Tau
Introduction

Lars Koesterke
(& Kent Milfeld, Sameer Shende)

Cornell University
Ithaca, NY

March 13, 2009

Outline

- General
  - Measurements
  - Instrumentation & Control
  - Example: matmult
- Profiling and Tracing
  - Event Tracing
  - Steps for Performance Evaluation
  - Tau Architecture
- A look at a task-parallel MxM Implementation
- Paraprof Interface

General

- Tuning and Analysis Utilities (11+ year project effort)

  www.cs.uoregon.edu/research/paracomp/tau/

- *Performance system **framework*** for parallel, shared & distributed memory systems

- Targets a general complex system computation model
  - Nodes / Contexts / Threads

- ***Integrated toolkit*** for performance instrumentation, measurement, analysis, and visualization

TAU = Profiler and Tracer + Hardware Counters + GUI + Database

TACC

Tau: Measurements

- Parallel profiling
  – Function-level, block (loop)-level, statement-level
  – Supports user-defined events
  – TAU parallel profile data stored during execution
  – Hardware counter 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

TACC

Tau: Instrumentation

PDT is used to instrument your code.

Replace mpicc and mpif90 in make files with tau_f90.sh and tau_cc.sh

It is necessary to specify all the components that will be used in the instrumentation (mpi, openmp, profiling, counters [PAPI], etc. However, these come in a limited number of combinations.)

Combinations: First determine what you want to do (profiling, PAPI counters, tracing, etc.) and the programming paradigm (mpi, openmp), and the compiler. PDT is a required component:

Instrumentation	Parallel Paradigm	Collectors	Compiler:
PDT Hand-code	MPI OMP …	PAPI Callpath …	intel pgi gnu

Tau: Instrumentation

You can view the available combinations

(alias tauTypes 'ls -C1 $TAU | grep Makefile ').

Your selected combination is made known to the compiler wrapper through the TAU_MAKEFILE environment variable.

E.g. the PDT instrumention (pdt) for the Intel compiler (icpc) for MPI (mpi) is set with this command:

```
setenv TAU_MAKEFILE   /…/Makefile.tau-icpc-mpi-pdt
```

Other run-time and instrumentation options are set through TAU_OPTIONS. For verbose:

```
setenv TAU_OPTIONS   ‘-optVerbose’
```

Tau Example

% tar –xvf ~train00/tau.tar	
% cd tau	★ READ the Instructions file
% source sourceme.csh or % source sourceme.sh	create env. (modules and TAU_MAKEFILE)
% make matmultf or % make matmultc	create executable(s)
% qsub job	submit job (edit and uncomment ibrun line)
% paraprof	(for GUI) Analyze performance data:

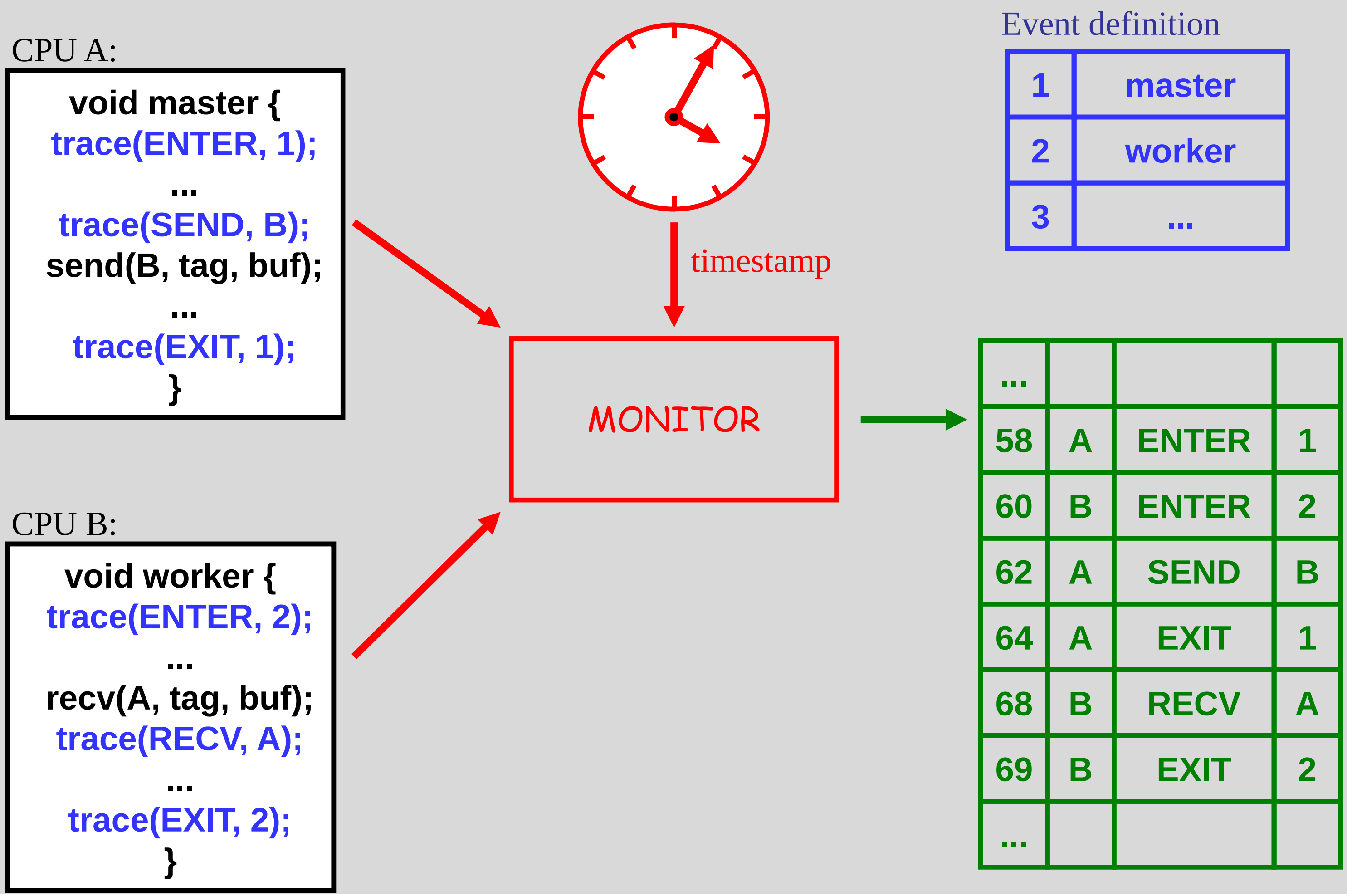
Definitions – Profiling

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

Definitions – Tracing

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

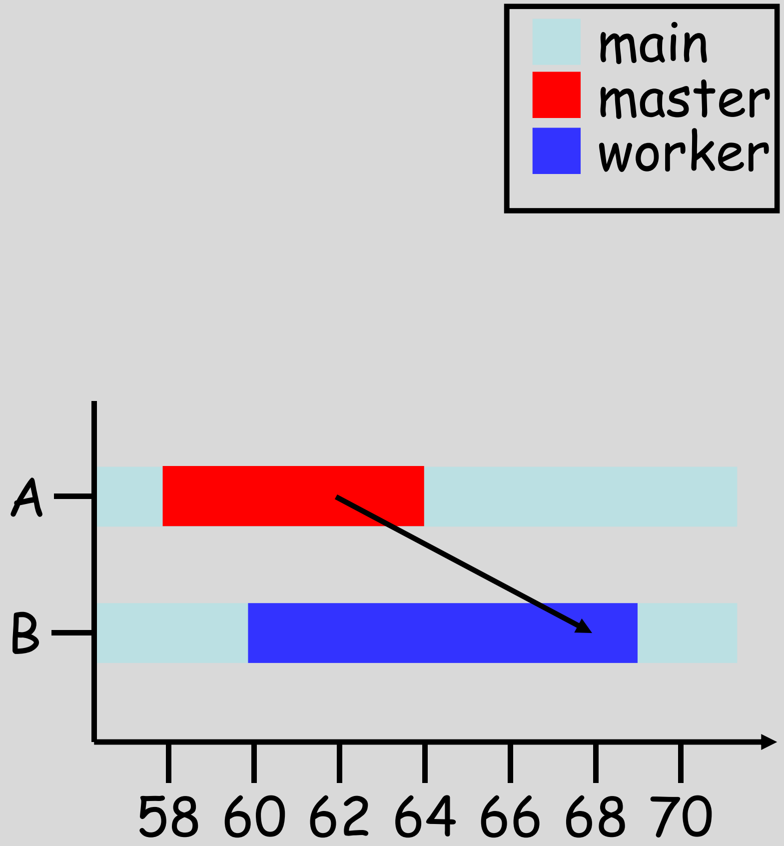
Event Tracing: Instrumentation, Monitor, Trace



Event Tracing: “Timeline” Visualization

1	master
2	worker
3	...

