Performance Analysis Tools and PAPI

Philip J. Mucci
mucci@cs.utk.edu

Innovative Computing Laboratory, Knoxville, TN
Center for Parallel Computers, Stockholm, Sweden

Linux Performance Infrastructure

- Contrary to popular belief, the Linux infrastructure is well established.
- PAPI is +7 years old.
- Wide complement of tools from which to choose.
- Some are production quality.
- Sun, IBM and HP are now focusing on Linux/HPC which means a focus on performance.

System Monitoring

- Evaluate the performance of a system as a whole.
- Snapshot, high-level views.
- Continuous collection, aggregation.
- No support for HT/CMT/SMT needed.

System Monitoring Applications

- PerfMiner
- Ganglia
- NWPerf
- SuperMon
- CluMon
- Nagios
- PCP

System Optimization

- Adaptive Kernel Subsystems
  - Dynamic page migration
  - TLB coalescing
  - Advanced HT/SMT scheduling.
- System throughput optimization
  - Profile samples that cross user/kernel domain.

System Optimization Mechanisms

- Oprofile
- Perfmon
- DCPI/ProfileMe
- KernInst
- DTrace

Application Monitoring

- Measure actual application performance via batch system. (or BSD like collection mechanisms.)
  - Workload characterization
- Per thread/per application metrics.
- Isolate deficits in throughput, efficiency and productivity.
- Dedicated CMT/SMT/HT counters.

Application Monitoring Systems

- PerfMiner (+ Easy)
- NWPerf
- Work at NCSA (+ OpenPBS)

Application Compilation, Analysis, Modeling and Optimization

- Focused on items code that the user has direct control over.
- Non-SUID/non-root/exclusive thread scope access and virtualization
- This is the focus of most user tools.
- Dedicated CMT/SMT/HT counters.

Compilers and Tools

- HPCToolkit
- PerfSuite
- SvPablo
- TAU
- Vampir
- Lots of vendor tools, compilers and modeling systems.

Hardware Performance Counters

- Performance Counters are hardware registers dedicated to counting certain types of events within the processor or system.
  - Usually a small number of these registers (2,4,8)
  - Sometimes they can count a lot of events or just a few
  - Symmetric or asymmetric
- Each register has a various modes of operation.
  - Interrupt on overflow
  - Edge detection (cycles vs. events)
  - User vs. kernel mode

Access to Performance Counters

- On some platforms there are APIs, on others nothing is available.
- When the APIs do exist, they are usually:
  - Not appropriate for the application engineer.
  - Not very well documented.
- The same can be said for the counter hardware itself.
  - Often not well documented.
  - Rarely are the events verified by the engineers.

Performance Counters

- Performance Counters are hardware registers dedicated to counting certain types of events within the processor or system.
  - Usually a small number of these registers (2,4,8)
  - Sometimes they can count a lot of events or just a few
  - Symmetric or asymmetric
- Each register has an associated control register that tells it what to count and how to do it.
  - Interrupt on overflow
  - Edge detection (cycles vs. events)
  - User vs. kernel mode

Performance Counters

- Most high performance processors include hardware performance counters.
  - AMD Athlon and Opteron
  - Compaq Alpha EV Series
  - CRAY T3E, X1
  - IBM Power Series
  - Intel Itanium, Pentium
  - SGI MIPS R1xK Series
  - Sun UltraSparc II+
  - And many others...

Available Performance Data

- Cycle count
- Instruction count
  - All instructions
  - Floating point
  - Integer
  - Load/store
- Branches
  - Taken / not taken
  - Mispredictions
- Pipeline stalls due to
  - Memory subsystem
  - Resource conflicts
- Cache
  - I/D cache misses for different levels
  - Invalidations
- TLB
  - Misses
  - Invalidations

PAPI

- **P**erformance **A**pplication **P**rogramming **I**nterface
- The purpose of PAPI is to implement a standardized portable and efficient API to access the hardware performance monitor counters found on most modern microprocessors.
- The goal of PAPI is to facilitate the optimization of parallel and serial code performance by encouraging the development of cross-platform optimization tools.

PAPI Counter Interfaces

- PAPI provides 2 interfaces to the underlying counter hardware:
  1. The high level interface provides the ability to start, stop and read the counters for a specified list of events.
  2. The low level interface manages hardware events in user defined groups called *EventSets*, and provides access to advanced features.

PAPI Features

- Standardized Access to Performance Counters
- Preset Performance Metrics
- Easy Access to Platform-Specific Metrics
- Multiplexed Event Measurement
- Dispatch on Overflow
- Full SVR4 Profiling
- Bindings for C, Fortran, Matlab, and Java

PAPI Preset Events

- PAPI supports around preset events
- Proposed set of events deemed most relevant for application performance tuning
- Preset events are mappings from symbolic names to machine specific definitions for a particular hardware resource.
  - Total Cycles is PAPI_TOT_CYC
- Mapped to native events on a given platform
- PAPI also supports presets that may be derived from the underlying hardware metrics

Native Events

- Any event countable by the CPU can be counted even if there is no matching preset PAPI event.
- Same interface as when setting up a preset event, but we use one additional call to translate a CPU-specific moniker into a PAPI event definition.

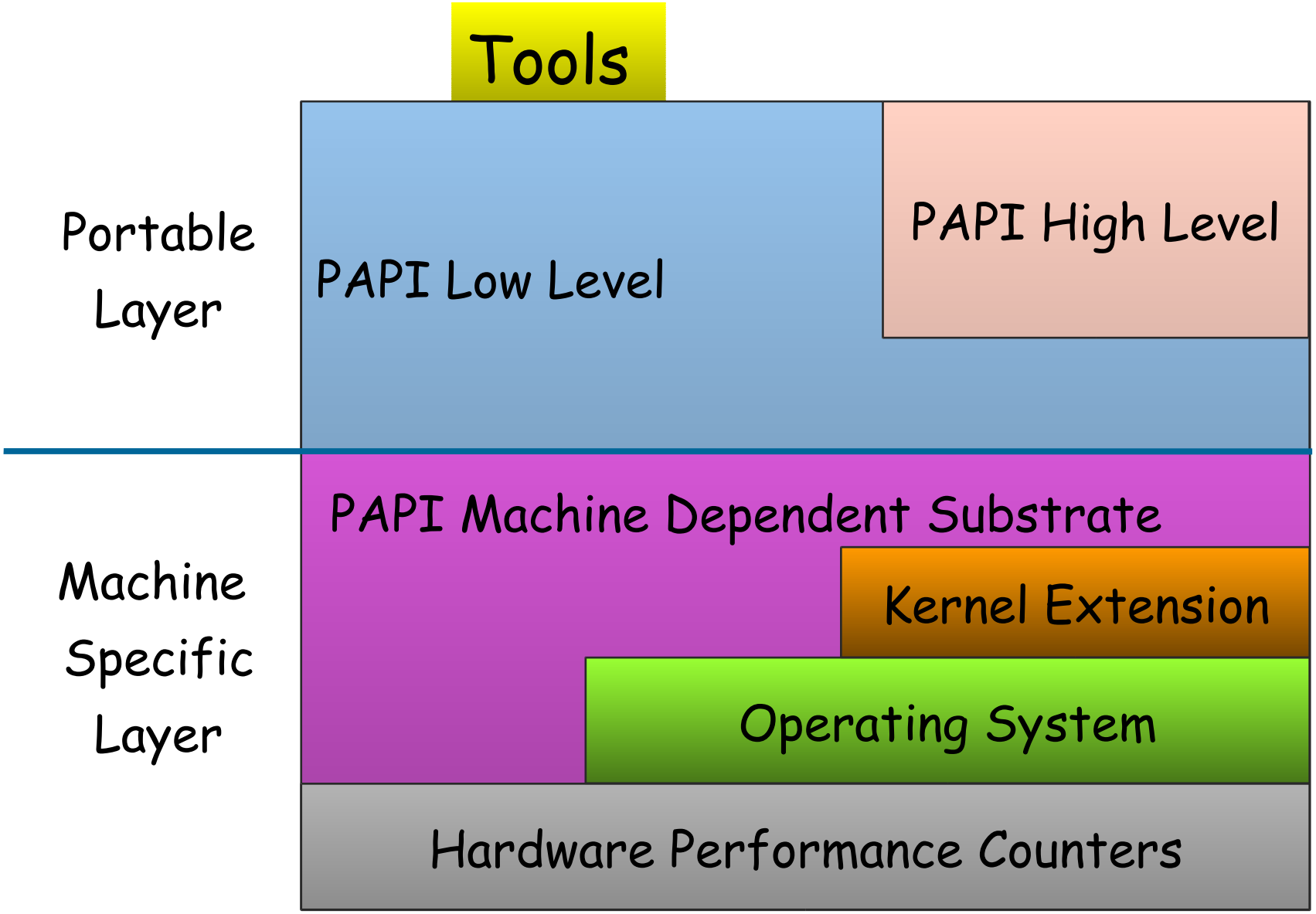
PAPI 3.0

- Full enumeration of platform-specific metrics
- Overflow and profiling on multiple events simultaneously
- Complete memory hierarchy information
- Complete shared library map
- Thread safe, high level API
- Efficient thread local storage and locking routines
- 32 and 64-bit profiling buckets (vs. 16-bit in SVR4/POSIX)

PAPI 3.0 Release

- Lower measurement overheads.
- New support for Intel EM64T and Cray X1 (SSP/MSP)
- Updated Web Site and Documentation:
  - Links to New tools, Example codes
  - Improved Web page
  - Bugzilla Database

