, HP (Compaq) SC (Tru64), Sun, Hitachi SR8000, NEC SX-5/6, Linux clusters (IA-32/64, Alpha, PPC, PA-RISC, Power, Opteron), Apple (G4/5, OS X), Windows
- Programming languages
  - C, C++, Fortran 77/90/95, HPF, Java, OpenMP, Python
- Thread libraries
  - pthreads, SGI sproc, Java,Windows, OpenMP
- Compilers (selected)
  - Intel KAI (KCC, KAP/Pro), PGI, GNU, Fujitsu, Sun, Microsoft, SGI, Cray, IBM (xlc, xlf), Compaq, NEC, Intel

Concluding Remarks

- Complex parallel systems and software pose challenging performance analysis problems that require robust methodologies and tools
- To build more sophisticated performance tools, existing proven performance technology must be utilized
- Performance tools must be integrated with software and systems models and technology
  - Performance engineered software
  - Function consistently and coherently in software and system environments
- TAU performance system offers robust performance technology that can be broadly integrated

Support Acknowledgements

- Department of Energy (DOE)
  - Office of Science contracts
  - University of Utah DOE ASCI Level 1 sub-contract
  - DOE ASCI Level 3 (LANL, LLNL)
- NSF National Young Investigator (NYI) award
- Research Centre Juelich
  - John von Neumann Institute for Computing
  - Dr. Bernd Mohr
- Los Alamos National Laboratory