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

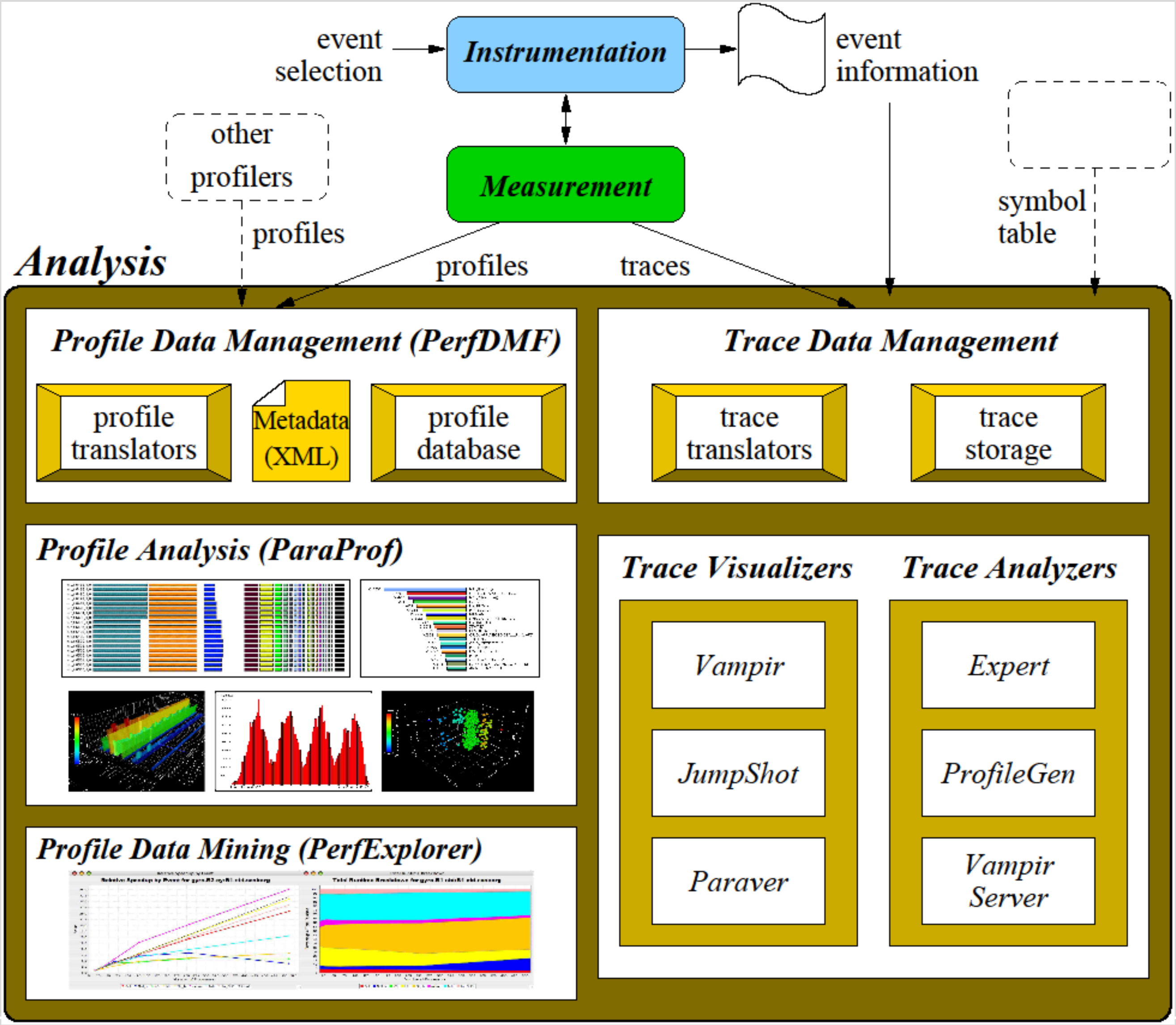


Steps of Performance Evaluation

- Collect basic routine-level timing profile to determine where most time is being spent
- Collect routine-level hardware counter data to determine types of performance problems
- Collect callpath profiles to determine sequence of events causing performance problems
- Conduct finer-grained profiling and/or tracing to pinpoint performance bottlenecks
  - Loop-level profiling with hardware counters
  - Tracing of communication operations

TACC

TAU Performance System Architecture



Overview of Matmult: C = A x B

C = A x B

Order N
P Tasks

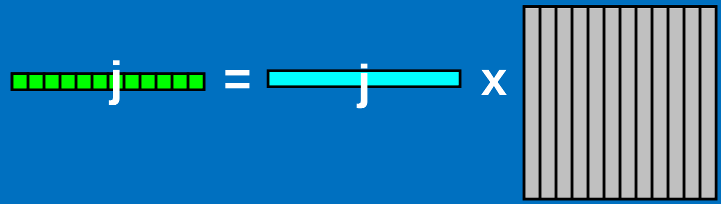
MASTER | **Worker**

Create A & B

Send B → **Receive B**

Send Row of A → **Receive a**

Multiply row a x B



Receive Row of C ← **Send Back row of C**

TACC

Preparation of Matmult: C = A x B

C = A x B

Order N
P Tasks

MASTER

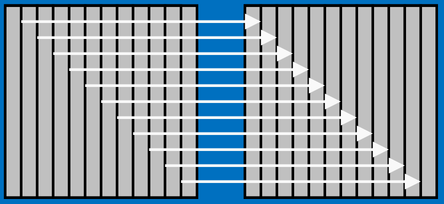
Generate A & B

PE 0 — **Create A**

PE 0 — **Create B**

Broadcast B to All by columns

PE 0 → PE x



loop over i (i=1→n)

MPI_Bcast(b(1,i)…

Master Ops of Matmult: C = A x B

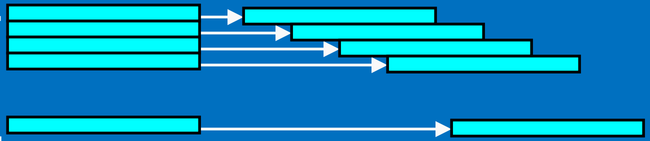
C = A x B

Order N
P Tasks

MASTER

Master (0) sends rows 1 through (p-1) to slaves (1→p-1) receives

PE 0 PE 1 -- p-1



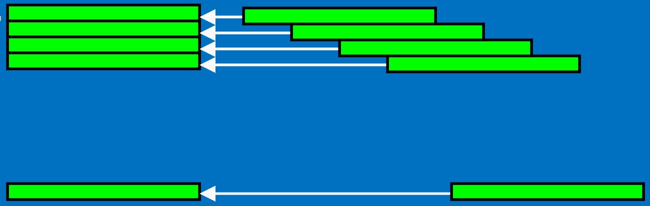
loop over i (i=1→p-1)

MPI_Send(arow … i,i

destination
tag

Master (0) receives rows 1 through (n) from Slaves.

PE 0 PE 1 -- p-1



loop over i (i=1→n)

source,tag

MPI_Recv(crow … ANY,k

MPI_Send(arow …idle,j

dest,tag

Master Ops of Matmult: C = A x B

C = A x B

Order N
P Tasks

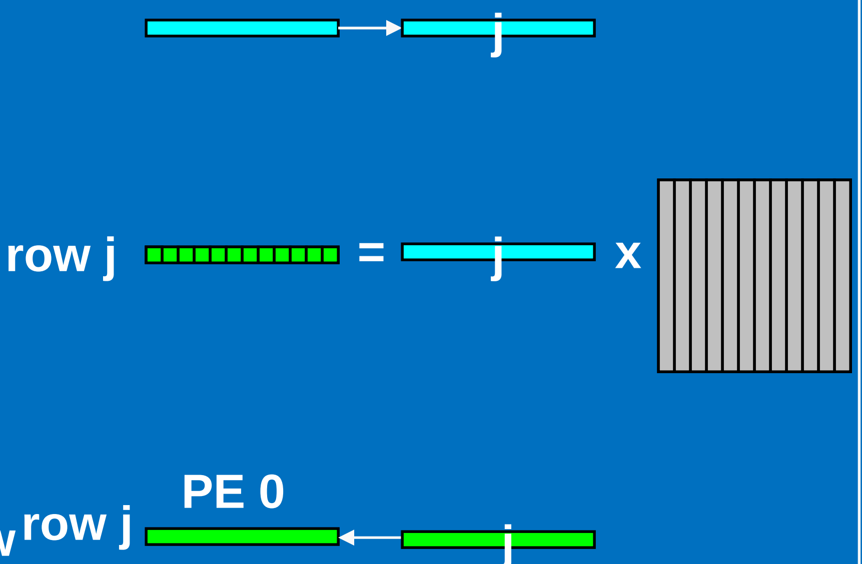
Worker

Pick up broadcast of B columns from PE 0

Slave receives any A row from PE 0

Slaves multiply all Columns of B into A (row i) to form row i of Matrix C

Slave(any) sends row j of C to master, PE 0



loop over i (i=1→n)

