TAU Performance Toolkit

(WOMPAT 2004 OpenMP Lab)

Sameer Shende, Allen D. Malony

University of Oregon

{sameer, malony}@cs.uoregon.edu

Tuning and Analysis Utilities

UNIVERSITY OF OREGON

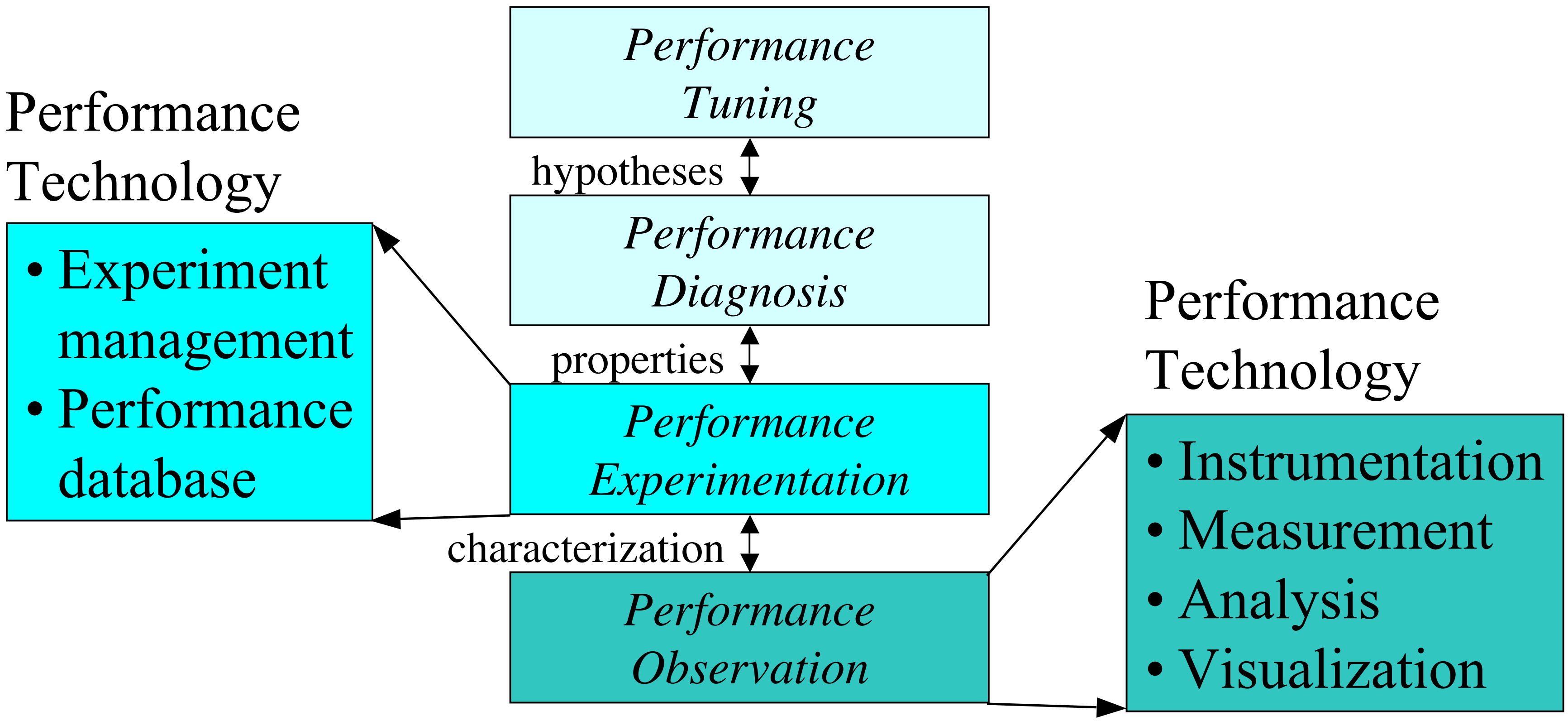
Los Alamos NATIONAL LABORATORY

John von Neumann - Institut für Computing NIC

Zentralinstitut für Angewandte Mathematik

Research Motivation

- Tools for performance problem solving
  - Empirical-based performance optimization process
  - Performance technology concerns



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 Systems 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 and applications

