TAU Performance System®

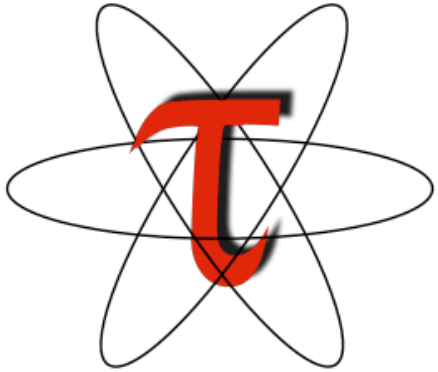
Leap to Petascale Workshop, ALCF

Room A, ANL Guest House, May 27-29, 2009

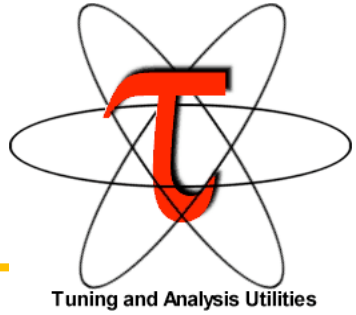
Sameer Shende and the TAU team

sameer@paratools.com

TAU: A Quick Reference



TAU Performance System

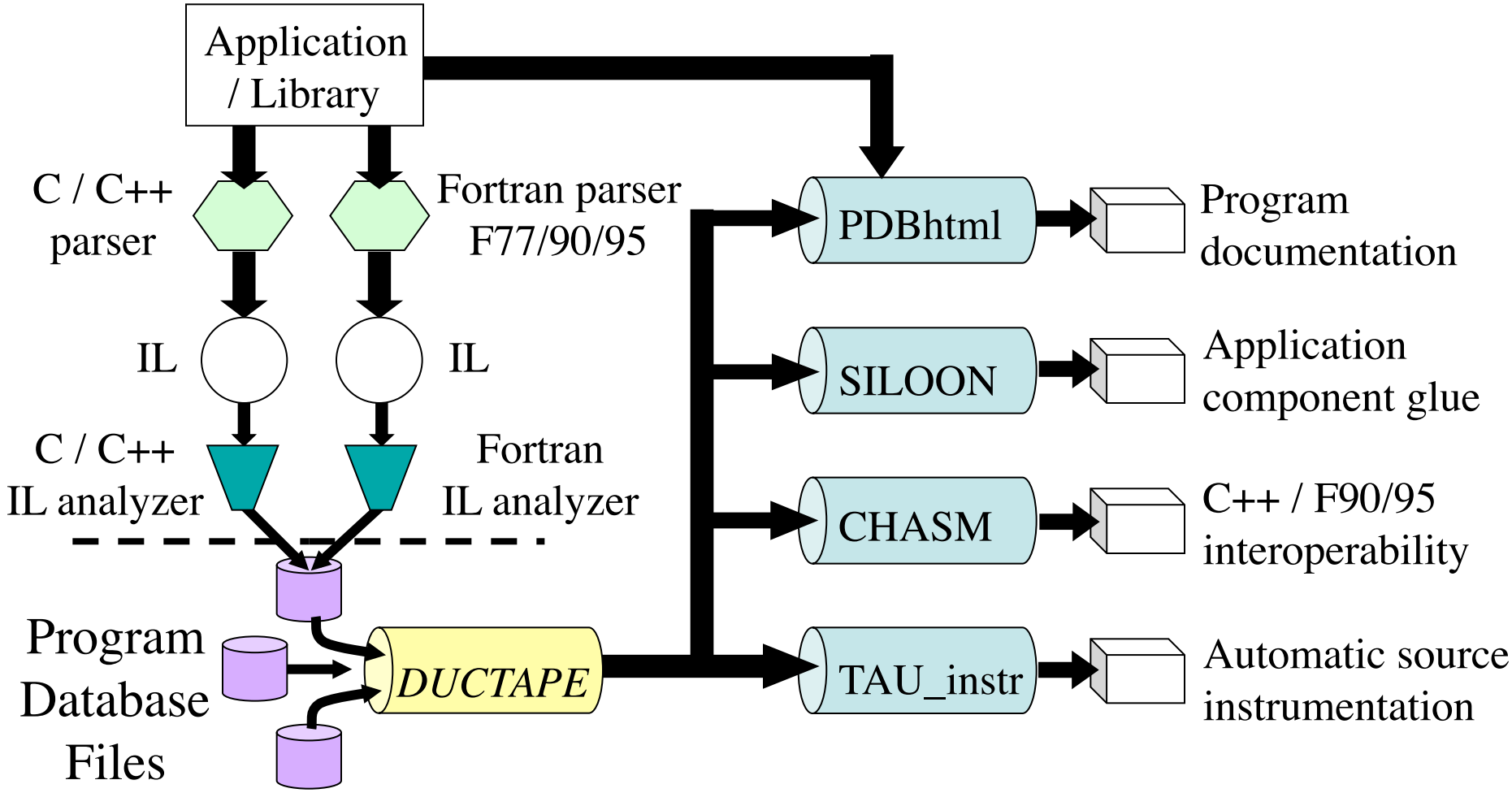


- http://tau.uoregon.edu/
- Multi-level performance instrumentation
  - Multi-language automatic source instrumentation
- Flexible and configurable performance measurement
- Widely-ported parallel performance profiling system
  - Computer system architectures and operating systems
  - Different programming languages and compilers
- Support for multiple parallel programming paradigms
  - Multi-threading, message passing, mixed-mode, hybrid
- Integration in complex software, systems, applications
- Partners: ANL, ORNL, LBL, LLNL, LANL, FZJ, UTK, TUD, PSC, UIUC

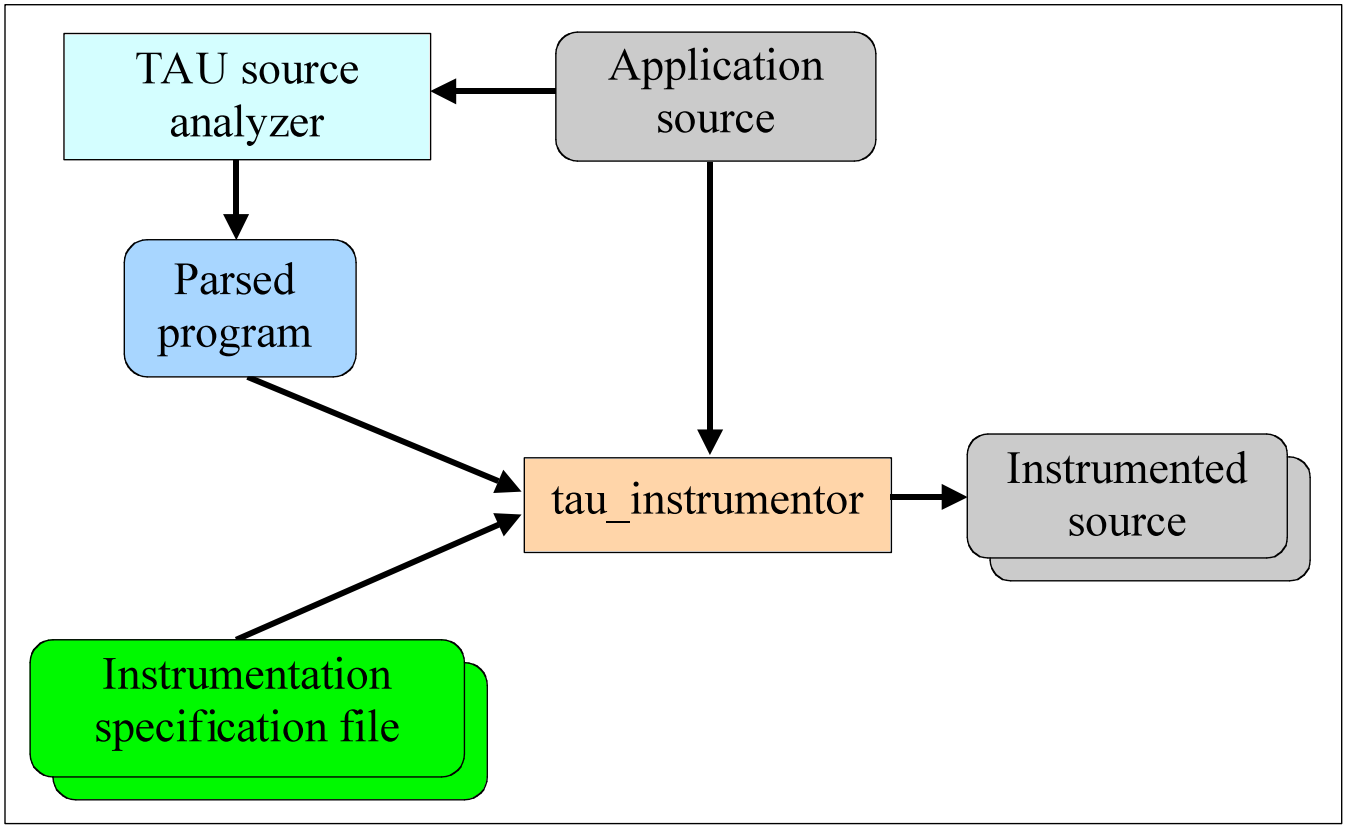
What is TAU?

