Performance Evaluation using the TAU Performance System®

Incite Getting Started Workshop at Argonne National Laboratory, ALCF, Bldg. 240, Jan 27-29, 2010, 8:30am - 5:00pm, Argonne, IL

Sameer Shende

sameer@paratools.com

http://www.paratools.com/anl10

Acknowledgements: University of Oregon

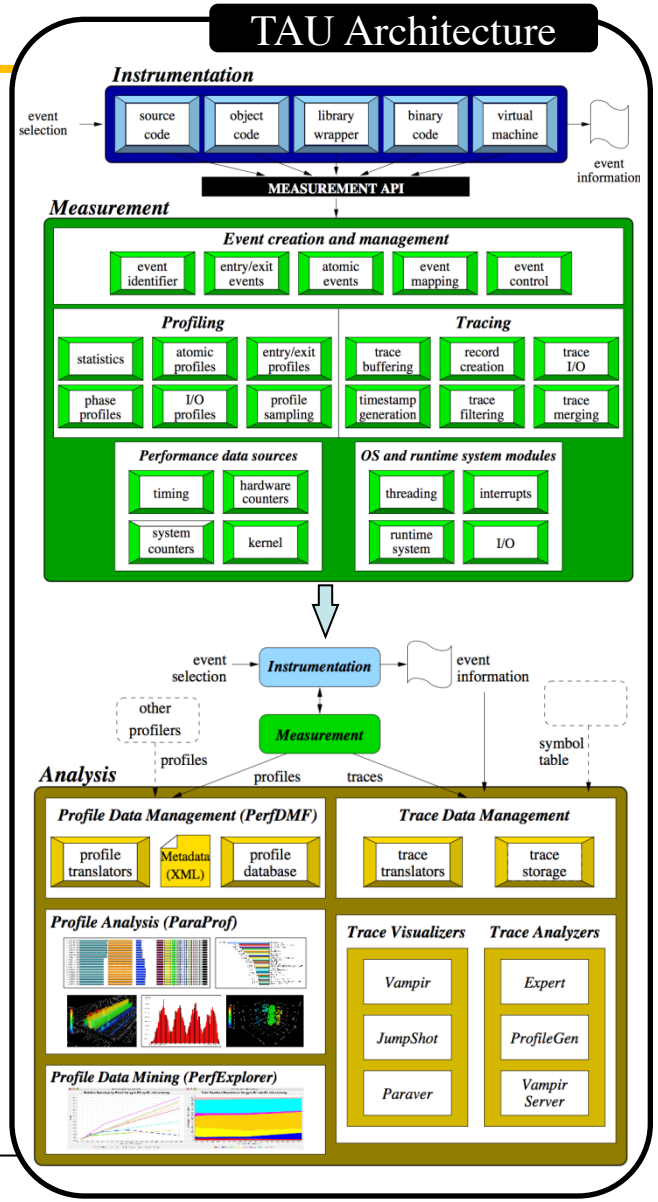
- Dr. Allen D. Malony, Professor, CIS Dept, and Director, NeuroInformatics Center, and CEO, ParaTools, Inc.
- Alan Morris, Senior software engineer
- Wyatt Spear, Software engineer
- Scott Biersdorff, Software engineer
- Dr. Robert Yelle, Research faculty
- Suzanne Millstein, Ph.D. student
- Ivan Pulleyn, Systems administrator

What is TAU?

- TAU is a performance evaluation tool
- It supports parallel profiling and tracing toolkit
- Profiling shows you how much (total) time was spent in each routine
- Tracing shows you *when* the events take place in each process along a timeline
- Profiling and tracing can measure time as well as hardware performance counters from your CPU
- TAU can automatically instrument your source code (routines, loops, I/O, memory, phases, etc.)
- It supports C++, C, Chapel, UPC, Fortran, Python and Java
- TAU runs on all HPC platforms and it is free (BSD style license)
- TAU has instrumentation, measurement and analysis tools
- To use TAU, you need to set a couple of environment variables and substitute the name of the compiler with a TAU shell script

TAU Performance System®

- Integrated toolkit for performance problem solving
  - Instrumentation, measurement, analysis, visualization
  - Portable performance profiling and tracing facility
  - Performance data management and data mining
- Based on direct performance measurement approach
- Open source
- Available on all HPC platforms
- http://tau.uoregon.edu

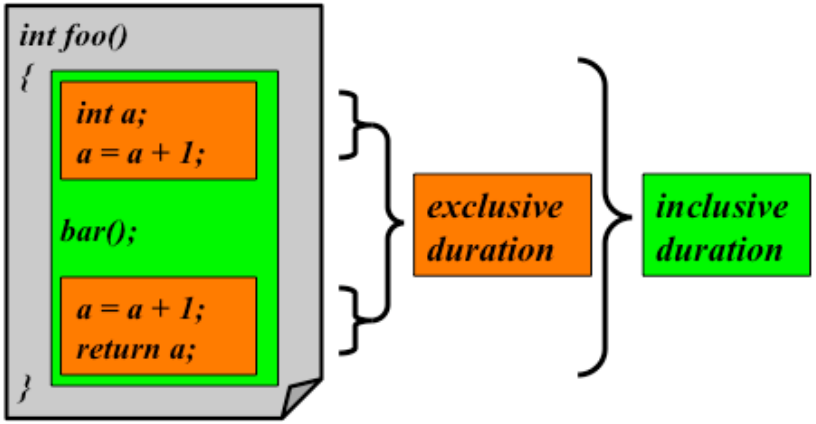


Performance Evaluation

- Profiling
  - Presents summary statistics of performance metrics
    - number of times a routine was invoked
    - exclusive, inclusive time/hpm counts spent executing it
    - number of instrumented child routines invoked, etc.
    - structure of invocations (calltrees/callgraphs)
    - memory, message communication sizes also tracked
- Tracing
  - Presents when and where events took place along a global timeline
    - timestamped log of events
    - message communication events (sends/receives) are tracked
      - shows when and where messages were sent
    - large volume of performance data generated leads to more perturbation in the program

TAU Performance Profiling

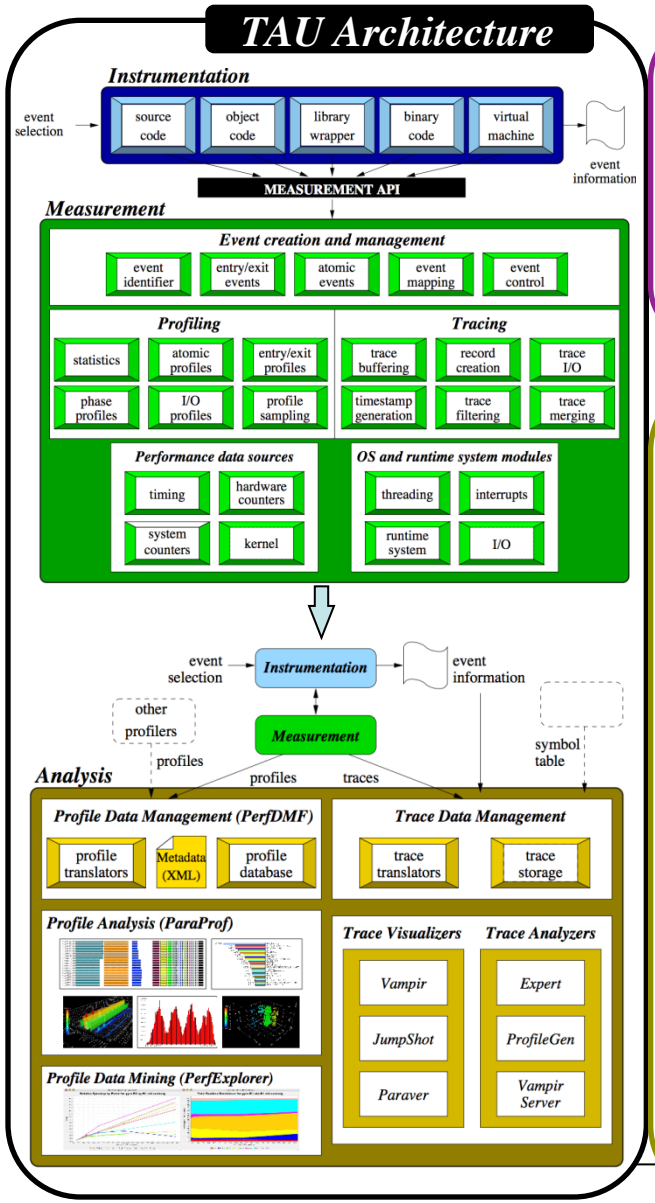
- Performance with respect to nested event regions
  - Program execution event stack (begin/end events)
- Profiling measures inclusive and exclusive data
- Exclusive measurements for region only performance
- Inclusive measurements includes nested “child” regions
- Support multiple profiling types
  - Flat, callpath, and phase profiling

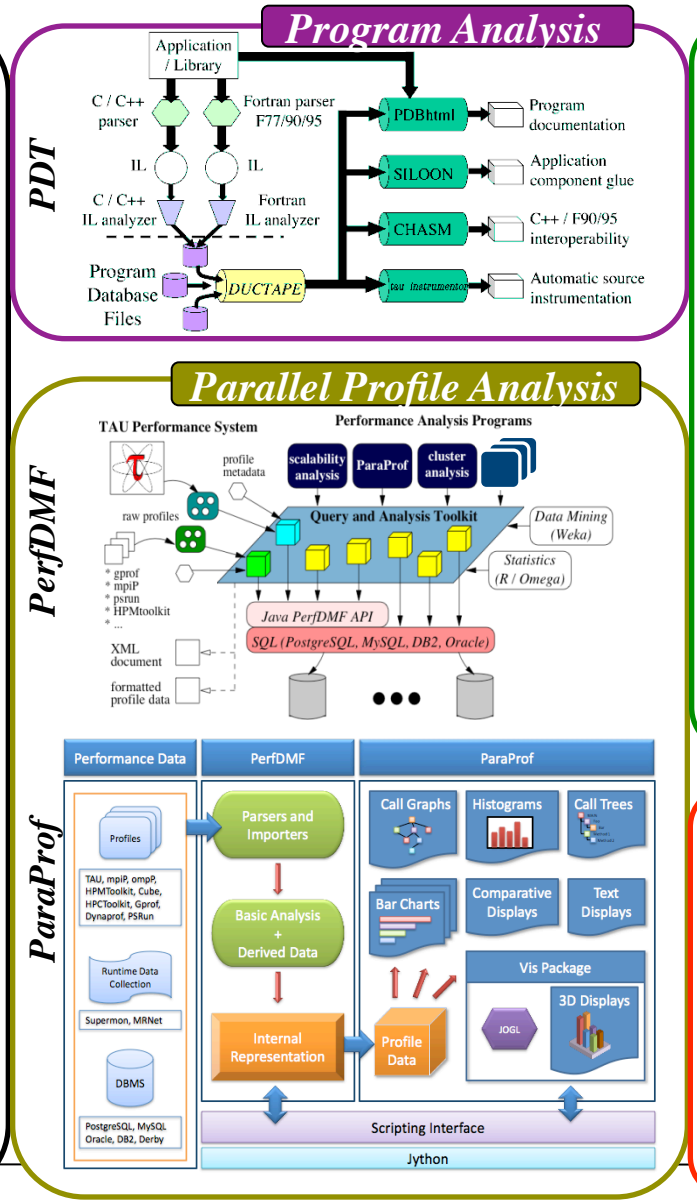


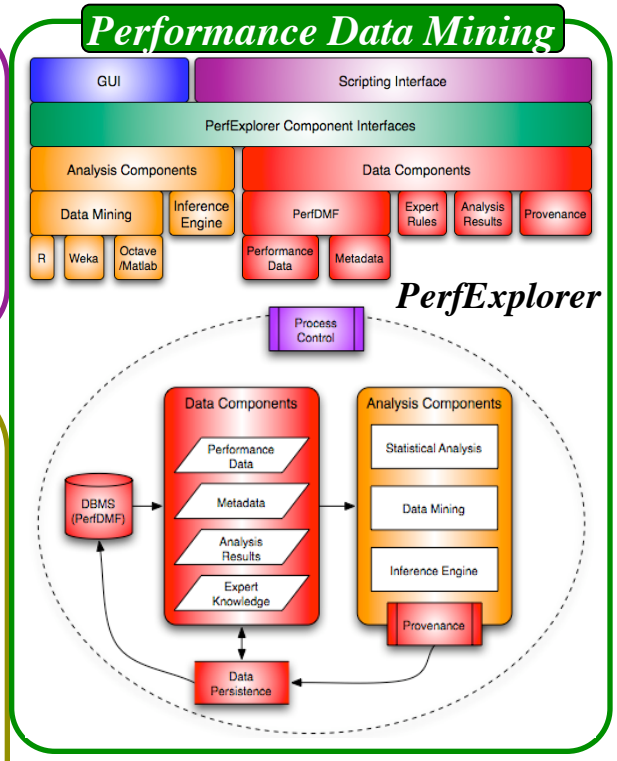
TAU Parallel Performance System Goals

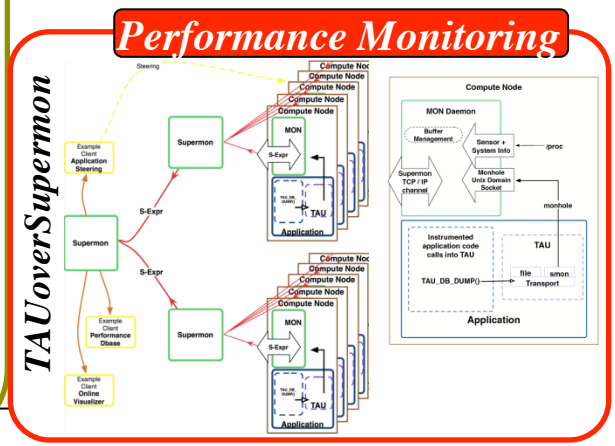
- **Portable (open source) parallel performance system**
  - Computer system architectures and operating systems
  - Different programming languages and compilers
- Multi-level, multi-language performance instrumentation
- **Flexible and configurable performance measurement**
- Support for multiple parallel programming paradigms
  - Multi-threading, message passing, mixed-mode, hybrid, object oriented (generic), component-based
- Support for performance mapping
- Integration of leading performance technology
- **Scalable (very large) parallel performance analysis**