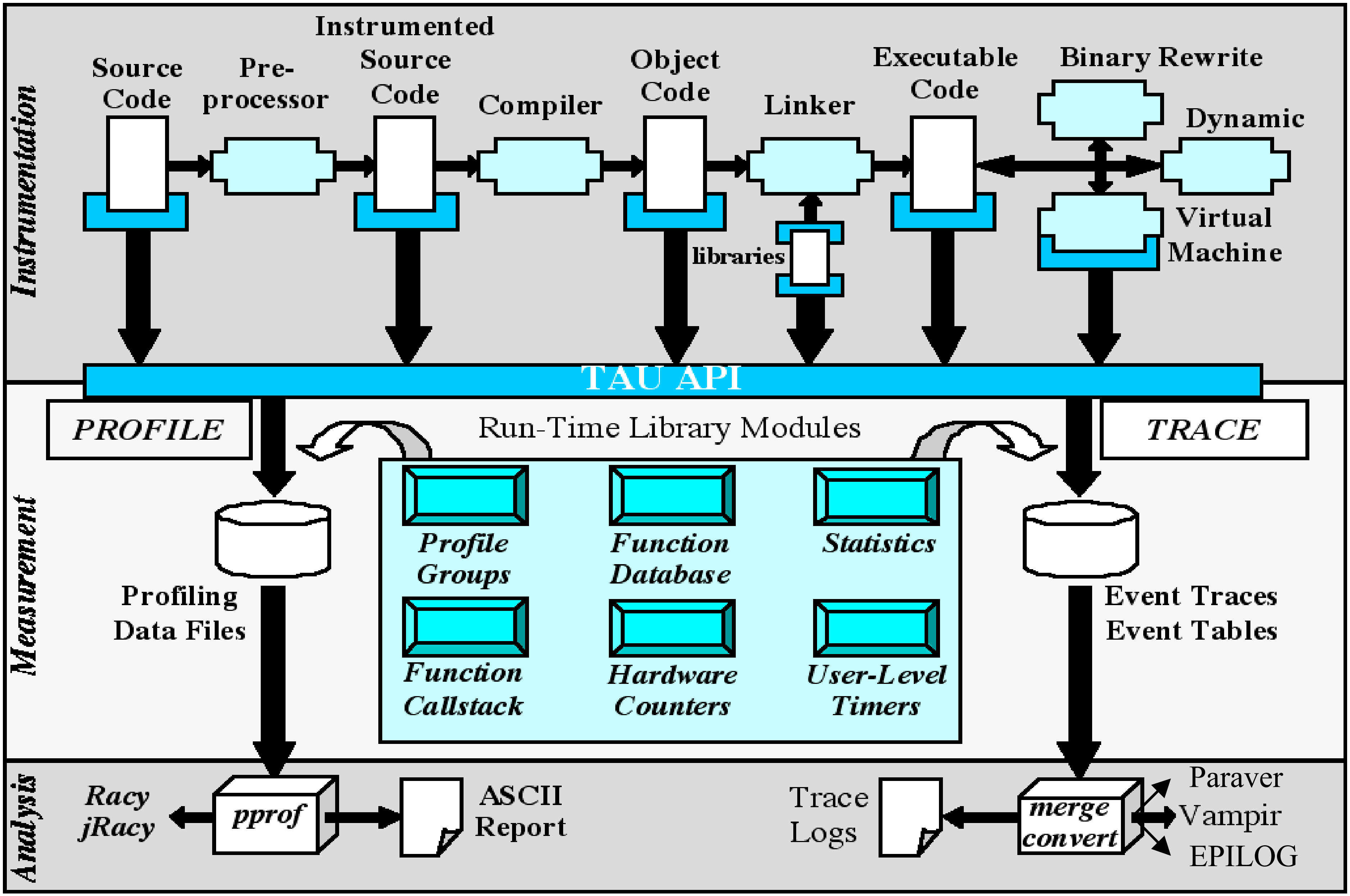
Definitions – Profiling

- 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 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*)