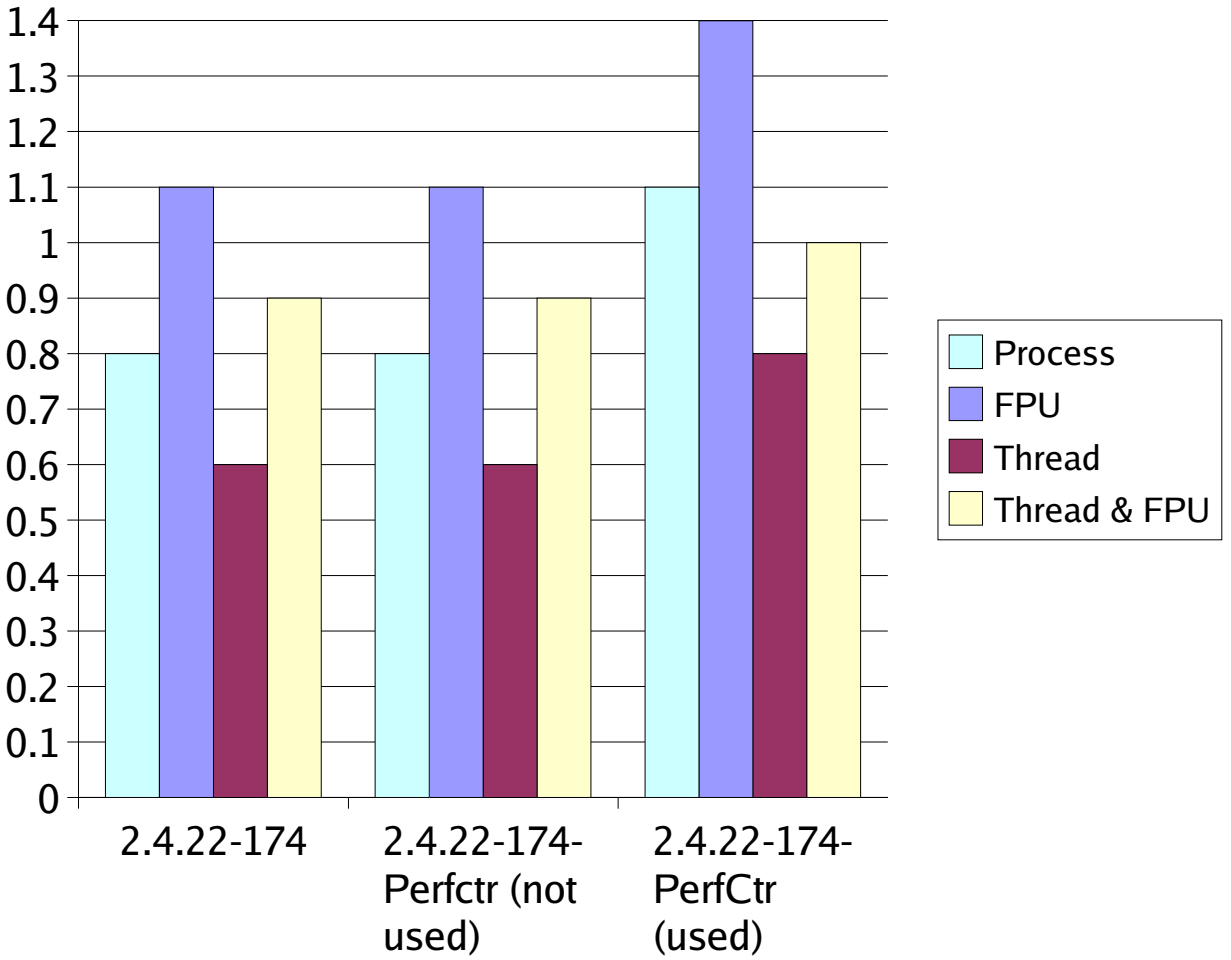
PAPI Implementation



Linux Kernel Support for PMC

- Performance counters are part of the thread context, just like FPU registers.
  - Dedicated, per-thread measurements
- Cost of switching is minimal when lazy-evaluation is used.
- Linux Kernel Integration
  - IA64: HP designed and pushed 'perfmon' into mainline by inheritance. (syscall based)
  - x86/x86_64: PerfCtr, designed by Mikael Pettersson in Uppsala. (mmap based)

PerfCtr 2.6 Context Switches



High-level Interface

- Meant for application programmers wanting coarse-grained measurements
- Not thread safe
- Calls the lower level API
- Allows only PAPI preset events
- Easier to use and less setup (additional code) than low-level

High-level API

- C interface
  PAPI_start_counters
  PAPI_read_counters
  PAPI_stop_counters
  PAPI_accum_counters
  PAPI_num_counters
  PAPI_flops

- Fortran interface
  PAPIF_start_counters
  PAPIF_read_counters
  PAPIF_stop_counters
  PAPIF_accum_counters
  PAPIF_num_counters
  PAPIF_flops

PAPI_flops

- **`int PAPI_flops(float *real_time, float *proc_time, long_long *flpins, float *mflops)`**
  - Only two calls needed, PAPI_flops before and after the code you want to monitor
  - real_time is the wall-clocktime between the two calls
  - proc_time is the “virtual” time or time the process was actually executing between the two calls (not as fine grained as real_time but better for longer measurements)
  - flpins is the total floating point instructions executed between the two calls
  - mflops is the Mflop/s rating between the two calls
  - If *flpins == -1 the counters are reset

High-level Interface Setup

- **`int PAPI_num_counters(void)`**
  - Initializes PAPI (if needed)
  - Returns number of hardware counters
- **`int PAPI_start_counters(int *events, int len)`**
  - Initializes PAPI (if needed)
  - Sets up an event set with the given counters
  - Starts counting in the event set
- **`int PAPI_library_init(int version)`**
  - Low-level routine implicitly called by above

Controlling the Counters

- **`PAPI_stop_counters(long_long *vals, int  alen)`**
  - Stop counters and put counter values in array
- **`PAPI_accum_counters(long_long *vals, int  alen)`**
  - Accumulate counters into array and reset
- **`PAPI_read_counters(long_long *vals, int  alen)`**
  - Copy counter values into array and reset counters
- **`PAPI_flops(float *rtime, float *ptime,`**
  `long_long *flpins, float *mflops)`
  - Wallclock time, process time, FP ins since start,
  - Mflop/s since last call

PAPI High-level Example

```
long long values[NUM_EVENTS];
unsigned int Events[NUM_EVENTS]=
  {PAPI_TOT_INS,PAPI_TOT_CYC};
  /* Start the counters */
  PAPI_start_counters((int*)Events,NUM_EVENTS);
  /* What we are monitoring… */
  do_work();
  /* Stop the counters and store the results in values */
  retval = PAPI_stop_counters(values,NUM_EVENTS);
```

Low-level Interface

- Increased efficiency and functionality over the high level PAPI interface
- About 56 functions (http://icl.cs.utk.edu/projects/papi/files/html_man/papi.html#4)
- Obtain information about the executable and the hardware
- Thread-safe
- Fully programmable (native events)
- Multiplexing
- Callbacks on counter overflow
- Profiling

Supported Architectures

- AMD Athlon and Opteron
- Cray T3E and X1
- HP Alpha (caveats)
- IBM POWER3, POWER4, POWER5
- Intel Pentium Pro,II,III,IV, Itanium 1 + 2
- MIPS R10K, R12K, R14K
- Sun UltraSparc I, II, III
- BlueGene
- Red Storm/Catamount

Example PAPI Data: Parallel Ocean Program Performance x1 Data Set, 2x2 Procs, 10 Steps

Raw Data	Debug	Optimized
PAPI_LD_INS	1.21E+011	2.104E+10
PAPI_SR_INS	2.02E+010	7.783E+09
PAPI_BR_INS	8.64E+009	5.043E+09
PAPI_FP_INS	2.21E+010	2.251E+10
PAPI_FMA_INS	1.04E+010	1.007E+10
PAPI_FPU_FDIV		2.551E+08
PAPI_FPU_FSQRT		1.317E+08
PAPI_TOT_INS	3.28E+011	6.257E+10
PAPI_TOT_CYC	3.63E+011	6.226E+10
PAPI_L1_LDM	1.03E+009	1.011E+09
PAPI_L1_STM	3.54E+008	3.475E+08
PAPI_L2_LDM	6.94E+008	6.894E+08
PAPI_FPU_IDL	1.66E+011	1.411E+10
PAPI_LSU_IDL	4.06E+010	1.483E+10
PAPI_MEM_RC	1.03E+011	1.368E+10
PAPI_MEM_SC	1.26E+011	2.413E+10
PAPI_STL_CCY	2.01E+011	3.367E+10

Metric	Debug	Optimize
% Ld Ins	36.86	33.63
% Sr Ins	6.17	12.44
% Br Ins	2.63	8.06
% FP Ins	6.75	35.98
% FMA Ins	3.16	16.09
% FP Divide		0.41
% FP SQRT		0.21
MFLIPS	12.19	72.31
% MFLIPS Peal	3.05	18.08
IPC	0.90	1.00
Mem Opts/FLIF	6.38	1.28
% L1 Ld HR	99.15	95.19
% L1 Sr HR	98.25	95.54
% L2 Ld HR	99.43	96.72
% FPU Idle Cyc	45.77	22.66
% LSU Idle Cyc	11.17	23.82
% Ld Stall Cyc	28.28	21.97
% Sr Stall Cyc	34.59	38.76
% No Ins. Cyc	55.25	54.08

Low-level Functionality

- Library initialization

  PAPI_library_init, PAPI_thread_init,
  PAPI_multiplex_init, PAPI_shutdown
- Timing functions: highest resolution and accuracy

  PAPI_get_real_usec, PAPI_get_virt_usec
  PAPI_get_real_cyc, PAPI_get_virt_cyc
- Inquiry functions
- EventSet management
- Thread specific data pointers
- Simple fast lock/unlock operators

  PAPI_lock/PAPI_unlock

Callbacks on Counter Overflow

- PAPI provides the ability to call user-defined handlers when a specified event exceeds a specified threshold.
- For systems that do not support counter overflow at the OS level, PAPI sets up a high resolution interval timer and installs a timer interrupt handler.

Statistical Profiling

- Callbacks on counter overflow allow us to do interesting things.
- PAPI provides support for SVR4-compatible execution profiling based on any counter event.
- PAPI_profil() creates a histogram of overflow counts for a specified region of the application code.
  - Range compression
  - 32/64 bit buckets
  - Unknown mapping counter

Simple Low Level Example in C

```
#include "papi.h”
#define NUM_EVENTS 2
int Events[NUM_EVENTS]={PAPI_FP_INS,PAPI_TOT_CYC};
int retval, EventSet = PAPI_NULL;
long long values[NUM_EVENTS];

/* Initialize the Library */
retval = PAPI_library_init(PAPI_VER_CURRENT);
/* Allocate space for the new eventset and do setup */
retval = PAPI_create_eventset(&EventSet);
/* Add Flops and total cycles to the eventset */
retval = PAPI_add_events(EventSet,Events,NUM_EVENTS);
/* Start the counters */
retval = PAPI_start(EventSet);
do_work();
/*Stop counters and store results in values */
retval = PAPI_read(EventSet,values);
do_more_work();
/*Stop counters and store results in values */
retval = PAPI_stop(EventSet,values);
```

Simple Low Level Example in Fortran

```
#include “fpapi.h”
integer evset, status, retval
integer*8 values(2)
retval = PAPI_VER_CURRENT
evset = PAPI_NULL
call papif_library_init(retval)
call papif_create_eventset(evset, status)
call papif_add_event(evset, PAPI_TOT_CYC, status)
call papif_add_event(evset, PAPI_FP_INS, status)
call papif_start(evset, status)
C
call do_work()
C
call papif_read(evset, values, status)
C
call do_more_work()
C
call papif_stop(evset, values, status)
```

Downloading PAPI

- Download the latest version of PAPI 3 from the PAPI website.
  - Latest and greatest from the CVS tree. (recommended)

    Cvs -d :pserver:anonymous@icl.cs.utk.edu:/cvs/homes/papi login

    <no password>

    Cvs -d :pserver:anonymous@icl.cs.utk.edu:/cvs/homes/papi co papi/src
  - Last stable release: 3.0.7