- TAU is a performance evaluation tool
- It supports both parallel profiling and tracing
- Profiling shows you how much (total) time was spent in each routine (event)
- Tracing shows you *when* events take place in each process along a timeline
- Profiling and tracing can measure time as well as hardware performance counters
- TAU uses a package called *PDT* for automatic instrumentation of the source code
- With PDT, TAU can instrument routine, loop, phase, I/O, and memory
- TAU can also use your compiler to insert the instrumentation at routine boundaries
- TAU can *throttle* the insignificant lightweight routines at runtime to reduce perturbation. It can also *subtract* the timer overhead at runtime to compensate.
- TAU runs on all HPC platforms and it is free (BSD style license)
- TAU has instrumentation, measurement and analysis tools
  - ParaProf is TAU's 3D profile browser, PerfDMF is the TAU database tool
- To use TAU, all you need to do is set a couple of environment variables and substitute the name of your compiler with a TAU shell script (e.g., tau_f90.sh)

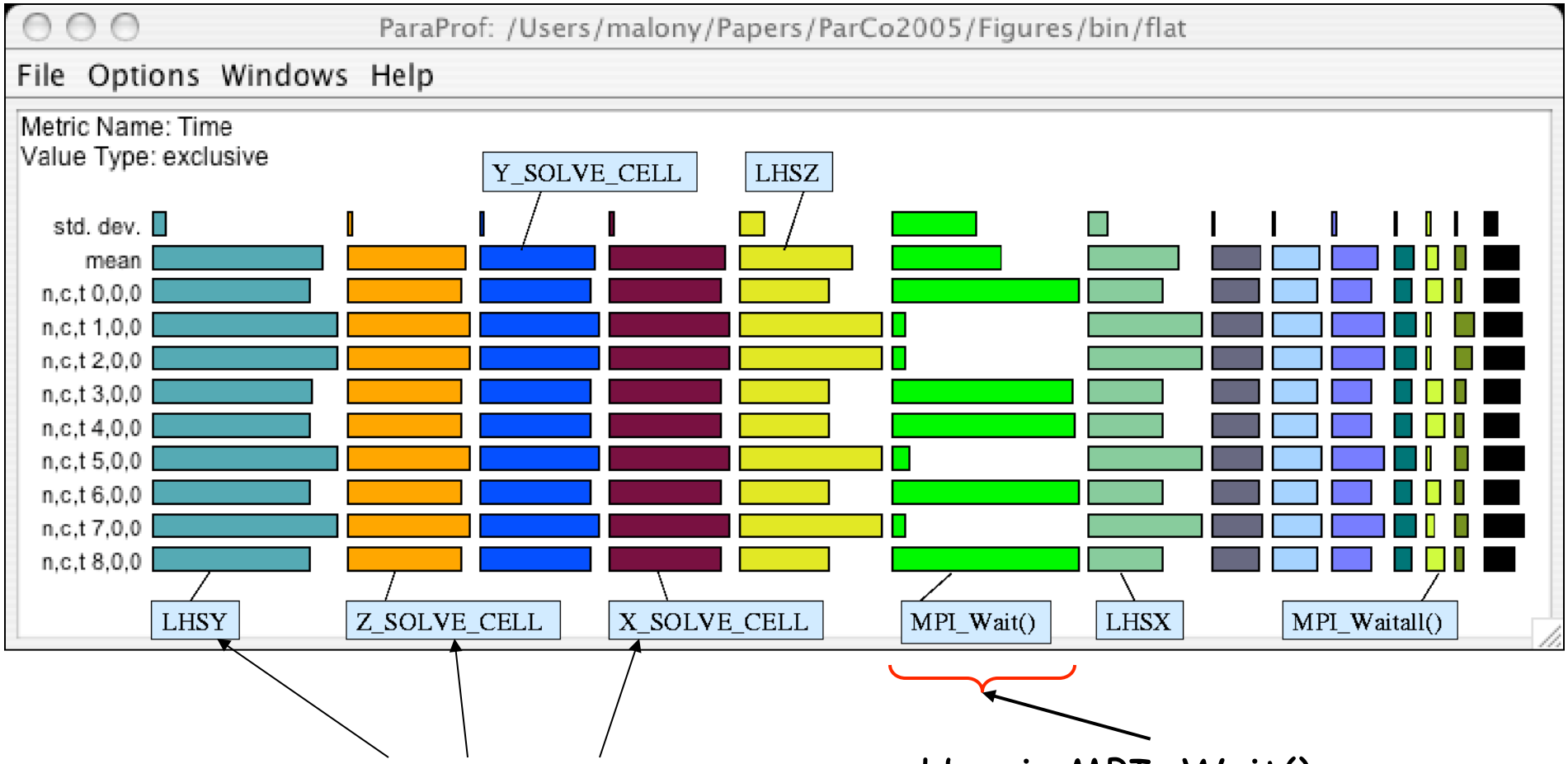
Program Database Toolkit (PDT)