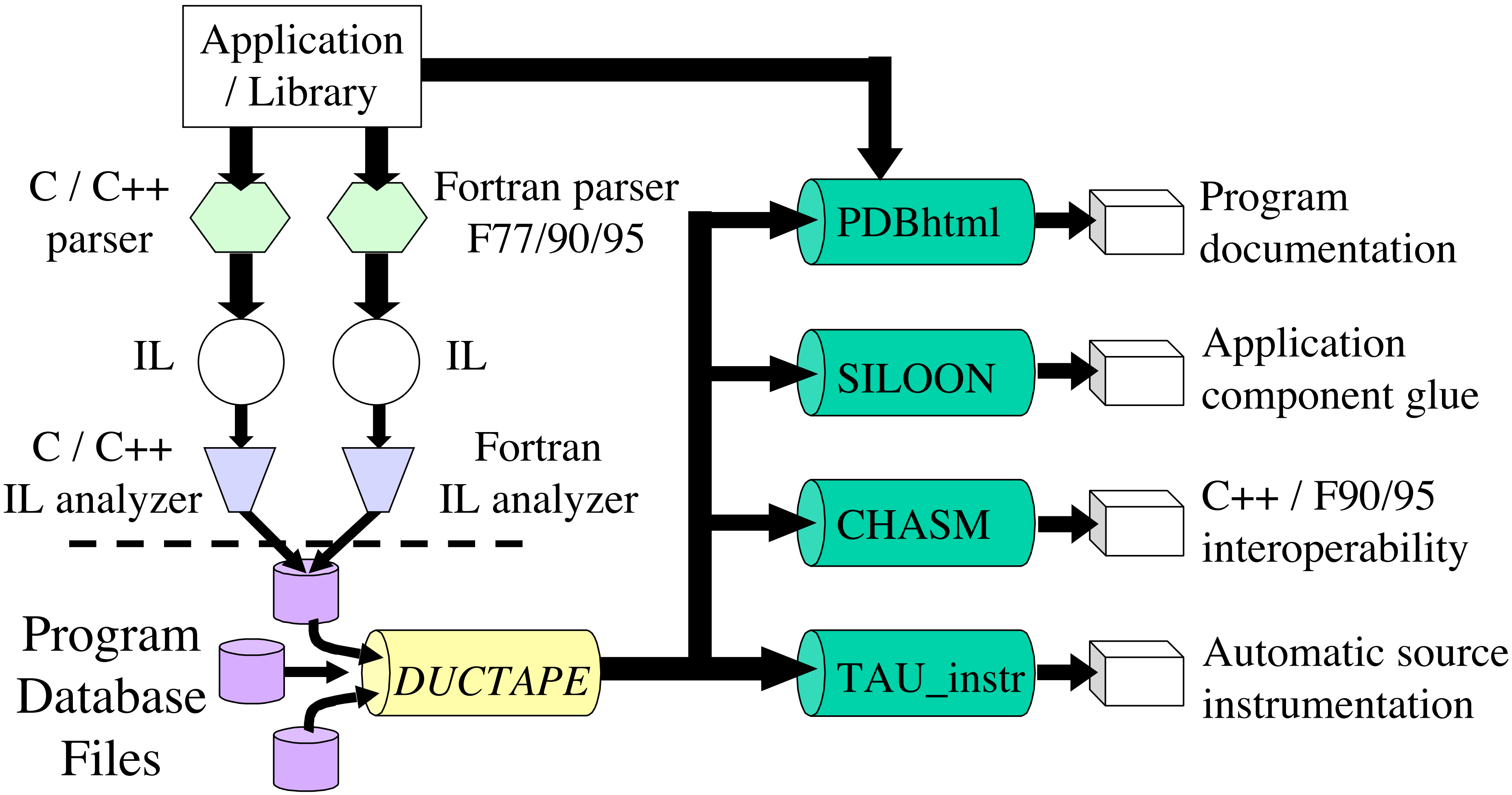
Multi-Level Instrumentation

- 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

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

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.sgi 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
```

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

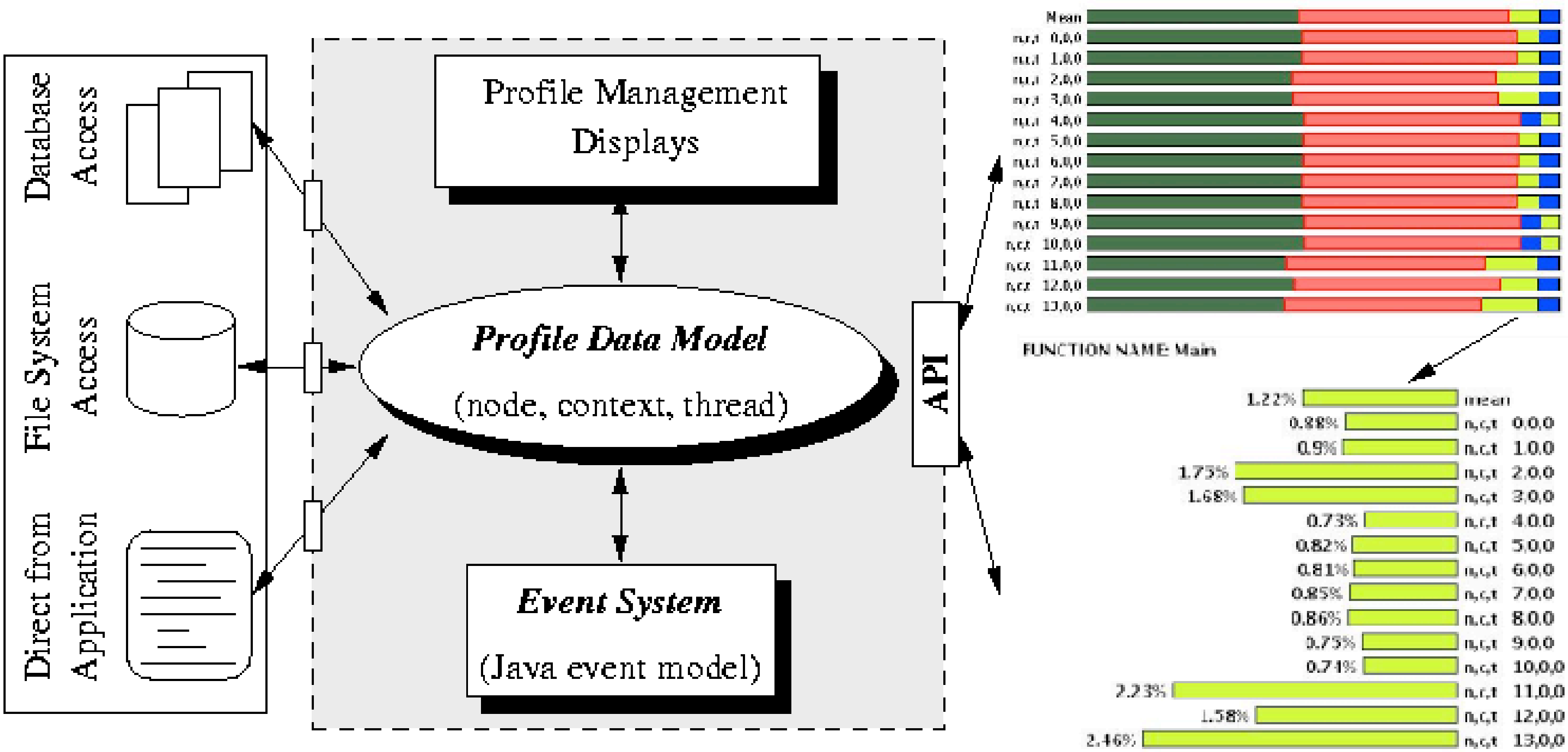
myfile1.o: myfile1.f90
myfile2.o: ...
```