TAU Performance System Components

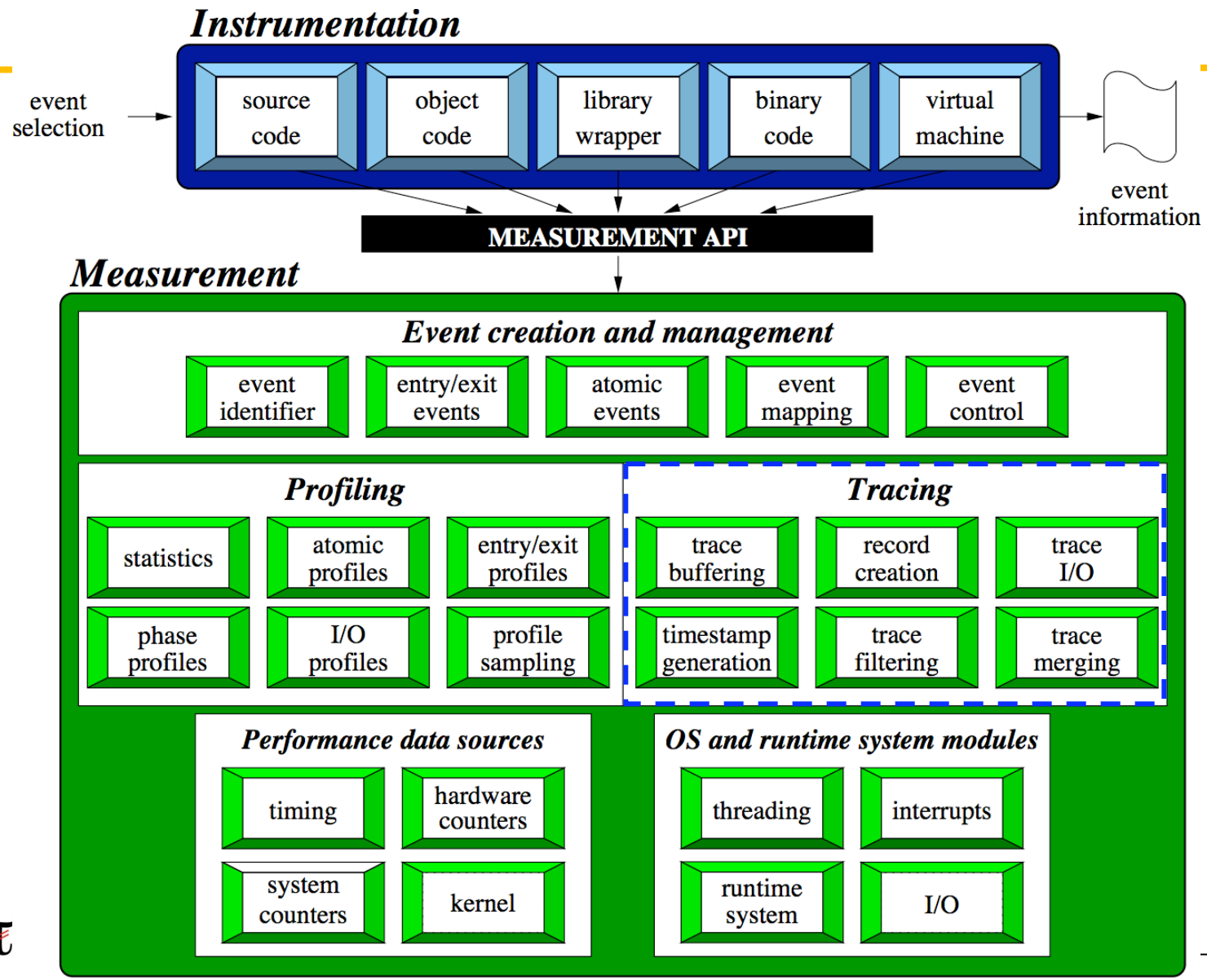






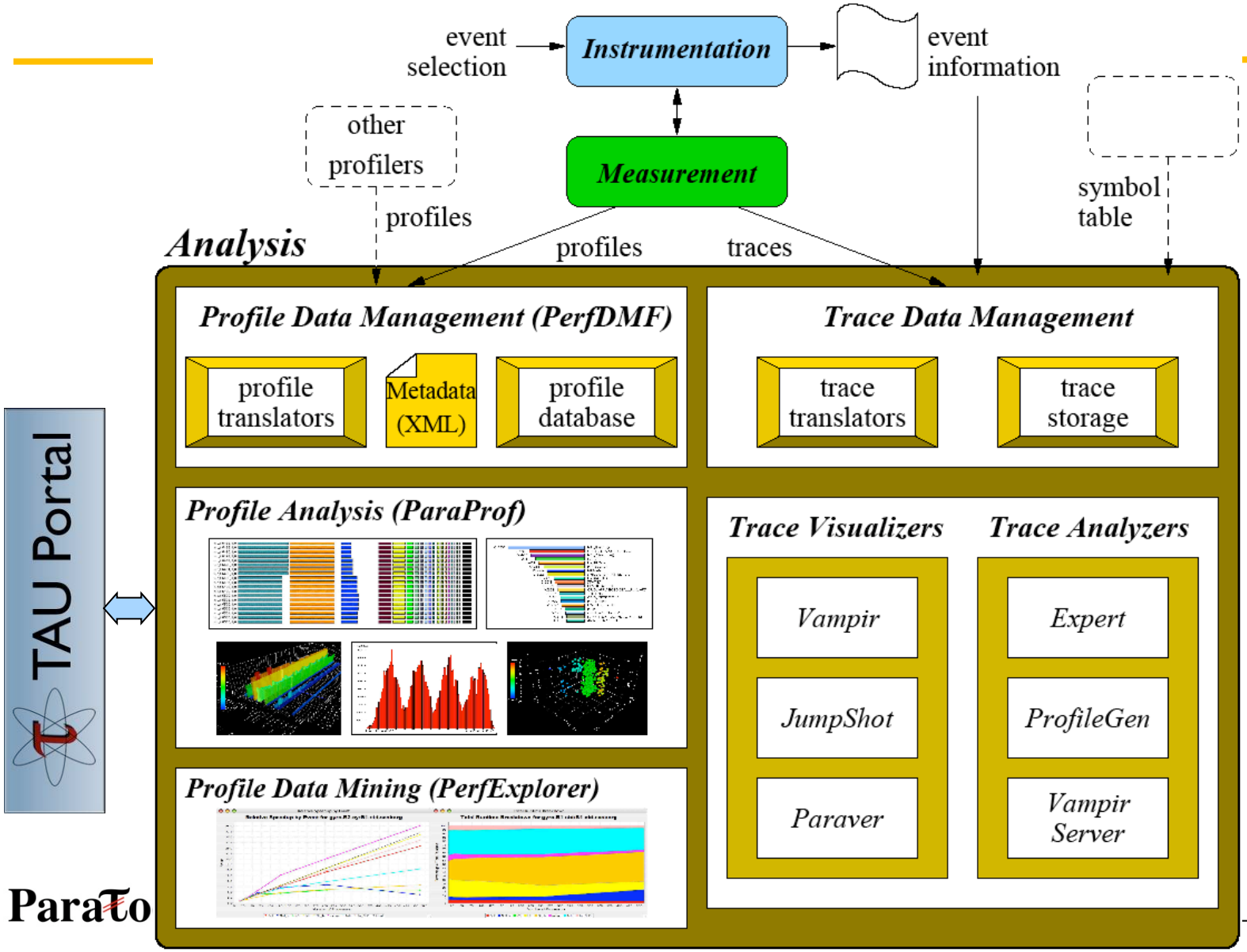


TAU Performance System Architecture

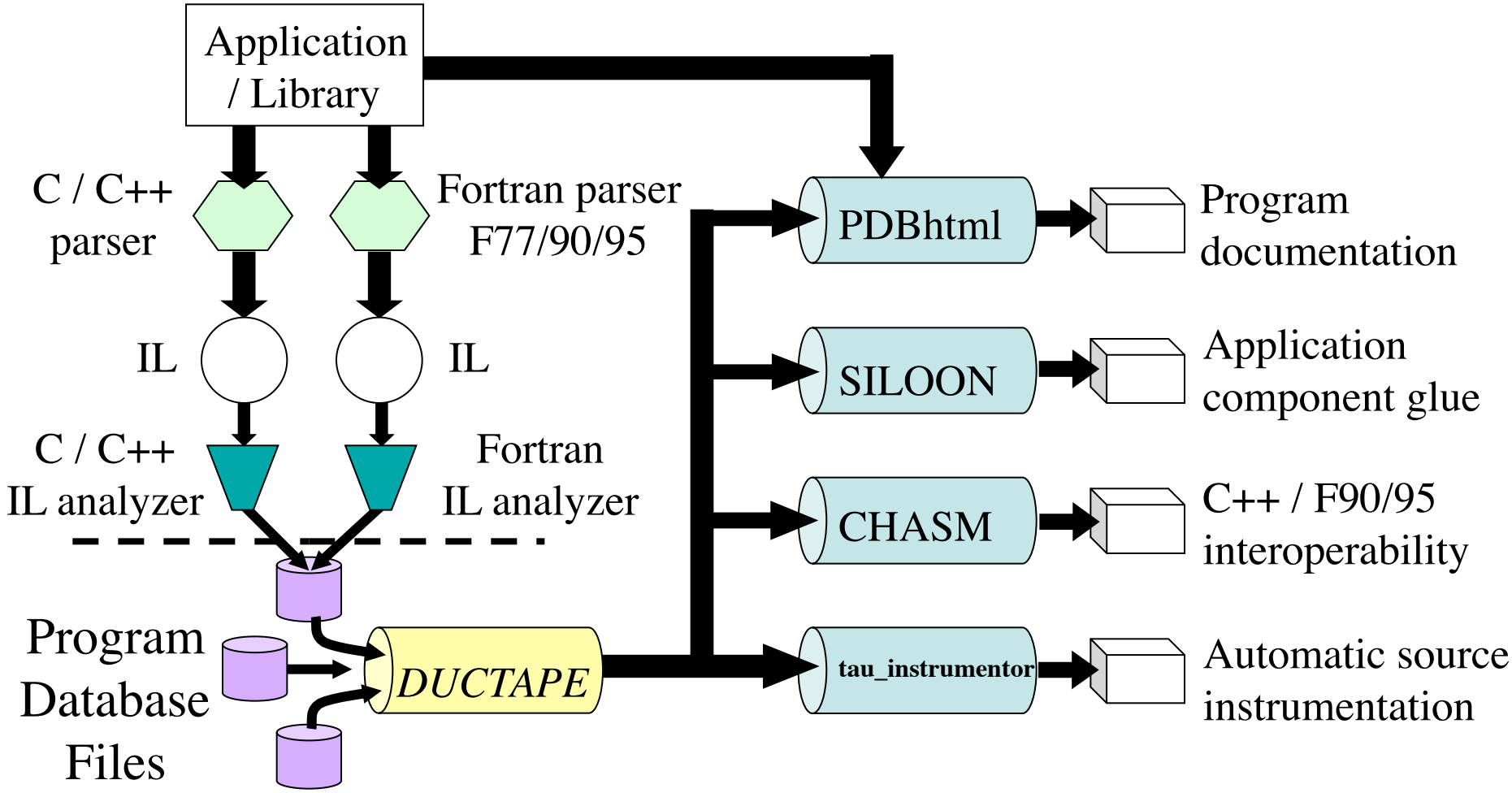


ParaT

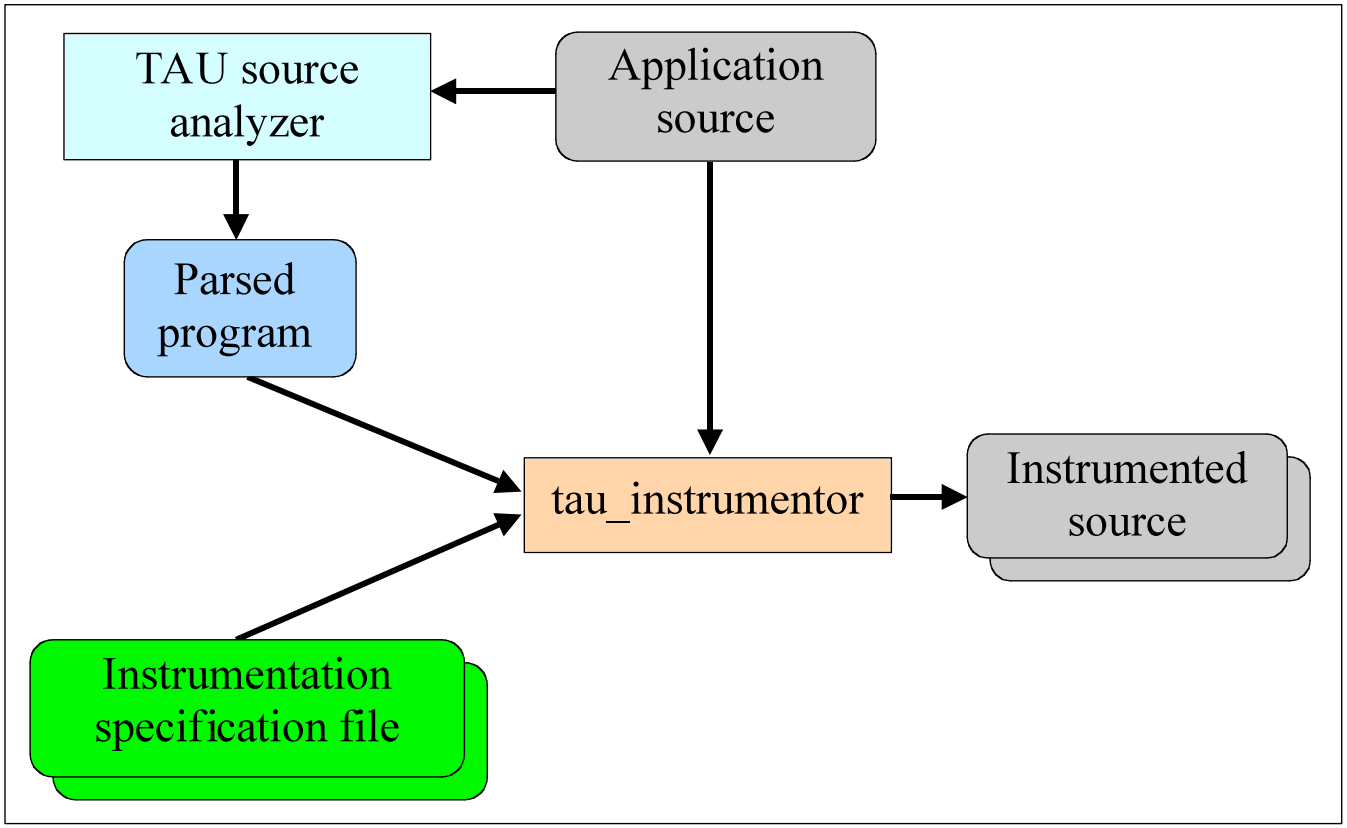
TAU Performance System Architecture



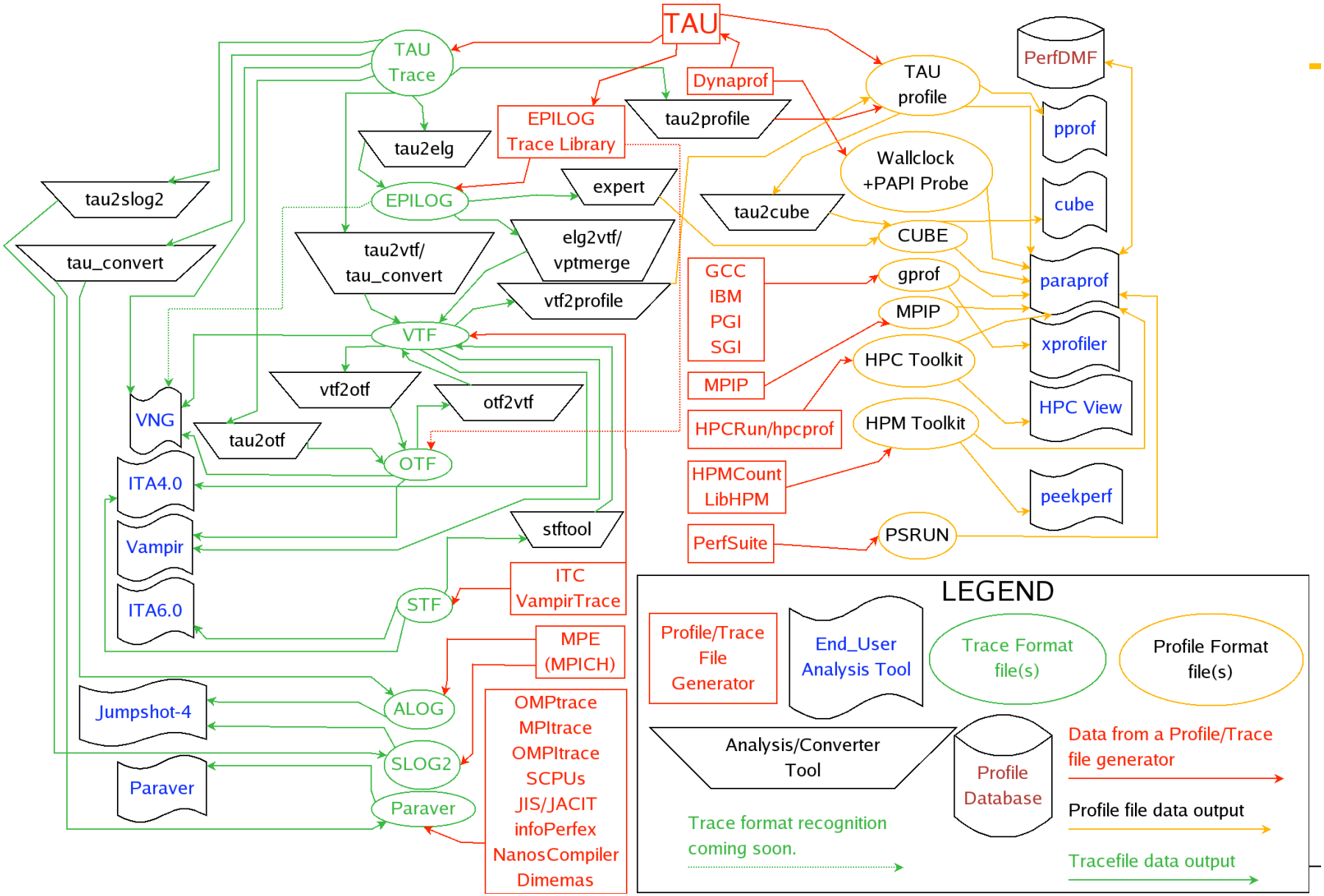
Program Database Toolkit (PDT)



Automatic Source-Level Instrumentation in TAU



Building Bridges to Other Tools



TAU Instrumentation Approach

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

Interval, Atomic and Context Events in TAU

```
File  Edit  View  Terminal  Tabs  Help
Reading Profile files in profile.*

NODE 0;CONTEXT 0;THREAD 0:
---------------------------------------------------------------------------------------
%Time    Exclusive    Inclusive       #Call      #Subrs  Inclusive Name
              msec   total msec                          usec/call
---------------------------------------------------------------------------------------
100.0        0.007        0.256           1           5        256 MAIN
 97.3        0.132        0.249           5           5         50 FOO
 40.6        0.104        0.104           5           0         21 BAR
 36.3        0.013        0.093           3           3         31 G
---------------------------------------------------------------------------------------

USER EVENTS Profile :NODE 0, CONTEXT 0, THREAD 0
---------------------------------------------------------------------------------------
NumSamples   MaxValue   MinValue  MeanValue  Std. Dev.  Event Name
---------------------------------------------------------------------------------------
         1         16         16         16          0  MEMORY LEAK! malloc size <file=foo.f90, variable=X, line=7> : MAIN => FOO => BAR
         2         52         48         50          2  MEMORY LEAK! malloc size <file=foo.f90, variable=X, line=7> : MAIN => FOO => G => BAR
         1         80         80         80          0  free size <file=foo.f90, variable=X, line=10>
         1         80         80         80          0  free size <file=foo.f90, variable=X, line=10> : MAIN => FOO => G => BAR
         1        180        180        180          0  free size <file=foo.f90, variable=X, line=15>
         1        180        180        180          0  free size <file=foo.f90, variable=X, line=15> : MAIN => FOO => BAR
         1        180        180        180          0  malloc size <file=foo.f90, variable=X, line=13>
         1        180        180        180          0  malloc size <file=foo.f90, variable=X, line=13> : MAIN => FOO => BAR
         4         80         16         49      22.69  malloc size <file=foo.f90, variable=X, line=7>
         1         16         16         16          0  malloc size <file=foo.f90, variable=X, line=7> : MAIN => FOO => BAR
         3         80         48         60      14.24  malloc size <file=foo.f90, variable=X, line=7> : MAIN => FOO => G => BAR
---------------------------------------------------------------------------------------
                                                                                    1,1           All
```

Interval Event

Context Event

Atomic Event

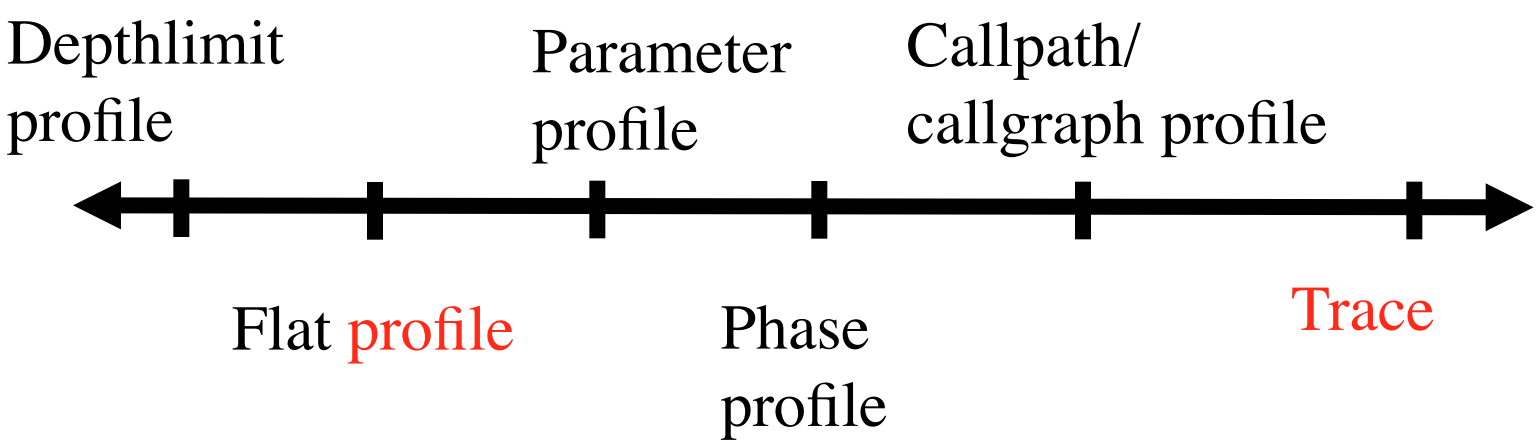
TAU Measurement Mechanisms

- **Parallel profiling**
  - Function-level, block-level, statement-level
  - Supports user-defined events and mapping events
  - Support for flat, callgraph/callpath, phase profiling
  - Support for memory profiling (headroom, malloc/leaks)
  - Support for tracking I/O (wrappers, read/write/print calls)
  - Parallel profiles written at end of execution
  - Parallel profile snapshots can be taken during execution
- **Tracing**
  - All profile-level events + inter-process communication
  - Inclusion of multiple counter data in traced events

Types of Parallel Performance Profiling

- ***Flat*** profiles
  - Metric (e.g., time) spent in an event (callgraph nodes)
  - Exclusive/inclusive, # of calls, child calls
- ***Callpath*** profiles (***Calldepth*** profiles)
  - Time spent along a calling path (edges in callgraph)
  - "*main=> f1 => f2 => MPI_Send*" (event name)
  - `TAU_CALLPATH_DEPTH` environment variable
- ***Phase*** profiles
  - Flat profiles under a phase (nested phases are allowed)
  - Default "main" phase
  - Supports static or dynamic (e.g., per-iteration) phases

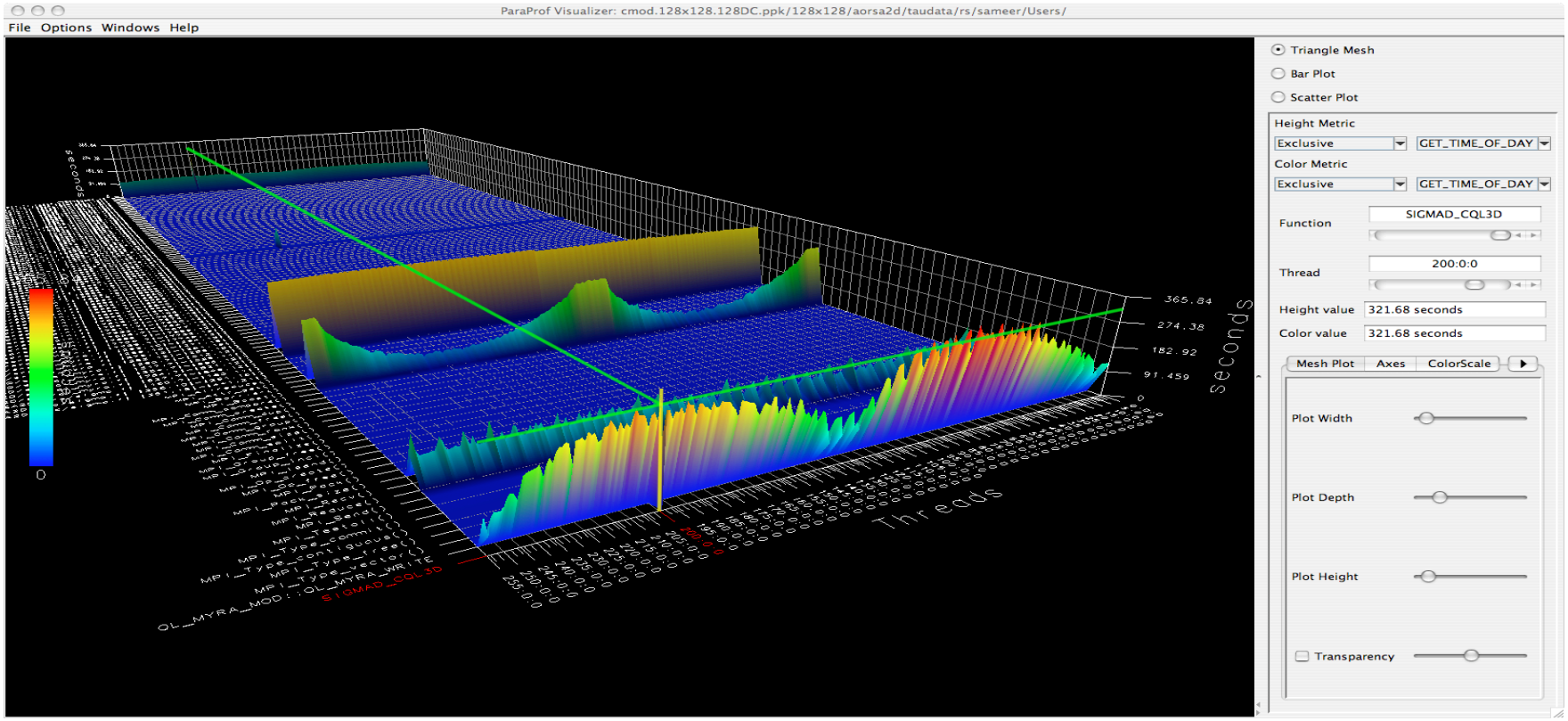
Performance Evaluation Alternatives



Each alternative has:
- one metric/counter
- multiple counters

Volume of performance data

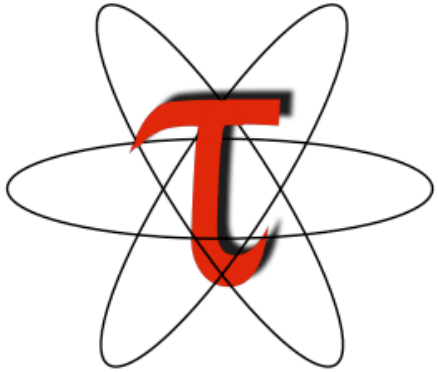
Parallel Profile Visualization: ParaProf



More Information

- PAPI References:
  - PAPI documentation page available from the PAPI website: http://icl.cs.utk.edu/papi/
- TAU References:
  - TAU Users Guide and papers available from the TAU website: http://tau.uoregon.edu/
- VAMPIR References
  - VAMPIR-NG website http://www.vampir-ng.de/
- Scalasca/KOJAK References
  - Scalasca documentation page http://www.scalasca.org/
- Eclipse PTP References
  - Documentation available from the Eclipse PTP website: http://www.eclipse.org/ptp/

TAU: A Quick Reference

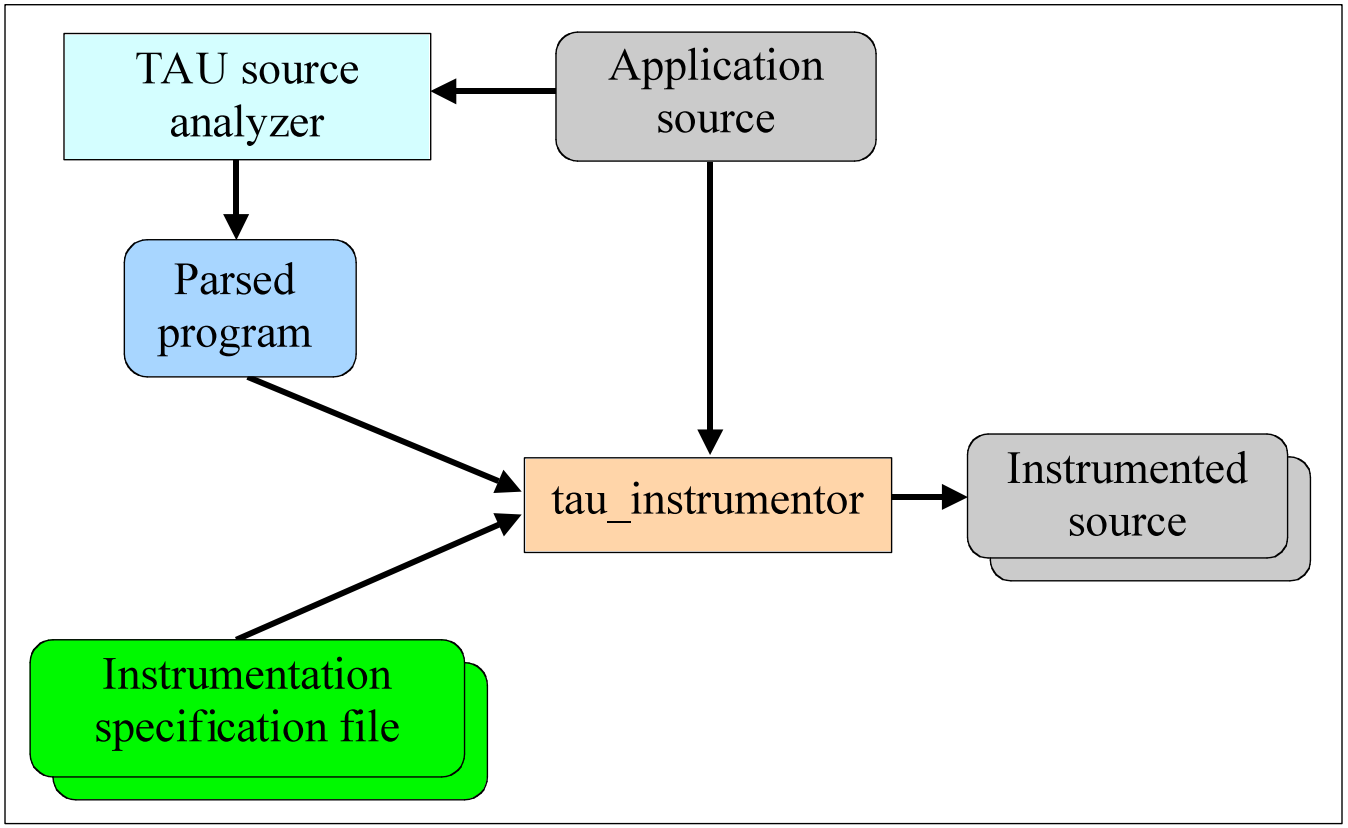


TAU Performance System

Tuning and Analysis Utilities

- http://tau.uoregon.edu/
- 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
- Integration in complex software, systems, applications

Automatic Source-Level Instrumentation in TAU using Program Database Toolkit (PDT)



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

Using TAU: A brief Introduction

- TAU supports several measurement options (profiling, tracing, profiling with hardware counters, etc.)
- Each measurement configuration of TAU corresponds to a unique stub makefile that is generated when you configure it
- To instrument source code using PDT
  - Choose an appropriate TAU stub makefile in <arch>/lib:

```
% export TAU_MAKEFILE=/soft/apps/tau/tau_latest/bgp/lib/Makefile.tau-mpi-pdt
% export TAU_OPTIONS='-optVerbose …' (see tau_compiler.sh -help)
```

  And use tau_f90.sh, tau_cxx.sh or tau_cc.sh as Fortran, C++ or C compilers:

```
% mpixlf90_r foo.f90
```

  changes to

```
% tau_f90.sh foo.f90
```

- Execute application and analyze performance data:

```
% pprof   (for text based profile display)
% paraprof  (for GUI)
```

TAU Measurement Configuration

```
% cd /soft/apps/tau/tau_latest/bgp/lib; ls Makefile.*
Makefile.tau-pdt
Makefile.tau-mpi-pdt
Makefile.tau-mpi-bgptimers-pdt
Makefile.tau-opari-openmp-mpi-pdt
Makefile.tau-mpi-scalasca-epilog-pdt
Makefile.tau-mpi-vampirtrace-pdt
Makefile.tau-mpi-papi-pdt
Makefile.tau-papi-mpi-openmp-opari-pdt
Makefile.tau-pthread-pdt…
```