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



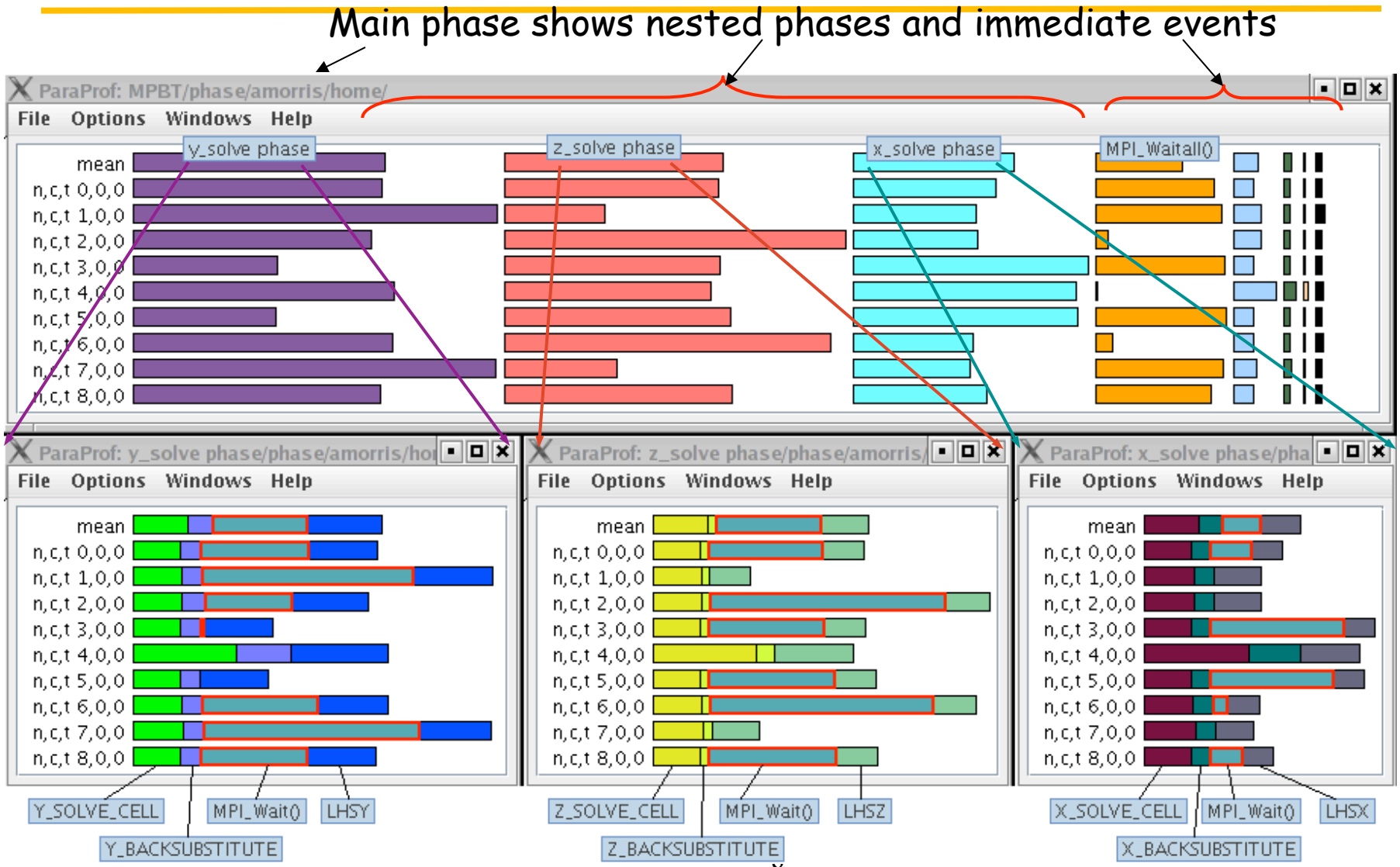
ParaProf – Flat Profile (NAS BT)



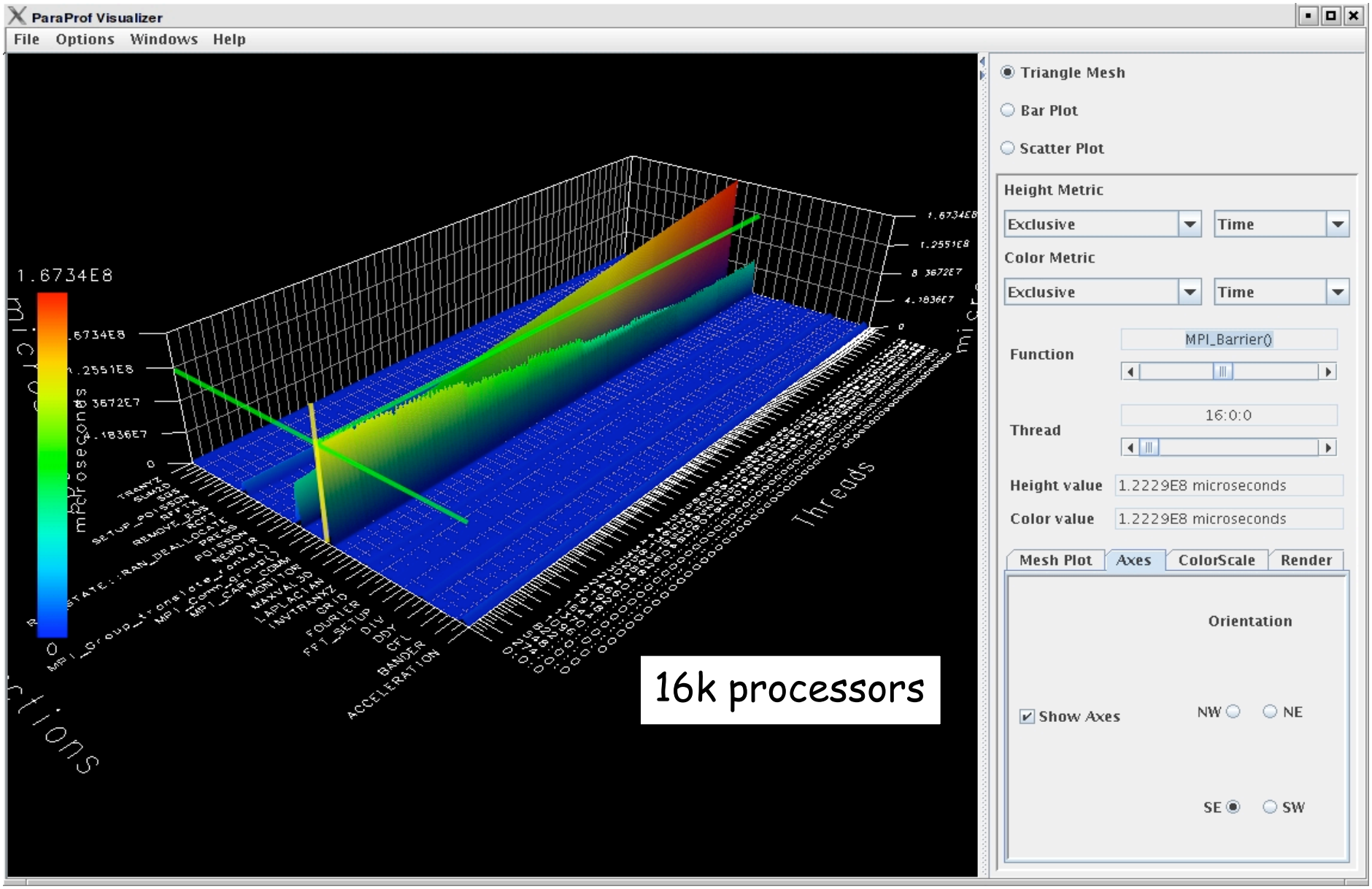
Application routine names reflect phase semantics

How is MPI_Wait() distributed relative to solver direction?