    Wget http://icl.cs.utk.edu/projects/papi/downloads/papi-3.0-beta2.tar.gz

Preset Listing from tests/avail -a

```
Test case 8: Available events and hardware information.
-------------------------------------------------------------------------
Vendor string and code   : AuthenticAMD (2)
Model string and code    : AMD K8 Revision C (15)
CPU Revision             : 8.000000
CPU Megahertz            : 1794.932983
CPU's in this Node       : 1
Nodes in this System     : 1
Total CPU's              : 1
Number Hardware Counters : 4
Max Multiplex Counters   : 32
-------------------------------------------------------------------------
Name            Derived Description (Mgr. Note)
PAPI_L1_DCM     No      Level 1 data cache misses (DC_MISS)
PAPI_L1_ICM     No      Level 1 instruction cache misses (IC_MISS)
PAPI_L2_DCM     No      Level 2 data cache misses (BU_L2_FILL_MISS_DC)
PAPI_L2_ICM     No      Level 2 instruction cache misses (BU_L2_FILL_MISS_IC)
PAPI_L1_TCM     Yes     Level 1 cache misses (DC_MISS, IC_MISS)
PAPI_L2_TCM     Yes     Level 2 cache misses (BU_L2_FILL_MISS_IC, BU_L2_FILL_MISS_DC)
PAPI_FPU_IDL    No      Cycles floating point units are idle (FP_NONE_RET)
PAPI_TLB_DM     No      Data translation lookaside buffer misses (DC_L1_DTLB_MISS_AND_L2_DTLB_MISS)
PAPI_TLB_IM     No      Instruction translation lookaside buffer misses
    (IC_L1ITLB_MISS_AND_L2ITLB_MISS)
```

The Right Performance Tool

- You must have the right tool for the job.
- What are your needs? Things to consider:
  - User Interface
    - Complex Suite
    - Quick and Dirty
  - Data Collection Mechanism
    - Aggregate
    - Trace based
    - Statistical

The Right Performance Tool 2

- Instrumentation Mechanism
  - Source
  - Binary (DPCL/DynInst)
  - Library interposition
- Data Management
  - Performance Database
  - User (Flat file)
- Data Visualization
  - Run Time
  - Post Mortem
  - Serial/Parallel Display
  - ASCII

Essential Tool Functionality

- Must work with Pthreads, OpenMP, MPI, fork() and exec().
- Passive Tools
  - Require no modification/instrumentation of source or object code.
    - Library preloading and/or name shifting.
- Active Tools
  - Instrumentation performed.
    - Binary
    - Source

Tool Methodology

- Direct Measurements read raw values of Metrics.
  - Overall/Global Measurements. (aka Quick & Dirty)
  - Site based.
    - Module/Function/Loop/Basic Block
    - Address Range

Tool Methodology

- Indirect Measurements infer values from probabilistic distributions.
- Statistical Profiling, developing a Histogram with X axis = Location, Y axis = Event Count.
- Event could equal:
  - Timer interrupts (like Gprof)
  - Hardware Counter Overflows on arbitrary Thresholds

PerfSuite from NCSA

- psrun/psprocess
- Command line tool similar to IRIX's perfex command.
- Does aggregate counting of the entire run. Also provides statistical profiling.
- Uses library preloading.
- Output is XML or Plain Text.
  - Machine information
  - Raw counter values/Derived metrics

PSRUN Sample Output

```
PerfSuite Hardware Performance Summary Report
Version      : 1.0
Created      : Mon Dec 30 11:31:53 AM Central Standard Time 2002
Generator    : psprocess 0.5
XML Source   : /u/ncsa/anyuser/performance/psrun-ia64.xml
Execution Information
================================================================================
Date         : Sun Dec 15 21:01:20 2002
Host         : user01
Processor and System Information
================================================================================
Node CPUs    : 2
Vendor       : Intel
Family       : IPF
Model        : Itanium
CPU Revision : 6
Clock (MHz)  : 800.136
                                    Memory (MB)   : 2007.16
Pagesize (KB): 16
Cache Information
================================================================================
Cache levels : 3
--------------------------------
Level 1
Type         : data
Size (KB)    : 16
Linesize (B) : 32
Assoc        : 4
Type         : instruction
Size (KB)    : 16
Linesize (B) : 32
Assoc        : 4
--------------------------------
Level 2
Type         : unified
Size (KB)    : 96
Linesize (B) : 64
Assoc        : 6
--------------------------------
Level 3
Type         : unified
Size (KB)    : 4096
Linesize (B) : 64
Assoc        : 4
```

PSRUN Sample Output

```
Index Description                                                                    Counter Value
================================================================================================
     1 Conditional branch instructions mispredicted.....................          4831072449
    2 Conditional branch instructions correctly predicted..............         52023705122
    3 Conditional branch instructions taken............................         47366258159
    4 Floating point instructions......................................         86124489172
    5 Total cycles.....................................................        594547754568
    6 Instructions completed...........................................       1049339828741
    7 Level 1 data cache accesses......................................         30238866204
    8 Level 1 data cache hits..........................................           972479062
    9 Level 1 data cache misses........................................         29224377672
   10 Level 1 instruction cache reads..................................        221828591306
   11 Level 1 cache misses.............................................         29312740738
   12 Level 2 data cache accesses......................................        129470315862
   13 Level 2 data cache misses........................................         15569536443
   14 Level 2 data cache reads.........................................        110524791561
   15 Level 2 data cache writes........................................         18622708948
   16 Level 2 instruction cache reads..................................           566330907
   17 Level 2 store misses.............................................          1208372120
   18 Level 2 cache misses.............................................         15401180750
   19 Level 3 data cache accesses......................................          4650999018
   20 Level 3 data cache hits..........................................           186108211
   21 Level 3 data cache misses........................................          4451199079
   22 Level 3 data cache reads.........................................          4613582451
   23 Level 3 data cache writes........................................            38456570
   24 Level 3 instruction cache misses.................................             3631385
   25 Level 3 instruction cache reads..................................            17631093
   26 Level 3 cache misses.............................................          4470968725
   27 Load instructions................................................        111438431677
   28 Load/store instructions completed................................        130391246662
   29 Cycles Stalled Waiting for memory accesses.......................        256484777623
   30 Store instructions...............................................         18840914540
   31 Cycles with no instruction issue.................................         61889609525
   32 Data translation lookaside buffer misses.........................             2832692

Event Index
================================================================================
    1: PAPI_BR_MSP        2: PAPI_BR_PRC        3: PAPI_BR_TKN        4: PAPI_FP_INS
    5: PAPI_TOT_CYC       6: PAPI_TOT_INS       7: PAPI_L1_DCA        8: PAPI_L1_DCH
    9: PAPI_L1_DCM       10: PAPI_L1_ICR       11: PAPI_L1_TCM       12: PAPI_L2_DCA
   13: PAPI_L2_DCM       14: PAPI_L2_DCR       15: PAPI_L2_DCW       16: PAPI_L2_ICR
   17: PAPI_L2_STM       18: PAPI_L2_TCM       19: PAPI_L3_DCA       20: PAPI_L3_DCH
   21: PAPI_L3_DCM       22: PAPI_L3_DCR       23: PAPI_L3_DCW       24: PAPI_L3_ICM
   25: PAPI_L3_ICR       26: PAPI_L3_TCM       27: PAPI_LD_INS       28: PAPI_LST_INS
   29: PAPI_MEM_SCY      30: PAPI_SR_INS       31: PAPI_STL_ICY      32: PAPI_TLB_DM
```

PSRUN Sample Output

```
Statistics
=============================================================================================
Graduated instructions per cycle.....                                                 1.765
Graduated floating point instructions per cycle.....                                  0.145
% graduated floating point instructions of all graduated instructions..               8.207
Graduated loads/stores per cycle.....                                                 0.219
Graduated loads/stores per graduated floating point instruction.....                  1.514
Mispredicted branches per correctly predicted branch.....                             0.093
Level 1 data cache accesses per graduated instruction.....                            2.882
Graduated floating point instructions per level 1 data cache access....               2.848
Level 1 cache line reuse (data).....                                                  3.462
Level 2 cache line reuse (data).....                                                  0.877
Level 3 cache line reuse (data).....                                                  2.498
Level 1 cache hit rate (data).....                                                    0.776
Level 2 cache hit rate (data).....                                                    0.467
Level 3 cache hit rate (data).....                                                    0.714
Level 1 cache miss ratio (instruction).....                                           0.003
Level 1 cache miss ratio (data).....                                                  0.966
Level 2 cache miss ratio (data).....                                                  0.120
Level 3 cache miss ratio (data).....                                                  0.957
Bandwidth used to level 1 cache (MB/s).....                                        1262.361
Bandwidth used to level 2 cache (MB/s).....                                        1326.512
Bandwidth used to level 3 cache (MB/s).....                                         385.087
% cycles with no instruction issue.....                                              10.410
% cycles stalled on memory access.....                                               43.139
MFLOPS (cycles).....                                                                115.905
MFLOPS (wallclock).....                                                             114.441
MIPS (cycles).....                                                                 1412.190
MIPS (wallclock).....                                                              1394.349
CPU time (seconds).....                                                             743.058
Wall clock time (seconds).....                                                      752.566
% CPU utilization.....                                                               98.737
```

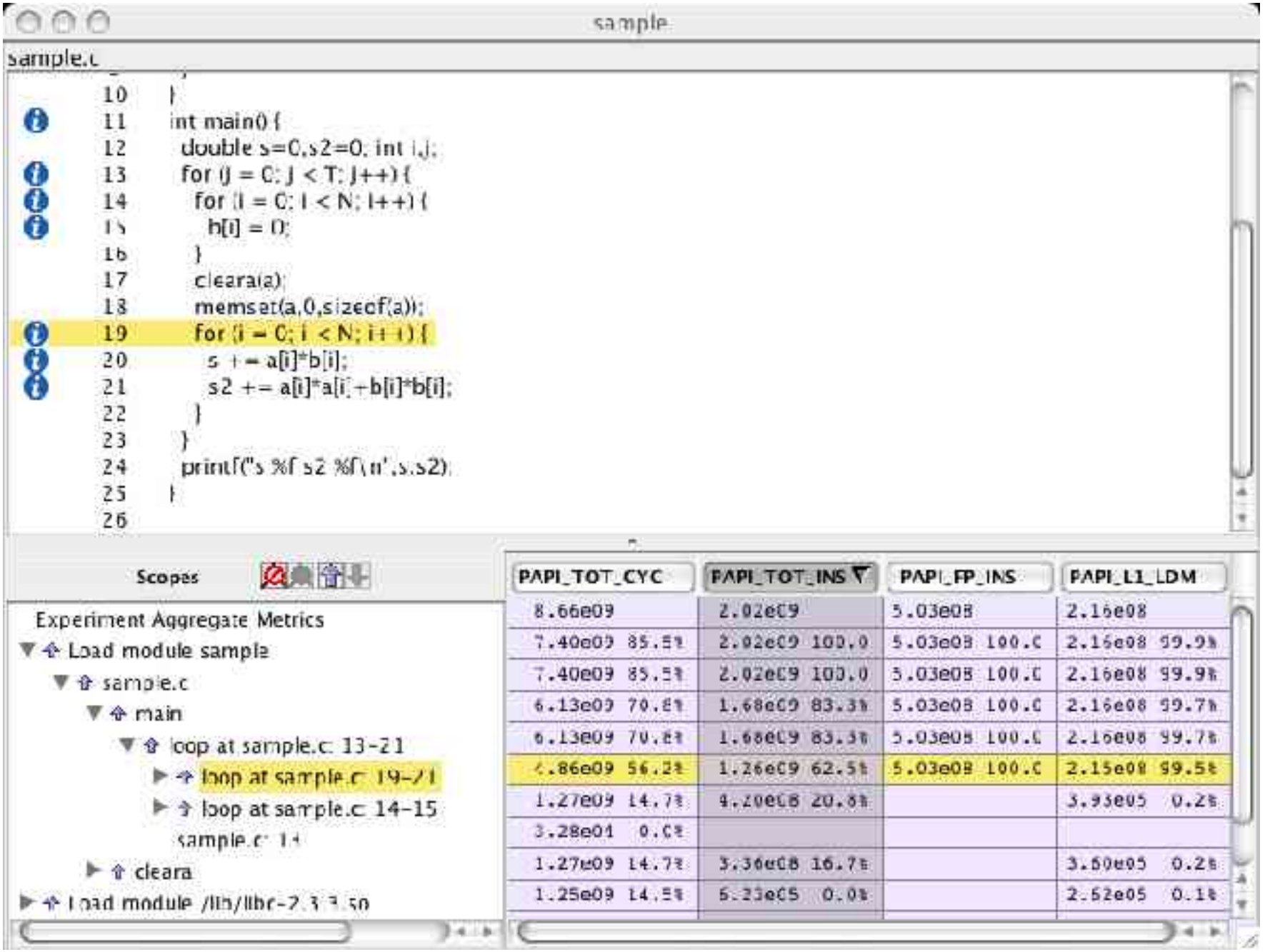
HPCToolkit from Rice U.