- For an MPI+F90 application, you may want to start with:

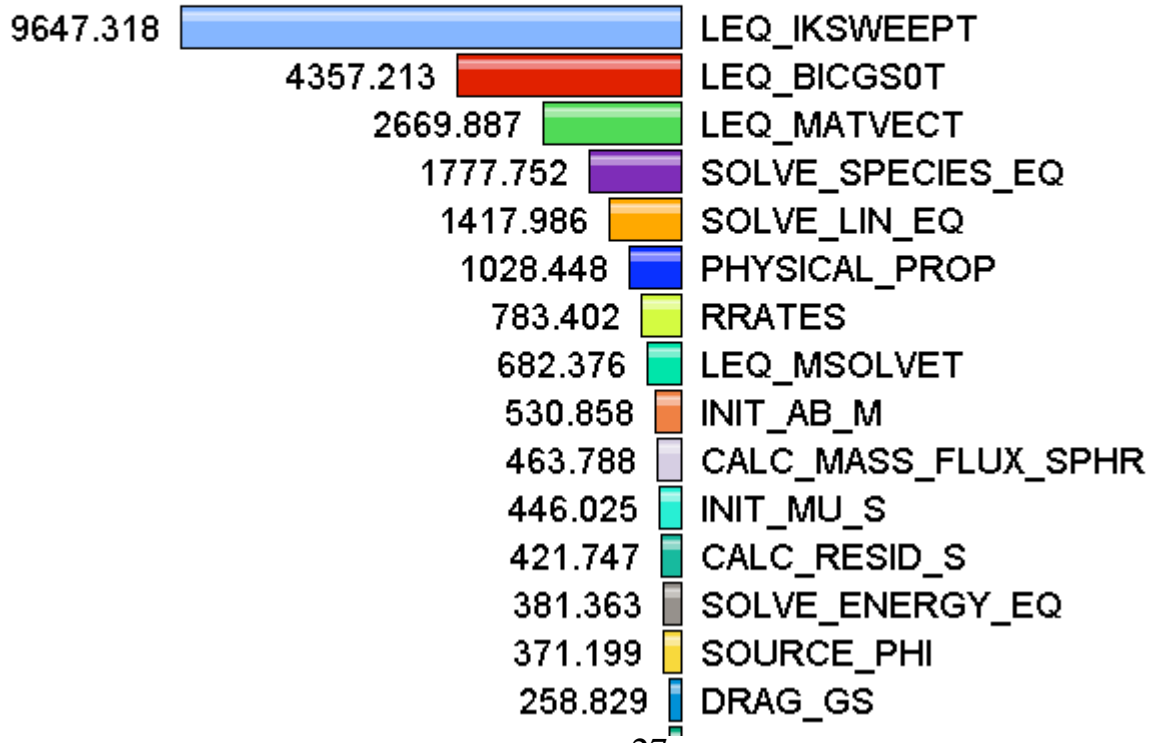
Makefile.tau-mpi-pdt

- Supports MPI instrumentation & PDT for automatic source instrumentation
- `% export TAU_MAKEFILE= /soft/apps/tau/tau_latest/bgp/lib/Makefile.tau-mpi-bgptimers-pdt`
- `% tau_f90.sh matrix.f90 -o matrix`

Usage Scenarios: Routine Level Profile

- Goal: What routines account for the most time? How much?
- Flat profile with wallclock time:

Metric: P_VIRTUAL_TIME
Value: Exclusive
Units: seconds



Solution: Generating a flat profile with MPI

```
% export TAU_MAKEFILE=/soft/apps/tau/tau_latest/bgp
                                        /lib/Makefile.tau-mpi-pdt
% export PATH=/soft/apps/tau/tau_latest/ppc64/bin:$PATH
OR
% source /soft/apps/tau/tau.bashrc  [ or tau.bashrc]
% tau_f90.sh matmult.f90 -o matmult
(Or edit Makefile and change F90=tau_f90.sh)

% mpirun -np 4 ./matmult
% paraprof --pack app.ppk
  Move the app.ppk file to your desktop.

% paraprof app.ppk
```

Usage Scenarios: Loop Level Instrumentation

- Goal: What loops account for the most time? How much?
- Flat profile with wallclock time with loop instrumentation:

Metric: GET_TIME_OF_DAY
Value: Exclusive
Units: microseconds

1729975.333	Loop: MULTIPLY_MATRICES [{matmult.f90} {31,9}-{36,14}]
443194	MPI_Recv()
81095	MAIN
49569	MPI_Bcast()
45669	Loop: MAIN [{matmult.f90} {86,9}-{106,14}]
12412	MPI_Send()
8959	Loop: INITIALIZE [{matmult.f90} {17,9}-{21,14}]
8953	Loop: INITIALIZE [{matmult.f90} {10,9}-{14,14}]
5609.2	MPI_Finalize()
2932.667	MULTIPLY_MATRICES
2577.667	Loop: MAIN [{matmult.f90} {117,9}-{128,14}]
2091.8	MPI_Barrier()
1875.667	Loop: MAIN [{matmult.f90} {112,9}-{115,14}]
1833	Loop: MAIN [{matmult.f90} {71,9}-{74,14}]
107	Loop: MAIN [{matmult.f90} {77,9}-{84,14}]
30	INITIALIZE
14.25	MPI_Comm_rank()
1	MPI_Comm_size()

ParaProf: Mflops Sorted by Exclusive Time

TAU: ParaProf: Mean Data – s3d.loops.12000.ppk

Metric: PAPI_FP_INS / LINUX_TIMERS
Value: Exclusive
Units: Derived metric shown in microseconds format

1142.273	Loop: CHEMKIN_M::REACTION_RATE_VEC [{chemkin_m.pp.f90} {457,3}-{471,8}]
884.23	Loop: TRANSPORT_M::COMPUTECOEFFICIENTS [{mixavg_transport_m.pp.f90} {492,5}-{520,9}]
0.014	MPI_Wait()
121.843	Loop: TRANSPORT_M::COMPUTESPECIESDIFFFLUX [{mixavg_transport_m.pp.f90} {630,5}-{656,19}]
0.021	MPI_Isend()
0.06	Loop: RHSF [{rhsf.pp.f90} {209,3}-{211,7}]
183.176	Loop: INTEGRATE [{integrate_erk.pp.f90} {73,3}-{93,13}]
0	MPI_Comm_split()
0	MPI_Barrier()
148.608	RHSF
269.897	Loop: TRANSPORT_M::COMPUTEHEATFLUX [{mixavg_transport_m.pp.f90} {782,5}-{790,19}]
742.83	Loop: THERMCHEM_M::CALC_TEMP [{thermchem_m.pp.f90} {175,5}-{216,9}]
163.317	Loop: RHSF [{rhsf.pp.f90} {537,3}-{543,16}]
174.081	Loop: RHSF [{rhsf.pp.f90} {545,3}-{551,16}]
124.129	Loop: RHSF [{rhsf.pp.f90} {515,3}-{535,16}]
1153.827	Loop: DERIVATIVE_X_CALC [{derivative_x.pp.f90} {432,10}-{441,15}]
2.177	DERIVATIVE_X_COMM
1422.272	Loop: DERIVATIVE_Y_CALC [{derivative_y.pp.f90} {431,10}-{440,15}]
1459.472	Loop: DERIVATIVE_Z_CALC [{derivative_z.pp.f90} {435,10}-{444,15}]
178.375	Loop: VARIABLES_M::GET_MASS_FRAC [{variables_m.pp.f90} {96,3}-{99,7}]
374.708	Loop: THERMCHEM_M::CALC_INV_AVG_MOL_WT [{thermchem_m.pp.f90} {127,5}-{129,9}]
0	MPI_Irecv()
3.778	DERIVATIVE_Y_COMM
244.308	Loop: THERMCHEM_M::CALC_SPECENTH_ALLPTS [{thermchem_m.pp.f90} {506,3}-{512,8}]
1.294	INTEGRATE
0.002	S3D
7.111	DERIVATIVE_Z_COMM
733.265	Loop: DERIVATIVE_X_CALC [{derivative_x.pp.f90} {490,13}-{509,18}]
765.947	Loop: DERIVATIVE_X_CALC [{derivative_x.pp.f90} {461,13}-{480,18}]
950.372	Loop: DERIVATIVE_Y_CALC [{derivative_y.pp.f90} {488,13}-{507,18}]
973.95	Loop: DERIVATIVE_Y_CALC [{derivative_y.pp.f90} {459,13}-{478,18}]
249.638	Loop: TRANSPORT_M::COMPUTESTRESSTENSOR [{mixavg_transport_m.pp.f90} {711,5}-{737,10}]
1119.025	Loop: DERIVATIVE_Z_CALC [{derivative_z.pp.f90} {464,13}-{483,18}]
0.328	TRANSPORT_M::COMPUTESPECIESDIFFFLUX
1257.338	Loop: DERIVATIVE_Z_CALC [{derivative_z.pp.f90} {493,13}-{512,18}]
84.649	RK_M::CONTROLLER
30.583	DERIVATIVE_X_CALC
33.374	DERIVATIVE_Y_CALC
33.405	DERIVATIVE_Z_CALC
61.032	THERMCHEM_M::CALC_INV_AVG_MOL_WT
95.032	GET_VELOCITY_VEC
103.526	Loop: TRANSPORT_M::COMPUTECOEFFICIENTS [{mixavg_transport_m.pp.f90} {527,5}-{529,10}]
196.839	Loop: FILTER_M::FILTER [{filter_m.pp.f90} {1232,5}-{1236,9}]
293.594	THERMCHEM_M::CALC_TEMP
141.127	TRANSPORT_M::COMPUTECOEFFICIENTS
24.465	COMPUTESCALARGRADIENT
0	Loop: INTEGRATE [{integrate_erk.pp.f90} {108,3}-{110,7}]
0.245	VARIABLES_M::GET_MASS_FRAC

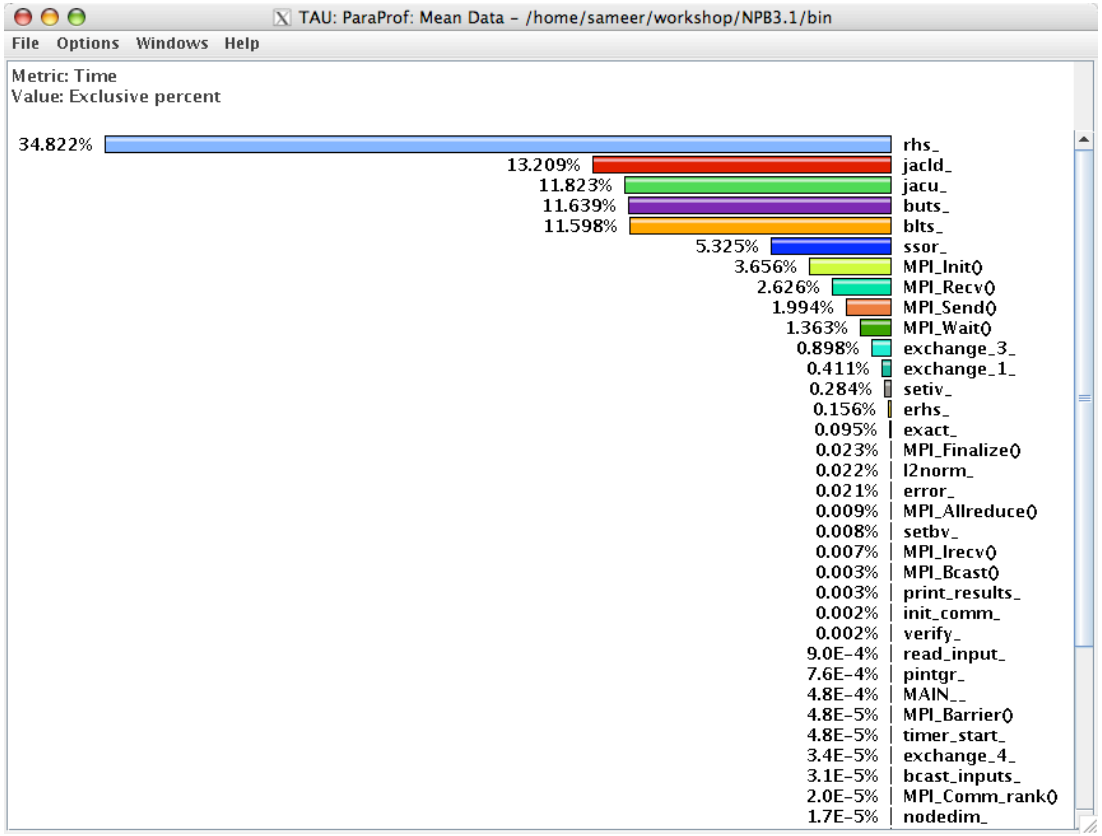
low mflops?

Solution: Generating a loop level profile

```
% export TAU_MAKEFILE=/soft/apps/tau/tau_latest/bgp
                                        /lib/Makefile.tau-mpi-pdt
% export TAU_OPTIONS='-optTauSelectFile=select.tau -optVerbose'
% cat select.tau
  BEGIN_INSTRUMENT_SECTION
  loops routine="#"
  END_INSTRUMENT_SECTION

% export PATH=/soft/apps/tau/tau_latest/ppc64/bin:$PATH
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)
% mpirun -np 4 ./a.out
% paraprof --pack app.ppk
  Move the app.ppk file to your desktop.

% paraprof app.ppk
```

Usage Scenarios: Compiler-based Instrumentation

- Goal: Easily generate routine level performance data using the compiler instead of PDT for parsing the source code



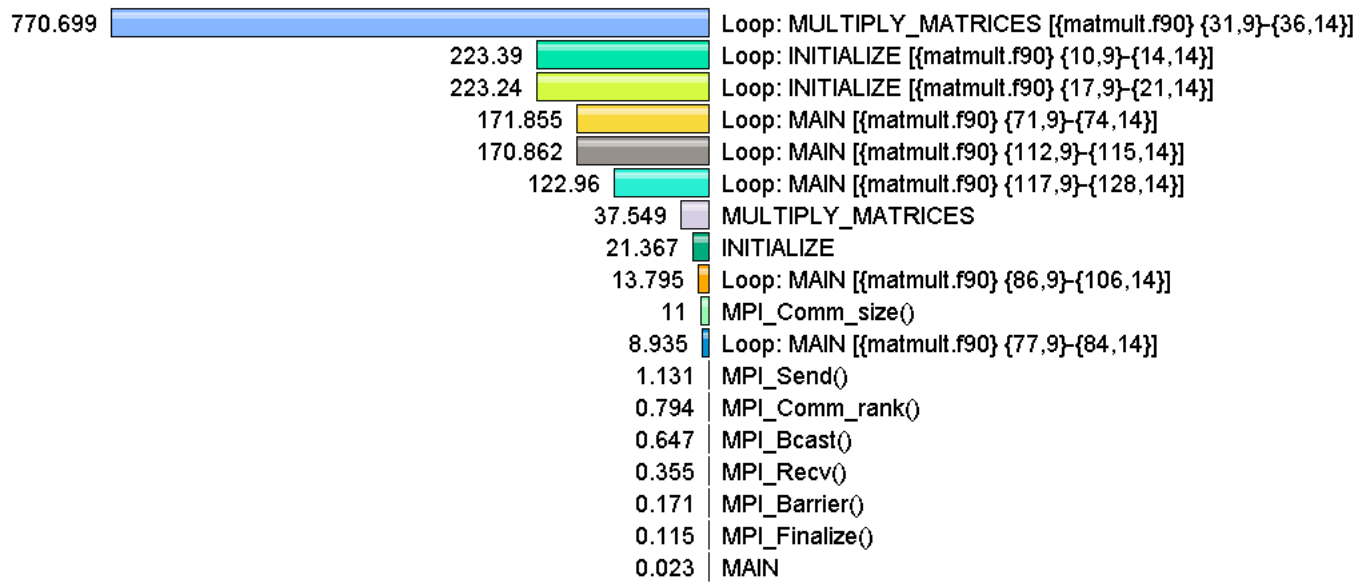
Use Compiler-Based Instrumentation

```
% export TAU_MAKEFILE=/soft/apps/tau/tau_latest/bgp
         /lib/Makefile.tau-mpi
% export TAU_OPTIONS='-optCompInst -optVerbose'
% export PATH=/soft/apps/tau/tau_latest/ppc64/bin:$PATH
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)

% qsub -A TAU09 -n 4 -t 10 -q R.TAU09 --mode vn ./a.out
% paraprof --pack app.ppk
  Move the app.ppk file to your desktop.
% paraprof app.ppk
```

Usage Scenarios: Calculate mflops in Loops

- Goal: What MFlops am I getting in all loops?
- Flat profile with PAPI_FP_INS/OPS and time with loop instrumentation:

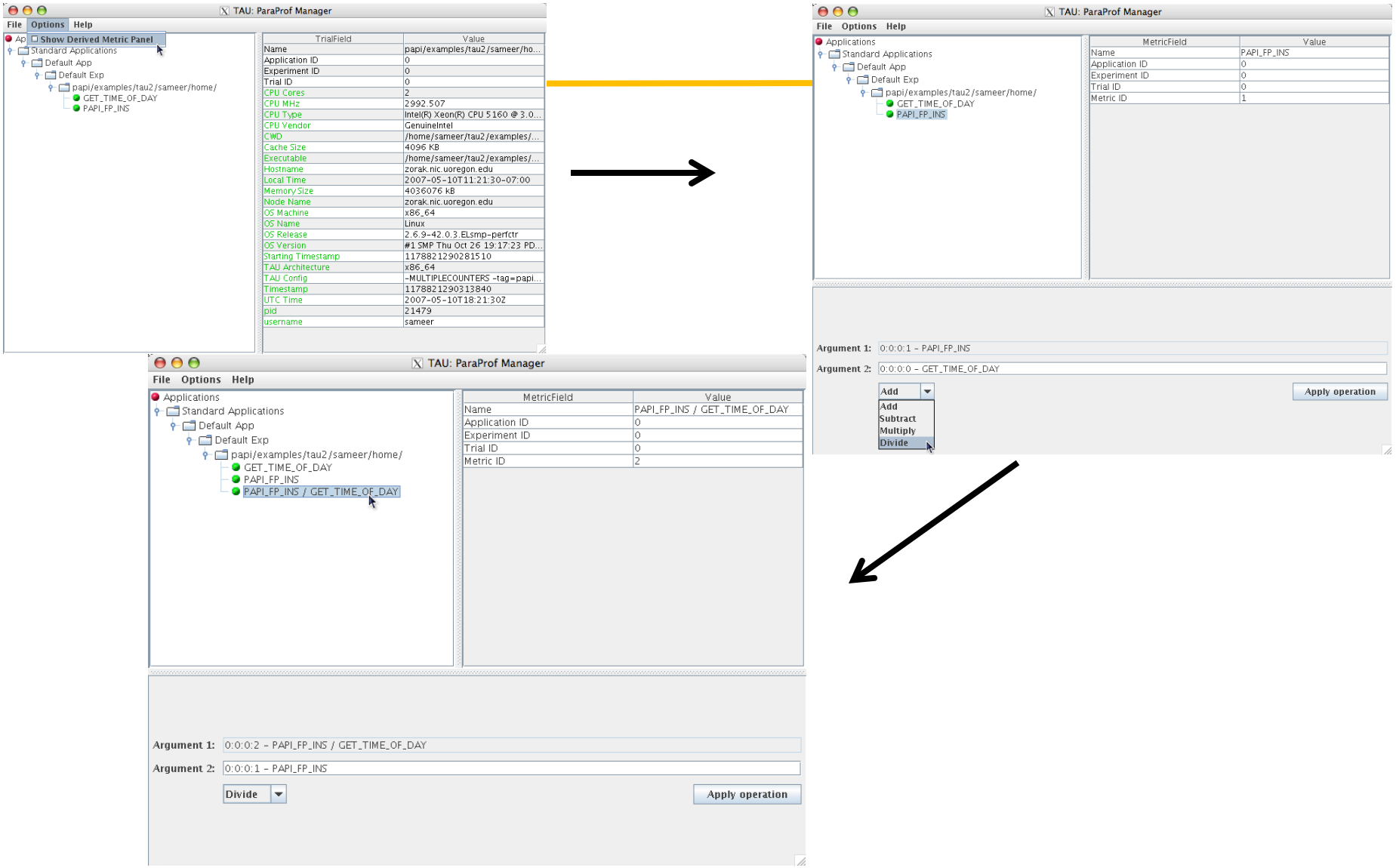
Metric: PAPI_FP_INS / GET_TIME_OF_DAY
Value: Exclusive
Units: Derived metric shown in microseconds format



Generate a PAPI profile with 2 or more counters

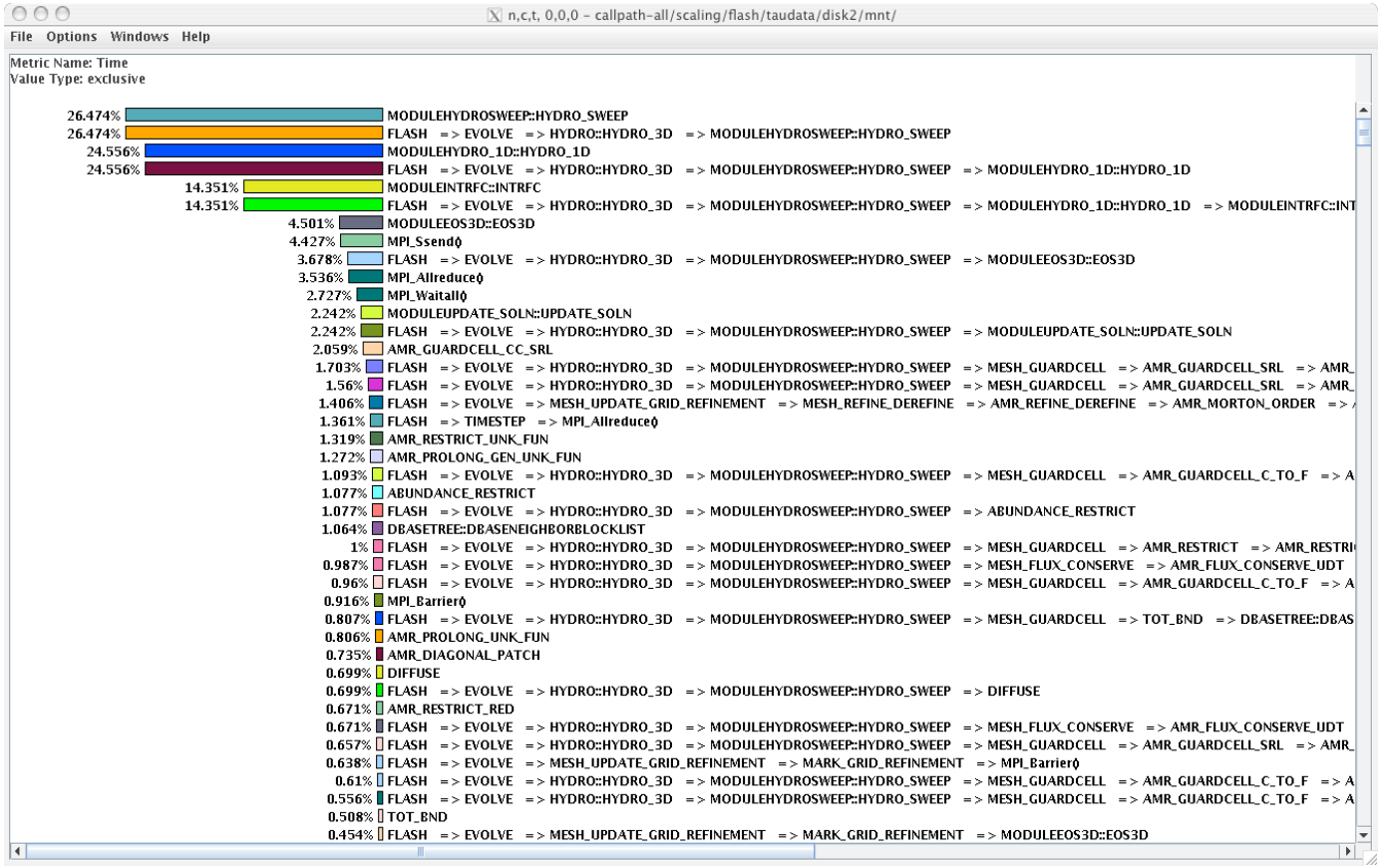
```
% export TAU_MAKEFILE=/soft/apps/tau/tau_latest/bgp
                              /lib/Makefile.tau-papi-mpi-pdt
% export TAU_OPTIONS='-optTauSelectFile=select.tau -optVerbose'
% cat select.tau
  BEGIN_INSTRUMENT_SECTION
  loops routine="#"
  END_INSTRUMENT_SECTION

% export PATH=/soft/apps/tau/tau_latest/ppc64/bin:$PATH
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)
% export COUNTER1=GET_TIME_OF_DAY
% export COUNTER2=PAPI_FP_INS
% qsub -A TAU09 -n 4 -t 10 -q R.TAU09 --mode vn ./a.out  ./a.out
% paraprof --pack app.ppk
   Move the app.ppk file to your desktop.
% paraprof app.ppk
  Choose Options -> Show Derived Panel -> Arg 1 = PAPI_FP_INS,
     Arg 2 = GET_TIME_OF_DAY, Operation = Divide -> Apply, choose.
```