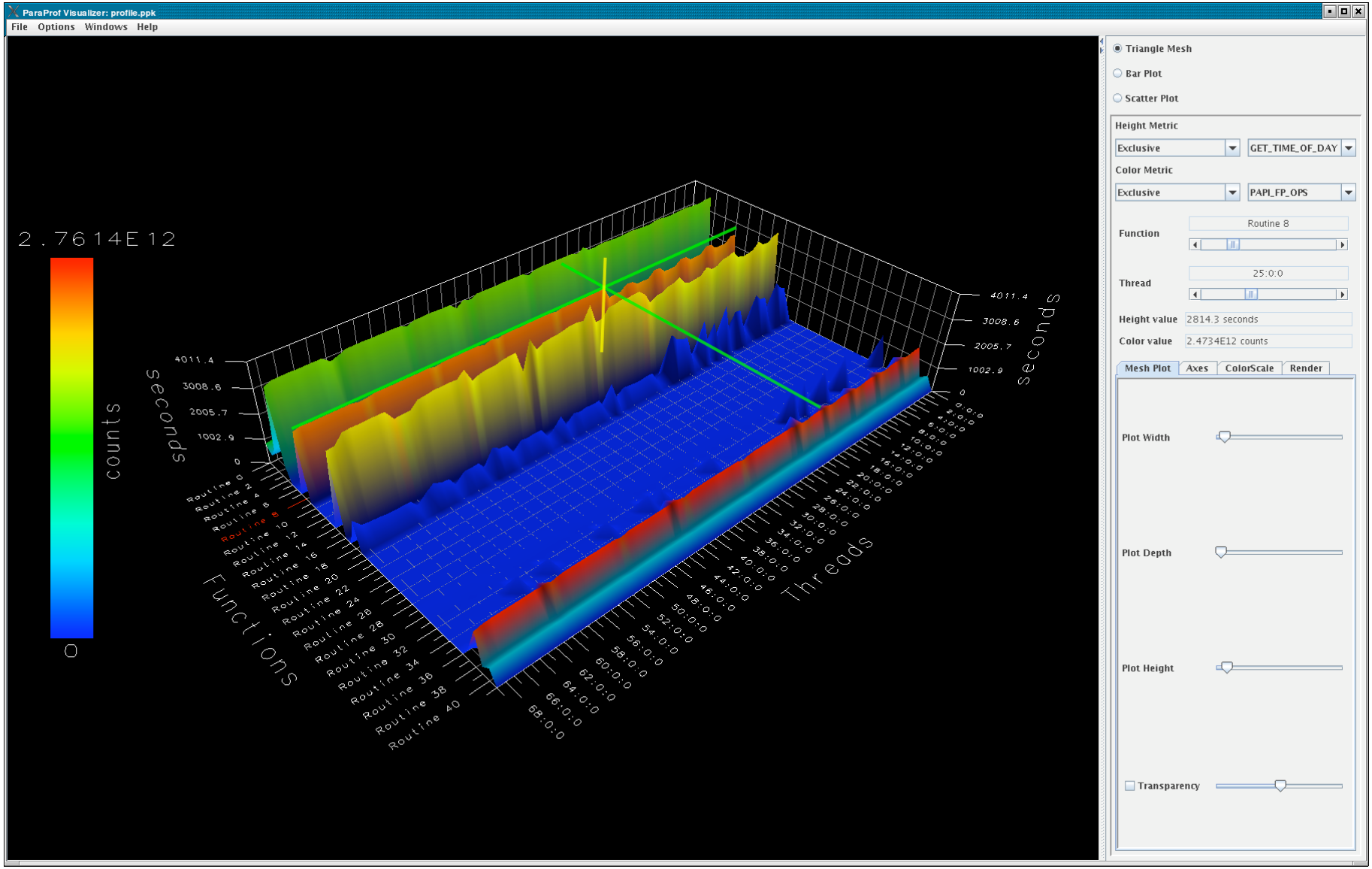
ParaProf – Phase Profile (NAS BT)



ParaProf – 3D Full Profile (Miranda, LLNL)

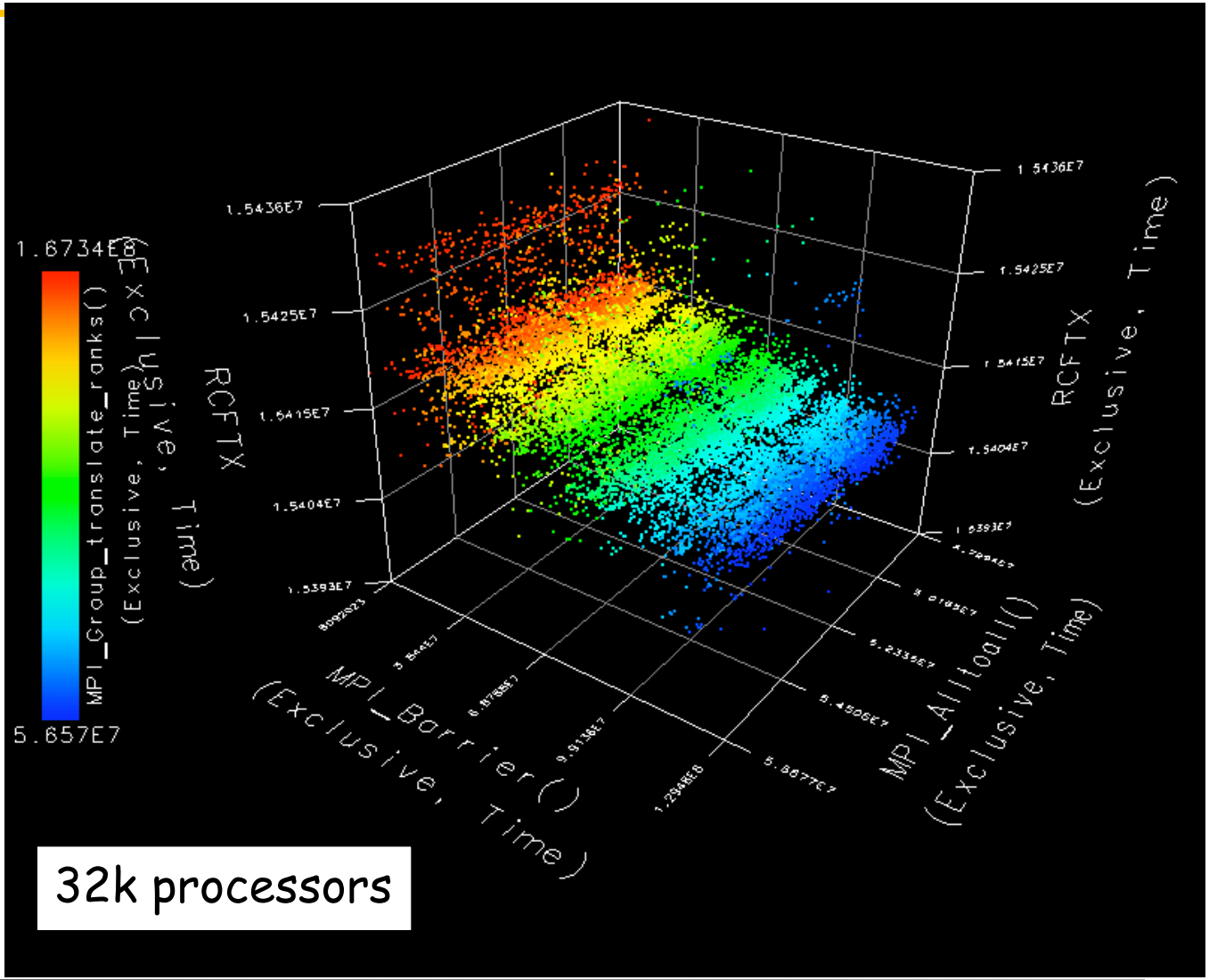


ParaProf – 3D Full Profile



ParaProf – 3D Scatterplot

- Each point is a “thread” of execution
- A total of four metrics shown in relation
- 3D profile visualization library
  - JOGL



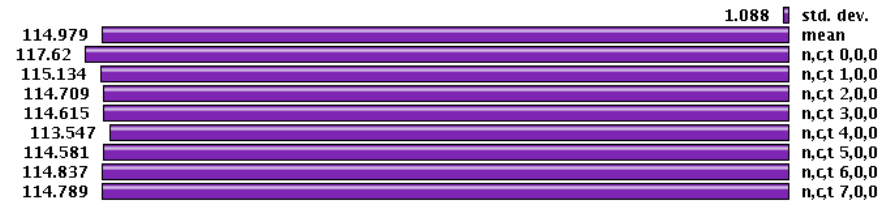
32k processors

ParaProf's Source Browser: Loop Level Instrumentation

TAU: ParaProf: Function Data Window: s3d_callpath_papi.ppk

File Options Windows Help

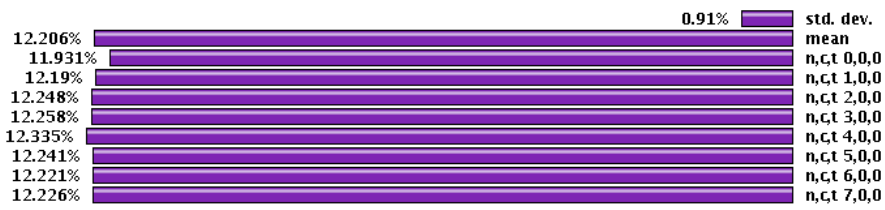
Name: Loop: TRANSPORT_M::COMPUTESPECIESDIFFFLUX [{mixavg_transport_m.pp.f90} {630,5}-{656,19}]
Metric Name: PAPI_FP_INS / GET_TIME_OF_DAY
Value: Exclusive
Units: Derived metric shown in microseconds format