MPI_Recv(arow …ANY,j

Matrix * Vector

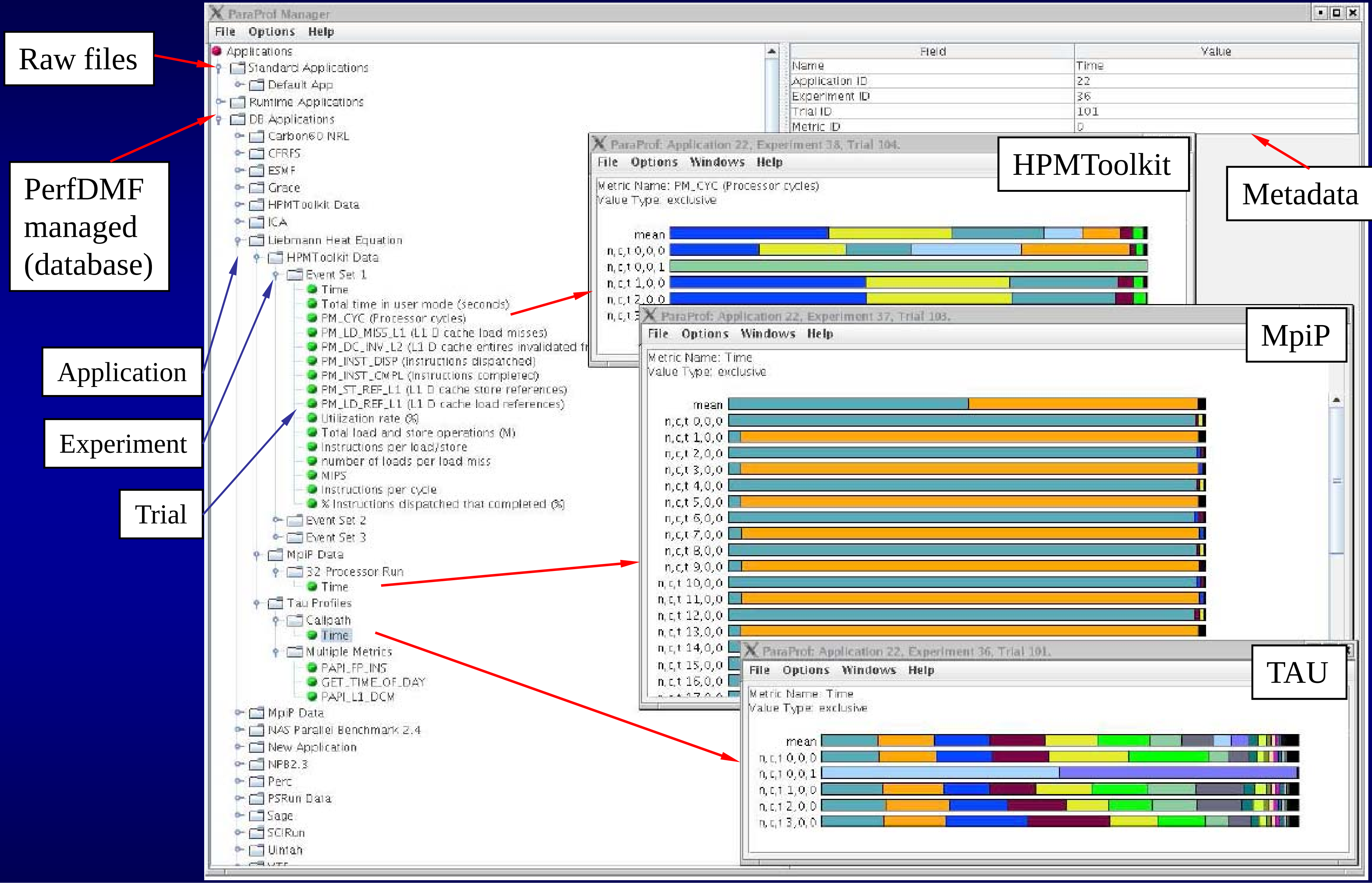
MPI_Send(crow … j

Paraprof and Pprof

- Execute application and analyze performance data:
- % qsub job
  - – Look for files: profile.<task_no>.
  - – With Multiple counters, look for directories for each counter.
- % pprof (for text based profile display)
- % paraprof (for GUI)
  - – pprof and paraprof will discover files/directories.
  - – paraprof runs on PCs,Files/Directories can be downloaded to laptop and analyzed there.

TACC

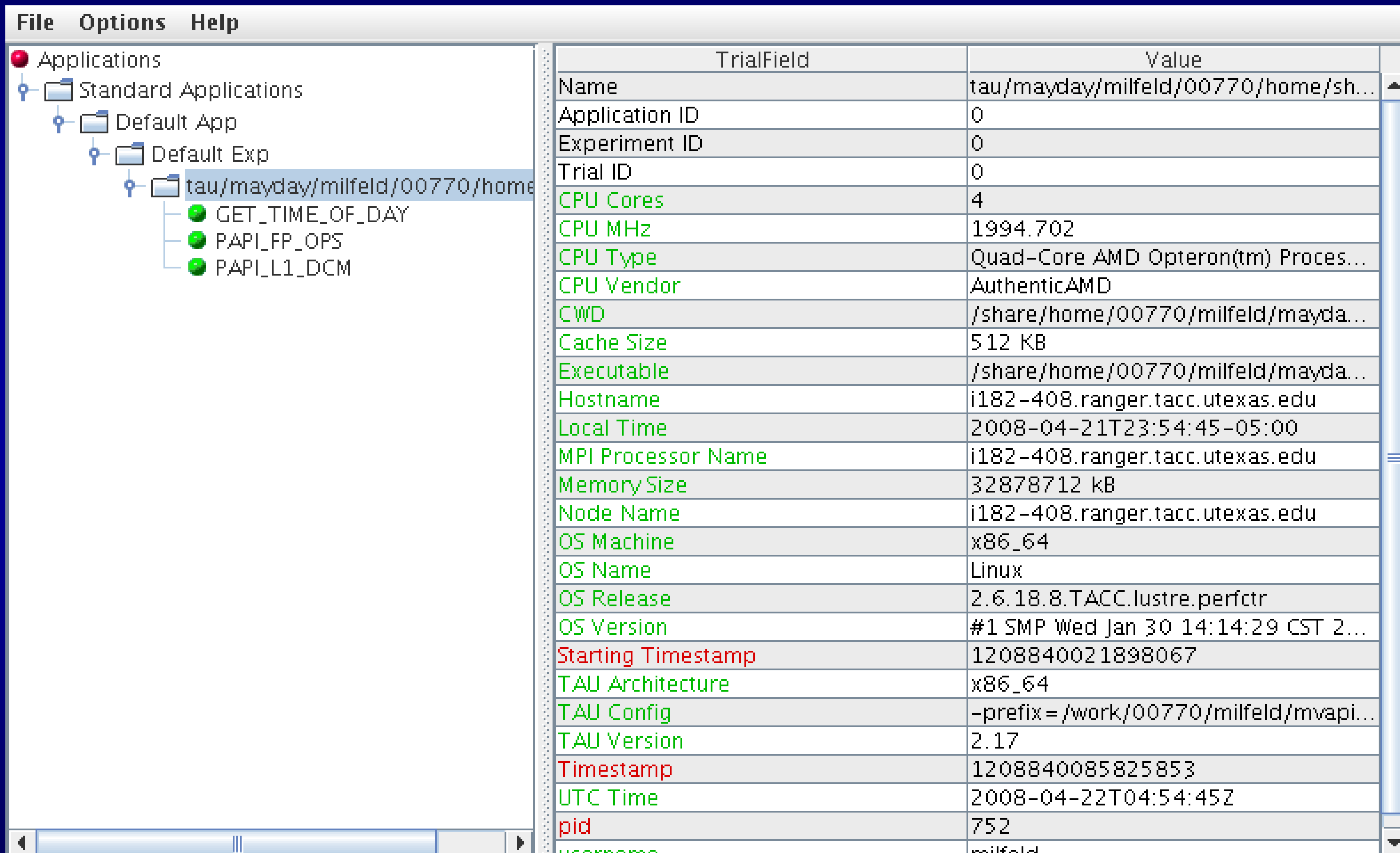
Tau Paraprof Overview



Tau Paraprof Manager Window

Provides Machine Details

Organizes Runs as: Applications, Experiments and Trials.