Derived Metrics in ParaProf



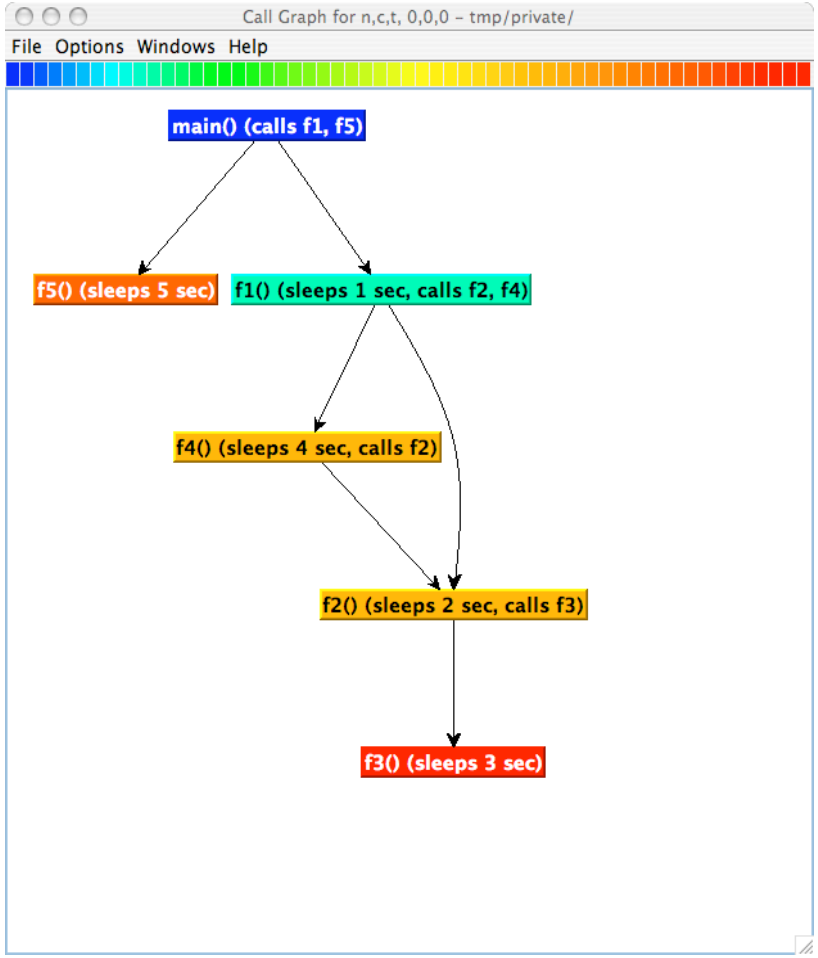
Usage Scenarios: Generating Callpath Profile

- Goal: Who calls my MPI_Barrier()? Where?
- Callpath profile for a given callpath depth:



Callpath Profile

- Generates program callgraph



Generate a Callpath Profile

```
% export TAU_MAKEFILE=/soft/apps/tau/tau_latest/bgp
                              /lib/Makefile.tau-callpath-mpi-pdt
% export PATH=/soft/apps/tau/tau_latest/ppc64/bin:$PATH
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)
% export TAU_CALLPATH_DEPTH=100

% qsub -A TAU09 -n 4 -t 10 -q R.TAU09 --mode vn ./a.out  ./a.out
% paraprof --pack app.ppk
   Move the app.ppk file to your desktop.
% paraprof app.ppk
(Windows -> Thread -> Call Graph)

NOTE: In TAU v2.18.1+, you may choose to just set:
% export TAU_CALLPATH=1
instead of recompiling your code with the above stub makefile.
Any TAU instrumented executable can generate callpath profiles.
```

Usage Scenario: Detect Memory Leaks

TAU: ParaProf: Mean Context Events - mem.ppk

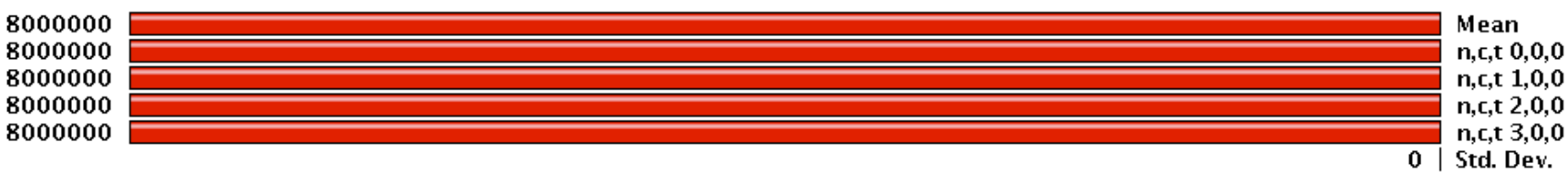
File Windows Help

Name	NumSamples	MaxValue	MinValue	MeanValue	Std. Dev.
MAIN [{matrix.f90} {141,7}-{146,22}]					
MATRICES::ALLOCATE_MATRICES [{matrix.f90} {10,7}-{13,38}]					
MEMORY LEAK! malloc size <file=matrix.f90, variable=C, line=11>	1	8,000,000	8,000,000	8,000,000	0
malloc size <file=matrix.f90, variable=A, line=11>	1	8,000,000	8,000,000	8,000,000	0
malloc size <file=matrix.f90, variable=B, line=11>	1	8,000,000	8,000,000	8,000,000	0
malloc size <file=matrix.f90, variable=C, line=11>	1	8,000,000	8,000,000	8,000,000	0
MATRICES::DEALLOCATE_MATRICES [{matrix.f90} {14,7}-{17,40}]					
free size <file=matrix.f90, variable=A, line=15>	1	8,000,000	8,000,000	8,000,000	0
free size <file=matrix.f90, variable=B , line=15>	1	8,000,000	8,000,000	8,000,000	0

User Event Window: mem.ppk

File Options Windows Help

Name: MEMORY LEAK! malloc size <file=matrix.f90, variable=C, line=11> : MAIN [{matrix.f90} {141,7}-{146,22}] => MATRICES::ALLOCATE_MATRICES [{matrix.f90} {10,7}-{13,38}]
Value Type: Max Value



Detect Memory Leaks

```
% export TAU_MAKEFILE=/soft/apps/tau/tau_latest/bgp
                          /lib/Makefile.tau-mpi-pdt
% export TAU_OPTIONS='-optDetectMemoryLeaks -optVerbose'
% export PATH=/soft/apps/tau/tau_latest/ppc64/bin:$PATH
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)
% export TAU_CALLPATH_DEPTH=100

% qsub -A TAU09 -n 4 -t 10 -q R.TAU09 --mode vn ./a.out  ./a.out
% paraprof --pack app.ppk
  Move the app.ppk file to your desktop.
% paraprof app.ppk
(Windows -> Thread -> Context Event Window -> Select thread -> select...
   expand tree)
(Windows -> Thread -> User Event Bar Chart -> right click LEAK
-> Show User Event Bar Chart)
```

Usage Scenarios: Instrument a Python program

- Goal: Generate a flat profile for a Python program

TAU: ParaProf: n,c,t 0,0,0 - flat.ppk

Metric: Time
Value: Exclusive
Units: seconds

5.473	dot
2.548	standard_normal
0.152	MachAr::__init__ [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/site-packages/numpy/lib/machar.py} {50}]
0.112	_parse [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/sre_parse.py} {385}]
0.105	_compile [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/sre_compile.py} {38}]
0.056	any [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/site-packages/numpy/core/fromnumeric.py} {540}]
0.048	append
0.046	len
0.046	Tokenizer::__next [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/sre_parse.py} {188}]
0.045	any
0.035	<module> [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/site-packages/numpy/linalg/linalg.py} {7}]
0.033	SubPattern::getwidth [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/sre_parse.py} {146}]
0.033	_optimize_charset [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/sre_compile.py} {213}]
0.028	Tokenizer::get [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/sre_parse.py} {207}]
0.023	_parse_sub [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/sre_parse.py} {307}]
0.022	add_newdoc [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/site-packages/numpy/lib/function_base.py} {1154}]
0.019	abs
0.019	_compile_charset [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/sre_compile.py} {184}]
0.016	<module> [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/site-packages/numpy/core/__init__.py} {2}]
0.015	<module> [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/site-packages/numpy/core/numeric.py} {1}]
0.014	SubPattern::__len__ [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/sre_parse.py} {132}]
0.011	_mk_bitmap [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/sre_compile.py} {264}]
0.011	Tokenizer::match [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/sre_parse.py} {201}]
0.01	<module> [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/site-packages/numpy/core/defmatrix.py} {1}]
0.009	SubPattern::__getitem__ [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/sre_parse.py} {136}]
0.009	_simple [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/sre_compile.py} {360}]
0.008	SubPattern::append [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/sre_parse.py} {144}]
0.007	min
0.007	<module> [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/ctypes/__init__.py} {4}]
0.007	extend_all [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/site-packages/numpy/core/numeric.py} {83}]
0.006	ord
0.005	<module> [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/site-packages/numpy/__init__.py} {17}]
0.005	<module> [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/site-packages/numpy/core/ma.py} {9}]
0.005	<module> [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/site-packages/numpy/core/numerictypes.py} {76}]
0.005	SubPattern::__getslice__ [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/sre_parse.py} {140}]
0.004	<module> [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/site-packages/numpy/lib/index_tricks.py} {3}]
0.004	<module> [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/site-packages/numpy/lib/__init__.py} {1}]
0.004	<module> [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/site-packages/numpy/random/__init__.py} {2}]
0.004	<module> [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/opcode.py} {5}]
0.004	_extendLine [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/site-packages/numpy/core/arrayprint.py} {270}]
0.004	_formatArray [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/site-packages/numpy/core/arrayprint.py} {278}]

Usage Scenarios: Instrument a Python program

Original code:

```
% cat foo.py
#!/usr/bin/env python
import numpy
ra=numpy.random
la=numpy.linalg

size=2000
a=ra.standard_normal((size,size))
b=ra.standard_normal((size,size))
c=la.linalg.dot(a,b)
print c
```

Create a wrapper:

```
% cat wrapper.py
#!/usr/bin/env python

# setenv PYTHONPATH $PET_HOME/pkgs/tau-2.17.3/ppc64/lib/bindings-gnu-python-pdt

import tau

def OurMain():
    import foo

tau.run('OurMain()')
```

Generate a Python Profile

```
% export TAU_MAKEFILE=/soft/apps/tau/tau_latest/bgp
                        /lib/Makefile.tau-python-pdt
% export PATH=/soft/apps/tau/tau_latest/ppc64/bin:$PATH
% cat wrapper.py
  import tau
  def OurMain():
      import foo
  tau.run('OurMain()')
Uninstrumented:
% ./foo.py
Instrumented:
% export PYTHONPATH= <taudir>/bgp/<lib>/bindings-python-pdt
(same options string as TAU_MAKEFILE)
% export LD_LIBRARY_PATH=<taudir>/bgp/lib/bindings-python-pdt:
$LD_LIBRARY_PATH
% ./wrapper.py

Wrapper invokes foo and generates performance data
% pprof/paraprof
```

Usage Scenarios: Mixed Python+F90+C+pyMPI

- Goal: Generate multi-level instrumentation for Python+MPI+C+F90+C++ ...

TAU: ParaProf: Mean Data - py.ppk

Metric: Time
Value: Exclusive percent

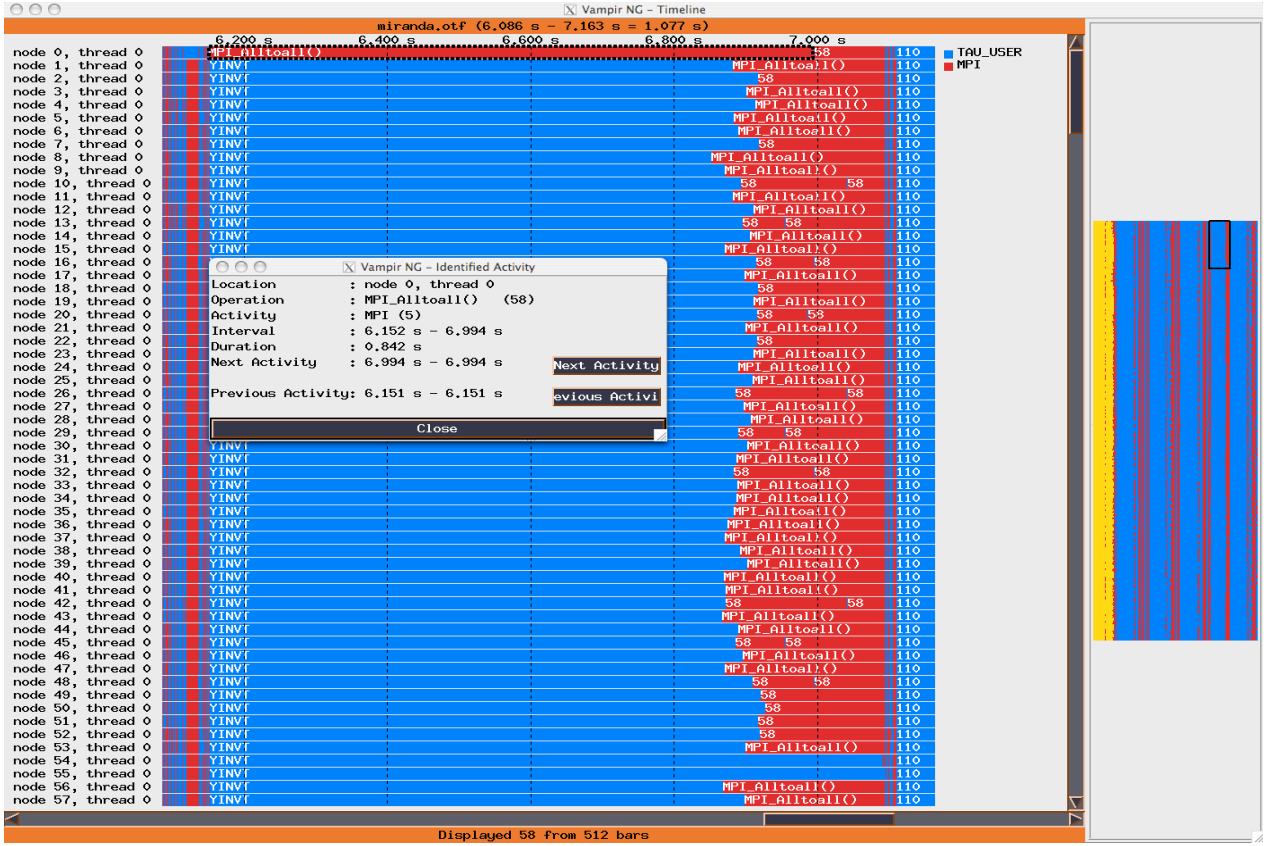
39.466%	void SAMINT::timestep(double, double) [{SAMINT.C} {72,1}-{78,1}]
6.563%	int pyMPI_Main_with_communicator(int, int *, char ***, MPI_Comm) C [{pyMPI_main.c} {23,1}-{80,1}]
5.18%	<module> [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/site-packages/numpy/linalg/linalg.py} {7}]
4.91%	void pyMPI_distutils_init(PyObject **) C [{pyMPI_distutils.c} {275,1}-{400,1}]
3.845%	<module> [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/site-packages/numpy/core/numeric.py} {1}]
3.633%	void pyMPI_site(void) C [{pyMPI_site.c} {23,1}-{121,1}]
3.607%	<module> [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/site-packages/numpy/core/__init__.py} {2}]
3.1%	MachAr::__init__ [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/site-packages/numpy/lib/machar.py} {50}]
2.261%	_parse [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/sre_parse.py} {385}]
2.123%	_compile [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/sre_compile.py} {38}]
1.187%	any [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/site-packages/numpy/core/fromnumeric.py} {540}]
1.034%	<module> [{/users3/sameer/py/py-c++-f90/example/samarcrun.py} {5}]
0.982%	<module> [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/site-packages/numpy/__init__.py} {17}]
0.941%	Tokenizer::__next [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/sre_parse.py} {188}]
0.929%	append
0.906%	<module> [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/ctypes/__init__.py} {4}]
0.895%	any
0.888%	<module> [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/site-packages/numpy/lib/__init__.py} {1}]
0.884%	len
0.863%	<module> [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/site-packages/numpy/lib/index_tricks.py} {3}]
0.806%	void pyMPI_pickle_init(PyObject **) C [{pyMPI_pickle.c} {43,1}-{61,1}]
0.739%	<module> [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/site-packages/numpy/random/__init__.py} {2}]
0.67%	SubPattern::getwidth [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/sre_parse.py} {146}]
0.658%	<module> [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/site-packages/numpy/testing/numpytest.py} {1}]
0.643%	_optimize_charset [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/sre_compile.py} {213}]
0.584%	Tokenizer::get [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/sre_parse.py} {207}]
0.552%	<module> [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/inspect.py} {24}]
0.541%	<module> [{/users3/sameer/py/py-c++-f90/samint.py} {7}]
0.537%	<module> [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/site-packages/numpy/core/memmap.py} {1}]
0.537%	<module> [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/site-packages/numpy/testing/__init__.py} {2}]
0.532%	<module> [{/users3/sameer/py/py-c++-f90/samarc.py} {9}]
0.475%	<module> [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/site-packages/numpy/ctypeslib.py} {1}]
0.462%	add_newdoc [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/site-packages/numpy/lib/function_base.py} {1154}]
0.453%	<module> [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/site-packages/numpy/lib/function_base.py} {1}]
0.451%	_parse_sub [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/sre_parse.py} {307}]
0.378%	_compile_charset [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/sre_compile.py} {184}]
0.367%	abs
0.365%	samarc::runStep [{/users3/sameer/py/py-c++-f90/samarc.py} {105}]
0.365%	<module> [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/site-packages/numpy/fft/fftpack.py} {21}]
0.296%	SubPattern::__len__ [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/sre_parse.py} {132}]
0.287%	<module> [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/site-packages/numpy/core/arrayprint.py} {4}]
0.27%	MPI_Finalize()
0.225%	Tokenizer::match [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/sre_parse.py} {201}]
0.213%	_mk_bitmap [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/sre_compile.py} {264}]
0.199%	<module> [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/site-packages/numpy/core/defmatrix.py} {1}]
0.18%	MPI_Init()
0.178%	SubPattern::__getitem__ [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/sre_parse.py} {136}]

Generate a Multi-Language Profile w/ Python

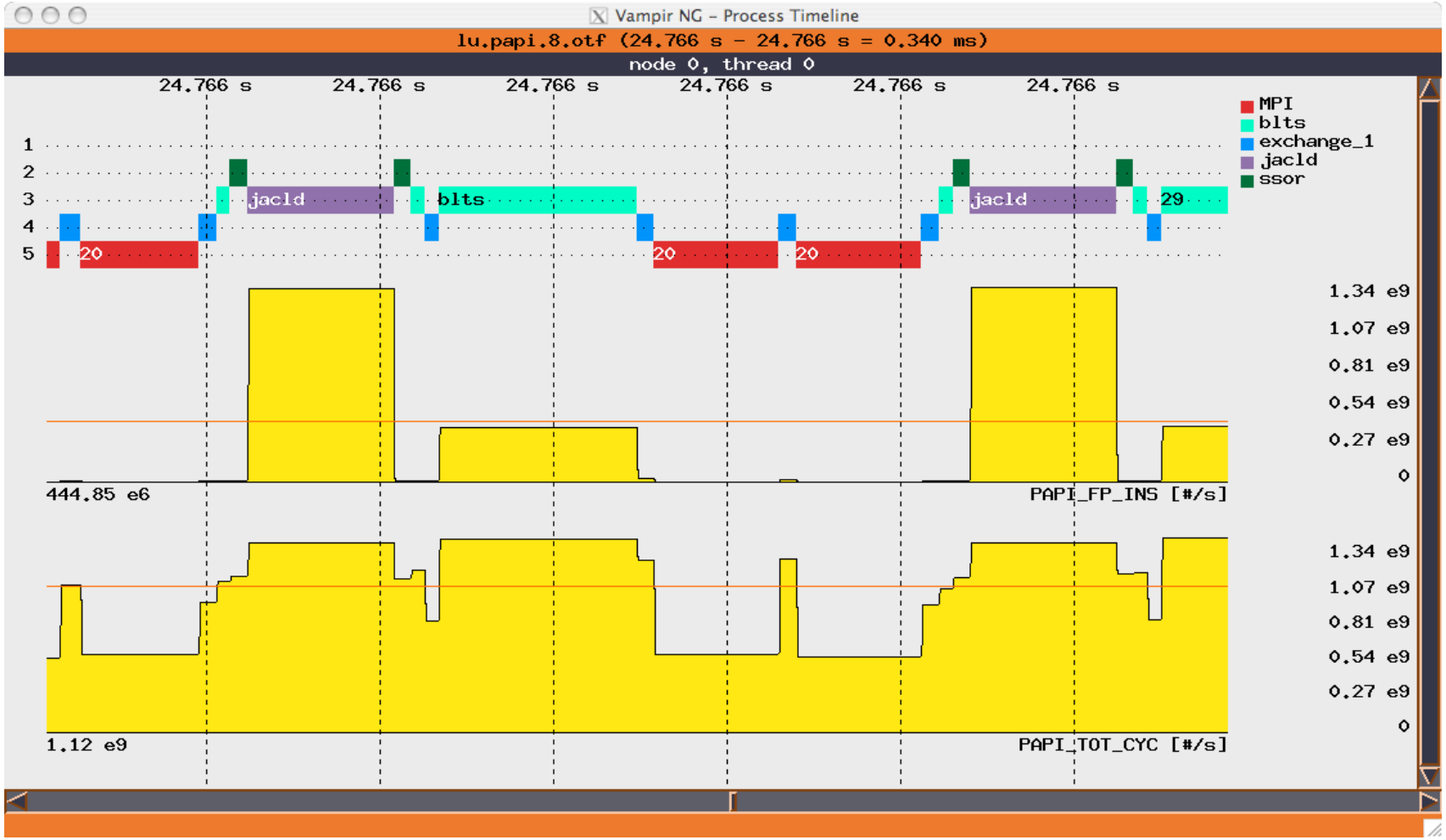
```
% export TAU_MAKEFILE=/soft/apps/tau/tau_latest/bgp
                        /lib/Makefile.tau-python-mpi-pdt
% export PATH=/soft/apps/tau/tau_latest/ppc64/bin:$PATH
% export TAU_OPTIONS='-optShared -optVerbose…'
(Python needs shared object based TAU library)
% make F90=tau_f90.sh CXX=tau_cxx.sh CC=tau_cc.sh  (build libs, pyMPI w/TAU)
% cat wrapper.py
  import tau
  def OurMain():
      import App
  tau.run('OurMain()')
Uninstrumented:
% cqsub -a -n 4 <dir>/pyMPI-2.5b0/bin/pyMPI ./App.py
Instrumented:
% export PYTHONPATH=<taudir>/bgp/<lib>/bindings-python-mpi-pdt
(same options string as TAU_MAKEFILE)
% export LD_LIBRARY_PATH=<taudir>/bgp/lib/bindings-python-mpi-pdt:
$LD_LIBRARY_PATH
% cqsub -a -n 4 <dir>/pyMPI-2.5b0-TAU/bin/pyMPI ./wrapper.py
(Instrumented pyMPI with wrapper.py)
```