TAU: ParaProf: Function Data Window: s3d_callpath_papi.ppk

File Options Windows Help

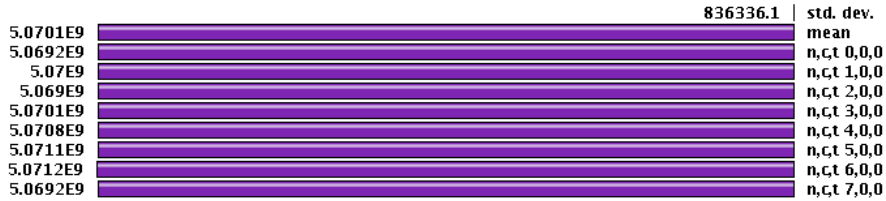
Name: Loop: TRANSPORT_M::COMPUTESPECIESDIFFFLUX [{mixavg_transport_m.pp.f90} {630,5}-{656,19}]
Metric Name: GET_TIME_OF_DAY
Value: Exclusive percent



TAU: ParaProf: Function Data Window: s3d_callpath_papi.ppk

File Options Windows Help

Name: Loop: TRANSPORT_M::COMPUTESPECIESDIFFFLUX [{mixavg_transport_m.pp.f90} {630,5}-{656,19}]
Metric Name: PAPI_L1_DCM
Value: Exclusive
Units: counts

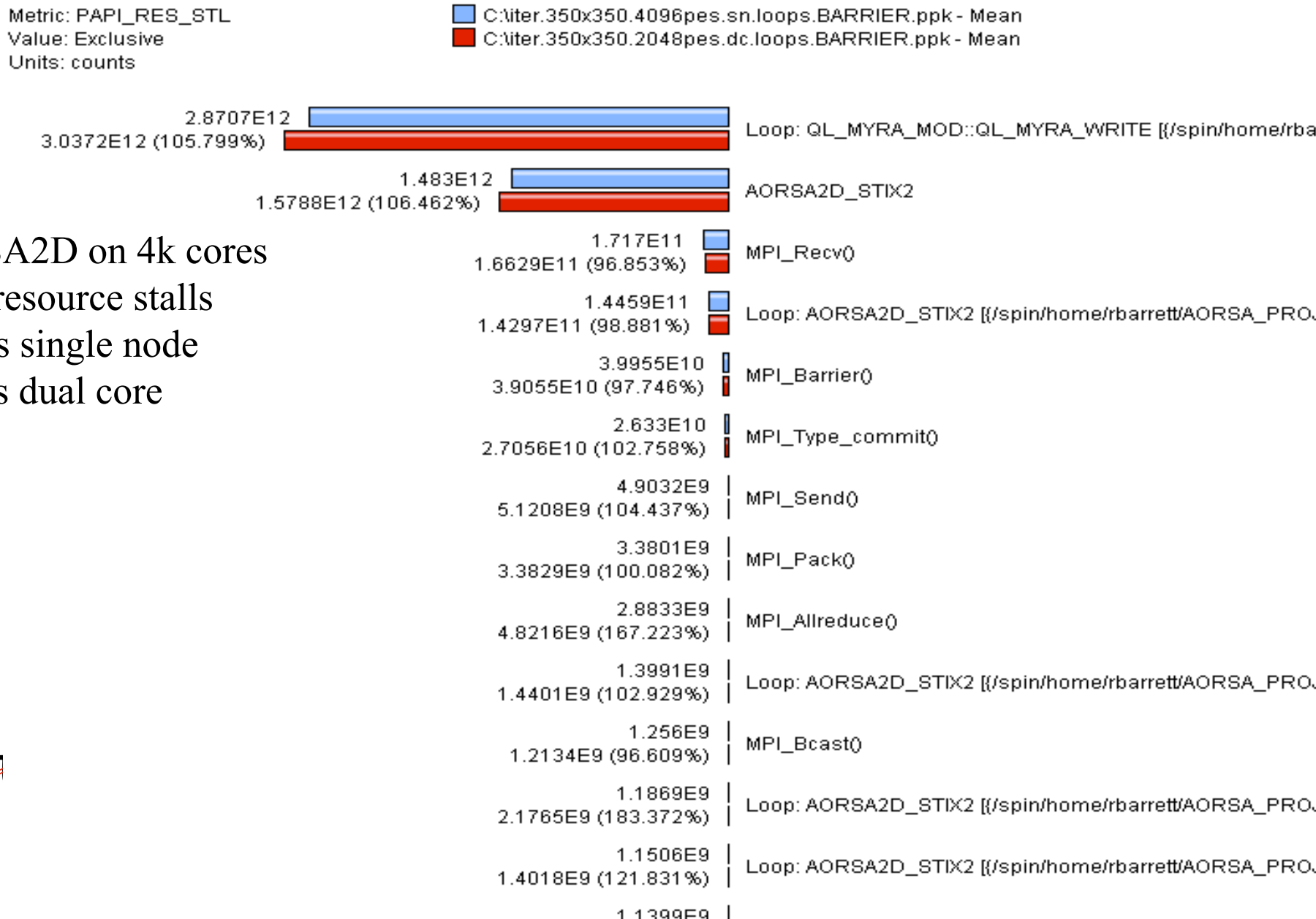


TAU: ParaProf: Source Browser: /mnt/epsilon/Users/sameer/rs/taudata/s3d/harness/flat/papi8

File Help

```
606      grad_mixMW(:,:,:,m) = grad_mixMW(:,:,:,m)*avmolwt(:,:,:)
607     end do
608
609     ! compute grad_P
610     if (baro_switch) then
611        allocate(grad_P(nx,ny,nz,3))
612        grad_P = 0.0
613        if (vary_in_x == 1) then
614           call derivative_x( nx,ny,nz, Press, grad_P(:,:,:,1), scale_1x, 1 )
615        endif
616        if (vary_in_y == 1) then
617           call derivative_y( nx,ny,nz, Press, grad_P(:,:,:,2), scale_1y, 1 )
618        endif
619        if (vary_in_z == 1) then
620           call derivative_z( nx,ny,nz, Press, grad_P(:,:,:,3), scale_1z, 1 )
621        endif
622     endif
623
624     ! Changed by Ramanan - 01/24/05
625     ! Ds_mixavg is now \rho*D
626     !
627     !grad_P/press and avmolwt*grad_T/Temp can be optimized by division before the loop.
628     ! compute diffusive flux for species n in direction m.
629     diffFlux(:,:,:,n_spec,:) = 0.0
630     DIRECTION: do m=1,3
631        SPECIES: do n=1,n_spec-1
632
633           if (baro_switch) then
634              ! driving force includes gradient in mole fraction and baro-diffusion:
635              diffFlux(:,:,:,n,m) = - Ds_mixavg(:,:,:,n) * ( grad_Ys(:,:,:,n,m)  &
636                                    + Ys(:,:,:,n) * ( grad_mixMW(:,:,:,m)    &
637                                    + (1 - molwt(n)*avmolwt) * grad_P(:,:,:,m)/Press))
638           else
639              ! driving force is just the gradient in mole fraction:
640              diffFlux(:,:,:,n,m) = - Ds_mixavg(:,:,:,n) * ( grad_Ys(:,:,:,n,m)  &
641                                    + Ys(:,:,:,n) * grad_mixMW(:,:,:,m) )
642           endif
643
644           ! Add thermal diffusion:
645           if (thermDiff_switch) then
646              diffFlux(:,:,:,n,m) = diffFlux(:,:,:,n,m)    &
647                   - Ds_mixavg(:,:,:,n) * Rs_therm_diff(:,:,:,n) * molwt(n) &
648                   * avmolwt * grad_T(:,:,:,m) / Temp
649           endif
650
651           ! compute contribution to nth species diffusive flux
652           ! this will ensure that the sum of the diffusive fluxes is zero.
653           diffFlux(:,:,:,n_spec,m) = diffFlux(:,:,:,n_spec,m) - diffFlux(:,:,:,n,m)
654
655        enddo SPECIES
656     enddo DIRECTION
657
658     if (baro_switch) then
659        deallocate(grad_P)
660     endif
661
662     return
663   end subroutine computeSpeciesDiffFlux
664
665 !!$=============================================================================
666
667
668   subroutine computeStressTensor( grad_u)
```