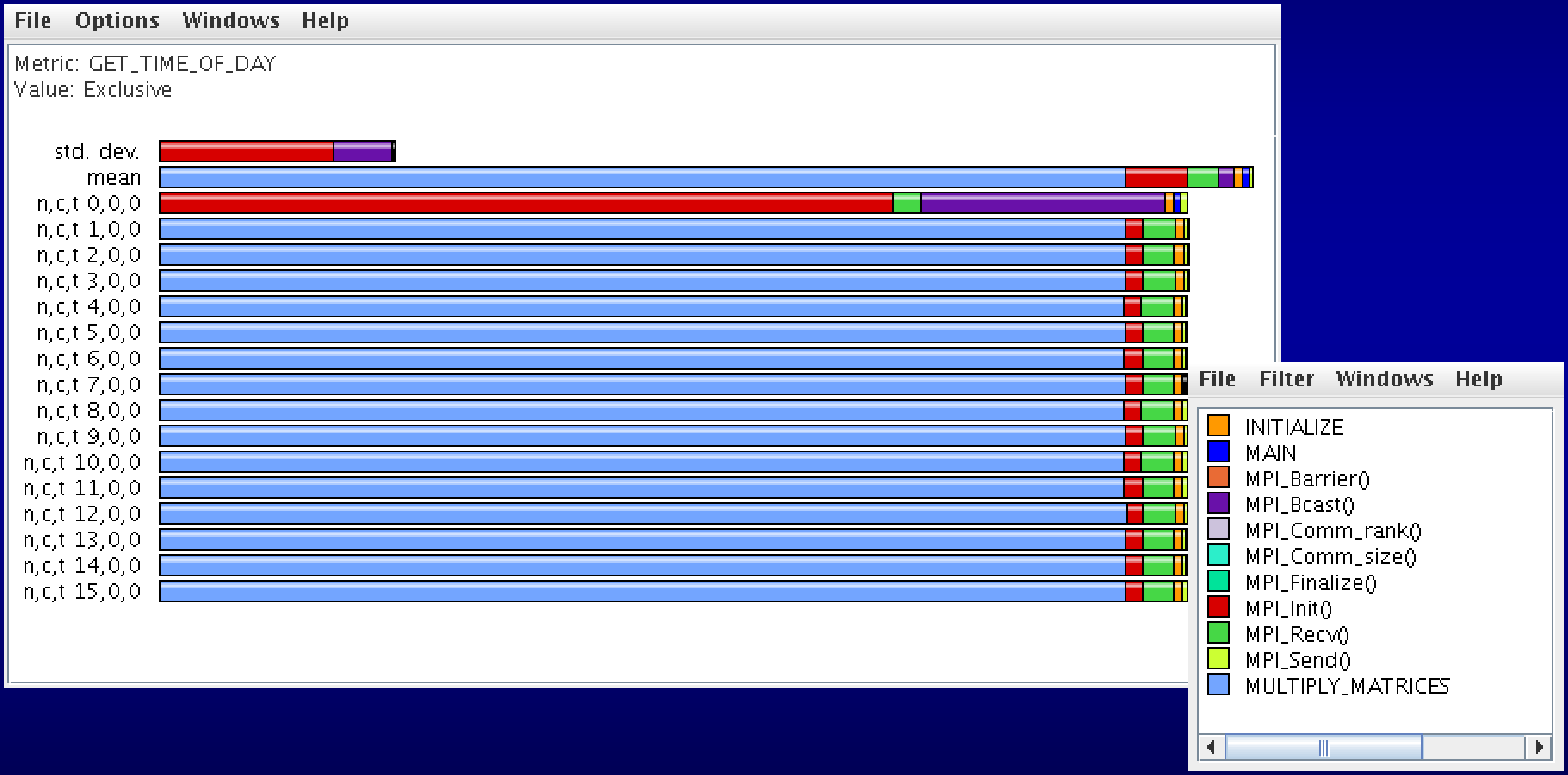


TACC

Routine Time Experiment

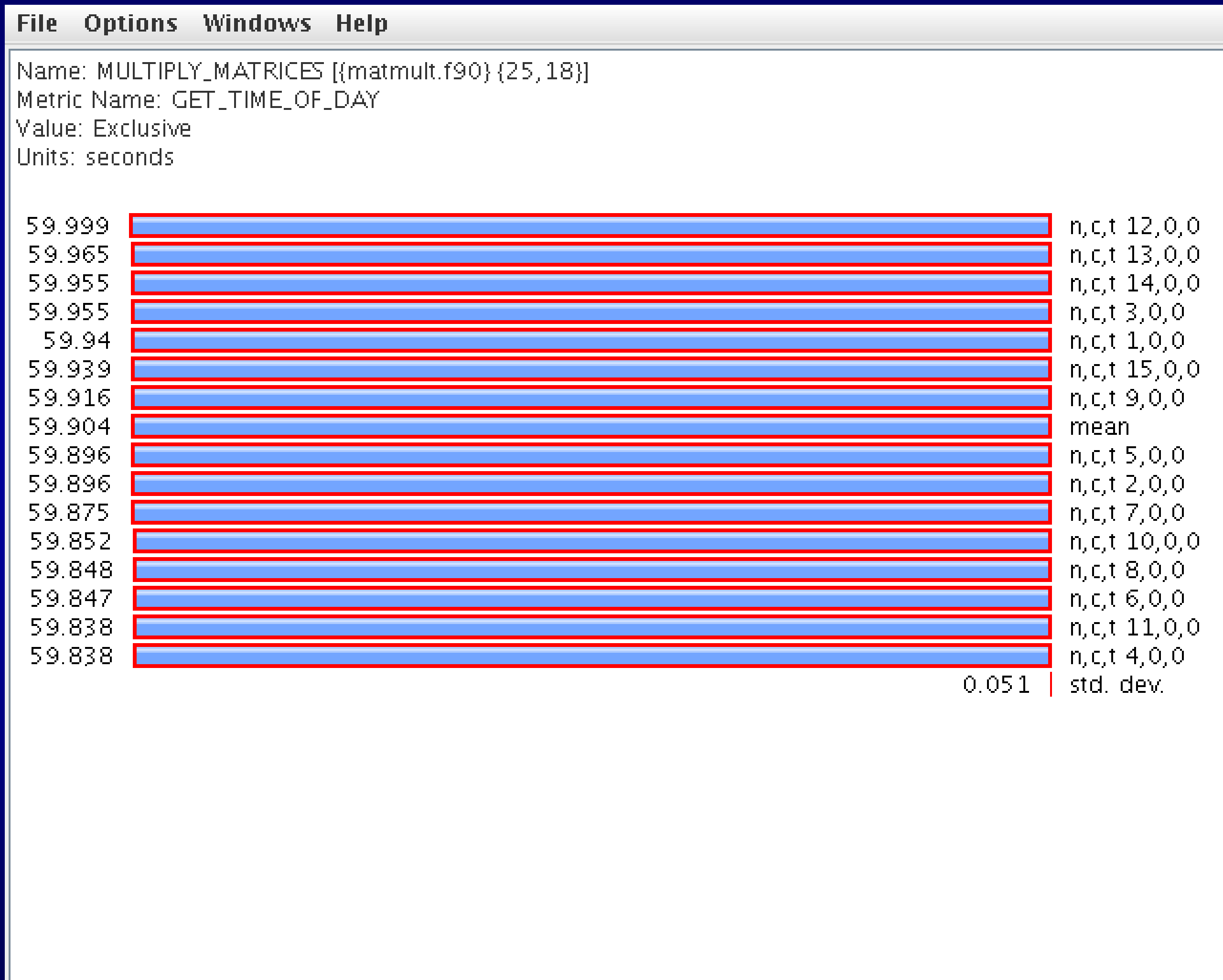
Profile Information is in “GET_TIME_OF_DAY” metric

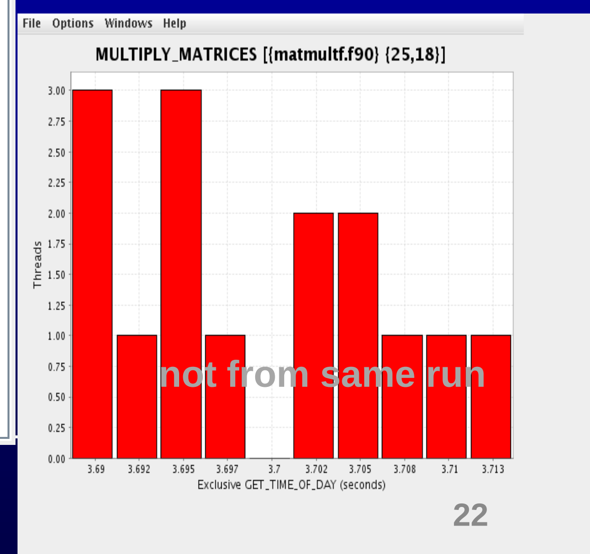
Mean and Standard Deviation Statistics given.