Performance Analysis and Visualization

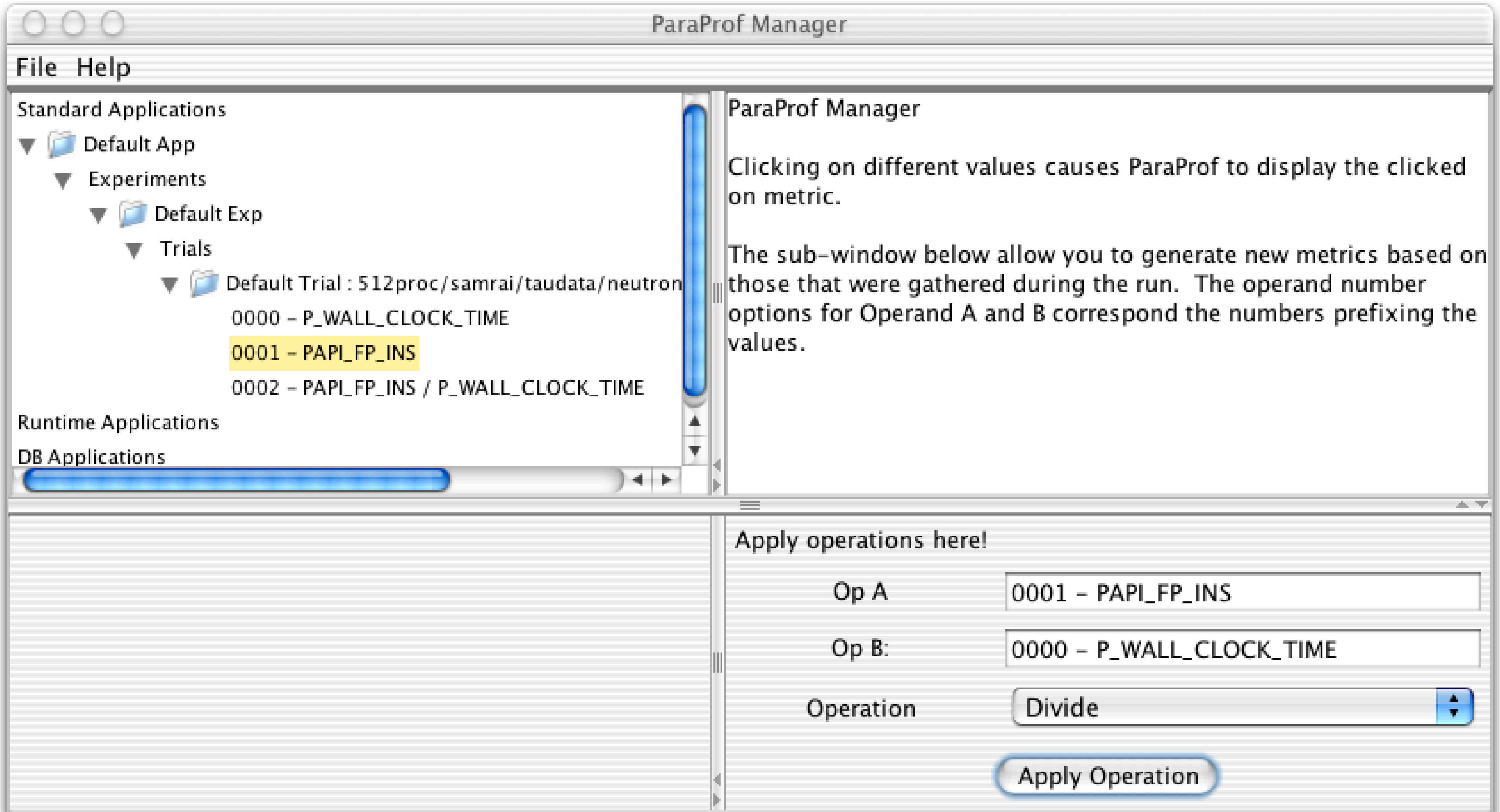
- Analysis of parallel profile and trace measurement
- Parallel profile analysis
  - ParaProf
  - Profile generation from trace data
- Performance database framework (PerfDBF)
- Parallel trace analysis
  - Translation to VTF 3.0 and EPILOG
  - Integration with VNG (Technical University of Dresden)
-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