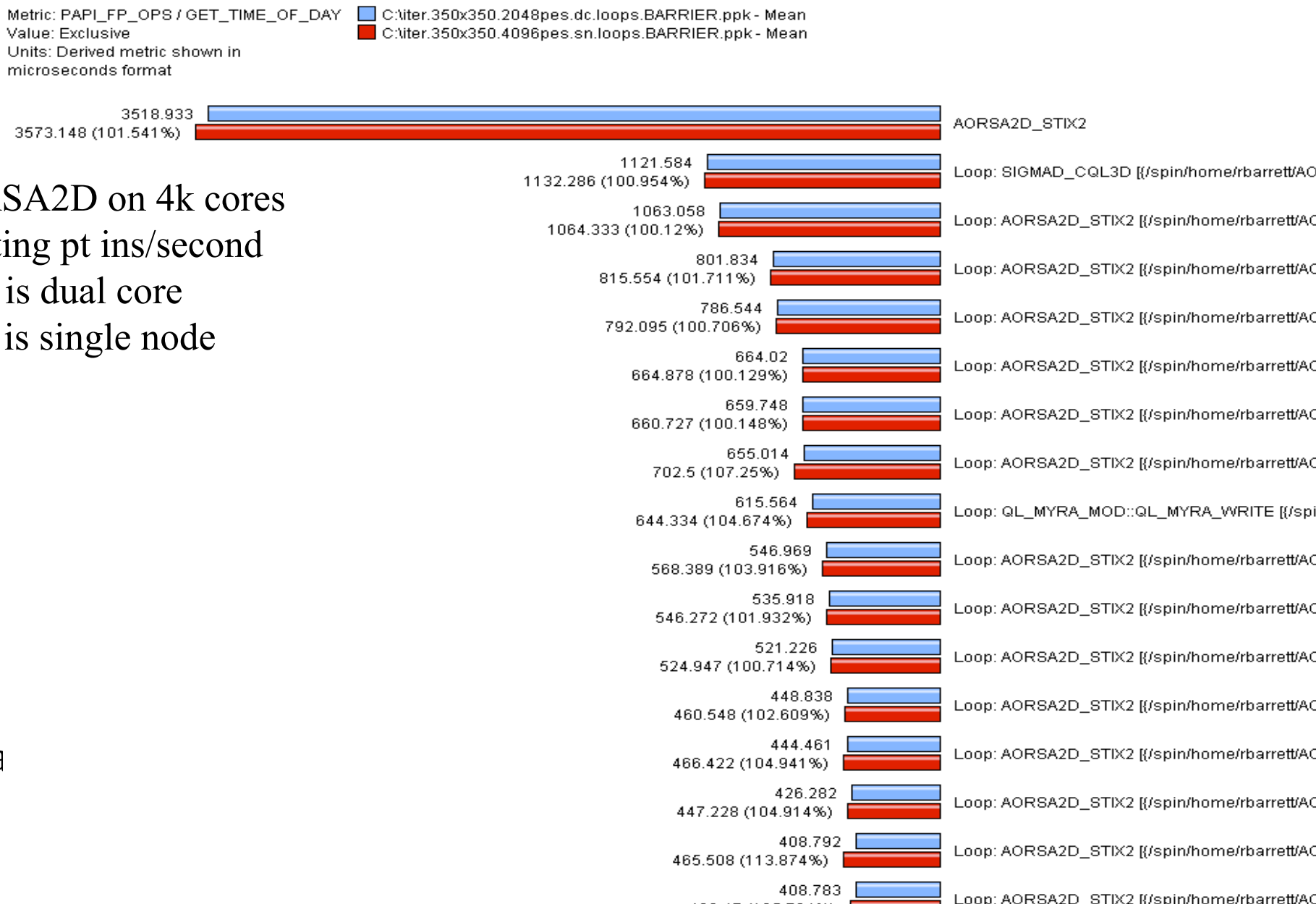
Comparing Effects of MultiCore Processors



- AORSA2D on 4k cores
- PAPI resource stalls
- Blue is single node
- Red is dual core