- Use event-based sampling and statistical profiling to profile unmodified applications: hpcrun
- Interpret program counter histograms: hpcprof
- Correlate source code, structure and performance metrics: hpcview/hpcquick
- Explore and analyze performance databases: hpcviewer
- Linux IA32, x86_64, IA64.

HPCToolkit Goals

- Support large, multi-lingual applications
  - Fortran, C, C++, external libraries (possibly binary only) with thousands of procedures, hundreds of thousands of lines
  - Avoid
    - Manual instrumentation
    - Significantly altering the build process
    - Frequent recompilation
- Collect execution measurements scalably and efficiently
  - Don't excessively dilate or perturb execution
  - Avoid large trace files for long running codes
- Support measurement and analysis of serial and parallel codes
- Present analysis results effectively
  - Top down analysis to cope with complex programs
  - Intuitive enough for physicists and engineers to use
  - Detailed enough to meet the needs of compiler writers
- Support a wide range of computer platforms

HPCToolkit Sample Output



TAU from U. Oregon

- Tuning and Analysis Utilities (11+ year project effort)
- Integrated toolkit for parallel and serial performance instrumentation, measurement, analysis, and visualization
- Open software approach with technology integration
- Robust timing and hardware performance support using PAPI
- TAU supports both profiling and tracing models.

Some TAU Features

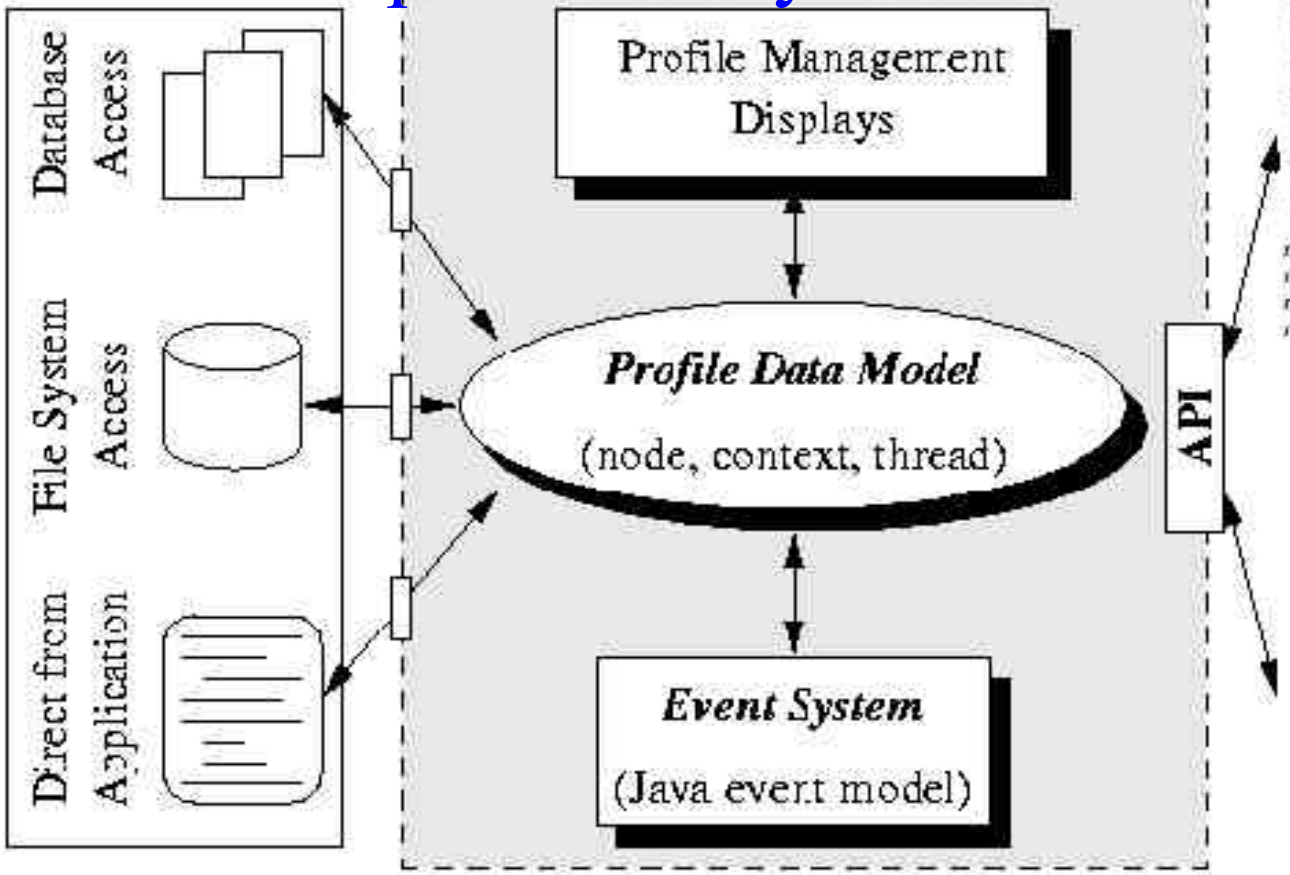
- Function-level, block-level, statement-level
- Support for callgraph and callpath profiling
- Parallel profiling and Inter-process communication events
- Supports user-defined events
- Trace merging and format conversion

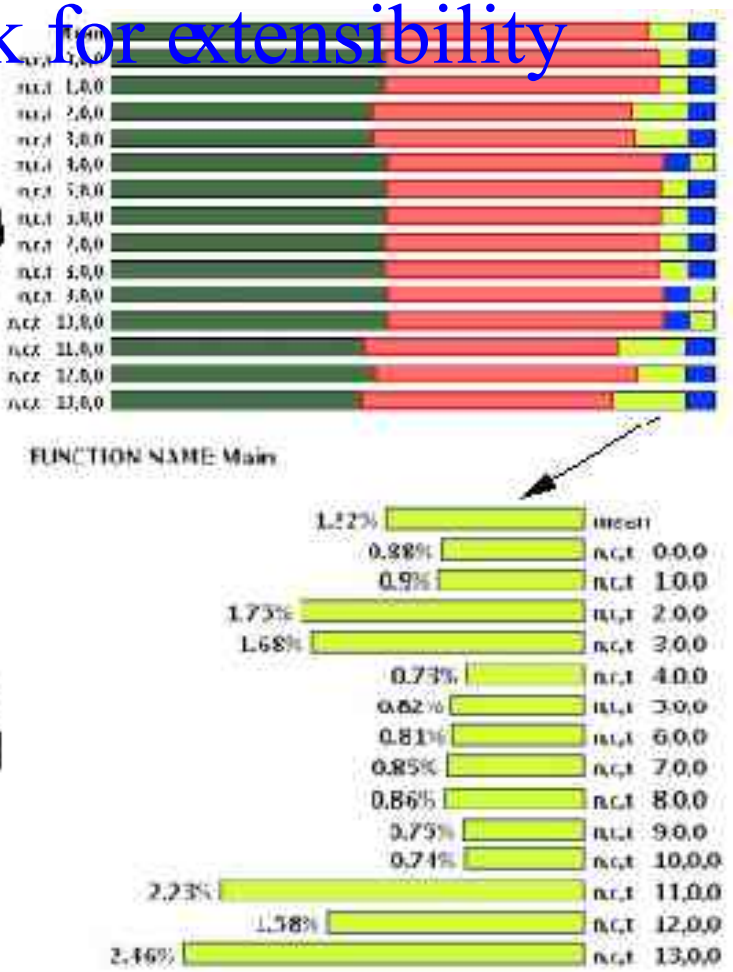
TAU Instrumentation

- Flexible mechanisms:
  - Source code both manual and 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)
  - Executable code
    - dynamic instrumentation (pre-execution) (DynInstAPI)