Multiply_Matrices Routine Results

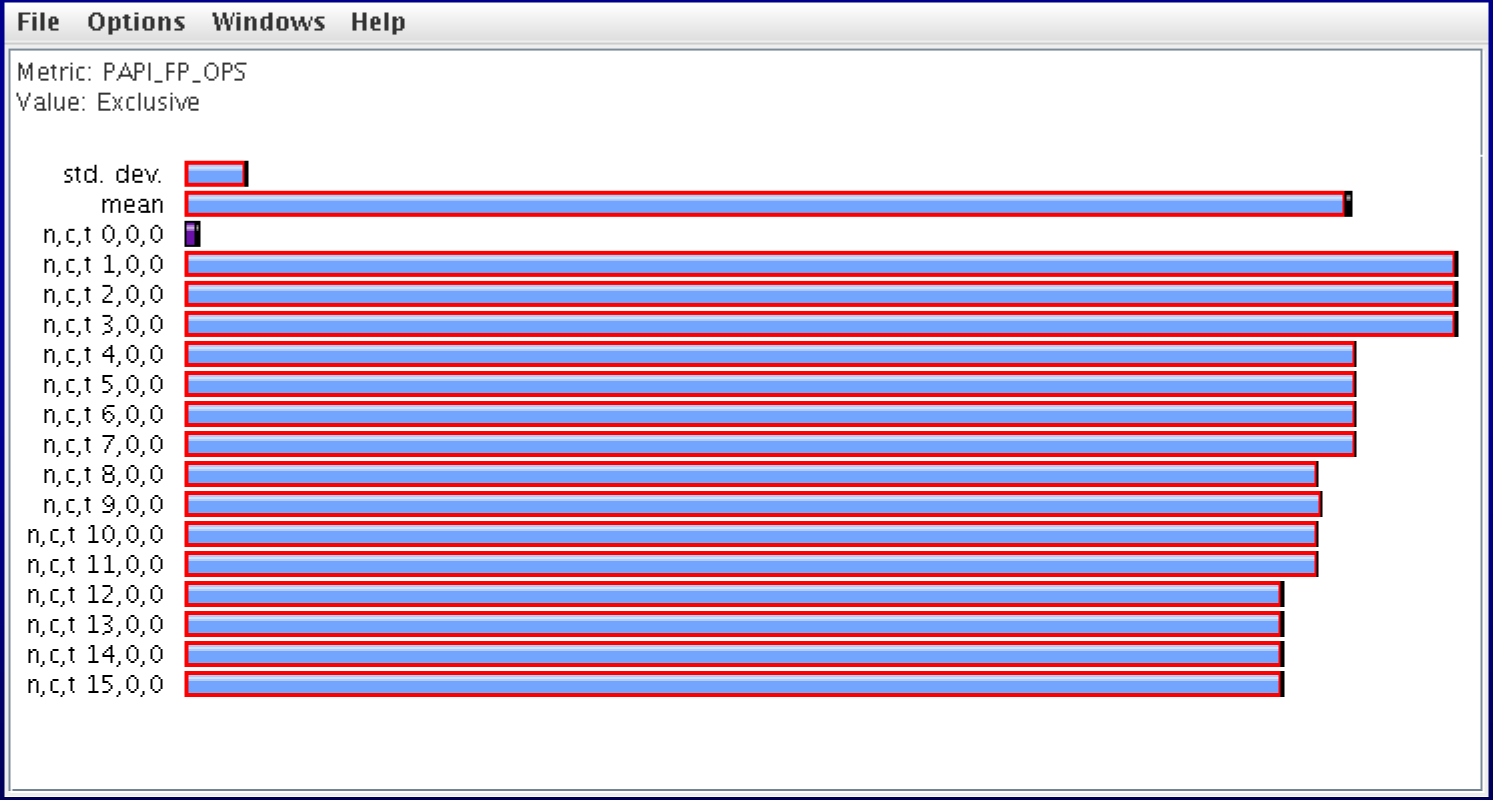
Function Data Window gives a closer look at a single function:





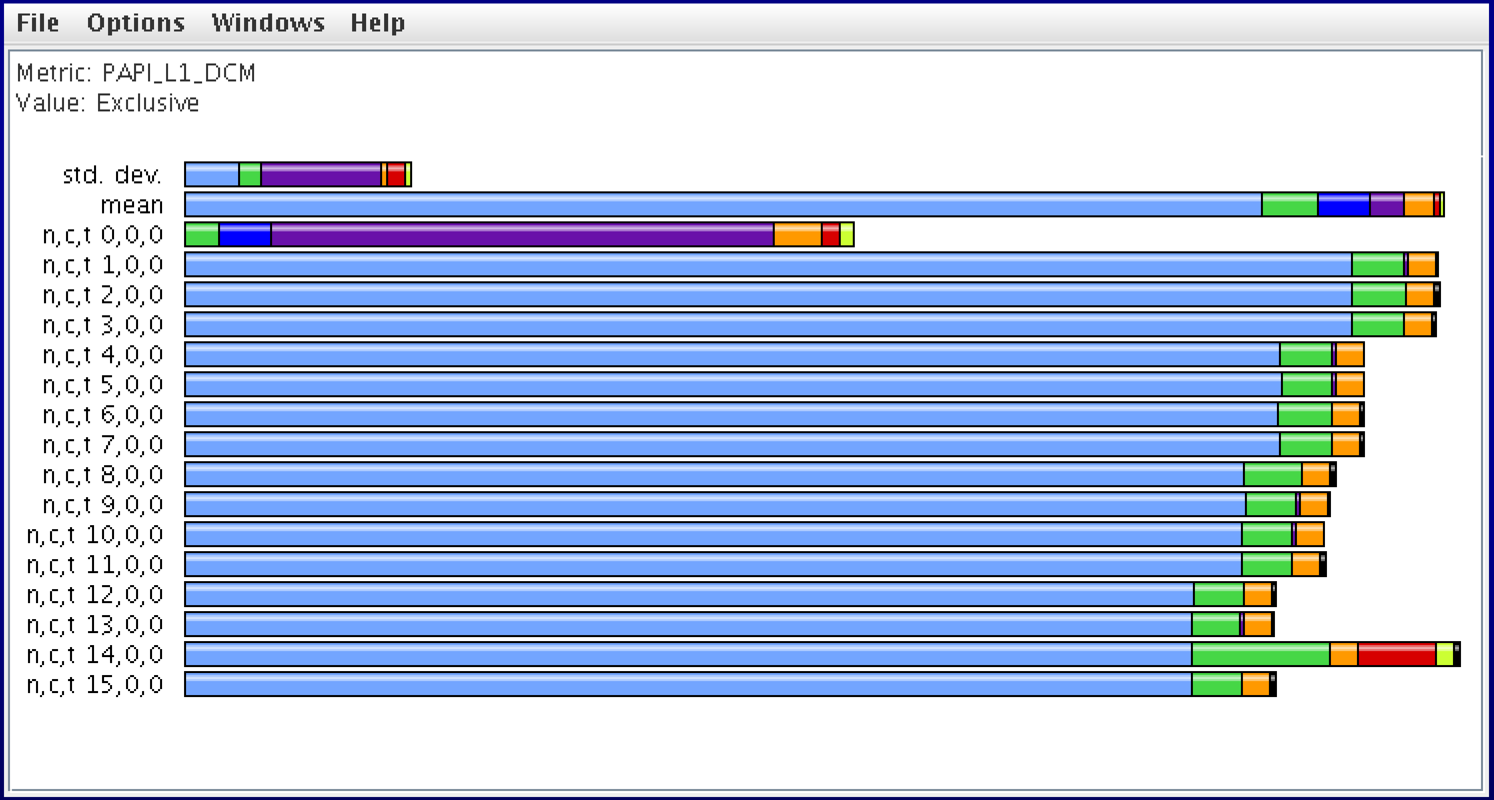
Float Point OPS trial

Hardware Counters provide Floating Point Operations (Function Data view).