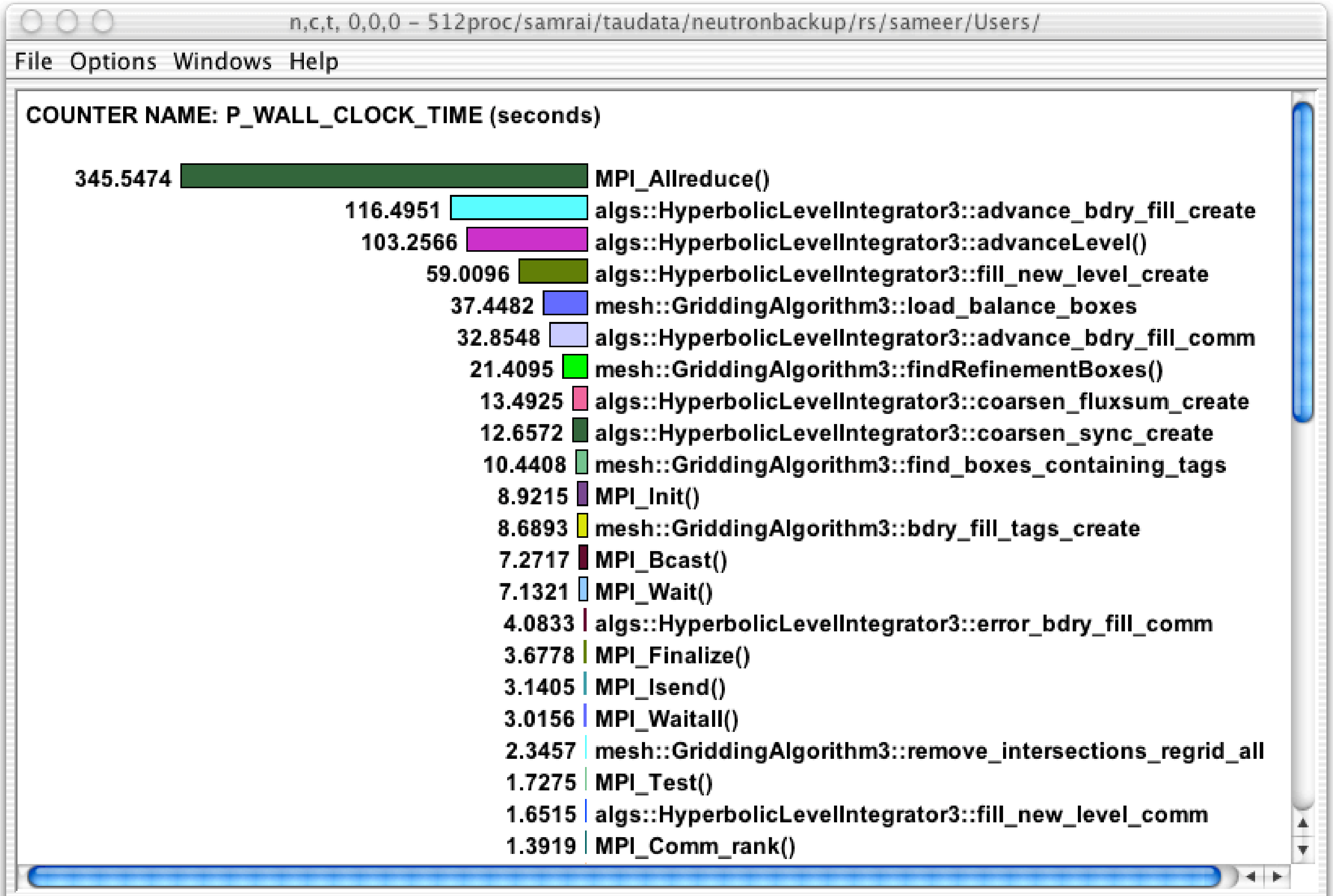


Profile Manager Window

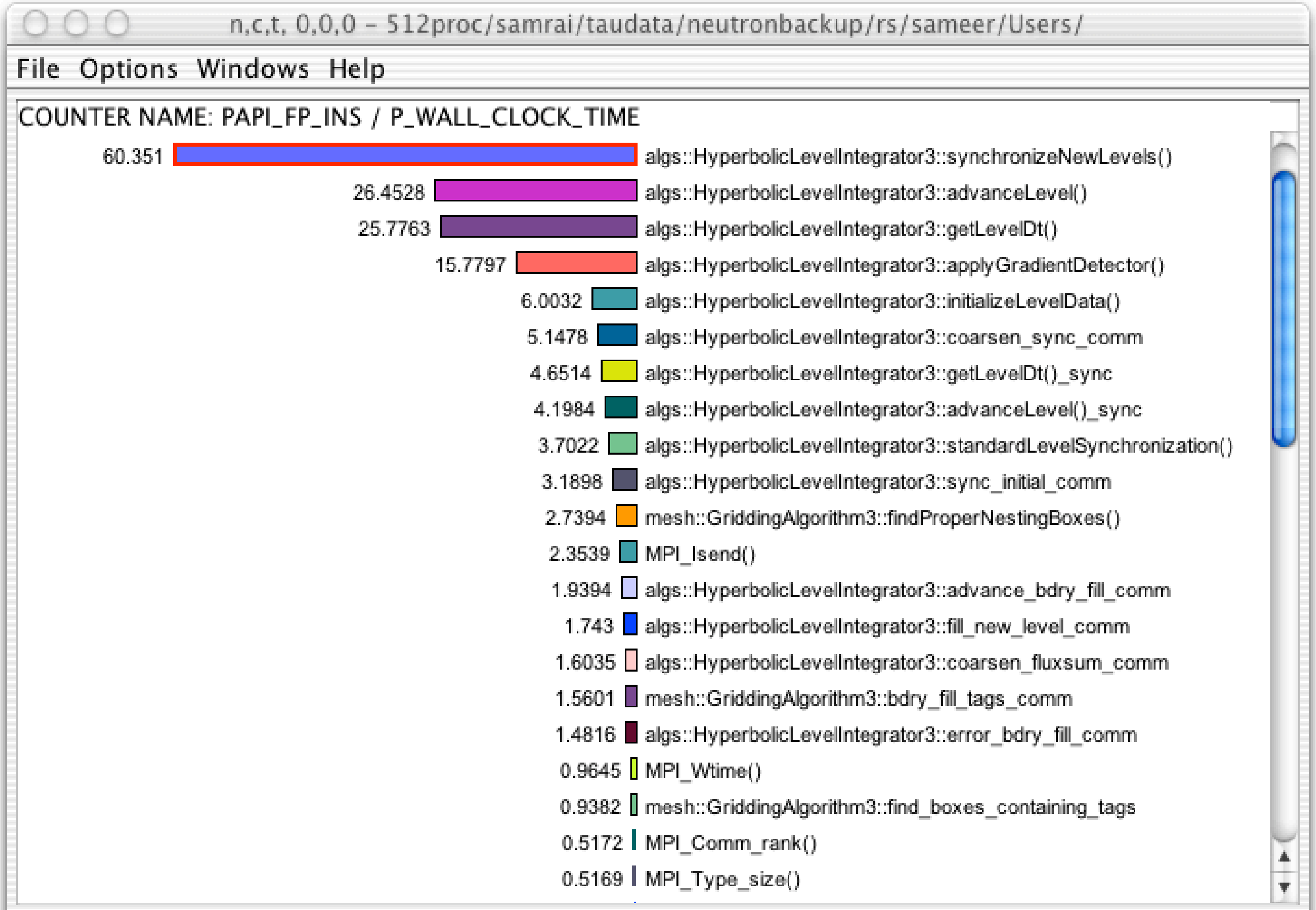