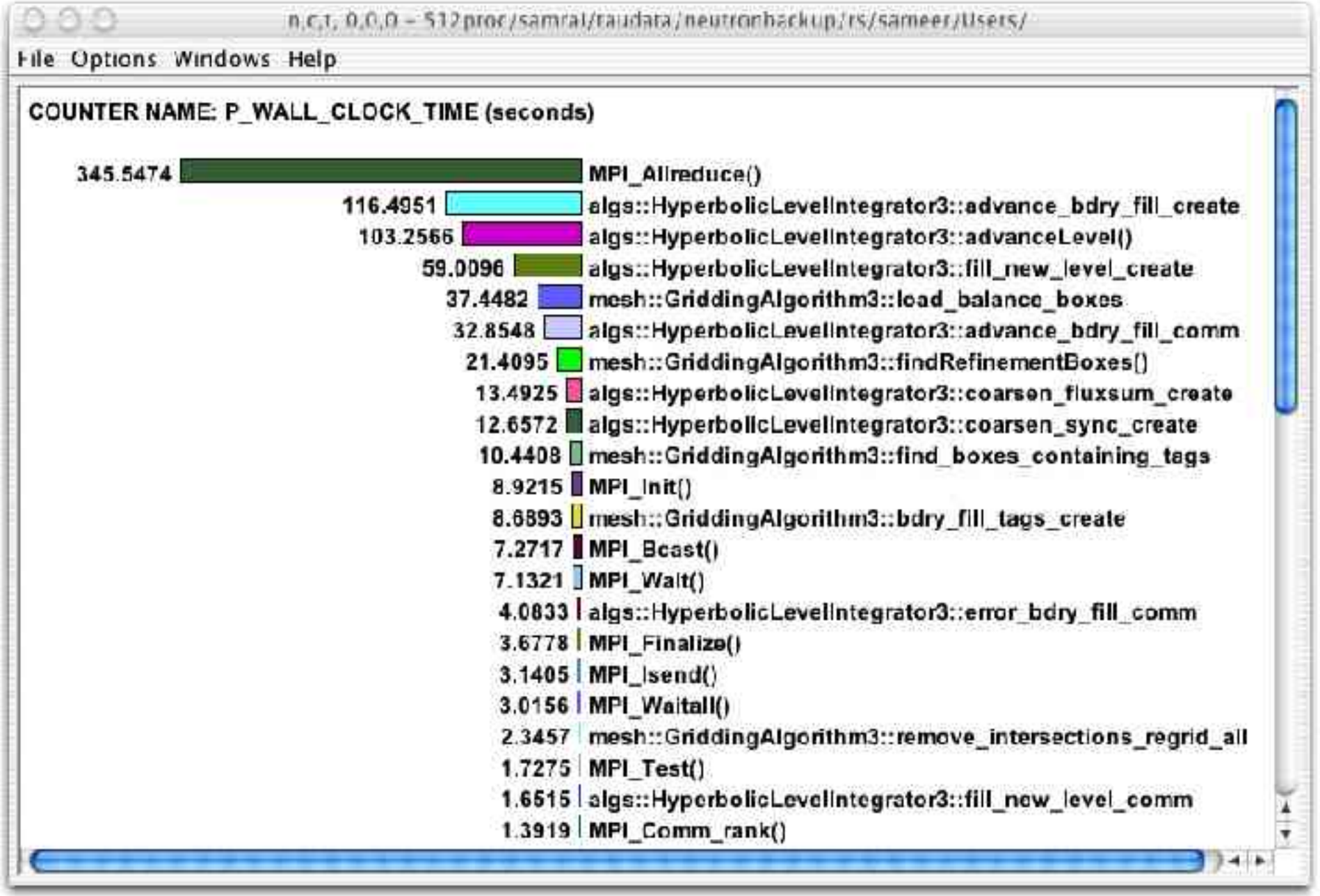
ParaProf Framework Architecture

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

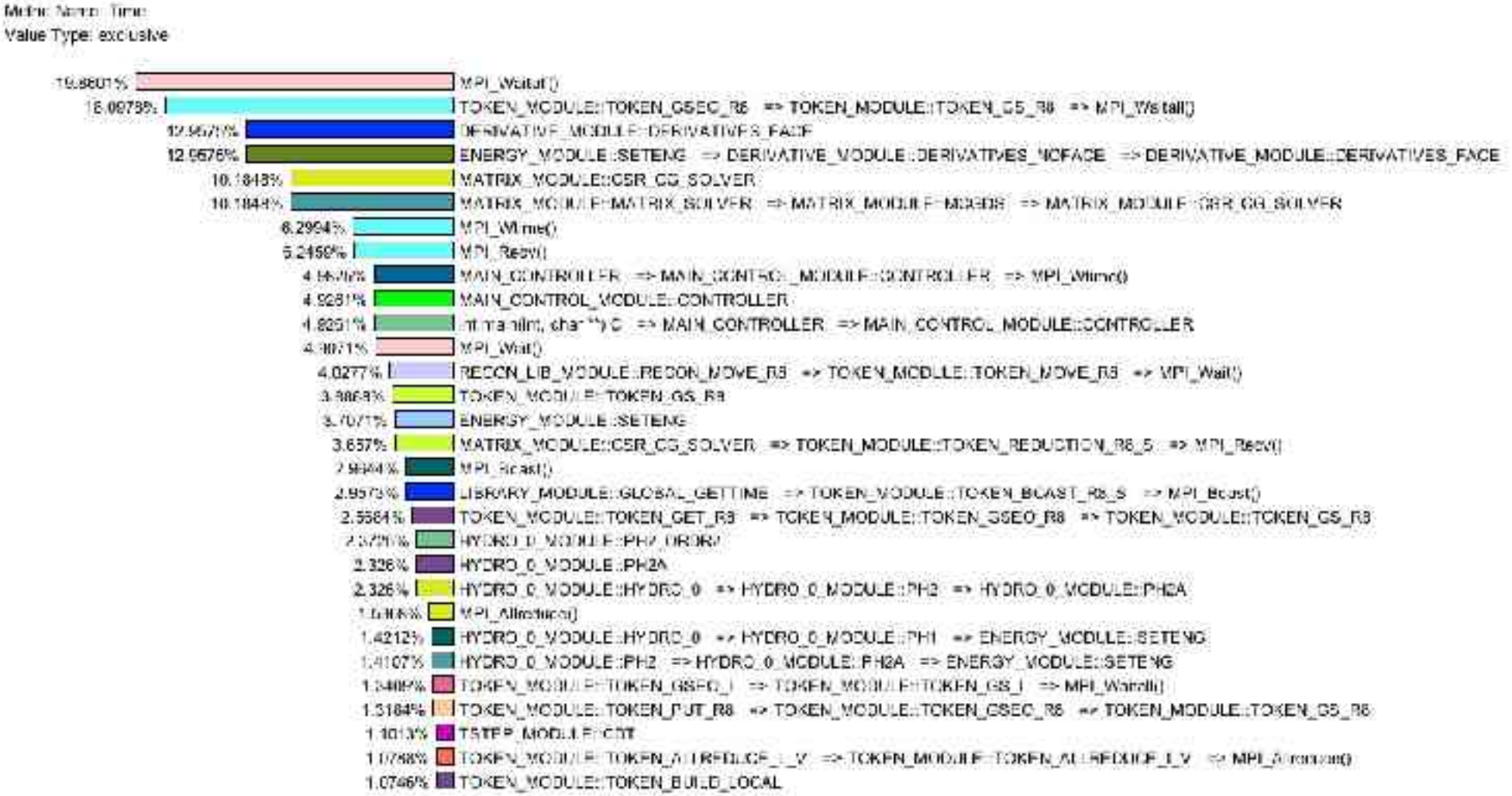




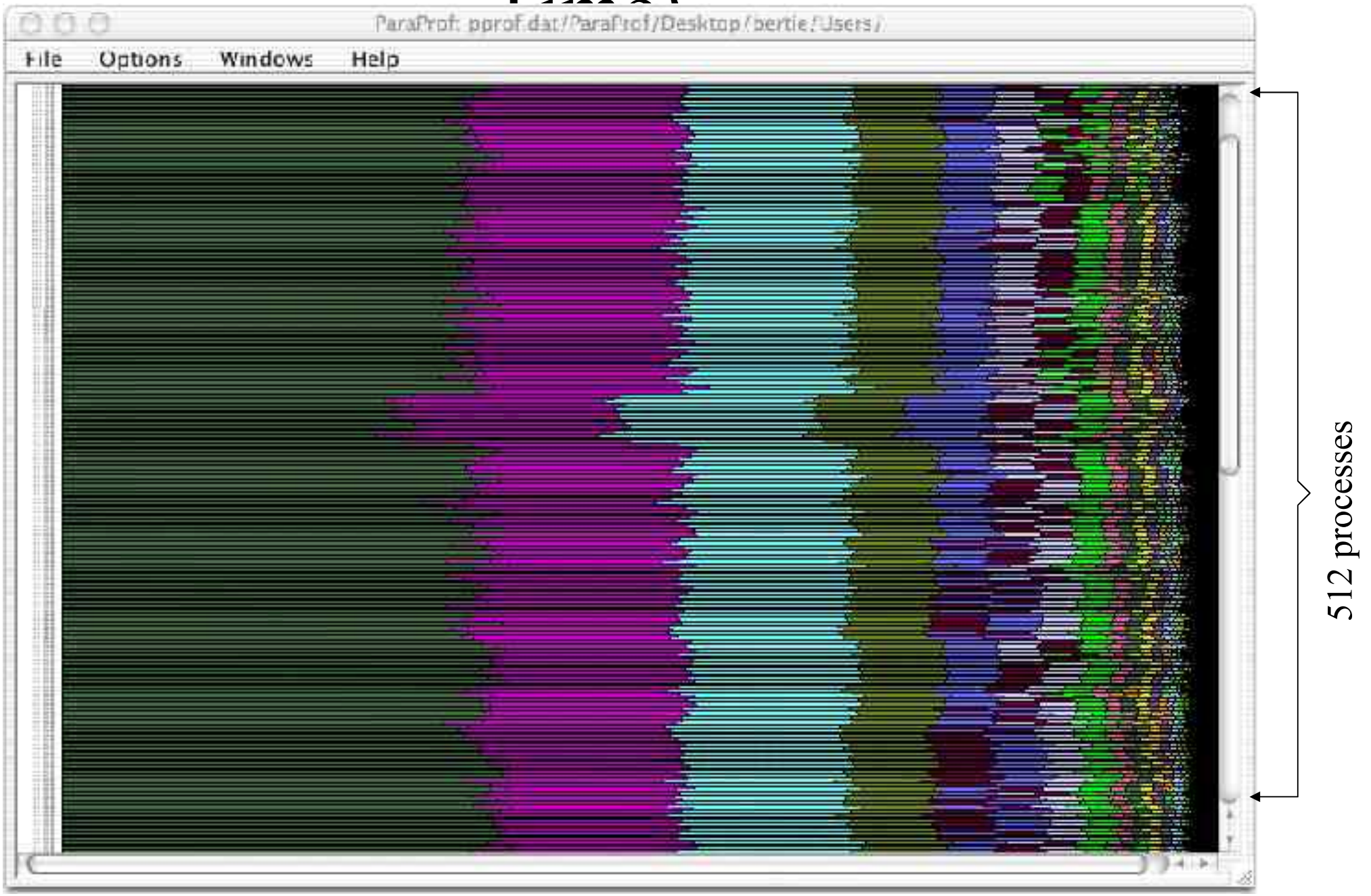