Usage Scenarios: Generating a Trace File

- Goal: Identify the temporal aspect of performance. What happens in my code at a given time? When?
- Event trace visualized in Vampir/Jumpshot



VNG Process Timeline with PAPI Counters



Vampir Counter Timeline Showing I/O BW

Generate a Trace File

```
% export TAU_MAKEFILE=/soft/apps/tau/tau_latest/bgp
                        /lib/Makefile.tau-mpi-pdt
% export PATH=/soft/apps/tau/tau_latest/ppc64/bin:$PATH
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)
% export TAU_TRACE=1
% mpirun -np 4  ./a.out
% tau_treemerge.pl
(merges binary traces to create tau.trc and tau.edf files)
JUMPSHOT:
% tau2slog2 tau.trc tau.edf -o app.slog2
% jumpshot app.slog2
   OR
VAMPIR:
% tau2otf tau.trc tau.edf app.otf -n 4 -z
(4 streams, compressed output trace)
% vampir app.otf
(or vng client with vngd server).
```

Measuring Performance of PGI Accelerator Code

TAU: ParaProf: n,c,t 0,0,0 - mat1k.ppk

Metric: TIME
Value: Exclusive percent

68.044%	__pgi_cu_launch multiply_matrices (pgi_kernel_7,gx=32,gy=32,gz=1,bx=16,by=16,bz=1) [{mm2.f90}{15}]
24.417%	__pgi_cu_init multiply_matrices [{mm2.f90}{9}]
3.206%	__pgi_cu_download2 multiply_matrices var=a [{mm2.f90}{20}]
1.572%	__pgi_cu_upload2 multiply_matrices var=b [{mm2.f90}{9}]
1.572%	__pgi_cu_upload2 multiply_matrices var=c [{mm2.f90}{9}]
0.782%	mymatrixmultiply [{mmdriv.f90} {1,0}]
0.142%	__pgi_cu_launch multiply_matrices (pgi_kernel_2,gx=32,gy=32,gz=1,bx=16,by=16,bz=1) [{mm2.f90}{11}]
0.122%	__pgi_cu_free multiply_matrices [{mm2.f90}]
0.12%	__pgi_cu_alloc multiply_matrices [{mm2.f90}{9}]
0.017%	multiply_matrices [{mm2.f90} {5,0}]
0.005%	pgi accelerator region
0.002%	__pgi_cu_module multiply_matrices [{mm2.f90}{9}]
2.1E-4%	__pgi_cu_module_function multiply_matrices [{mm2.f90}{11}]
1.2E-4%	__pgi_cu_paramset multiply_matrices [{mm2.f90}]
1.2E-4%	__pgi_cu_module_function multiply_matrices [{mm2.f90}{15}]

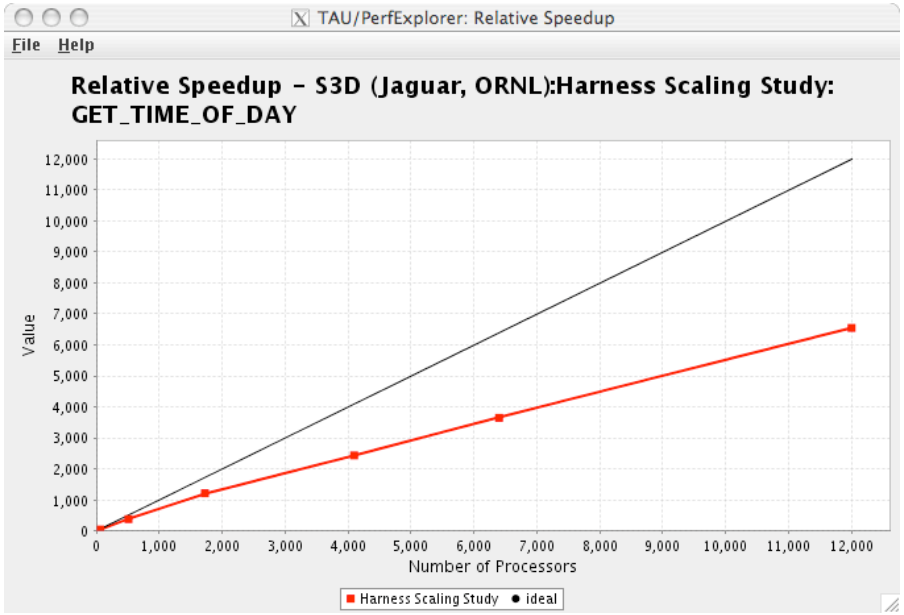
TAU: ParaProf: Thread Statistics: n,c,t, 0,0,0 - mat1k.ppk

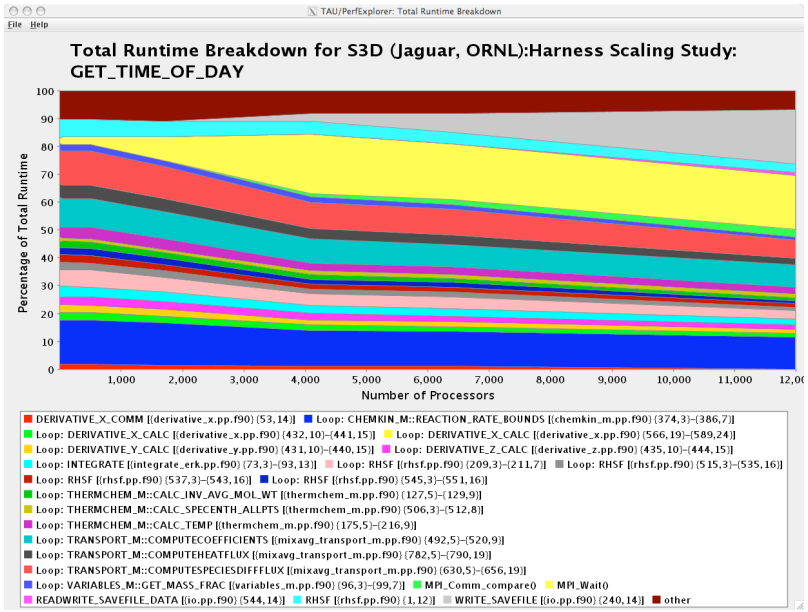
Name	Exclusive TIME	Inclusive TIME	Calls	Child Calls
__pgi_cu_launch multiply_matrices (pgi_kernel_7,gx=32,gy=32,gz=1,bx=16,by=16,bz=1) [{mm2.f90}{15}]	10.901	10.901	5	0
__pgi_cu_init multiply_matrices [{mm2.f90}{9}]	3.912	3.912	5	0
__pgi_cu_download2 multiply_matrices var=a [{mm2.f90}{20}]	0.514	0.514	5	0
__pgi_cu_upload2 multiply_matrices var=b [{mm2.f90}{9}]	0.252	0.252	5	0
__pgi_cu_upload2 multiply_matrices var=c [{mm2.f90}{9}]	0.252	0.252	5	0
mymatrixmultiply [{mmdriv.f90} {1,0}]	0.125	16.021	1	1
__pgi_cu_launch multiply_matrices (pgi_kernel_2,gx=32,gy=32,gz=1,bx=16,by=16,bz=1) [{mm2.f90}{11}]	0.023	0.023	5	0
__pgi_cu_free multiply_matrices [{mm2.f90}]	0.02	0.02	15	0
__pgi_cu_alloc multiply_matrices [{mm2.f90}{9}]	0.019	0.019	15	0
multiply_matrices [{mm2.f90} {5,0}]	0.003	15.895	5	9
pgi accelerator region	0.001	15.893	5	85
__pgi_cu_module multiply_matrices [{mm2.f90}{9}]	0	0	5	0
__pgi_cu_module_function multiply_matrices [{mm2.f90}{11}]	0	0	5	0
__pgi_cu_paramset multiply_matrices [{mm2.f90}]	0	0	10	0

- Show Source Code
- Show Function Bar Chart
- Show Function Histogram
- Assign Function Color
- Reset to Default Color

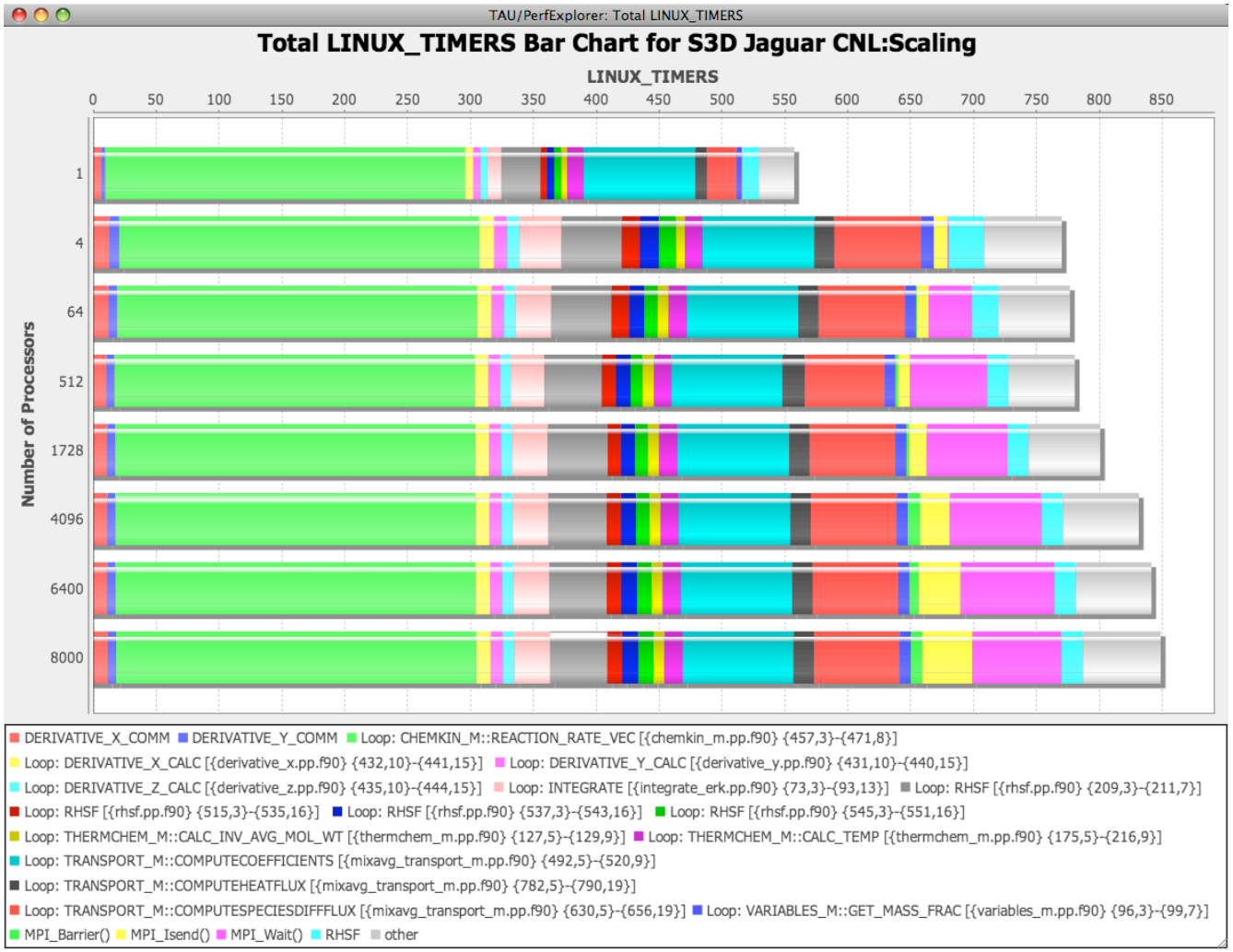
Usage Scenarios: Evaluate Scalability

- Goal: How does my application scale? What bottlenecks occur at what core counts?
- Load profiles in PerfDMF database and examine with PerfExplorer

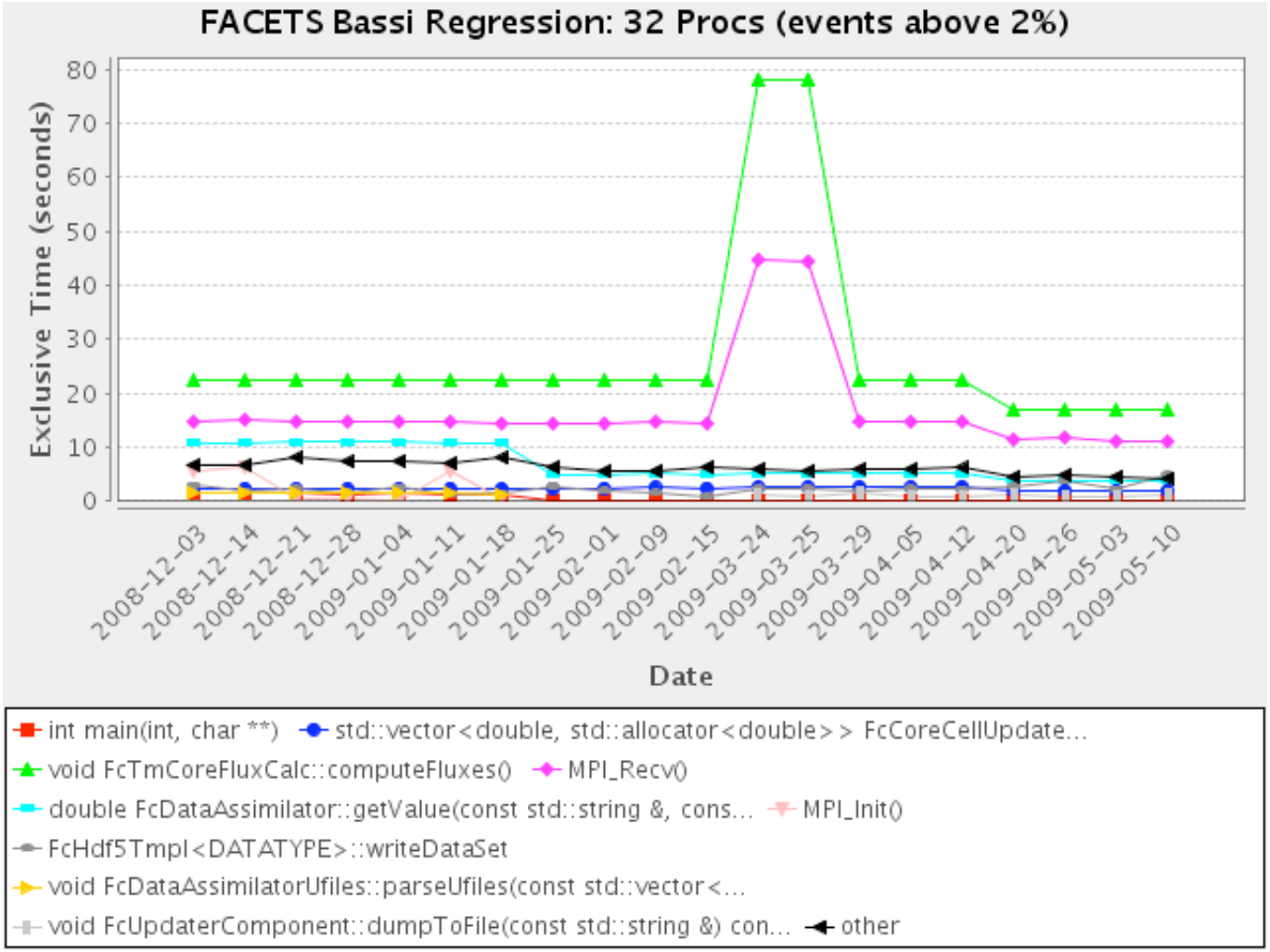




Usage Scenarios: Evaluate Scalability



Performance Regression Testing

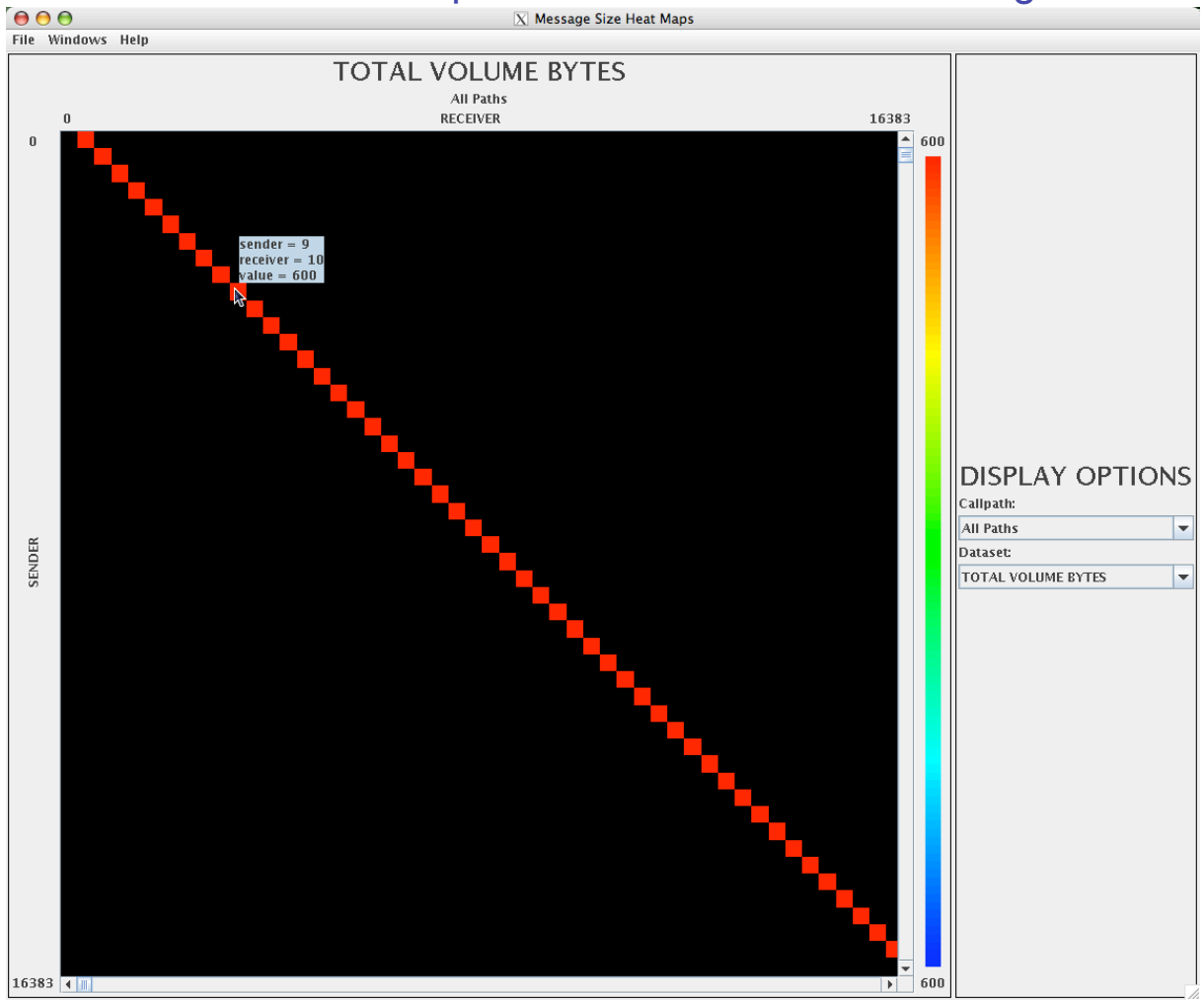


Evaluate Scalability using PerfExplorer Charts

```
% export TAU_MAKEFILE=/soft/apps/tau/tau_latest/bgp
                              /lib/Makefile.tau-mpi-pdt
% export PATH=/soft/apps/tau/tau_latest/ppc64/bin:$PATH
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)
% mpirun -np 1 ./a.out
% paraprof --pack 1p.ppk
% mpirun -np 2 ./a.out …
% paraprof --pack 2p.ppk … and so on.
On your client:
% perfdmf_configure --create-default
(Chooses derby, blank user/passwd, yes to save passwd, defaults)
% perfexplorer_configure
(Yes to load schema, defaults)
% paraprof
(load each trial: DB -> Add Trial -> Type (Paraprof Packed Profile) -> OK) OR use
    perfdmf_loadtrial
Then,
% perfexplorer
(Select experiment, Menu: Charts -> Speedup)
```

Communication Matrix Display

- Goal: What is the volume of inter-process communication? Along which calling path?

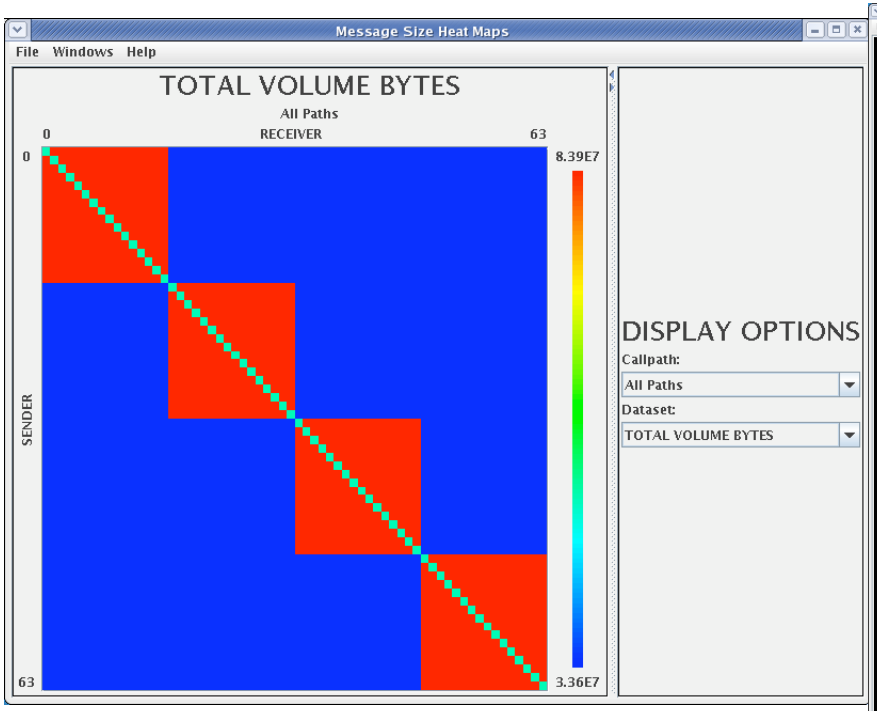


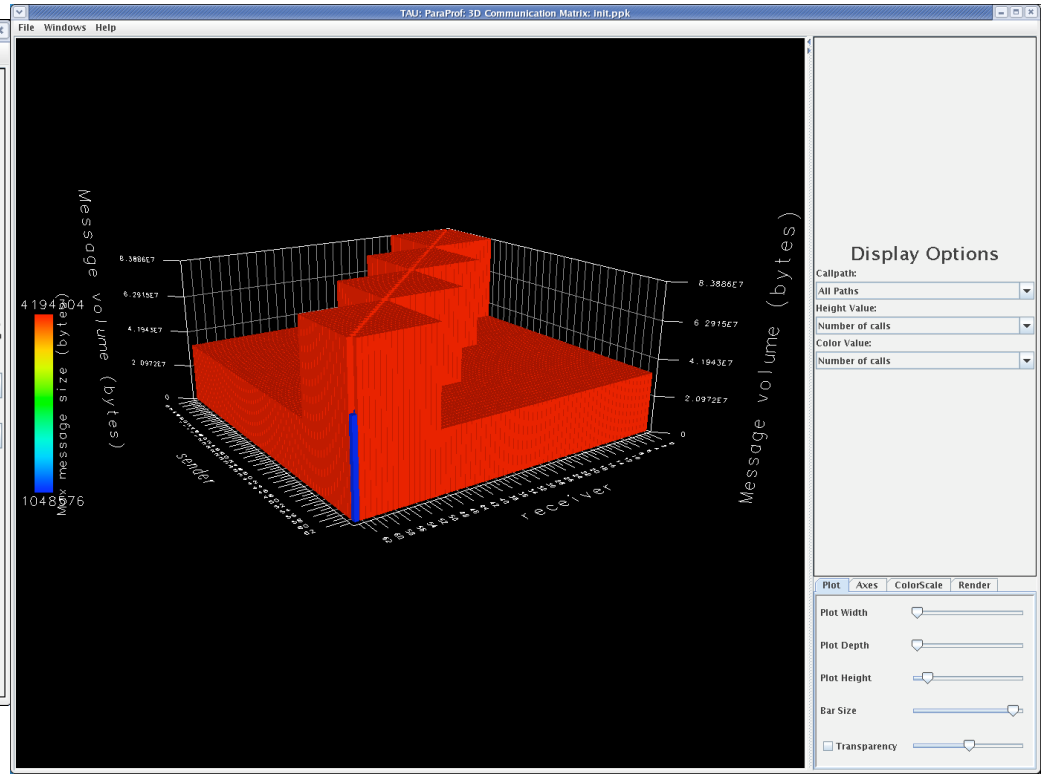
Evaluate Scalability using PerfExplorer Charts

```
% export TAU_MAKEFILE=/soft/apps/tau/tau_latest/bgp
                              /lib/Makefile.tau-mpi-pdt

% export PATH=/soft/apps/tau/tau_latest/ppc64/bin:$PATH

% make F90=tau_f90.sh

(Or edit Makefile and change F90=tau_f90.sh)

% export TAU_COMM_MATRIX=1


% qsub -A TAU09 -n 4 -t 10 -q R.TAU09 --mode vn ./a.out (setting the environment
    variables)


% paraprof

(Windows -> Communication Matrix, Windows -> 3D Communication Matrix)
```

Communication Matrix Display

- Goal: What is the volume of inter-process communication? Along which calling path?