- Structured AMR toolkit (SAMRAI++), LLNL

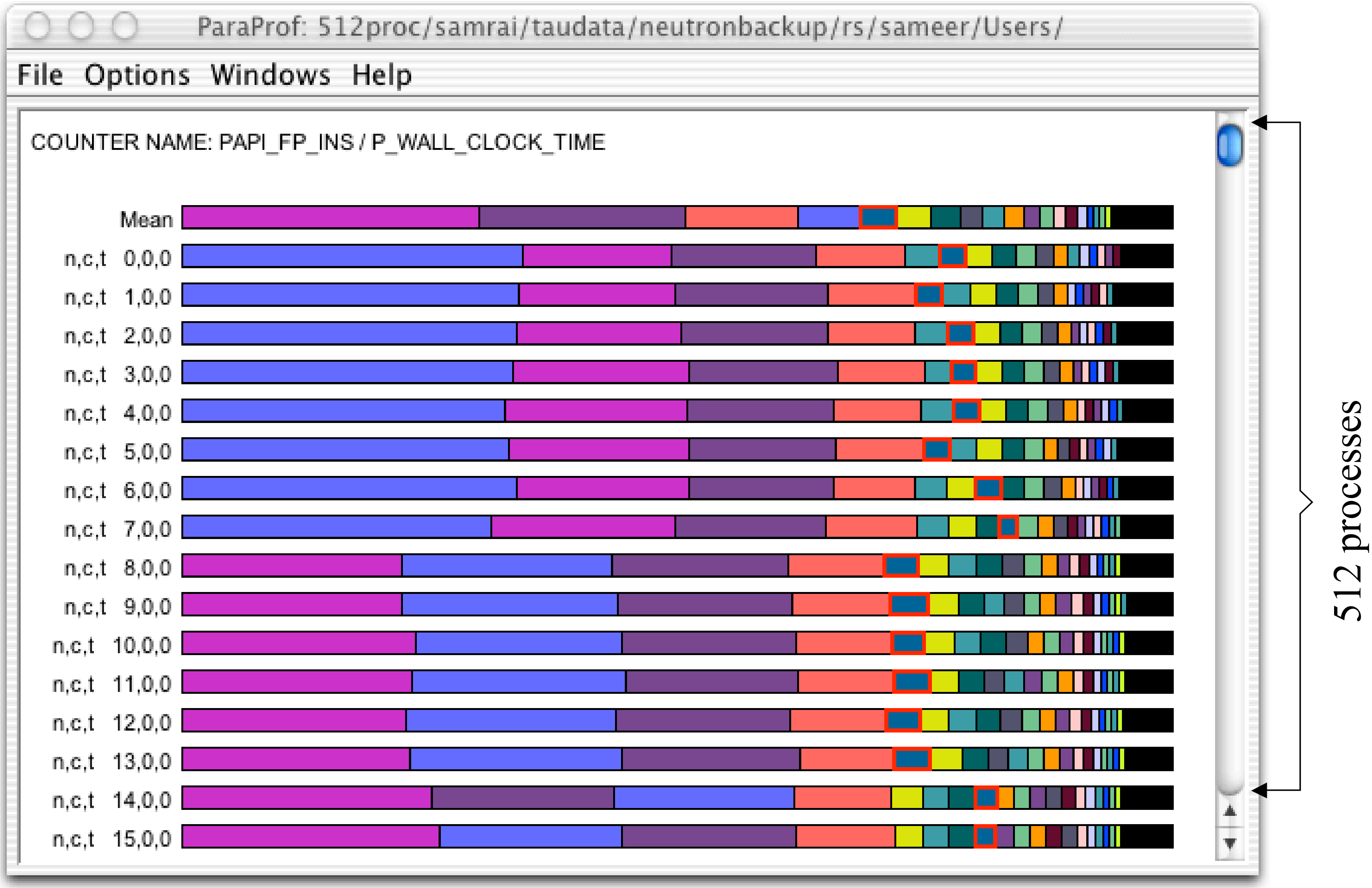
Node / Context / Thread Profile Window