Node / Context / Thread Profile Window



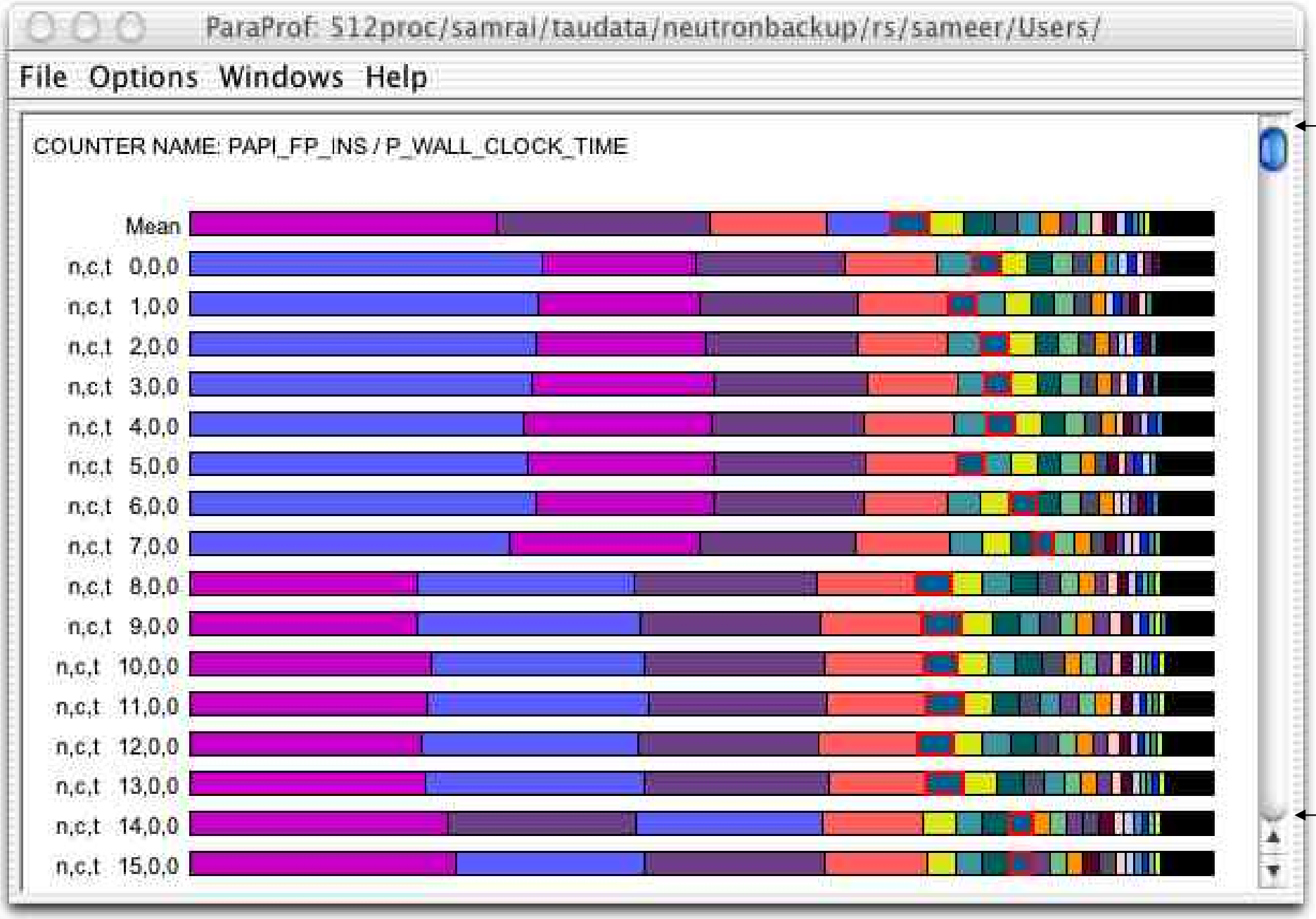
Example Callpath Data



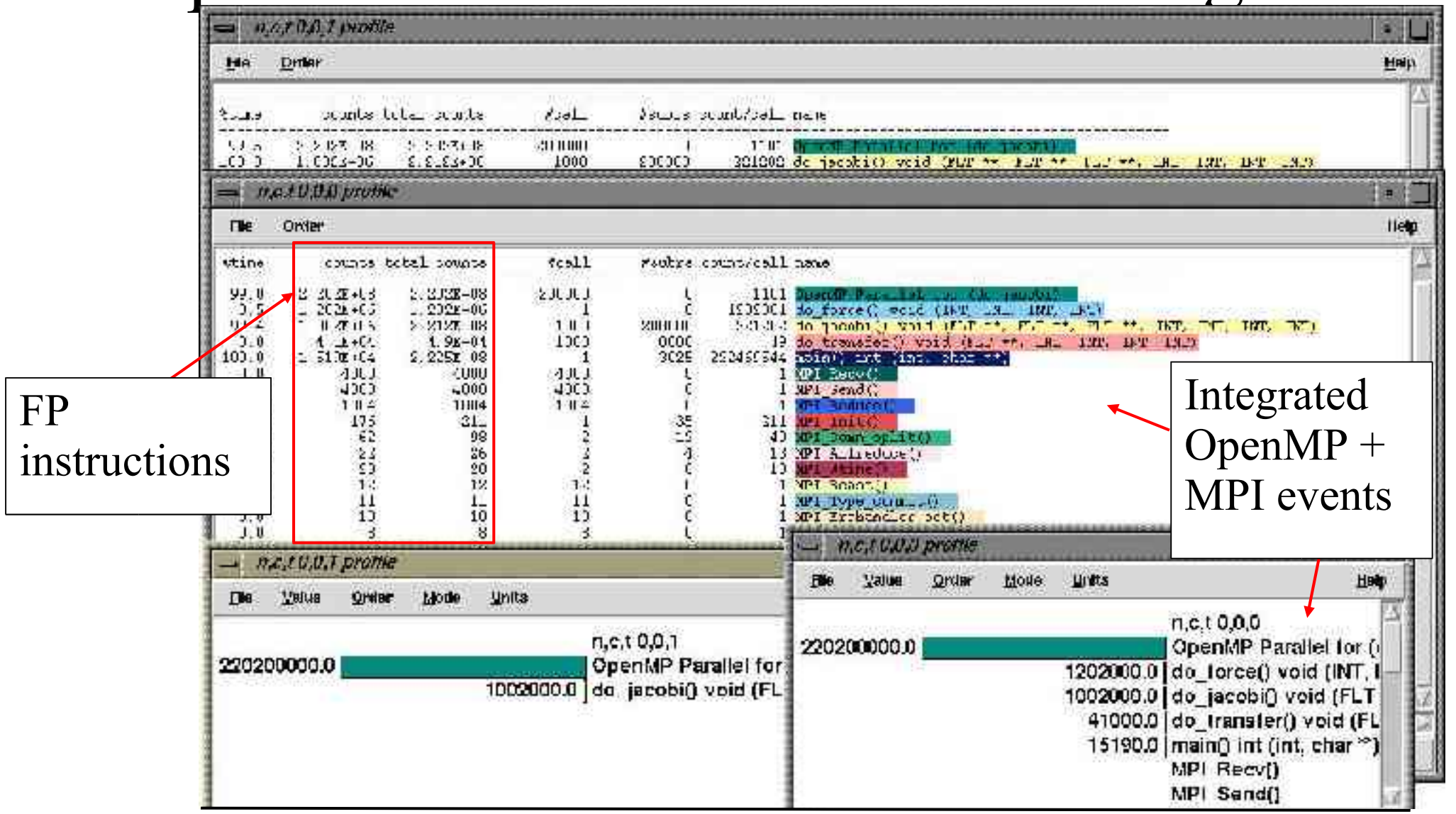
Full Profile Window (Exclusive Time)