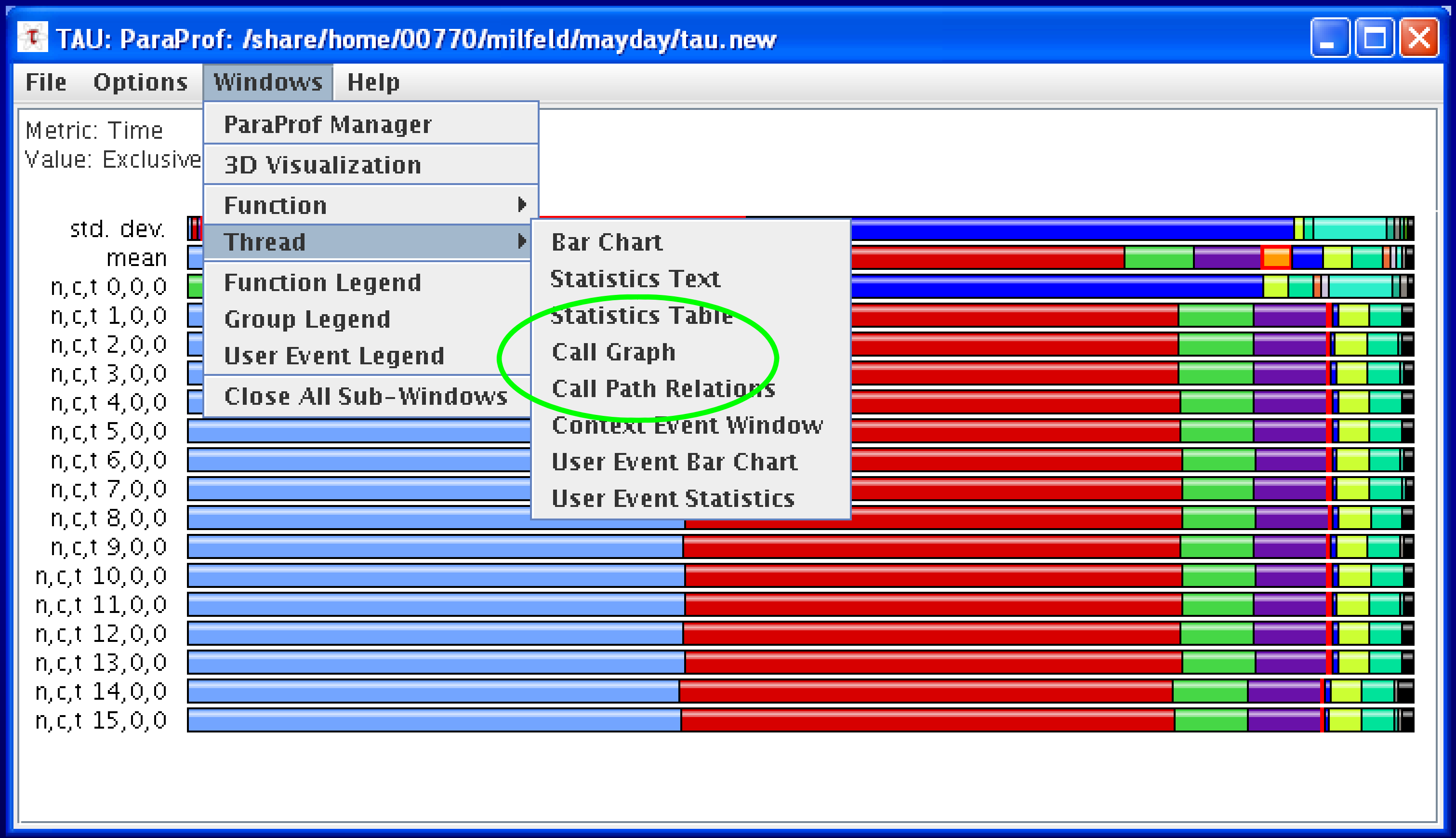
L1 Data Cache Miss trial

Hardware Counters provide L1 Cache Miss Operations.



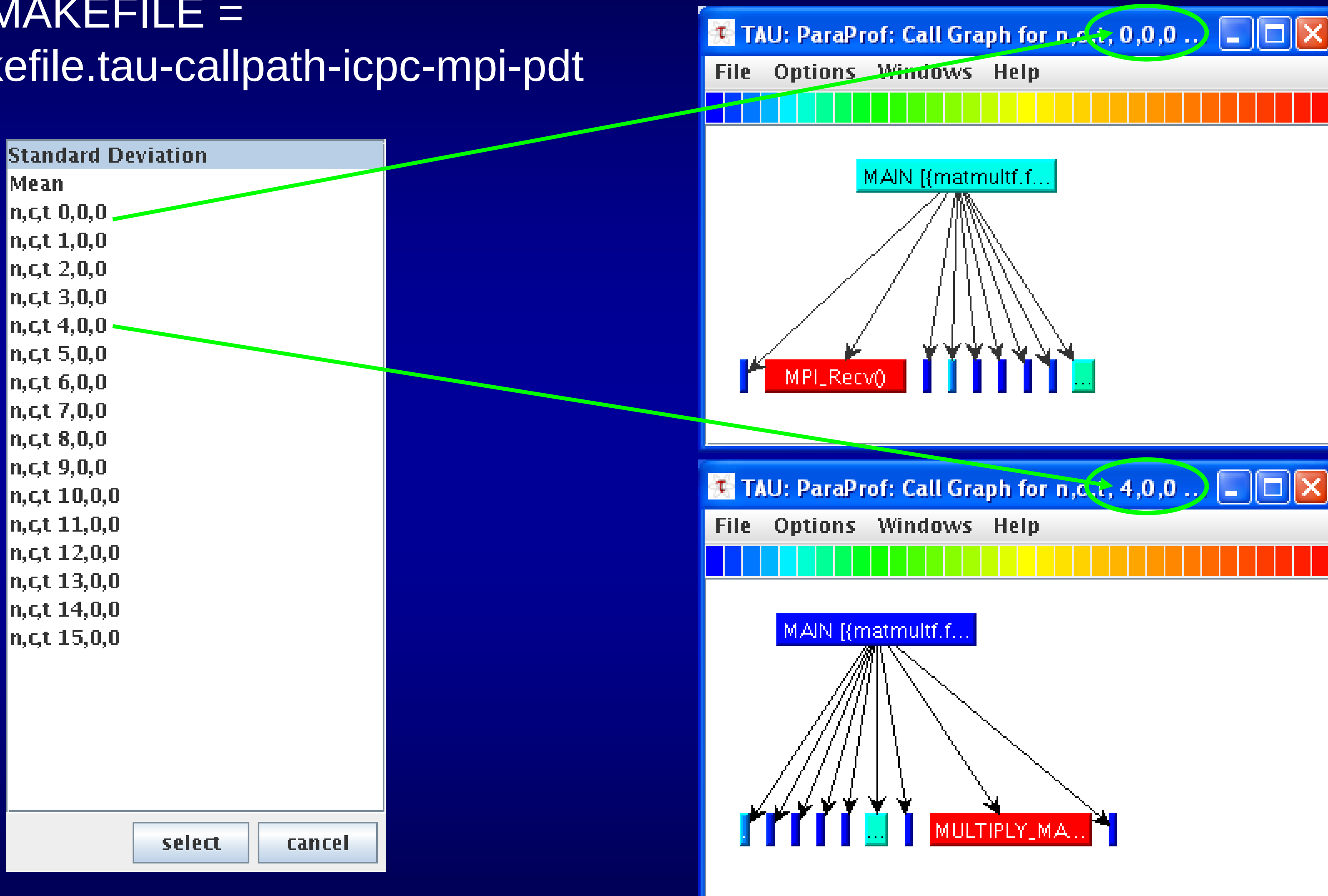
Call Path

Call Graph Paths (Must select through “thread” menu.)