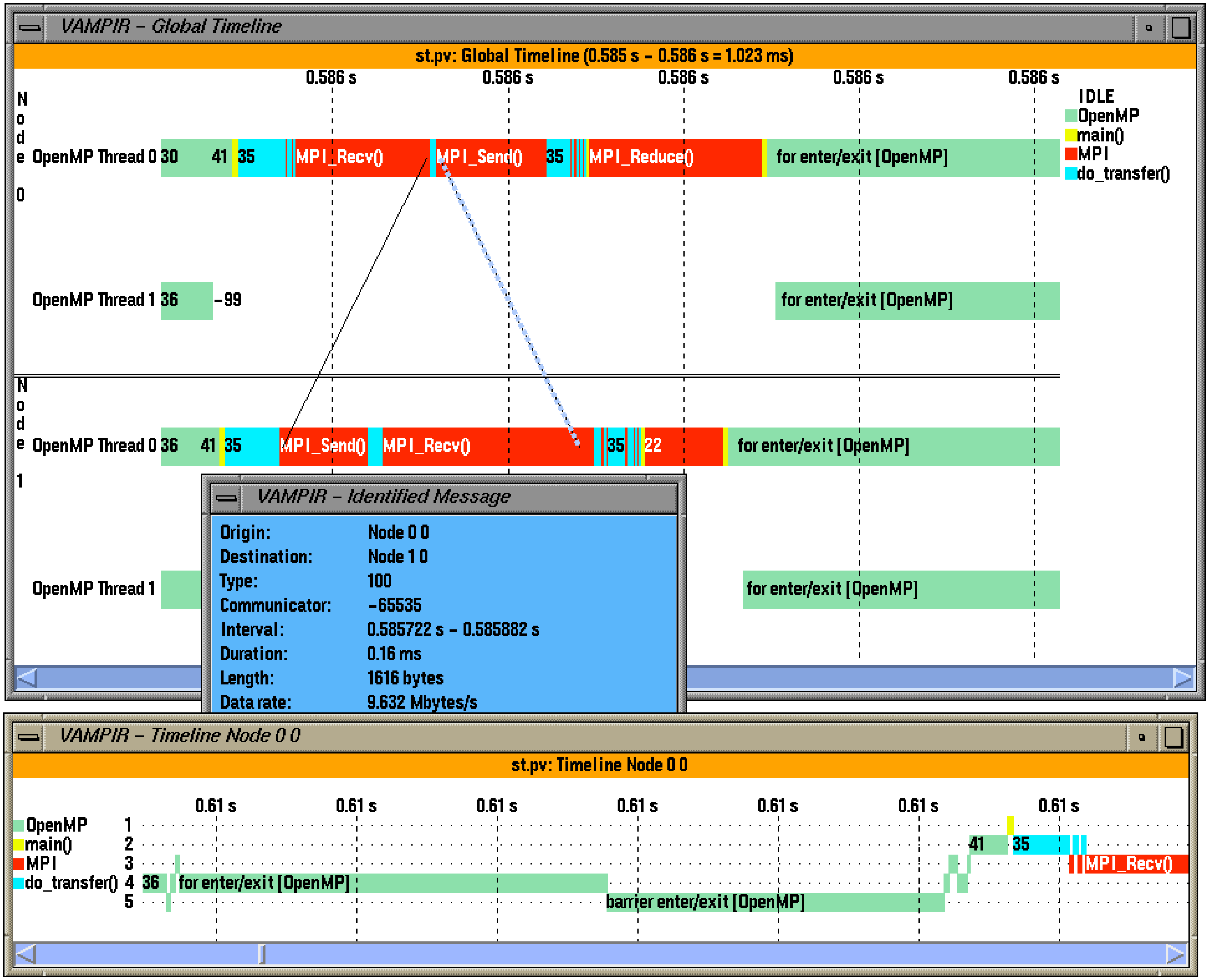
Derived Metrics



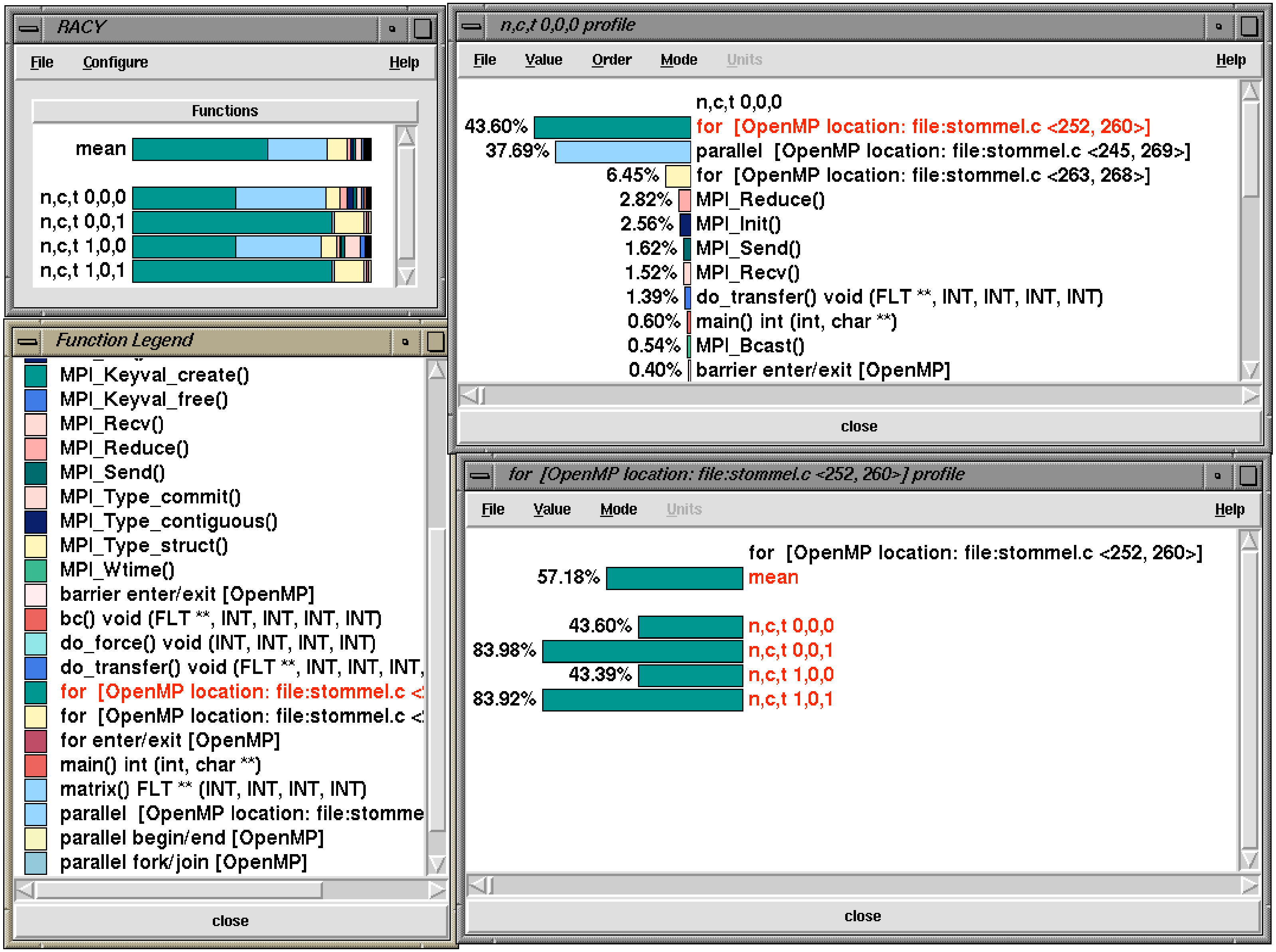
Full Profile Window (Metric-specific)