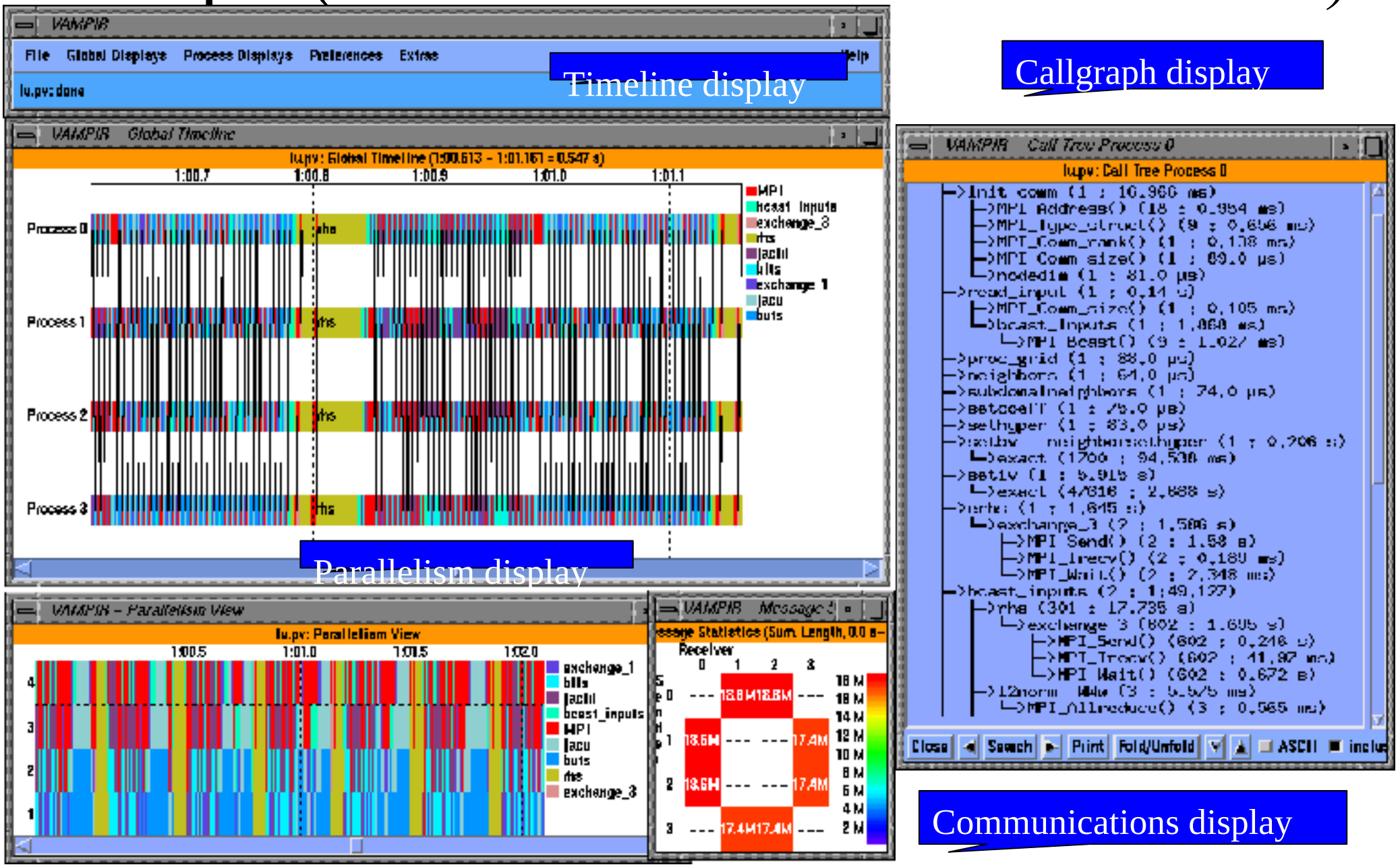
Profile Window for Derived Metrics



OpenMP + MPI Ocean Modeling

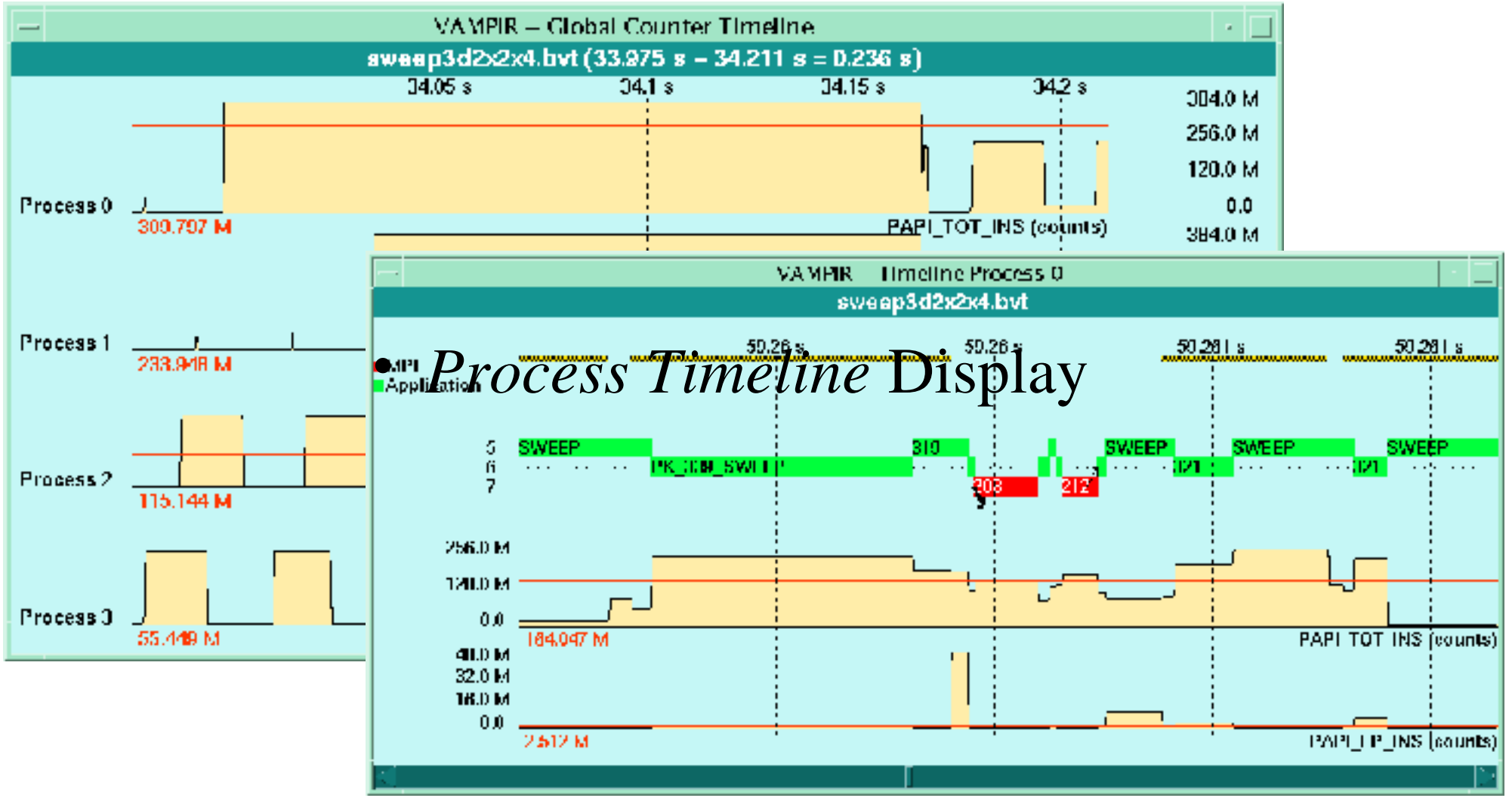


Vampir (NAS Parallel Benchmark – LU)



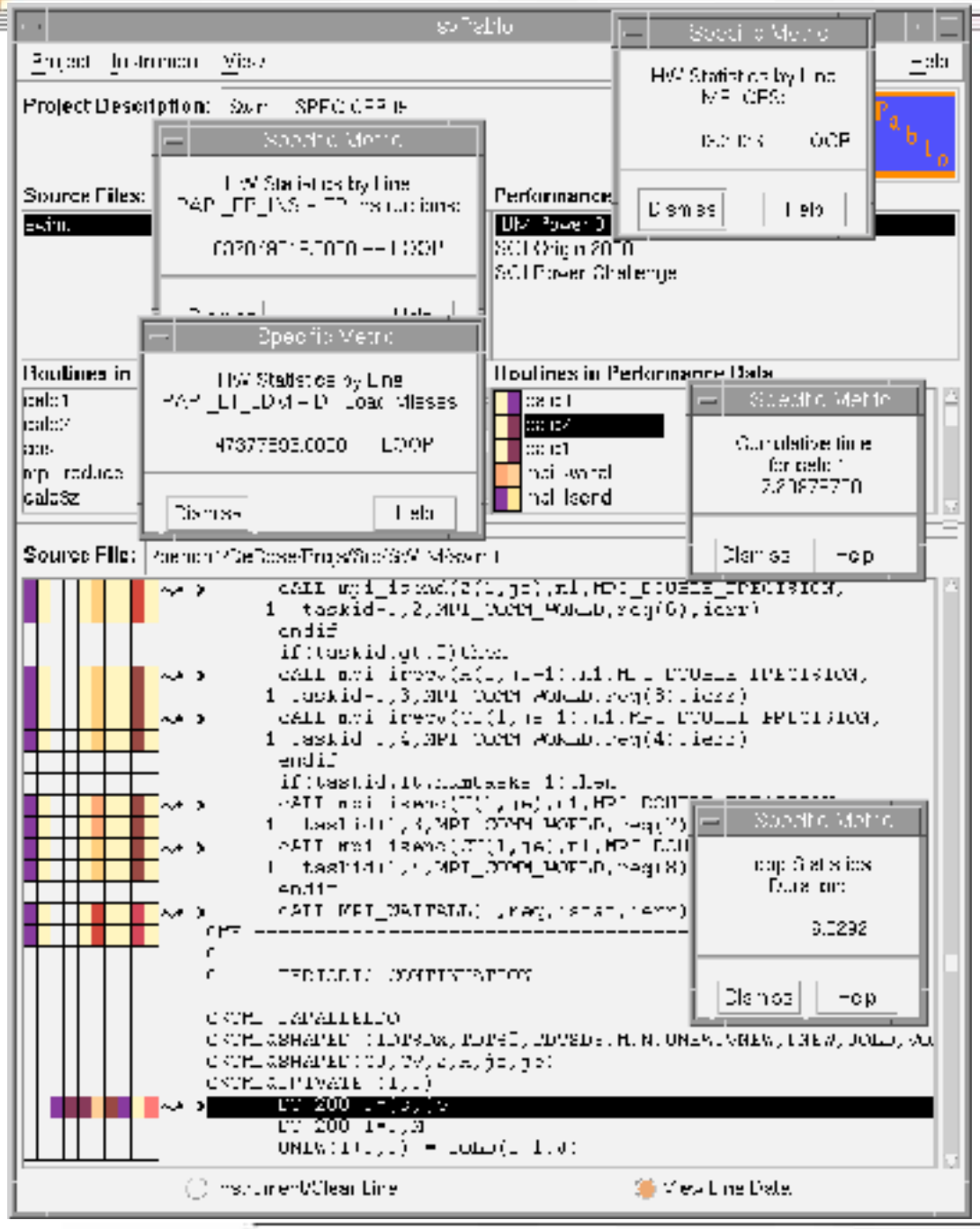
Vampir v3.x: HPM Counter

- *Counter Timeline* Display



Process Timeline Display

SvPablo from UIUC



- Source based instrumentation of loops and function calls
- Supports serial and MPI jobs
- Freely available
- Rough F90 parser

HPMToolkit

- Command line utility to gather aggregate counts.
  - PAPI version has been tested on IA32 & IA64
  - User can manually instrument code for more specific information
  - Reports derived metrics like SGI's perfex
- Developed by Luis Derose at IBM ACTC. (now at Cray)
- Hpmviz is a GUI to view results

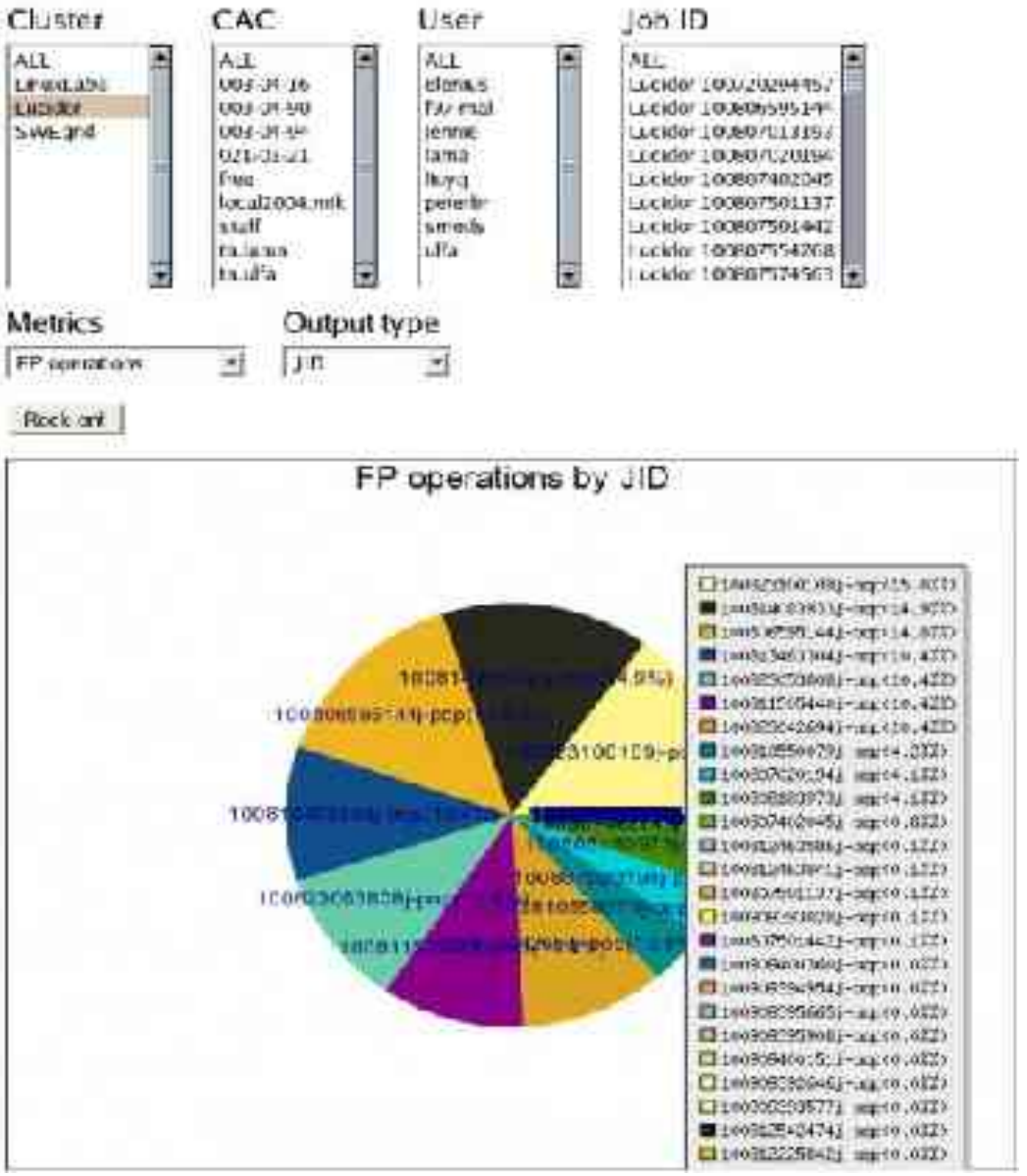
Site Wide Performance Monitoring at PDC

- Integrate complete job monitoring in the batch system itself.
- Track every cluster, group, user, job, node all the way down to individual threads.
- Zero overhead monitoring, no source code modifications.
- Near 100% accuracy.