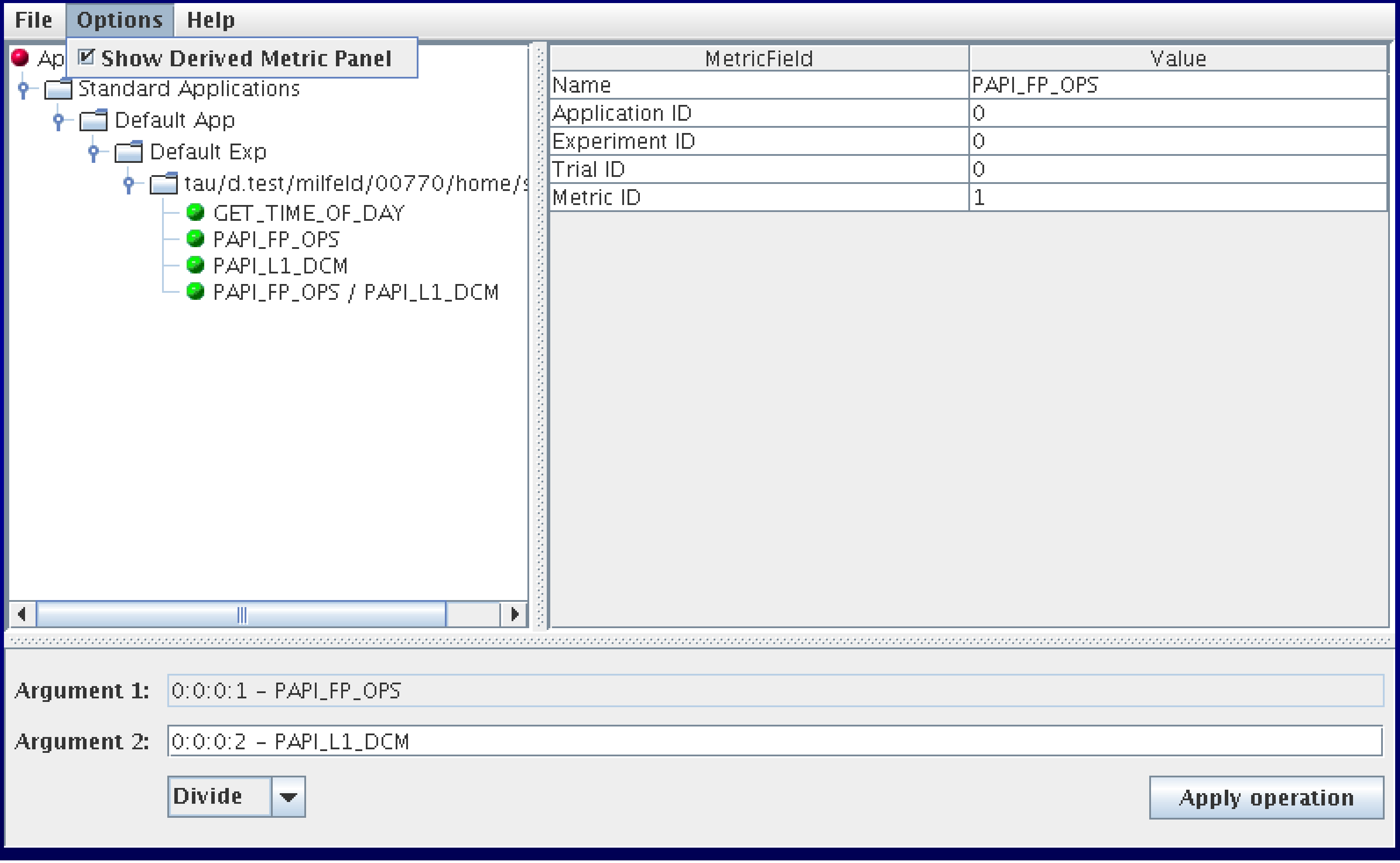
Call Path

TAU_MAKEFILE =
…Makefile.tau-callpath-icpc-mpi-pdt



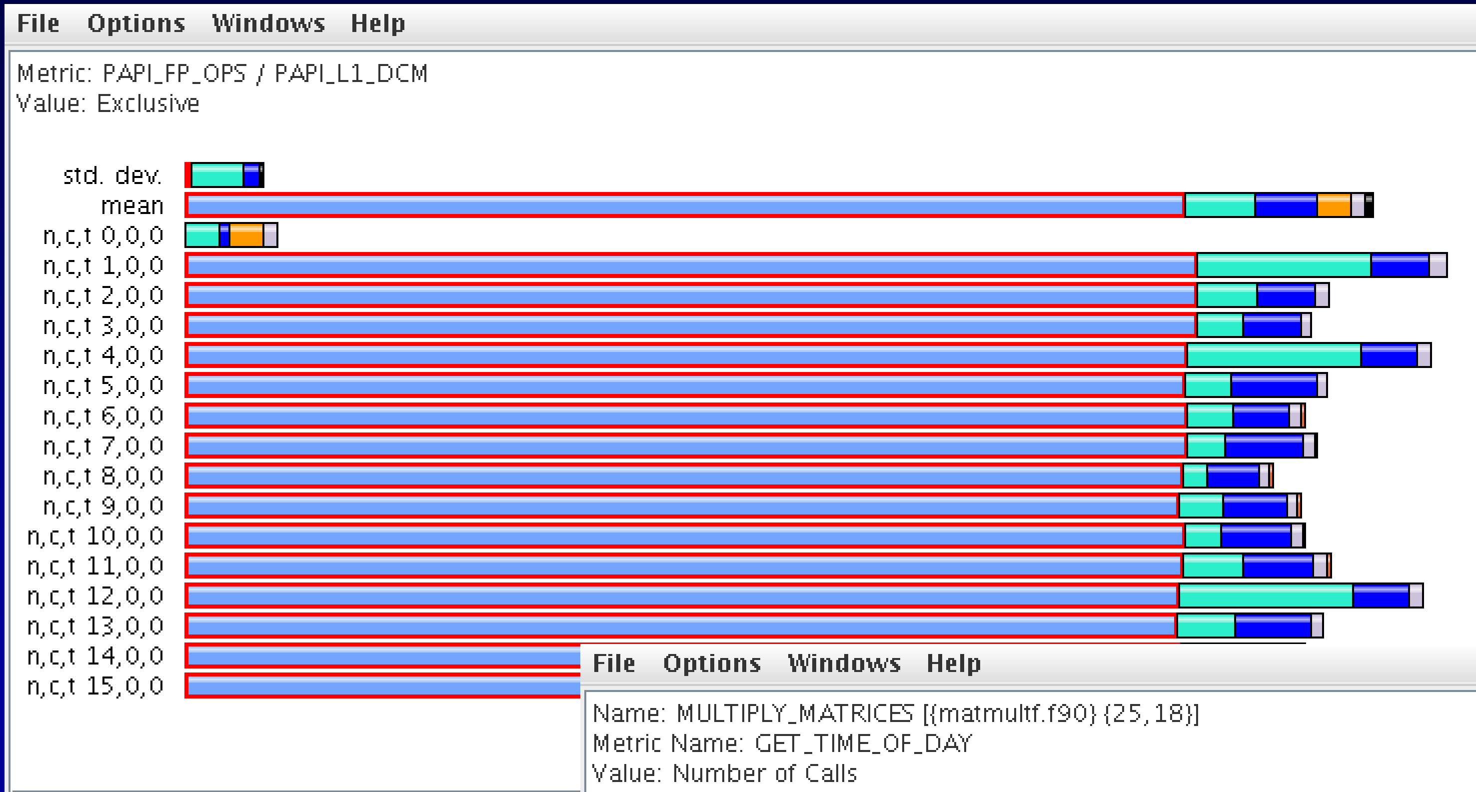
Derived Metrics

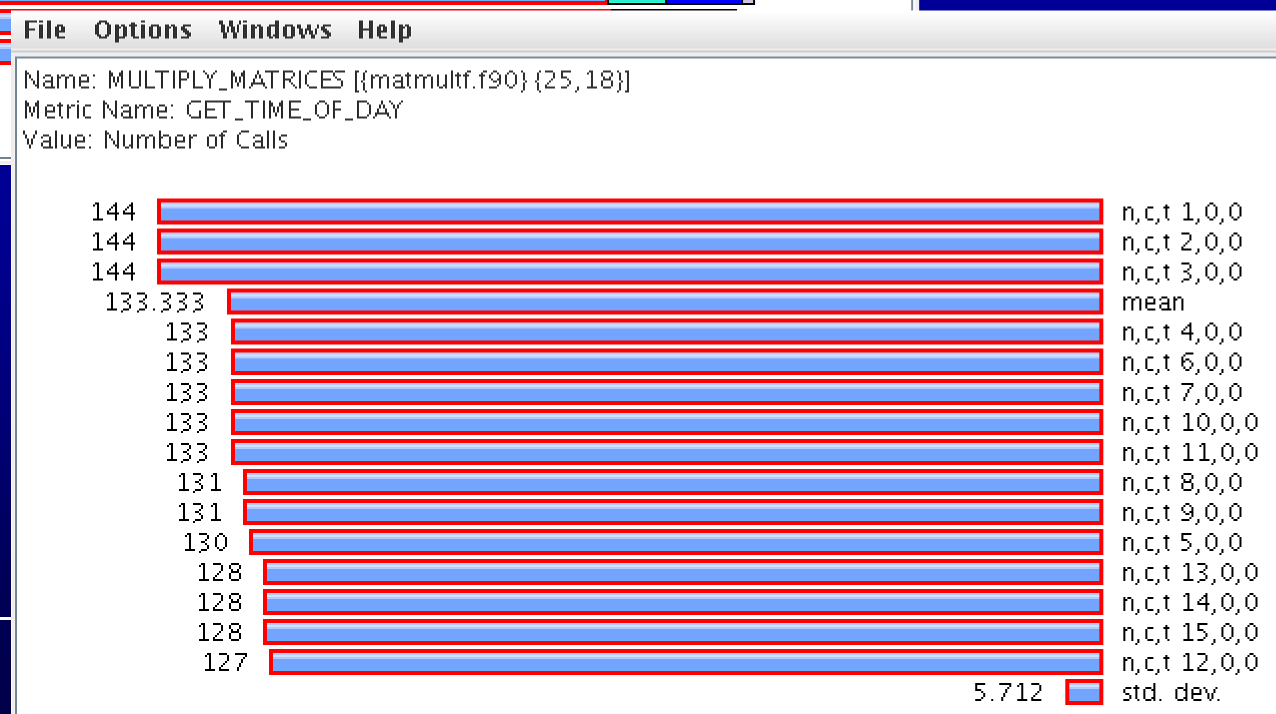
Select Argument 1 (green ball); Select Argument 2 (green ball);
Select Operation; then Apply. Derived Metric will appear as a new trial.



Derived Metrics

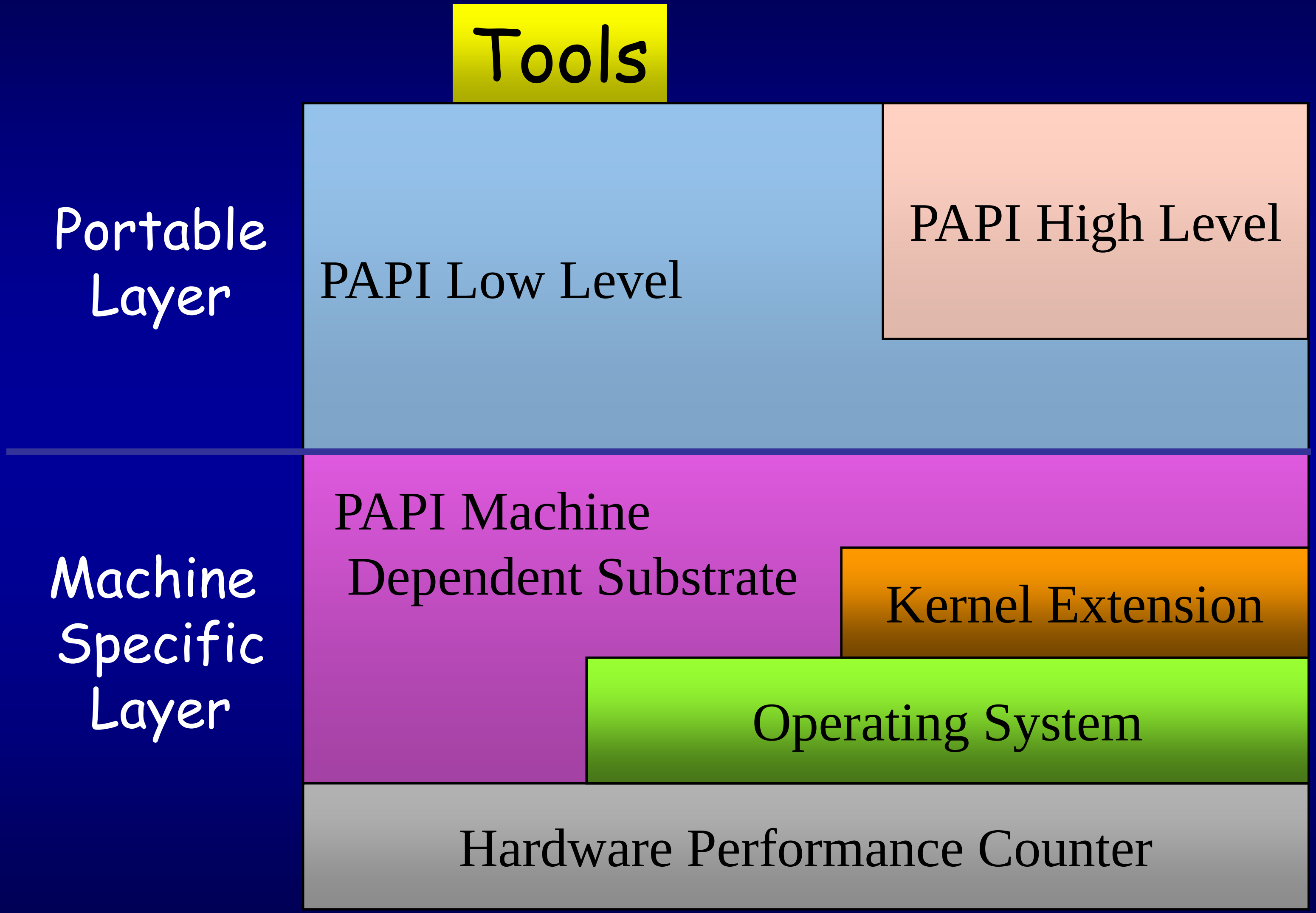
Since FP/Miss ratios are constant– must be memory access problem.





Be careful→ even though ratios are constant, cores may do different amounts of work/operations per call.

PAPI Implementation



PAPI Performance Monitor

- Provides high level counters for events:
  - Floating point instructions/operations,
  - Total instructions and cycles
  - Cache accesses and misses
  - Translation Lookaside Buffer (TLB) counts
  - Branch instructions taken, predicted, mispredicted
- PAPI_flops routine for basic performance analysis
  - Wall and processor times
  - Total floating point operations and MFLOPS

    http://icl.cs.utk.edu/projects/papi
- Low level functions are thread-safe, high level are not

PAPI Preset Events

- Proposed standard set of events deemed most relevant for application performance tuning
- Defined in papiStdEventDefs.h
- Mapped to native events on a given platform
  - Run tests/avail to see list of PAPI preset events available on a platform

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_flips
  PAPI_ipc

- Fortran interface
  PAPIF_start_counters
  PAPIF_read_counters
  PAPIF_stop_counters
  PAPIF_accum_counters
  PAPIF_num_counters
  PAPIF_flips