Para

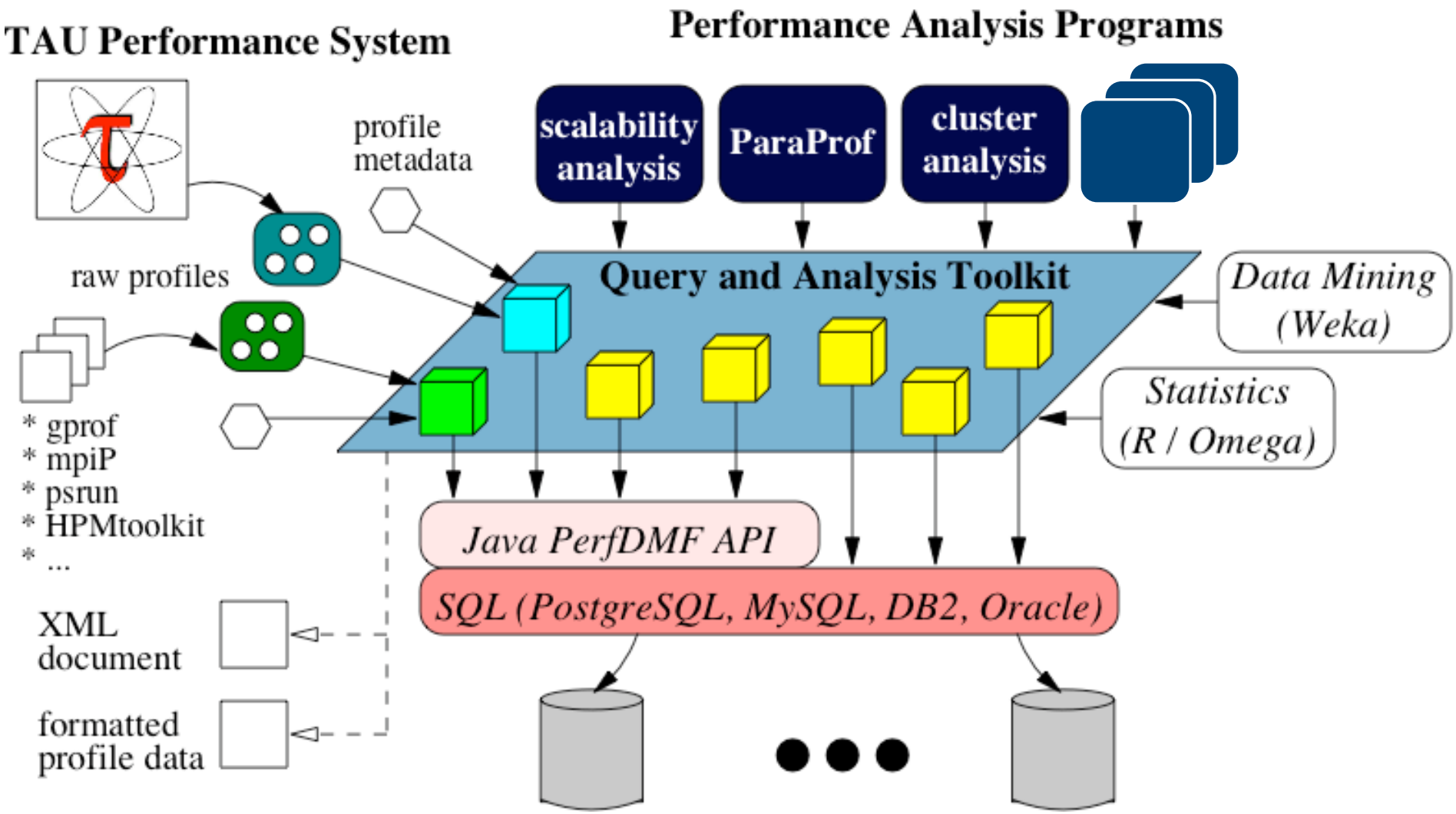
Comparing FLOPS: MultiCore Processors



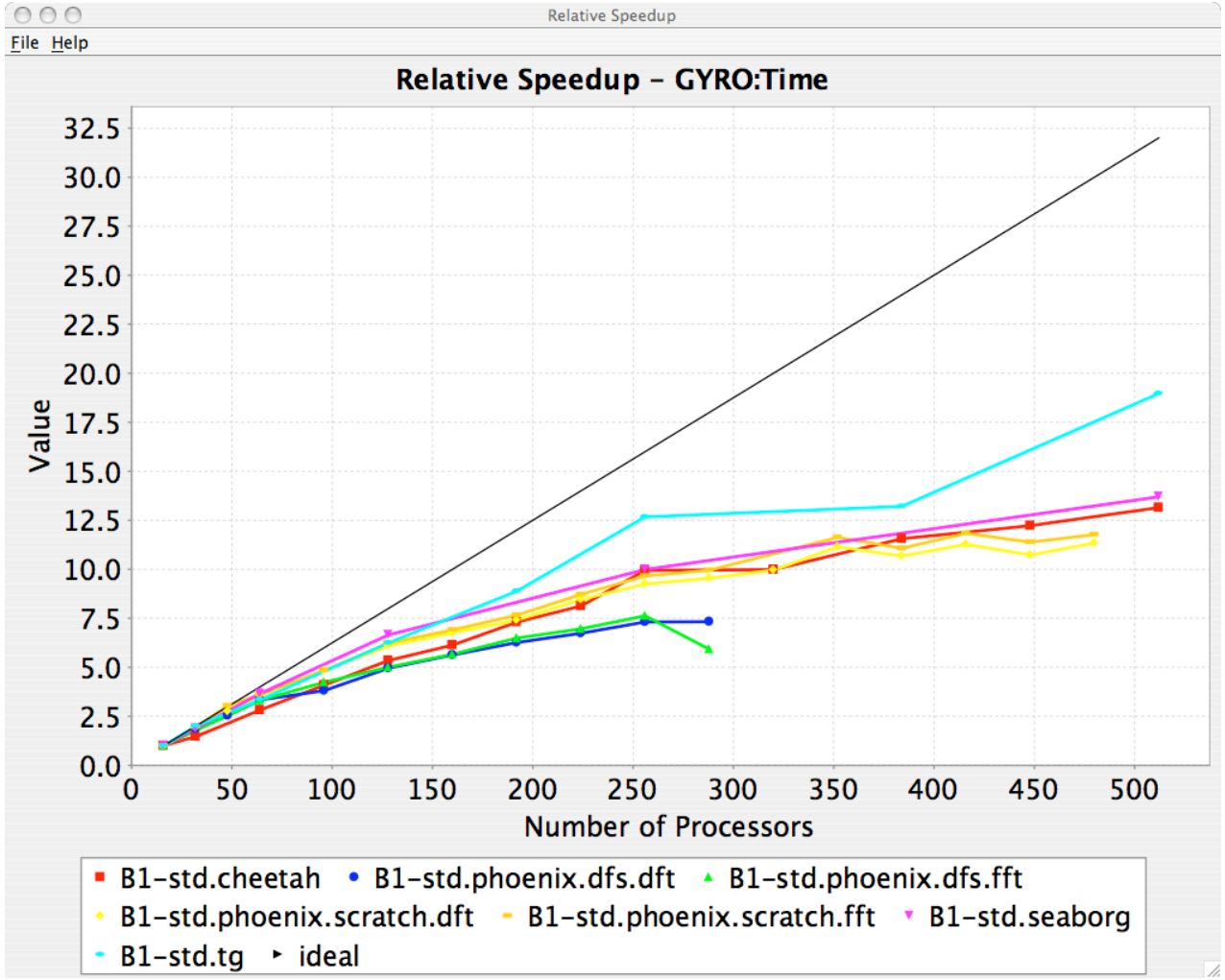
- AORSA2D on 4k cores
- Floating pt ins/second
- Blue is dual core
- Red is single node

Para

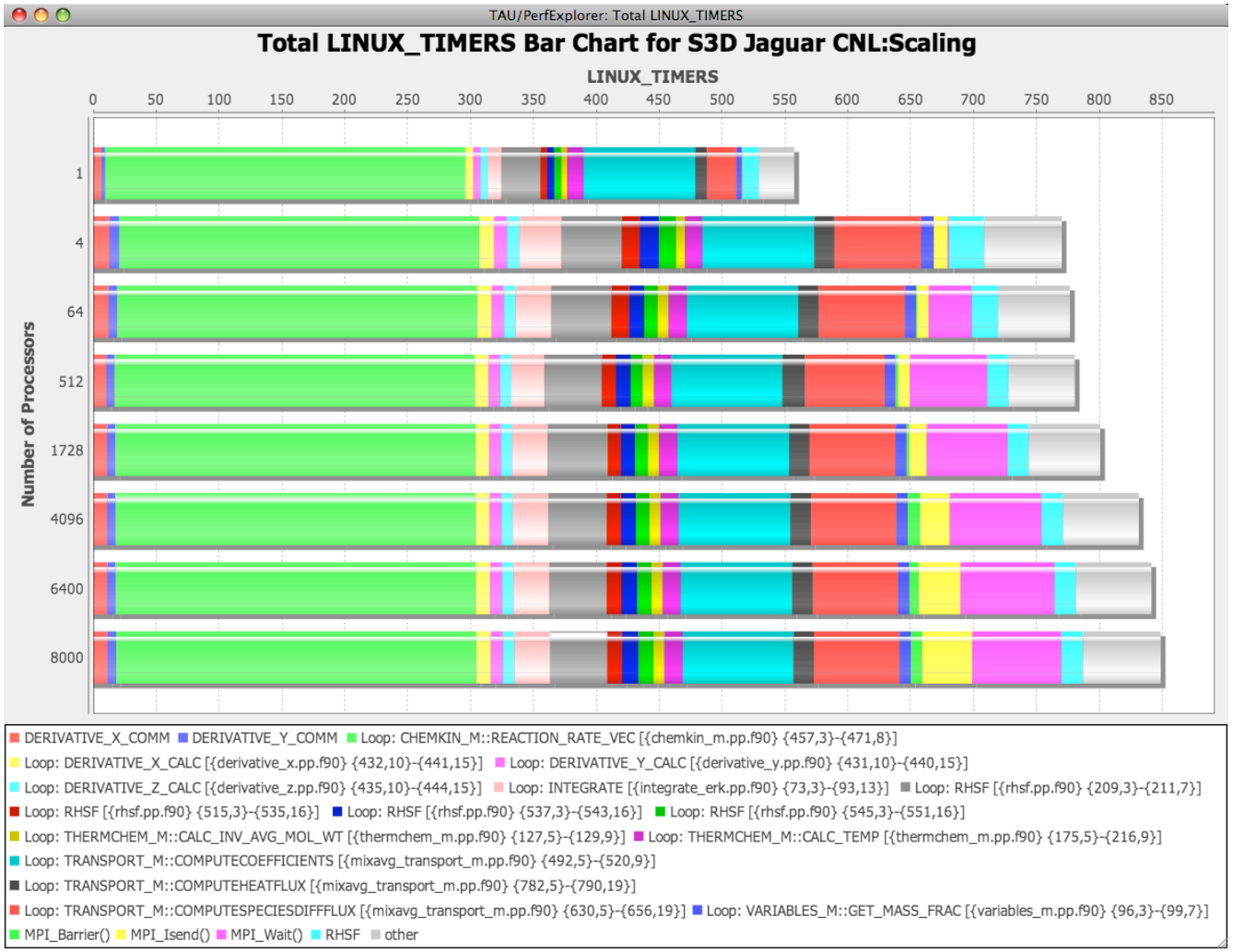
PerfDMF: Performance Data Mgmt. Framework