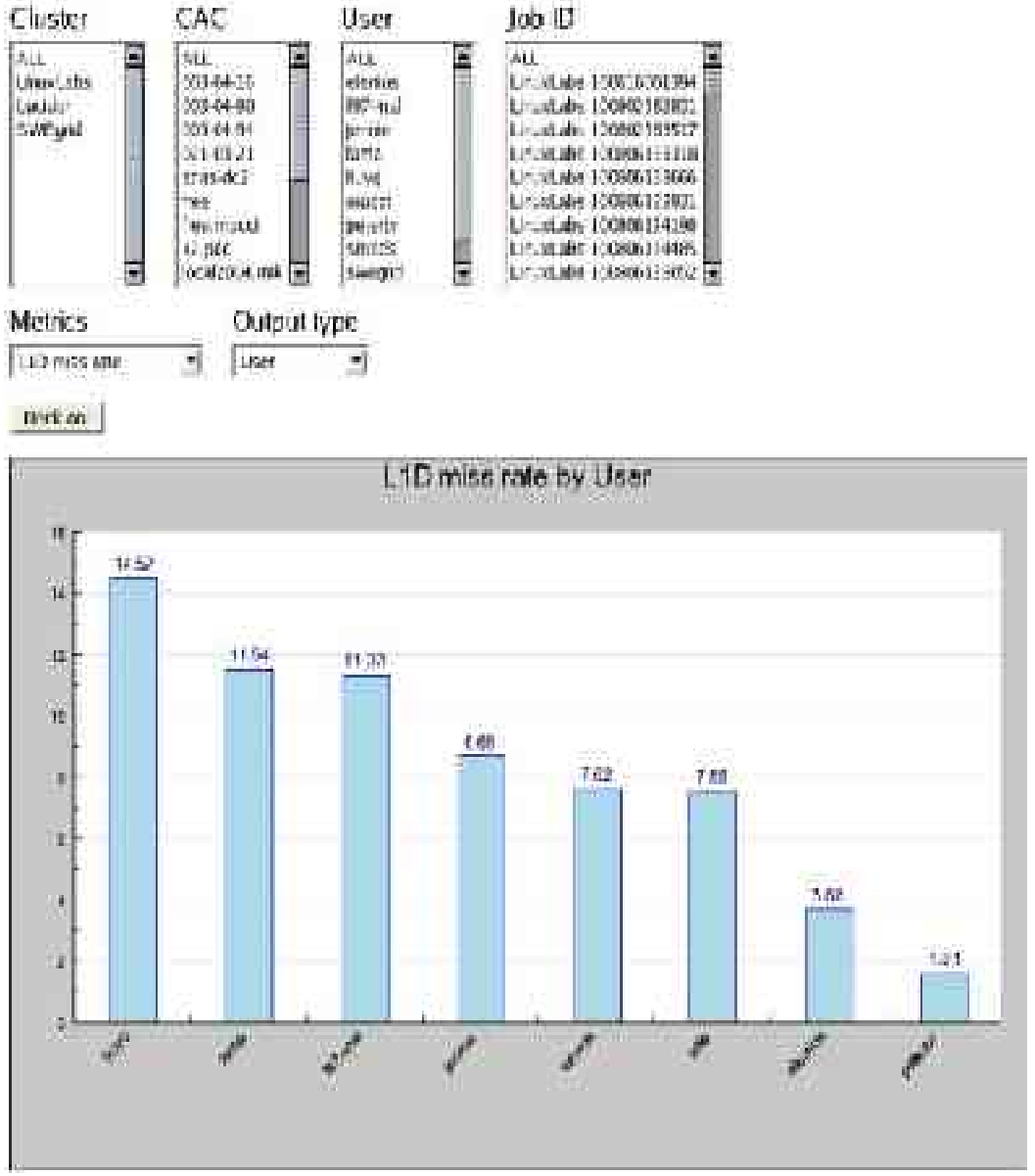
Site Wide Performance Monitoring at PDC

- Allow performance characterization of all aspects of a technical compute center:
  - Application Workloads
  - System Performance
  - Resource Utilization
- Provide users, managers and administrators with a quick and easy way to track and visualize performance of their jobs/system.
- Complete integration from batch system to database to PHP web interface.

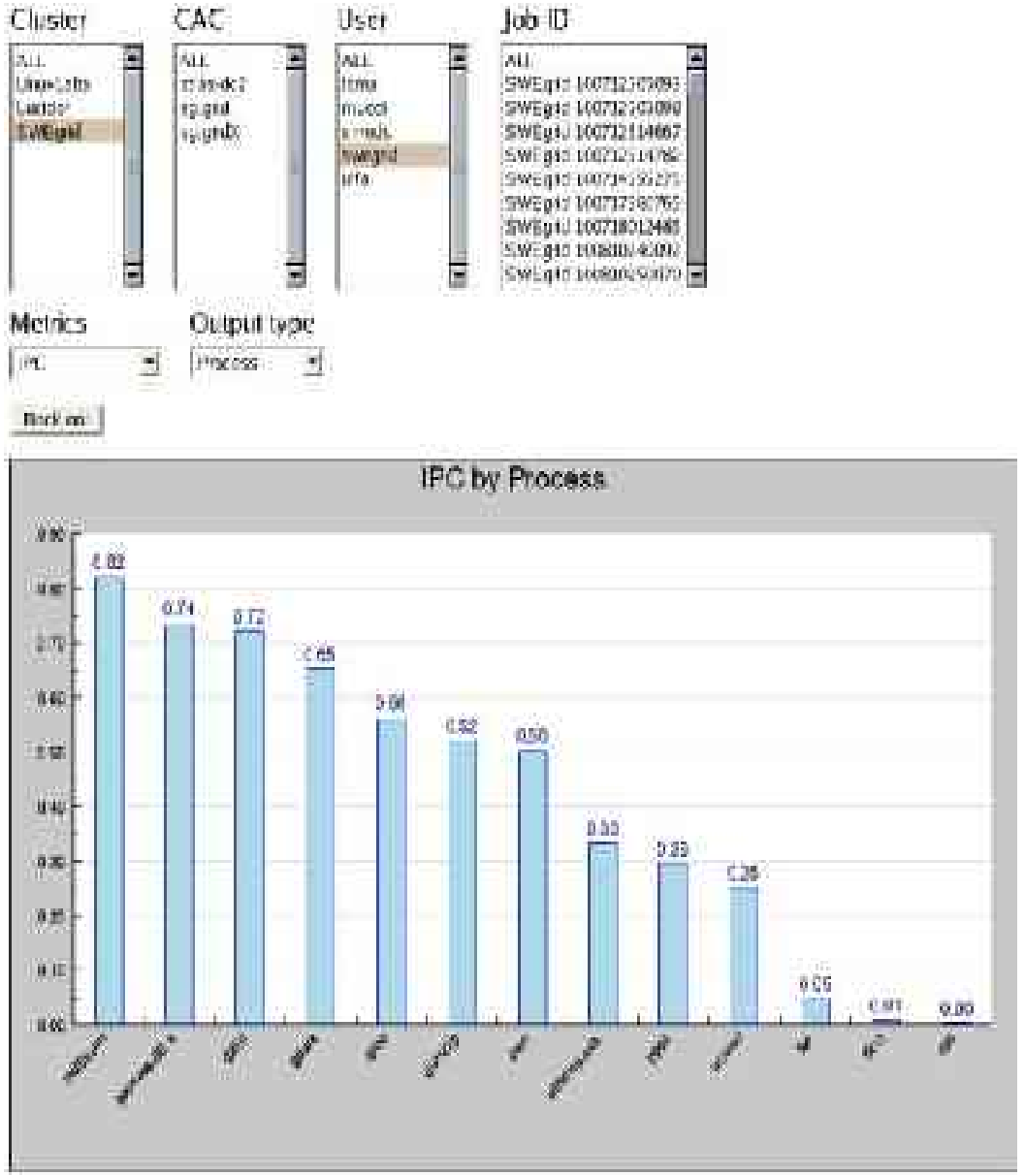
PDC Performance Miner
FP Ops by Job ID



PDC Performance Miner
L1 Miss Rate by User



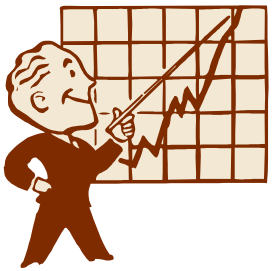
PDC Performance Miner
IPC by Process for SweGrid



Some Performance Tools

- TAU (U. Oregon)
  - Source/dynamic instrumentation and tracing system
  - http://www.cs.uoregon.edu/research/paracomp/tau/
- HPCToolkit (Rice U.)
  - Command line statistical profiling (including shlibs)
  - http://hipersoft.cs.rice.edu/hpctoolkit/
- PerfSuite and PSRUN (NCSA)
  - Command line aggregate and statistical profiling
  - http://perfsuite.ncsa.uiuc.edu

More Performance Tools



- KOJAK (Juelich, UTK)
  - Instrumentation, tracing and analysis system for MPI, OpenMP and Performance Counters.
  - http://www.fz-juelich.de/zam/kojak/
- SvPablo (UIUC)
  - Instrumentation system for Performance Counters
  - http://www-pablo.cs.uiuc.edu/Project/SVPablo
- Q-Tools (HP) (non-PAPI)
  - Statistical profiling of system and user processes
  - http://www.hpl.hp.com/research/linux/q-tools

More Performance Tools

- PapiEx: PAPI Execute
  - Passive aggregate counter measurement tool.
  - http://www.cs.utk.edu/~mucci/papiex
- DynaProf (P. Mucci, U Tenn)
  - Dynamic instrumentation tool.
  - http://www.cs.utk.edu/~mucci/dynaprof

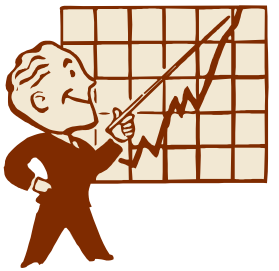
Non Open Source Tools (Why?)

- SCALEA (U Innsbruck)
  - Instrumentation system for MPI, OpenMP and Performance Counters
  - http://www.par.univie.ac.at/project/scalea/
- ParaVer (CEPBA)
  - Performance tracing for MPI, OpenMP and Performance Counters
  - http://www.cepba.upc.es/paraver
- VAMPIR (Pallas)
  - Trace visualizer for MPI and Performance Counters (when used with TAU and other systems)
  - http://www.pallas.com/e/products/vampir/index.htm

Tools that use PAPI

- TAU (Sameer Shende, U Oregon)
  http://www.cs.uoregon.edu/research/paracomp/tau/
- SvPablo (Celso Mendes, UIUC)
  http://www-pablo.cs.uiuc.edu/Project/SVPablo/
- HPCToolkit (J. Mellor-Crummey, Rice U)
  http://hipersoft.cs.rice.edu/hpctoolkit/
- psrun (Rick Kufrin, NCSA, UIUC)
  http://www.ncsa.uiuc.edu/~rkufrin/perfsuite/psrun/
- Titanium (Dan Bonachea, UC Berkeley)
  http://www.cs.berkeley.edu/Research/Projects/titanium/

Tools that use PAPI



- SCALEA (Thomas Fahringer, U Innsbruck)
  http://www.par.univie.ac.at/project/scalea/
- KOJAK (Bernd Mohr, FZ Juelich; U Tenn)
  http://www.fz-juelich.de/zam/kojak
- Cone (Felix Wolf, U Tenn)
  http://icl.cs.utk.edu/kojak/cone
- HPMtoolkit (Luiz Derose, IBM)
  http://www.alphaworks.ibm.com/tech/hpmtoolkit
- CUBE (Felix Wolf, U Tenn)
  http://icl.cs.utk.edu/kojak/cube

Tools that use PAPI

- ParaVer (J. Labarta, CEPBA)
  http://www.cepba.upc.es/paraver
- VAMPIR (Pallas)

  http://www.pallas.com/e/products/vampir/index.htm
- DynaProf (P. Mucci, U Tenn)

  http://www.cs.utk.edu/~mucci/dynaprof