  PAPIF_ipc

Low-level Interface

- Increased efficiency and functionality over the high level PAPI interface
- About 40 functions
- Obtain information about the executable and the hardware
- Thread-safe
- Fully programmable
- Callbacks on counter overflow

PAPI counters in Tau

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

Important Environment Variables

- Choose the measurement option and compile your code:
- setenv TAU_MAKEFILE $TAU/Makefile.tau-icpc-mpi-pdt
- setenv TAU_OPTIONS '-optVerbose -optKeepFiles -optPreProcess'
- setenv TAU_THROTTLE 1   At runtime to keep instrumentation overhead in check

TACC

Fortran TAU Tips

- If your Fortran code uses free format in .f files (fixed is default for .f), you may use:
- % setenv TAU_OPTIONS '-optPdtF95Opts="-R free" -optVerbose '

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

  % setenv TAU_OPTIONS '-optPdtGnuFortranParser -optVerbose'

- If your Fortran code uses C preprocessor directives (#include, #ifdef, #endif):

  % setenv TAU_OPTIONS '-optPreProcess -optVerbose -optDetectMemoryLeaks'

- To use an instrumentation specification file:

  % setenv TAU_OPTIONS '-optTauSelectFile=mycmd.tau -optVerbose -optPreProcess'

  % cat mycmd.tau

```
BEGIN_INSTRUMENT_SECTION
memory file="foo.f90" routine="#"
# instruments all allocate/deallocate statements in all routines in foo.f90
loops file="*" routine="#"
io file="abc.f90" routine="FOO"
END_INSTRUMENT_SECTION
```

References

- Performance Research Laboratory, University of Oregon, Eugene, sameer@cs.uoregon.edu
- http://www.cs.uoregon.edu/research/tau/about.php