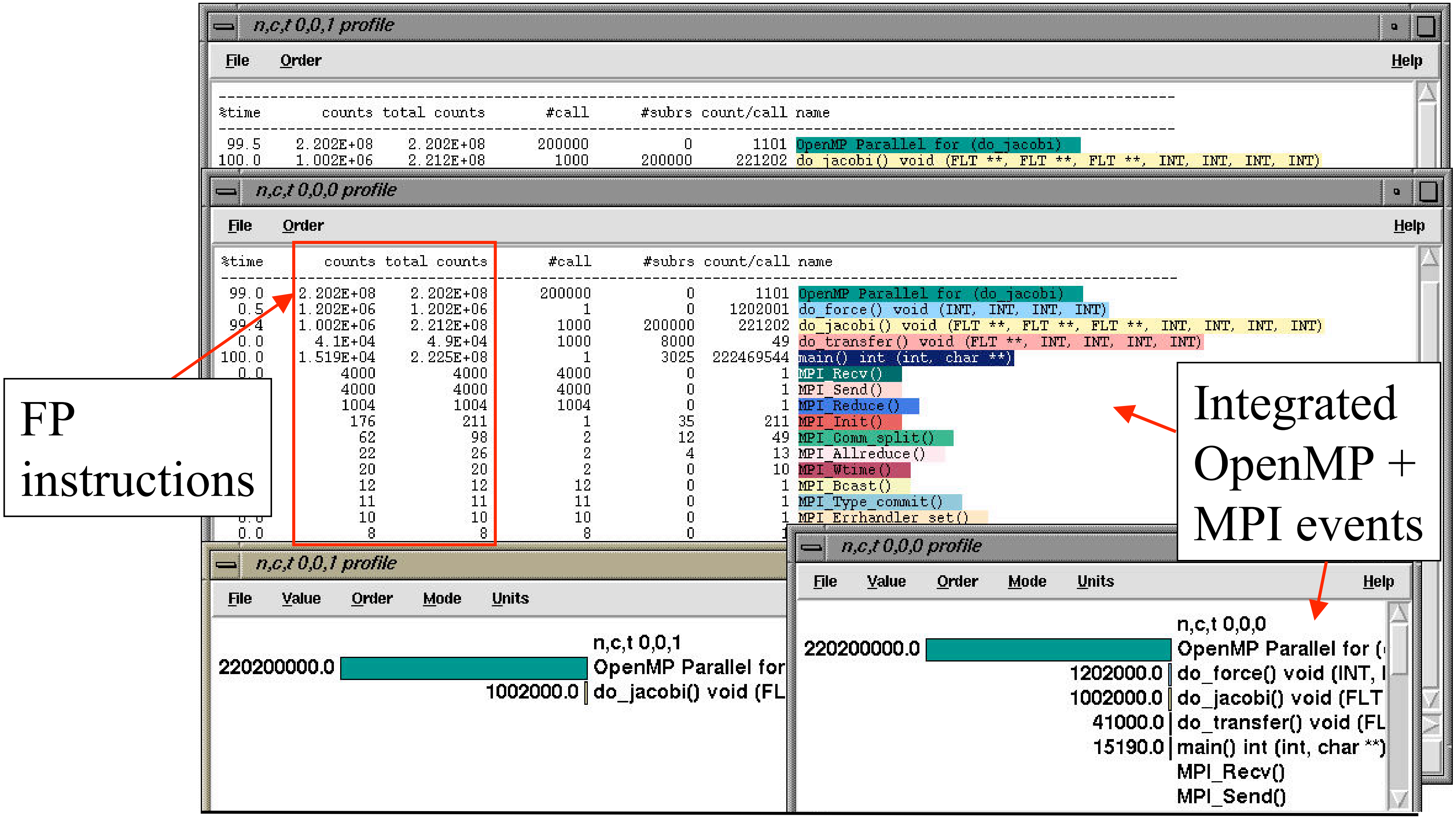
Tracing Hybrid Executions – TAU and Vampir



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