Interval Events, Atomic Events in TAU

```
xterm
NODE 0;CONTEXT 0;THREAD 0:
---------------------------------------------------------------------------------------
%Time    Exclusive    Inclusive       #Call      #Subrs  Inclusive Name
              msec   total msec                          usec/call
---------------------------------------------------------------------------------------
100.0        0.187        1,105           1          44    1105659 int main(int, char **) C
 93.2        1,030        1,030           1           0    1030654 MPI_Init()
  5.9        0.879           65          40         320       1637 void func(int, int) C
  4.6           51           51          40           0       1277 MPI_Barrier()
  1.2           13           13         120           0        111 MPI_Recv()
  0.8            9            9           1           0       9328 MPI_Finalize()
  0.0        0.137        0.137         120           0          1 MPI_Send()
  0.0        0.086        0.086          40           0          2 MPI_Bcast()
  0.0        0.002        0.002           1           0          2 MPI_Comm_size()
  0.0        0.001        0.001           1           0          1 MPI_Comm_rank()
---------------------------------------------------------------------------------------

USER EVENTS Profile :NODE 0, CONTEXT 0, THREAD 0
---------------------------------------------------------------------------------------
NumSamples   MaxValue   MinValue  MeanValue  Std. Dev.  Event Name
---------------------------------------------------------------------------------------
       365  5.138E+04      44.39   3.09E+04  1.234E+04  Heap Memory Used (KB) : Entry
       365  5.138E+04       2064  3.115E+04   1.21E+04  Heap Memory Used (KB) : Exit
        40         40         40         40          0  Message size for broadcast
---------------------------------------------------------------------------------------
                                                                  27,1          1%
```

Interval event e.g., routines (start/stop)

Atomic events (trigger with value)

% setenv TAU_CALLPATH_DEPTH 0
% setenv TAU_TRACK_HEAP 1

Atomic Events, Context Events

```
% setenv TAU_CALLPATH_DEPTH      1
% setenv TAU_TRACK_HEAP          1
```

Context Events (default)

```
xterm
NODE 0;CONTEXT 0;THREAD 0:
---------------------------------------------------------------------------------------
%Time    Exclusive    Inclusive       #Call      #Subrs  Inclusive Name
              msec   total msec                          usec/call
---------------------------------------------------------------------------------------
100.0        0.357        1,114           1          44    1114040 int main(int, char **) C
 92.6        1,031        1,031           1           0    1031066 MPI_Init()
  6.7           72           74          40         320       1865 void func(int, int) C
  0.7            8            8           1           0       8002 MPI_Finalize()
  0.1            1            1         120           0         12 MPI_Recv()
  0.1        0.608        0.608          40           0         15 MPI_Barrier()
  0.0        0.136        0.136         120           0          1 MPI_Send()
  0.0        0.095        0.095          40           0          2 MPI_Bcast()
  0.0        0.001        0.001           1           0          1 MPI_Comm_size()
  0.0            0            0           1           0          0 MPI_Comm_rank()
---------------------------------------------------------------------------------------

USER EVENTS Profile :NODE 0, CONTEXT 0, THREAD 0
---------------------------------------------------------------------------------------
NumSamples   MaxValue   MinValue  MeanValue  Std. Dev.  Event Name
---------------------------------------------------------------------------------------
       365  5.139E+04      44.39  3.091E+04  1.234E+04  Heap Memory Used (KB) : Entry
         1      44.39      44.39      44.39          0  Heap Memory Used (KB) : Entry : int main(int, char **) C
         1       2068       2068       2068          0  Heap Memory Used (KB) : Entry : int main(int, char **) C => MPI_Comm_rank()
         1       2066       2066       2066          0  Heap Memory Used (KB) : Entry : int main(int, char **) C => MPI_Comm_size()
         1  5.139E+04  5.139E+04  5.139E+04          0  Heap Memory Used (KB) : Entry : int main(int, char **) C => MPI_Finalize()
         1      57.58      57.58      57.58          0  Heap Memory Used (KB) : Entry : int main(int, char **) C => MPI_Init()
        40  5.036E+04       2069  3.011E+04  1.228E+04  Heap Memory Used (KB) : Entry : int main(int, char **) C => void func(int, int) C
        40  5.139E+04       3098  3.114E+04  1.227E+04  Heap Memory Used (KB) : Entry : void func(int, int) C => MPI_Barrier()
        40  5.139E+04   1.13E+04  3.134E+04  1.187E+04  Heap Memory Used (KB) : Entry : void func(int, int) C => MPI_Bcast()
       120  5.139E+04   1.13E+04  3.134E+04  1.187E+04  Heap Memory Used (KB) : Entry : void func(int, int) C => MPI_Recv()
       120  5.139E+04   1.13E+04  3.134E+04  1.187E+04  Heap Memory Used (KB) : Entry : void func(int, int) C => MPI_Send()
       365  5.139E+04       2065  3.116E+04   1.21E+04  Heap Memory Used (KB) : Exit
                                                                                    3.7
```

Context event = atomic event + executing context

% setenv TAU_CALLPATH_DEPTH 2
% setenv TAU_TRACK_HEAP 1

Binary Rewriting: DyninstAPI [U.Wisc] and TAU

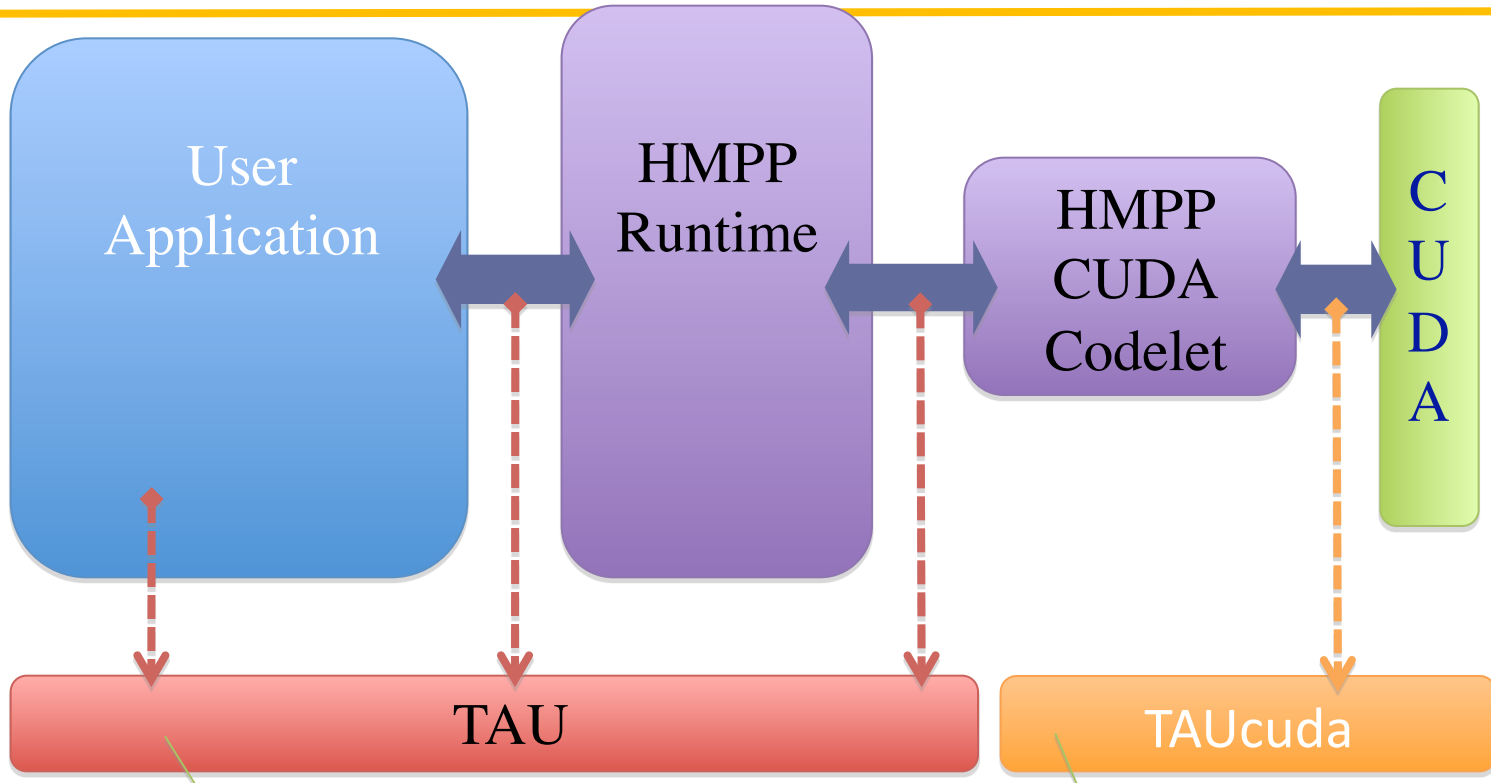
livetau@paratools01:~

```
/home/livetau% cd ~/tutorial
/home/livetau/tutorial% # Build an uninstrumented bt NAS Parallel Benchmark
/home/livetau/tutorial% make bt CLASS=W NPROCS=4
/home/livetau/tutorial% cd bin
/home/livetau/tutorial/bin% # Run the instrumented code
/home/livetau/tutorial/bin% mpirun -np 4 ./bt_W.4
/home/livetau/tutorial/bin%
/home/livetau/tutorial/bin% # Instrument the executable using TAU with DyninstAPI
/home/livetau/tutorial/bin%
/home/livetau/tutorial/bin% tau_run ./bt_W.4 -o ./bt.i
/home/livetau/tutorial/bin% rm -rf profile.* MULT*
/home/livetau/tutorial/bin% mpirun -np 4 ./bt.i
/home/livetau/tutorial/bin% paraprof
/home/livetau/tutorial/bin%
/home/livetau/tutorial/bin% # Choose a different TAU configuration
/home/livetau/tutorial/bin% ls $TAU/libTAUsh
libTAUsh-depthlimit-mpi-pdt.so*              libTAUsh-papi-pdt.so*
libTAUsh-mpi-pdt.so*                         libTAUsh-papi-pthread-pdt.so*
libTAUsh-mpi-pdt-upc.so*                     libTAUsh-param-mpi-pdt.so*
libTAUsh-mpi-python-pdt.so*                  libTAUsh-pdt.so*
libTAUsh-papi-mpi-pdt.so*                    libTAUsh-pdt-trace.so*
libTAUsh-papi-mpi-pdt-upc.so*                libTAUsh-phase-papi-mpi-pdt.so*
libTAUsh-papi-mpi-pdt-upc-udp.so*            libTAUsh-pthread-pdt.so*
libTAUsh-papi-mpi-pdt-vampirtrace-trace.so* libTAUsh-python-pdt.so*
libTAUsh-papi-mpi-python-pdt.so*
/home/livetau/tutorial/bin% ls $TAU/libTAUsh-
/home/livetau/tutorial/bin%
/home/livetau/tutorial/bin% tau_run -XrunTAUsh-papi-mpi-pdt-vampirtrace-trace bt_W.4 -o bt.vpt
/home/livetau/tutorial/bin% setenv VT_METRICS PAPI_FP_INS:PAPI_L1_DCM
/home/livetau/tutorial/bin% mpirun -np 4 ./bt.vpt
/home/livetau/tutorial/bin% vampir bt.vpt.otf &
/home/livetau/tutorial/bin%
```

Performance Evaluation on GPGPUs: HMPP [Caps Entreprise]

- Goal
  - Automated, seamless instrumentation of HMPP applications
  - Instrumentation support for codelet target generation
- Workflow
  - Replace compiler with TAU compiler
    - allows instrumentation of application-level events
  - Indicate HMPP compiler (as TAU's compiler target)
    - TAUcuda instrumentation automatically generated
    - placed in codelet around CUDA kernel statements
- HMPP TAU management
  - Default support in the HMPP runtime
  - Activated using compiler flags

HMPP-TAU Event Instrumentation/Measurement

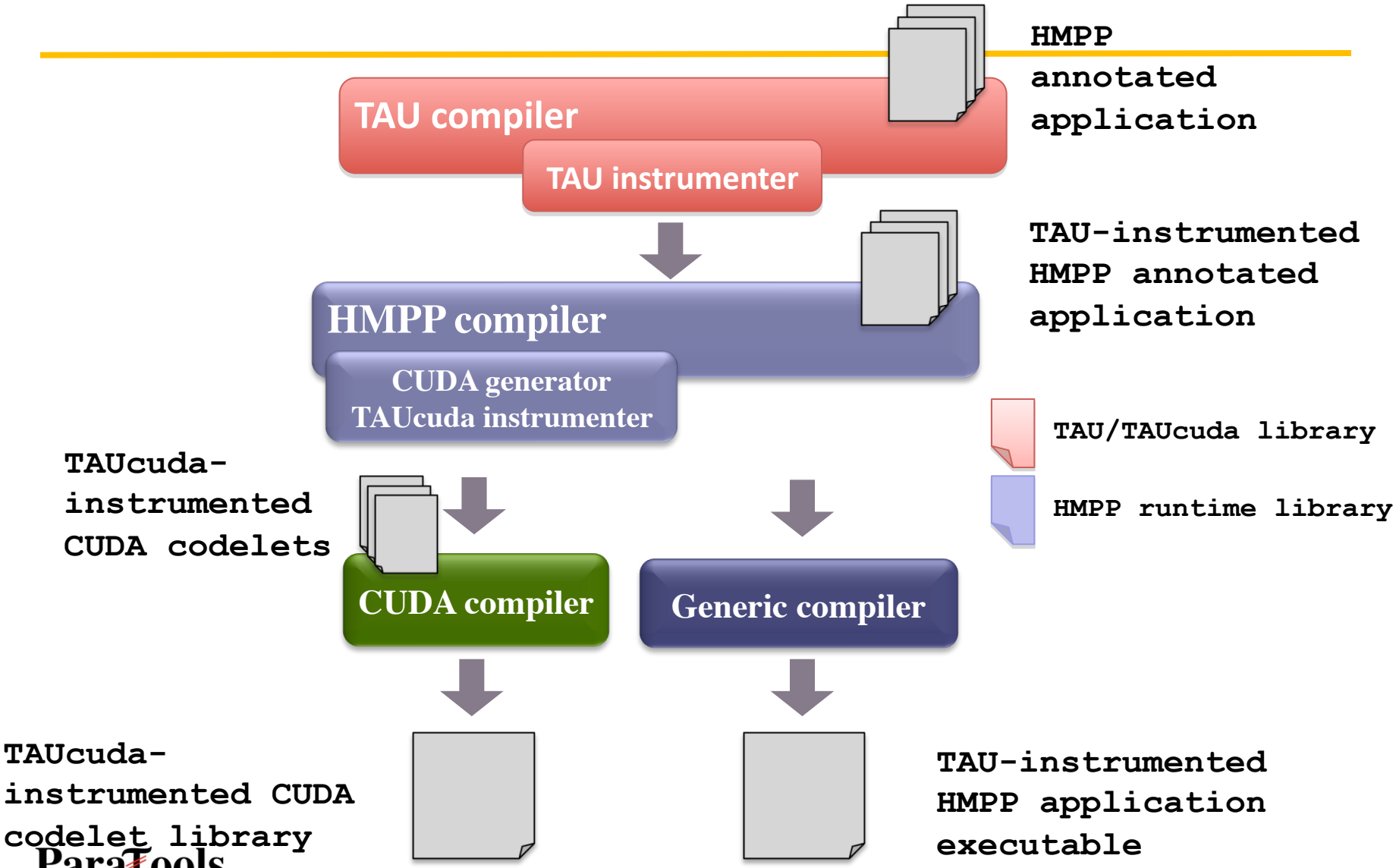


Measurement
- User events
- HMPP events
- Codelet events

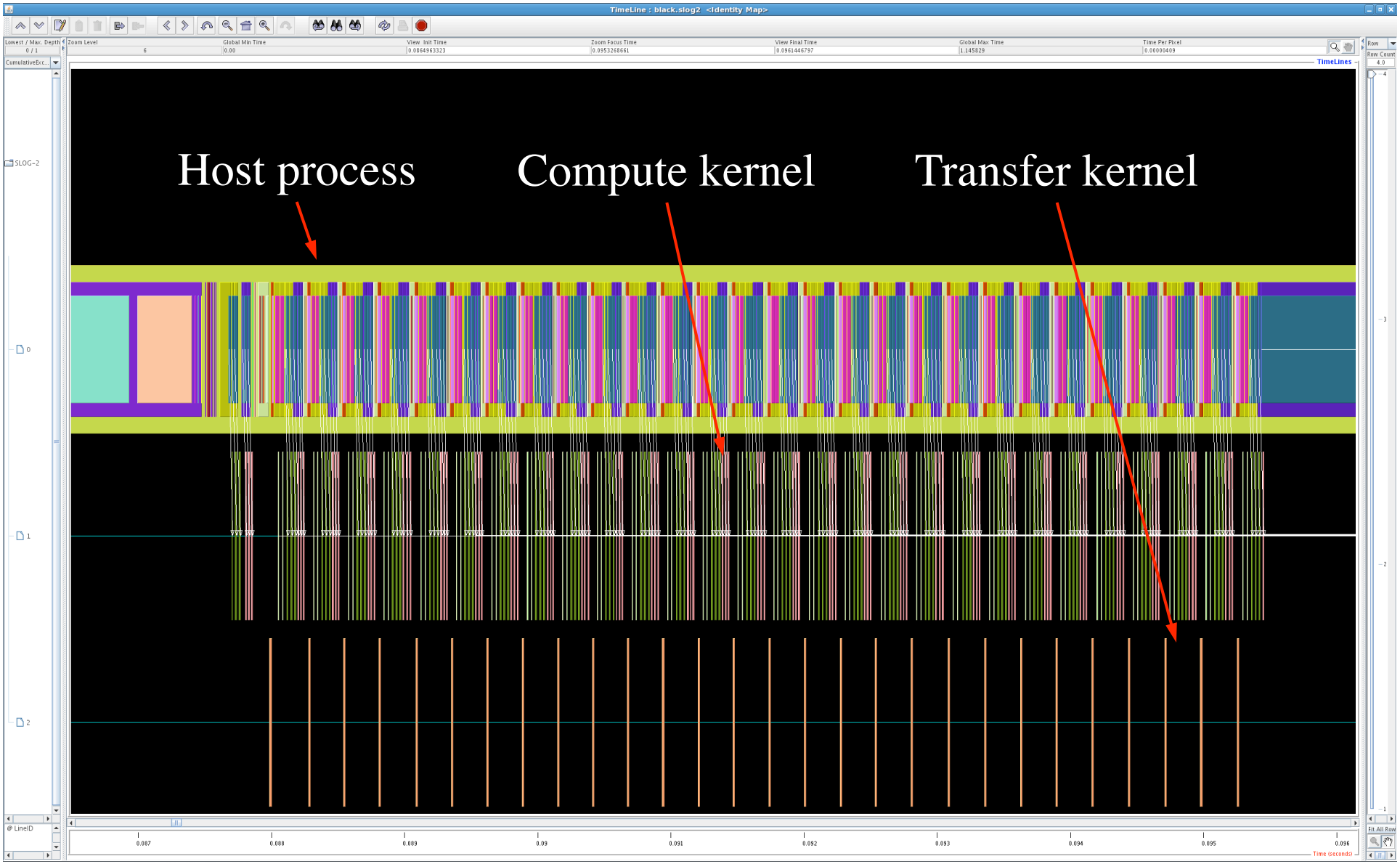
Measurement
- CUDA stream events
- Waiting information

ParaTools

HMPP-TAU Compilation Workflow

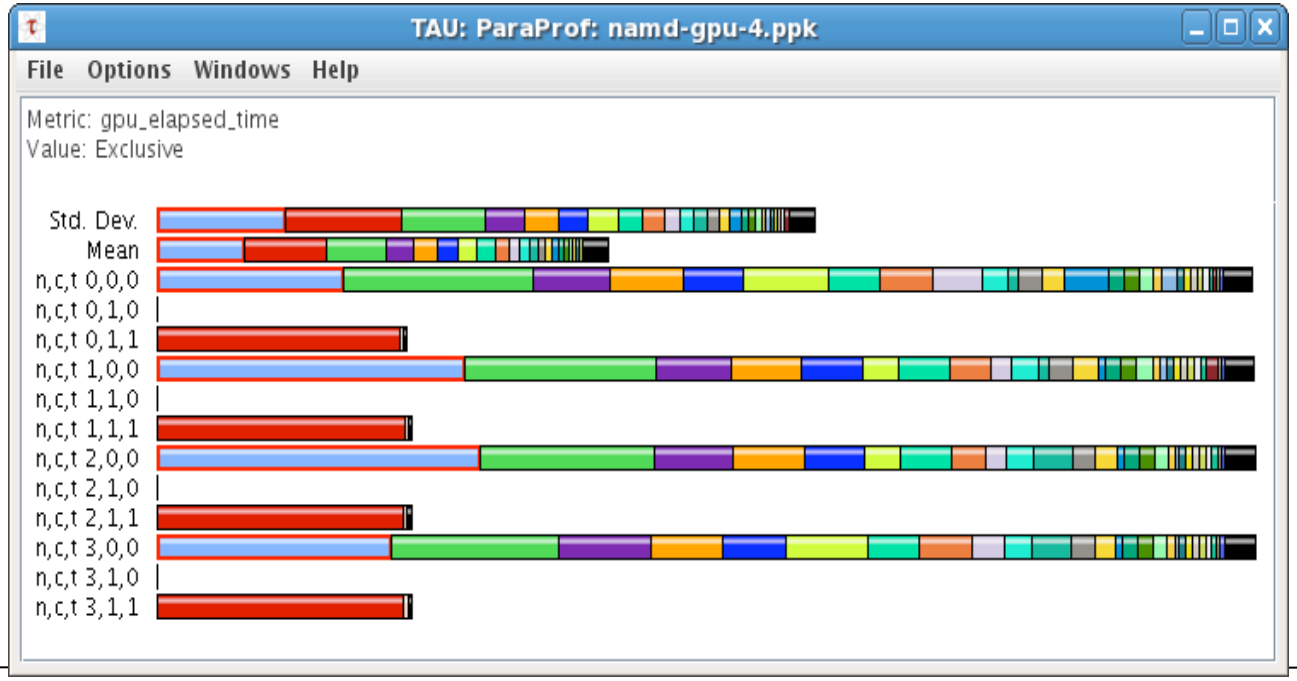


HMPP Workbench with TAUcuda

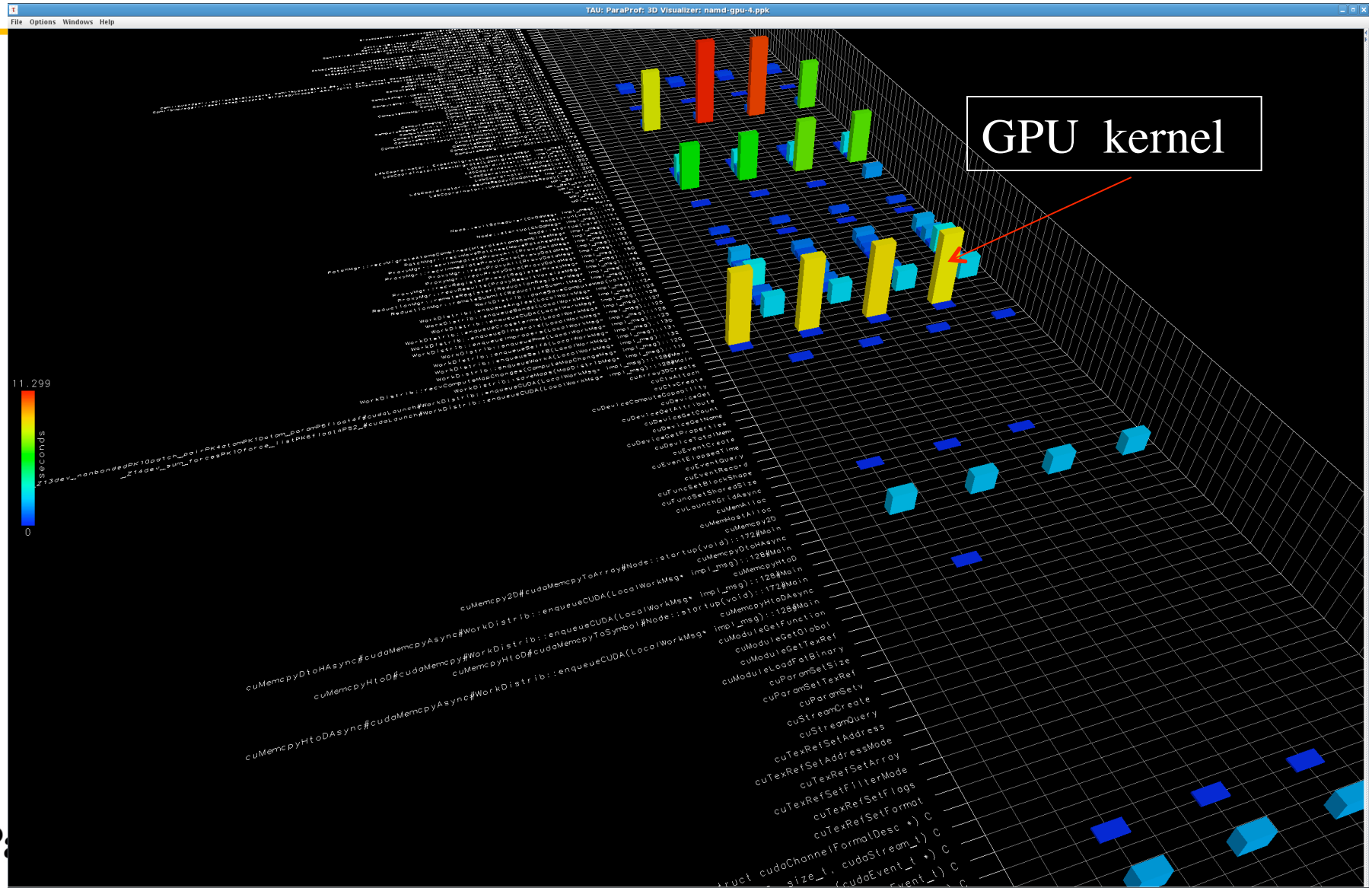


NAMD with CUDA

- NAMD is a molecular dynamics application (Charm++)
- NAMD has been accelerated with CUDA
- TAU integrated in Charm++
- Apply TAUcuda to NAMD
  - Four processes with one Tesla GPU for each



NAMD with CUDA (4 processes)



Scaling NAMD with CUDA

good GPU performance

ParaProf: Comparison Window

File Options Windows Help

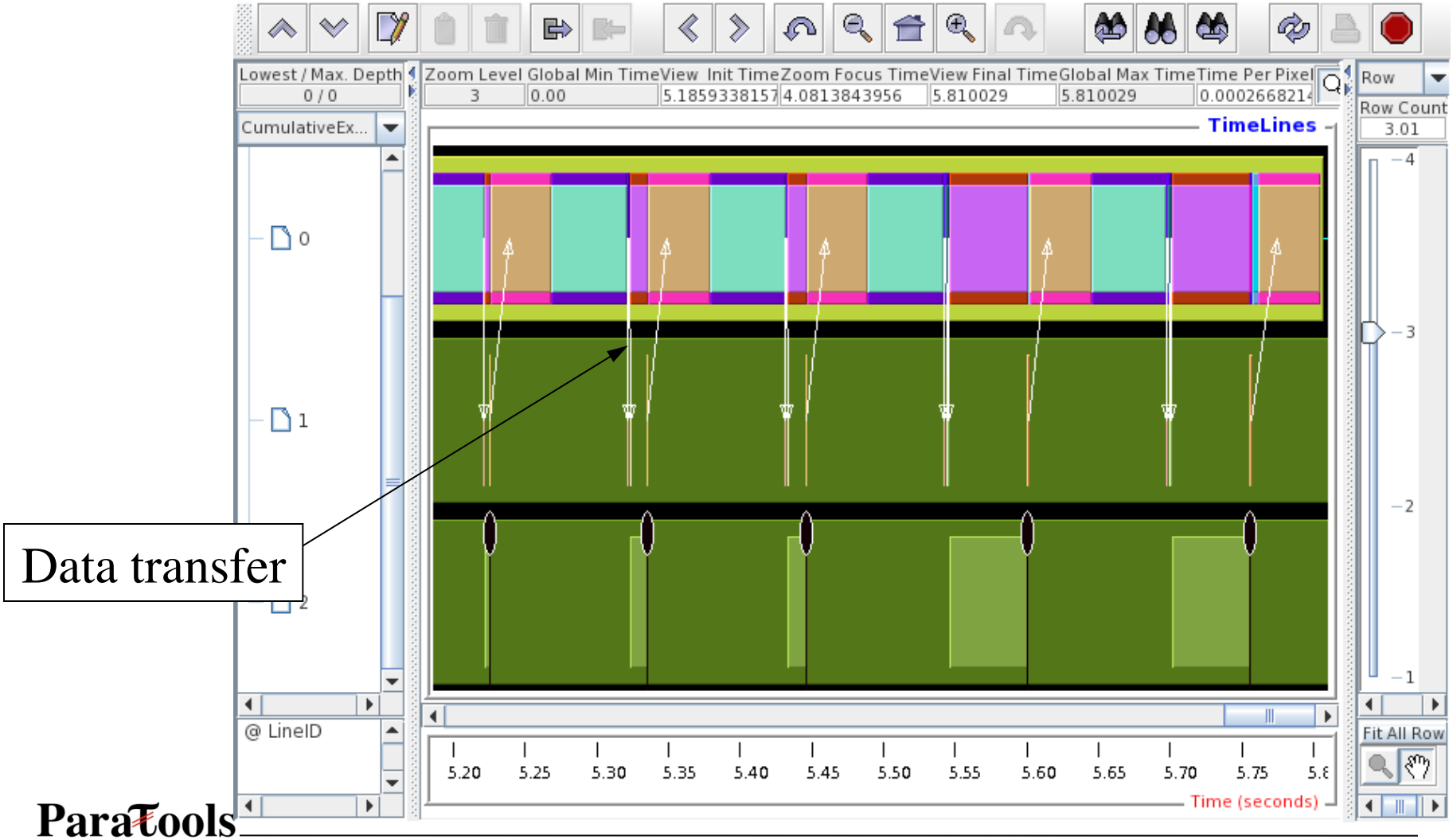
Metric: TIME
Value: Exclusive
Units: seconds

namd-gpu-1.ppk - Mean
namd-gpu-2.ppk - Mean
namd-gpu-4.ppk - Mean

10.563 5.524 (52.299%) 2.868 (27.147%)	_Z13dev_nonbondedPK10patch_pairPK4atomPK10atom_paramP6float4
4.467 5.462 (122.274%) 2.101 (47.025%)	Main
3.573 2.031 (56.831%) 3.064 (85.745%)	Idle
3.307 1.672 (50.545%) 0.834 (25.228%)	WorkDistrib::enqueueWorkA(LocalWorkMsg* impl_msg)::132
2.608 1.215 (46.61%) 0.692 (26.524%)	WorkDistrib::enqueueDihedrals(LocalWorkMsg* impl_msg)::125
1.929 0.926 (47.974%) 0.522 (27.054%)	WorkDistrib::enqueueAngles(LocalWorkMsg* impl_msg)::124
1.727 0.669 (38.719%) 0.318 (18.413%)	WorkDistrib::enqueuePme(LocalWorkMsg* impl_msg)::129
1.326 0.602 (45.368%) 0.273 (20.616%)	ComputePmeMgr::ungridCalc(void)::279
0.876 0.943	MPI_Recv()
0.924 0.464 (50.215%) 0.262 (28.322%)	WorkDistrib::enqueueBonds(LocalWorkMsg* impl_msg)::123
0.715 0.351 (49.17%) 0.175 (24.416%)	ComputePmeMgr::gridCalc2(void)::273
0.037 0.709 (1937.536%) 0.709 (1937.593%)	cuMemcpyDtoHAsync
0.677 0.347 (51.2%) 0.178 (26.333%)	WorkDistrib::enqueueSelfA(LocalWorkMsg* impl_msg)::130
0.672 0.027 (4.032%) 0.031 (4.556%)	cuLaunchGridAsync

Scaling NAMD with CUDA: Jumpshot Timeline

Scaling NAMD with CUDA



Environment Variables in TAU

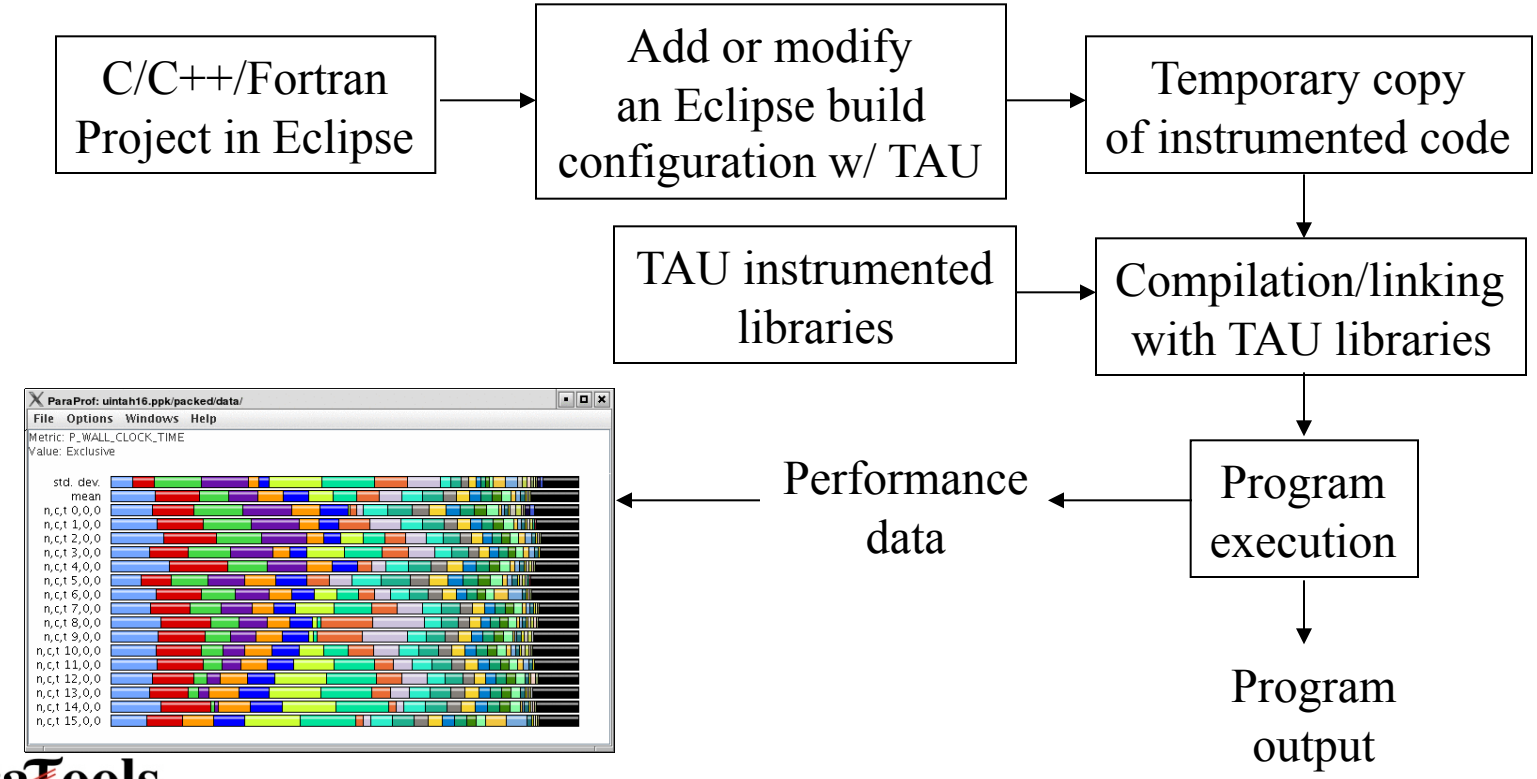
Environment Variable	Default	Description
TAU_TRACE	0	Setting to 1 turns on tracing
TAU_CALLPATH	0	Setting to 1 turns on callpath profiling
TAU_TRACK_HEAP or TAU_TRACK_HEADROOM	0	Setting to 1 turns on tracking heap memory/headroom at routine entry & exit using context events (e.g., Heap at Entry: main=>foo=>bar)
TAU_CALLPATH_DEPTH	2	Specifies depth of callpath. Setting to 0 generates no callpath or routine information, setting to 1 generates flat profile and context events have just parent information (e.g., Heap Entry: foo)
TAU_SYNCHRONIZE_CLOCKS	1	Synchronize clocks across nodes to correct timestamps in traces
TAU_COMM_MATRIX	0	Setting to 1 generates communication matrix display using context events
TAU_THROTTLE	1	Setting to 0 turns off throttling. Enabled by default to remove instrumentation in lightweight routines that are called frequently
TAU_THROTTLE_NUMCALLS	100000	Specifies the number of calls before testing for throttling
TAU_THROTTLE_PERCALL	10	Specifies value in microseconds. Throttle a routine if it is called over 100000 times and takes less than 10 usec of inclusive time per call
TAU_COMPENSATE	0	Setting to 1 enables runtime compensation of instrumentation overhead
TAU_PROFILE_FORMAT	Profile	Setting to “merged” generates a single file. “snapshot” generates xml format
TAU_METRICS	TIME	Setting to a comma separted list generates other metrics. (e.g., TIME:linuxtimers:PAPI_FP_OPS:PAPI_NATIVE_<event>)

TAU Integration with IDEs

- High performance software development environments
  - Tools may be complicated to use
  - Interfaces and mechanisms differ between platforms / OS
- Integrated development environments
  - Consistent development environment
  - Numerous enhancements to development process
  - Standard in industrial software development
- Integrated performance analysis
  - Tools limited to single platform or programming language
  - Rarely compatible with 3rd party analysis tools
  - Little or no support for parallel projects

TAU and Eclipse

- Provide an interface for configuring TAU's automatic instrumentation within Eclipse's build system
- Manage runtime configuration settings and environment variables for execution of TAU instrumented programs



TAU and Eclipse

Fortran – matmult.f90 – Eclipse SDK

File Edit Refactor Navigate Search Project Run Window Help

Fortran Projects | Navigator

matmultiply
- Binaries
- Includes
- .settings
- Debug_ia32
- Debug_ia32(tau-icpc-mpi-pdt)
- matmult.f90
- .cdtbuild
- .cdtproject
- .project
- profile.0.0.0
- profile.1.0.0
- profile.2.0.0
- profile.3.0.0

matmult.f90

```
!*********************************************************************
!     matmult.f90 - simple matrix multiply implementation
!*********************************************************************
      subroutine initialize(a, b, n)
        double precision a(n,n)
        double precision b(n,n)
        integer n

! first initialize the A matrix
        do i = 1,n
          do j = 1,n
            a(j,i) = i
          end do
        end do

! then initialize the B matrix
        do i = 1,n
          do j = 1,n
            b(j,i) = i
          end do
        end do

      end subroutine initialize

      subroutine multiply_matrices(answer, buffer, b, matsize)
        double precision buffer(matsize), answer(matsize)
        double precision b(matsize, matsize)
        integer i, j
! multiply the row with the column
```

Outline | Make ...

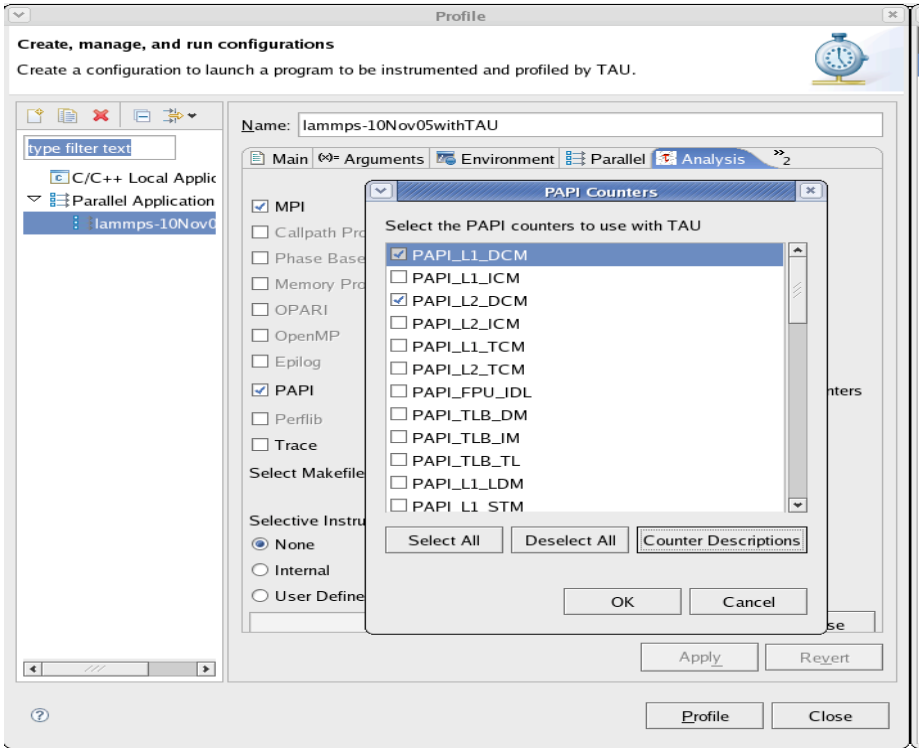
- initialize
- multiply_matrices
- main

Problems | Console | Properties | Performance Data Manager | Launch ParaProf

- AORSA2D
- matmultiply
  - Experiment
    - The New Trial: 2006-12-02 20:36:59
- mm
- ring

PerfDMF

Choosing PAPI Counters with TAU in Eclipse



Counter	Definition
PAPI_L1_DCM	Level 1 data cache misses
PAPI_L1_ICM	Level 1 instruction cache misses
PAPI_L2_DCM	Level 2 data cache misses
PAPI_L2_ICM	Level 2 instruction cache misses
PAPI_L1_TCM	Level 1 cache misses
PAPI_L2_TCM	Level 2 cache misses
PAPI_FPU_IDL	Cycles floating point units are idle
PAPI_TLB_DM	Data translation lookaside buffer misses
PAPI_TLB_IM	Instruction translation lookaside buffer misses
PAPI_TLB_TL	Total translation lookaside buffer misses
PAPI_L1_LDM	Level 1 load misses
PAPI_L1_STM	Level 1 store misses
PAPI_L2_LDM	Level 2 load misses
PAPI_L2_STM	Level 2 store misses
PAPI_STL_ICY	Cycles with no instruction issue
PAPI_HW_INT	Hardware interrupts
PAPI_BR_TKN	Conditional branch instructions taken
PAPI_BR_MSP	Conditional branch instructions mispredicted
PAPI_TOT_INS	Instructions completed
PAPI_FP_INS	Floating point instructions
PAPI_BR_INS	Branch instructions
PAPI_VEC_INS	Vector/SIMD instructions
PAPI_RES_STL	Cycles stalled on any resource
PAPI_TOT_CYC	Total cycles
PAPI_L1_DCH	Level 1 data cache hits
PAPI_L2_DCH	Level 2 data cache hits
PAPI_L1_DCA	Level 1 data cache accesses
PAPI_L2_DCA	Level 2 data cache accesses
PAPI_L2_DCR	Level 2 data cache reads
PAPI_L2_DCW	Level 2 data cache writes

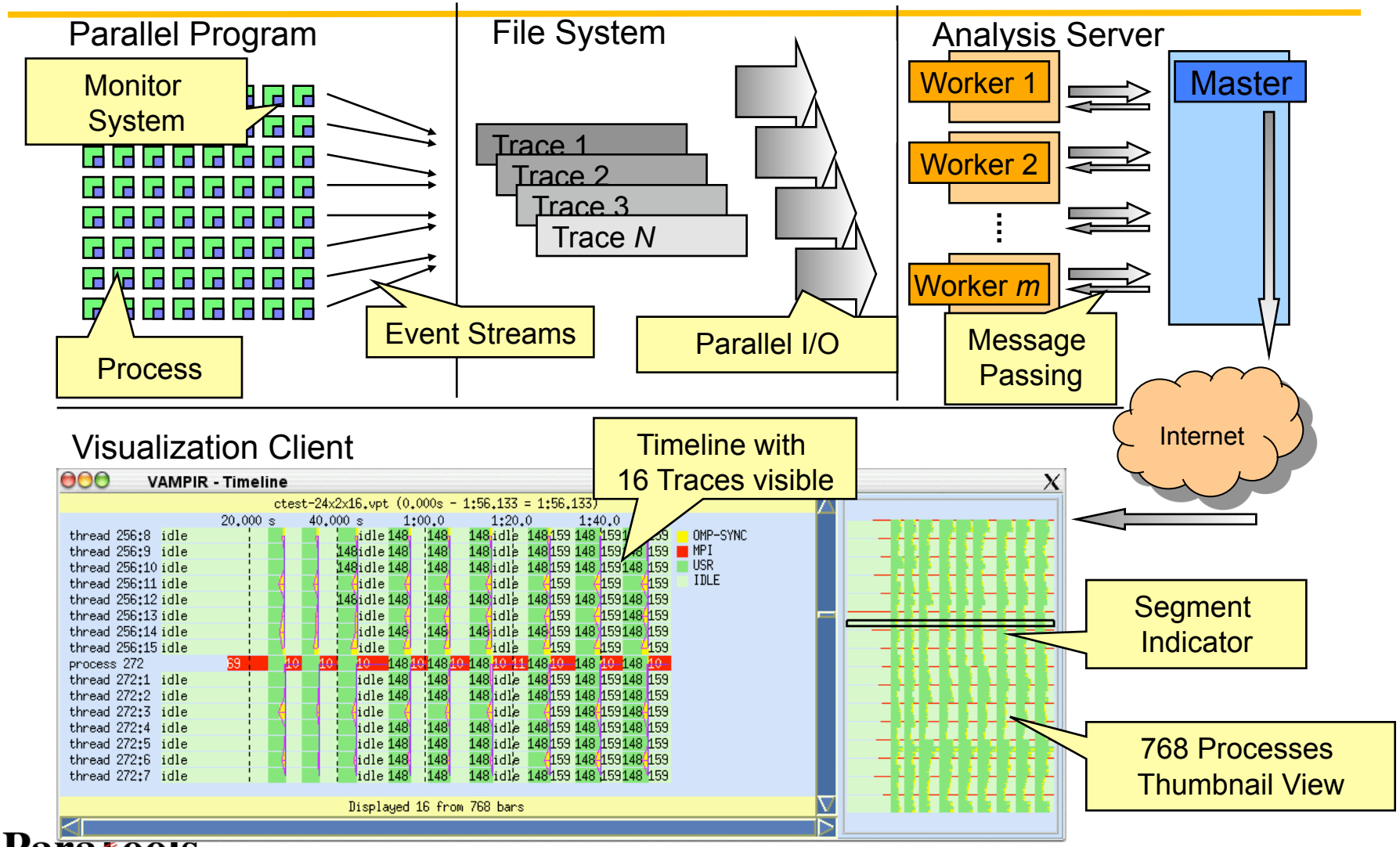
```
% /soft/apps/tau/eclipse/eclipse
```

VAMPIRTRACE & VAMPIR
INTRODUCTION AND OVERVIEW

Overview

- Introduction
- Event Trace Visualization
- Vampir & VampirServer
- The Vampir Displays
  - Timeline
  - Process Timeline with Performance Counters
  - Summary Display
  - Message Statistics
- VampirTrace
  - Instrumentation & Run-Time Measurement
- Conclusions

VampirServer Architecture

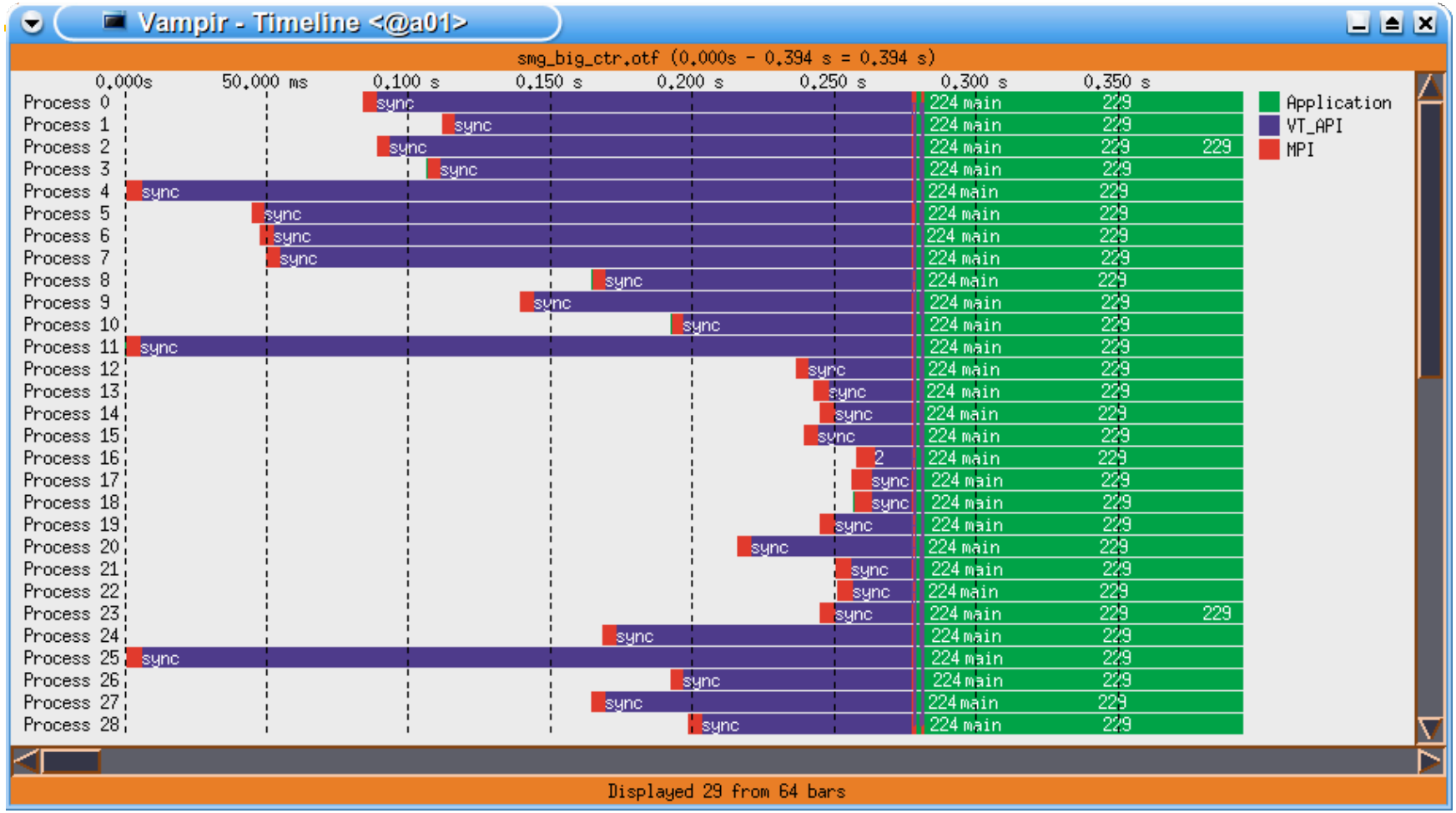


Vampir Displays

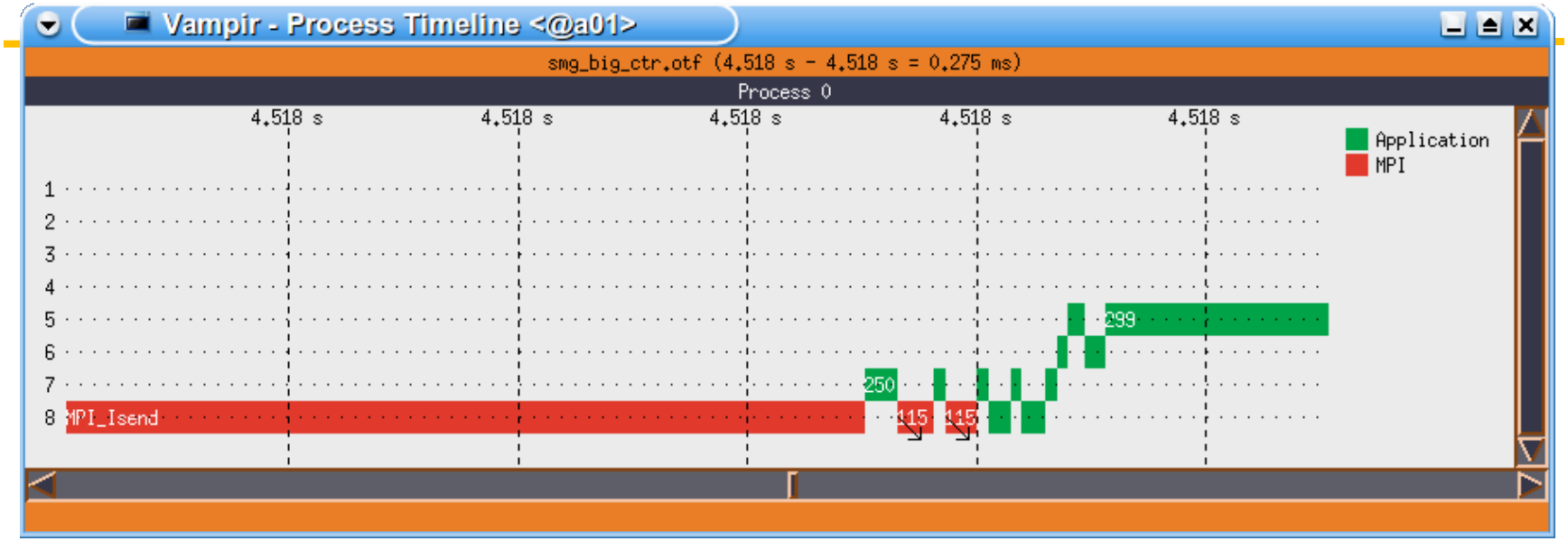
The main displays of Vampir:

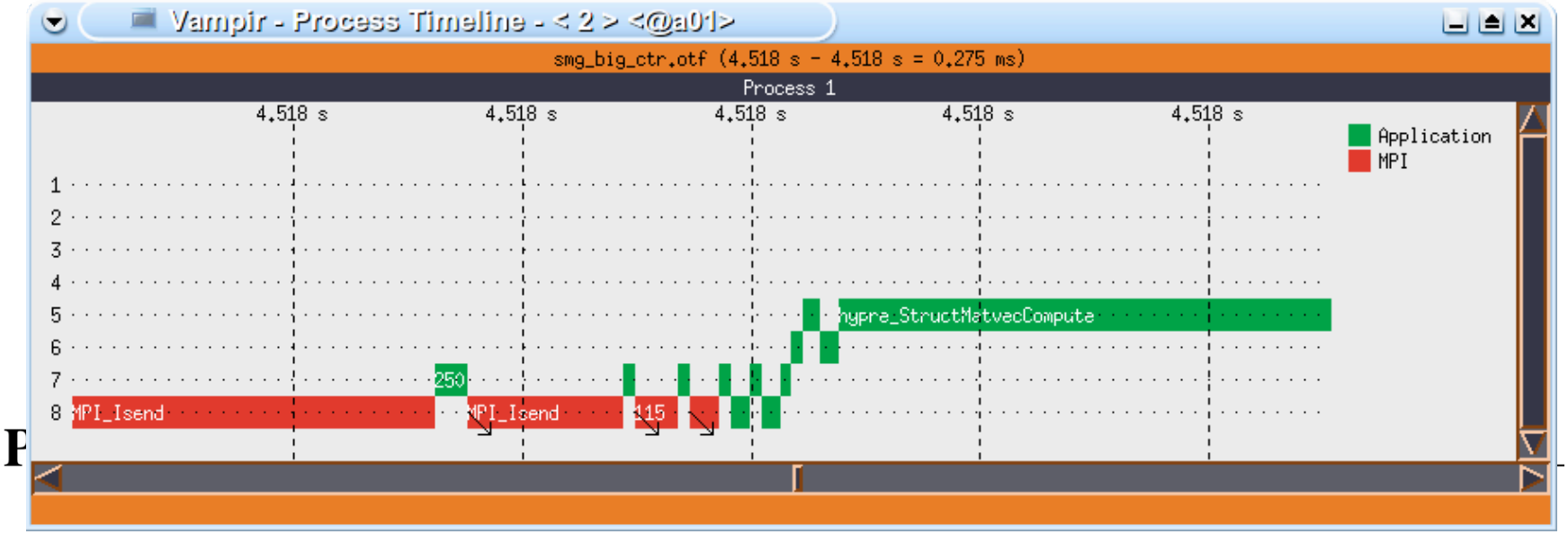
- Global Timeline
- Process Timeline w/o Counters
- Statistic Summary
- Summary Timeline
- Message Statistics
- Collective Operation Statistics
- Counter Timeline
- Call Tree

Vampir Global Timeline Display

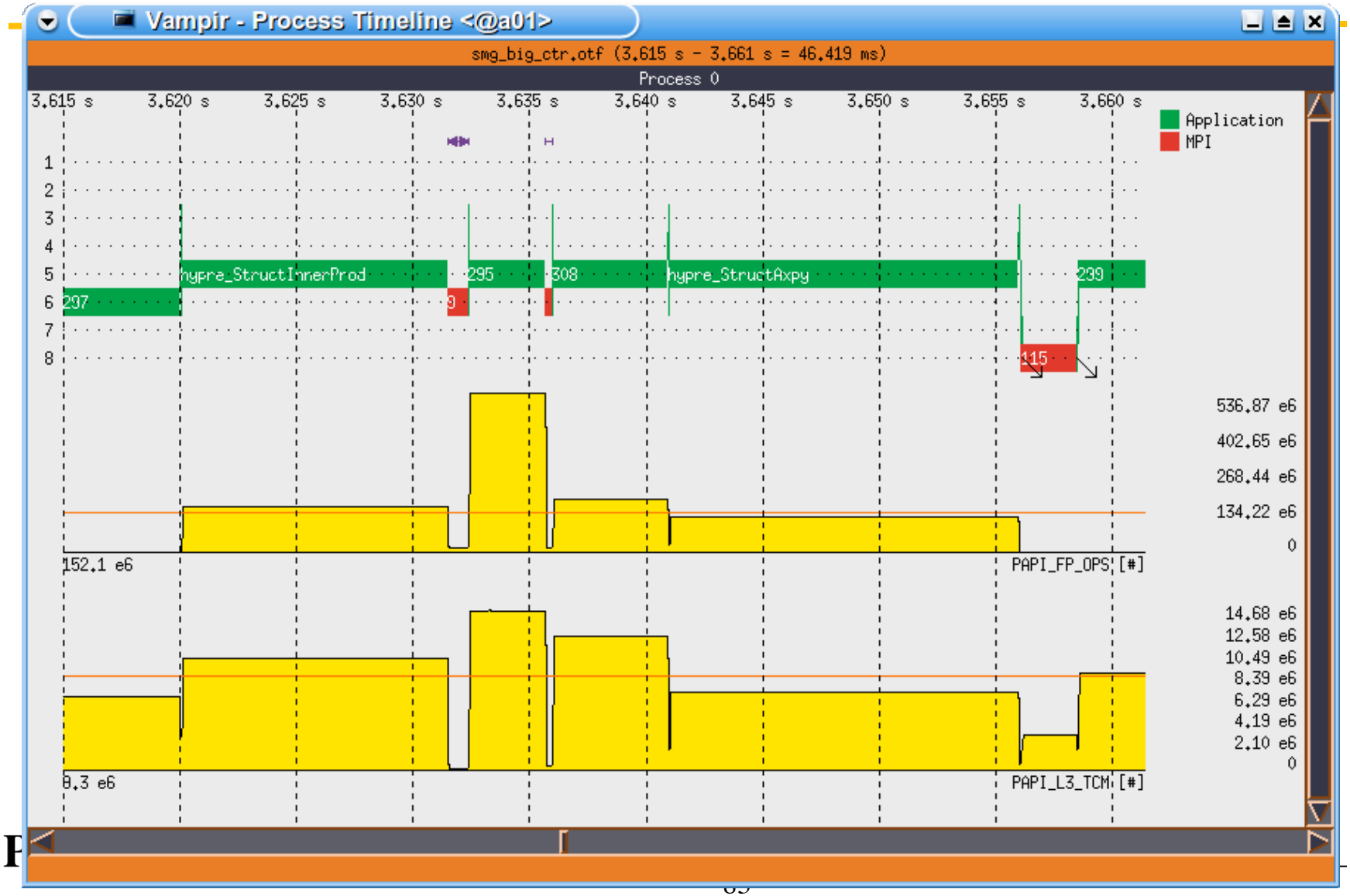


Process Timeline Display





Process Timeline with Counters



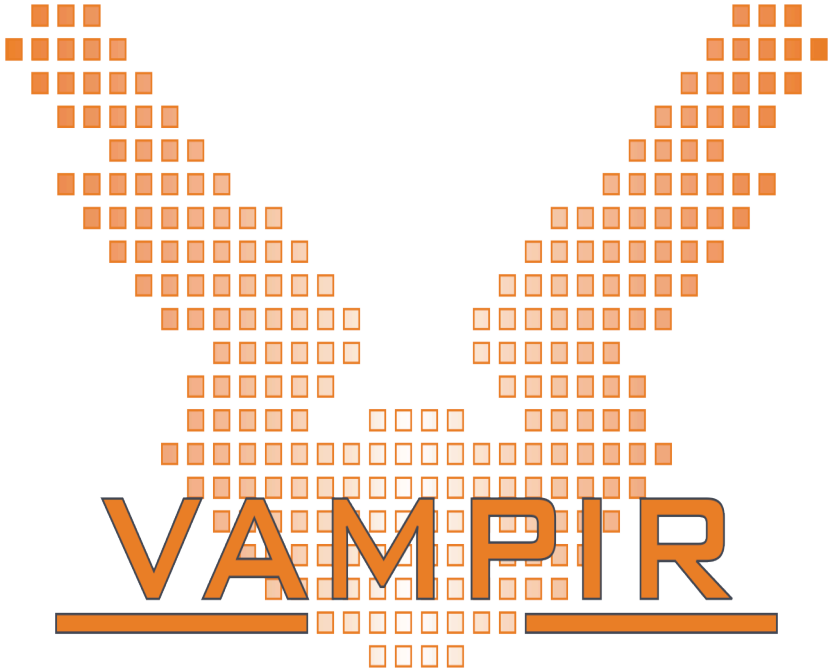
Statistic Summary Display

Vampir - Summary Chart - < 2 > <@a01>

Name	Token	Value
hypre_StructMatvecCompute	[299]	2:15.199
hypre_StructAxpy	[306]	2:10.744
hypre_StructInnerProd	[295]	1:14.087
MPI_Finalize	[82]	1:04.179
hypre_StructCopy	[297]	51.516 s
MPI_Waitall	[163]	20.135 s
hypre_StructVectorSetConstantVa	[303]	20.124 s
hypre_StructScale	[308]	15.580 s
MPI_Allreduce	[9]	13.283 s
MPI_Isend	[115]	9.010 s
hypre_StructMatrixSetBoxValues	[229]	8.455 s
sync	[2]	5.654 s
main	[184]	4.661 s
hypre_CAlloc	[186]	2.050 s
hypre_StructVectorSetBoxValues	[260]	1.827 s
hypre_StructMatrixInitializeDat	[224]	0.738 s
hypre_StructKrylovAxpy	[305]	0.668 s
MPI_Init	[108]	0.436 s
hypre_StructKrylovCopyVector	[296]	0.221 s
hypre_StructKrylovMatvec	[298]	0.215 s
hypre_PCGSolve	[293]	0.212 s
MPI_Irecv	[113]	0.190 s
hypre_BoxGetSize	[227]	0.182 s
hypre_Free	[187]	0.169 s
hypre_InitializeCommunication	[250]	0.160 s

Sorted by Value Down

All Symbols: Exclusive Times



Vampir and VampirTraces are

available at http://www.vampir.eu and

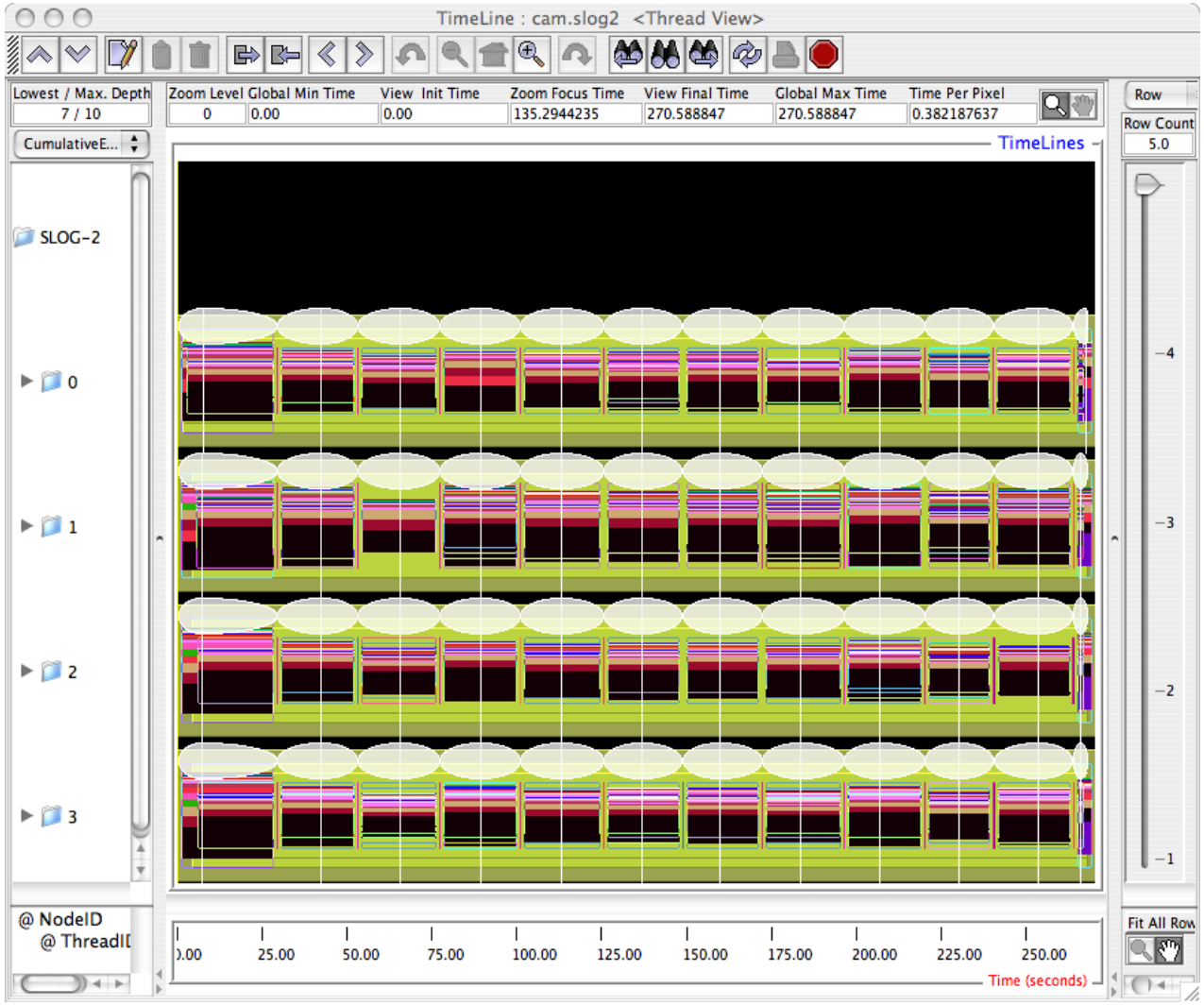
http://www.tu-dresden.de/zih/vampirtrace/ ,

get support via vampirsupport@zih.tu-dresden.de

Jumpshot

- http://www-unix.mcs.anl.gov/perfvis/software/viewers/index.htm
- Developed at Argonne National Laboratory as part of the MPICH project
  - Also works with other MPI implementations
  - Installed on IBM BG/P
  - Jumpshot is bundled with the TAU package
- Java-based tracefile visualization tool for postmortem performance analysis of MPI programs
- Latest version is Jumpshot-4 for SLOG-2 format
  - Scalable level of detail support
  - Timeline and histogram views
  - Scrolling and zooming
  - Search/scan facility

Jumpshot


TimeLine : cam.slog2 <Thread View>
Lowest / Max. Depth
7 / 10
Zoom Level
0
Global Min Time
0.00
View Init Time
0.00
Zoom Focus Time
135.2944235
View Final Time
270.588847
Global Max Time
270.588847
Time Per Pixel
0.382187637
Row
Row Count
5.0
CumulativeE...
TimeLines
SLOG-2
0
1
2
3
@ NodeID
@ ThreadID
25.00
50.00
75.00
100.00
125.00
150.00
175.00
200.00
225.00
250.00
Time (seconds)
Fit All Row

Labs!

Lab Instructions

```
Get workshop.tar.gz using:
% wget http://www.paratools.com/anl10/workshop.tar.gz
Or
% cp /soft/apps/tau/workshop.tar.gz .
% tar zxf workshop.tar.gz
source /soft/apps/tau/tau.bashrc
OR
source /soft/apps/tau/tau.bashrc
    in your .login file and then follow the instructions in
      the README file. These files contain ANL specific
      location information.Or
    % soft add  +tau-latest
For LiveDVD, see ~/workshop-point/README and follow.
```

Lab Instructions

To profile a code using TAU:

1. Change the compiler name to tau_cxx.sh, tau_f90.sh, tau_cc.sh:
   F90 = tau_f90.sh
2. Choose TAU stub makefile
   % export TAU_MAKEFILE=
   /soft/apps/tau/tau_latest/bgp/lib/Makefile.tau-[options]
   % soft add +tau_latest
   % make F90=tau_f90.sh
3. If stub makefile has -papi in its name, set the TAU_METRICS environment variable:
   % export TAU_METRICS=TIME:PAPI_L2_DCM:PAPI_TOT_CYC...
4. Build and run workshop examples, then run pprof/paraprof

Acknowledgements

- Department of Energy
  - Office of Science
  - Argonne National Laboratory
  - ORNL
  - NNSA/ASC Trilabs (SNL, LLNL, LANL)
- HPCMP DoD PET Program
- National Science Foundation
- University of Tennessee
  - David Cronk, Shirley Moore
  - Daniel Terpstra
- University of Oregon
  - Allen D. Malony, A. Morris, K. Huck, W. Spear, S. Biersdorff
- TU Dresden
  - Holger Brunst, Andreas Knupfer
  - Wolfgang Nagel
- Research Centre Juelich, Germany
  - Bernd Mohr
  - Felix Wolf