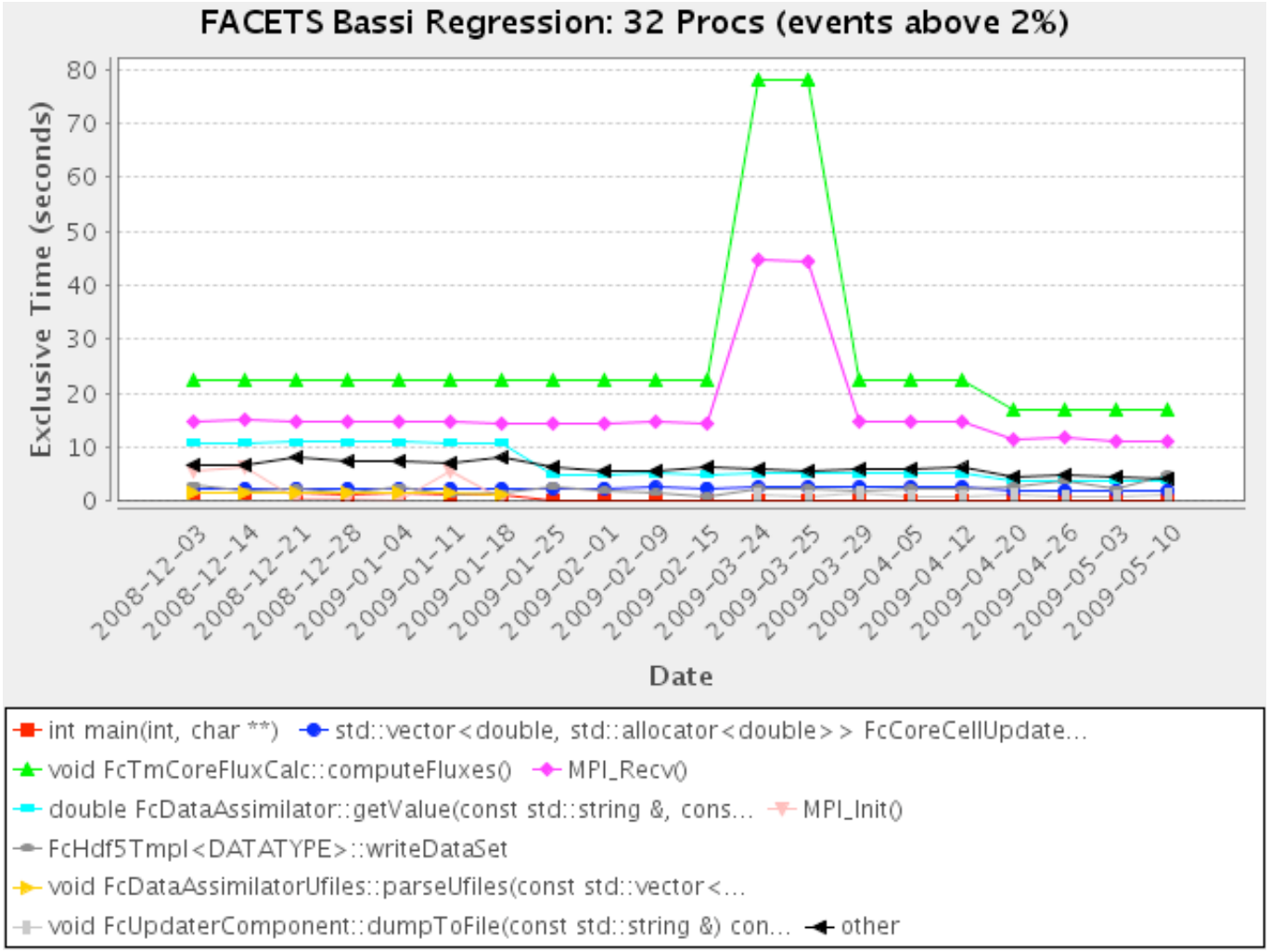
PerfExplorer: Comparing Relative Speedup on Different Architectures



Usage Scenarios: Evaluate Scalability



Performance Regression Testing



Profiling

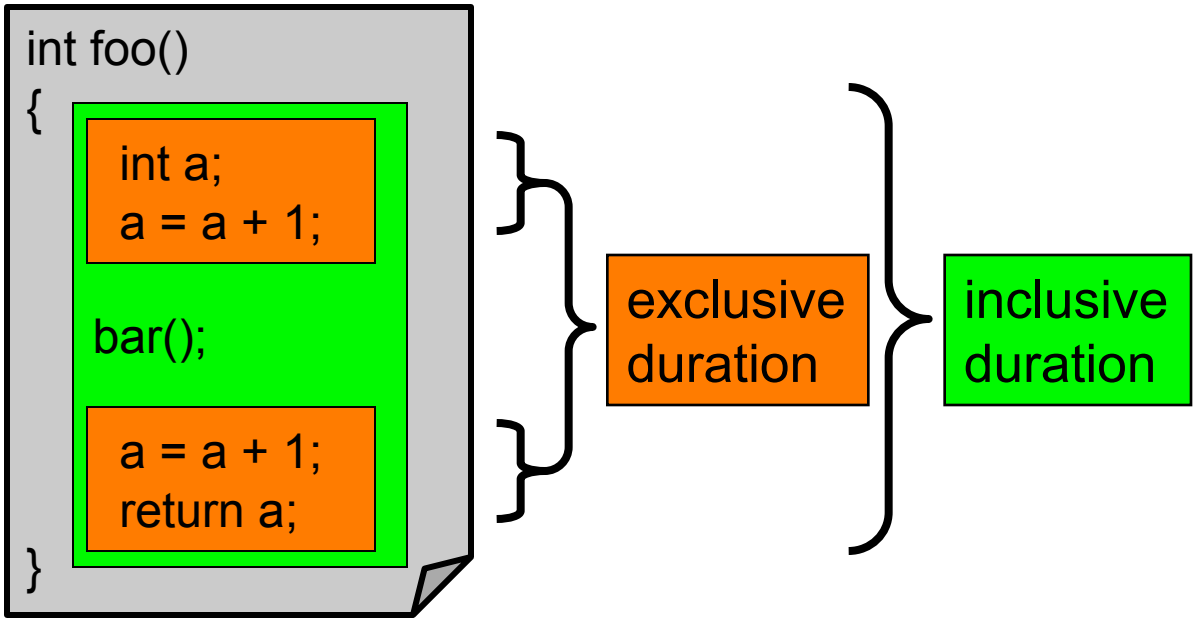
- Recording of aggregated information
  - Counts, time, …
- … about program and system entities
  - Functions, loops, basic blocks, …
  - Processes, threads
- Methods
  - Event-based sampling (indirect, statistical)
    - Open|SpeedShop, PerfSuite, HPCToolkit, gprof,…
  - Direct measurement (deterministic)
    - TAU, VampirTrace, Scalasca,…

Direct Observation: Events

- Event types
  - Interval events (begin/end events)
    - measures performance between begin and end
    - metrics monotonically increase
  - Atomic events
    - used to capture performance data state
- Code events
  - Routines, classes, templates
  - Statement-level blocks, loops
- User-defined events
  - Specified by the user
- Abstract mapping events

Inclusive and Exclusive Profiles

- Performance with respect to code regions
- Exclusive measurements for region only
- Inclusive measurements includes child regions



Terminology – Example

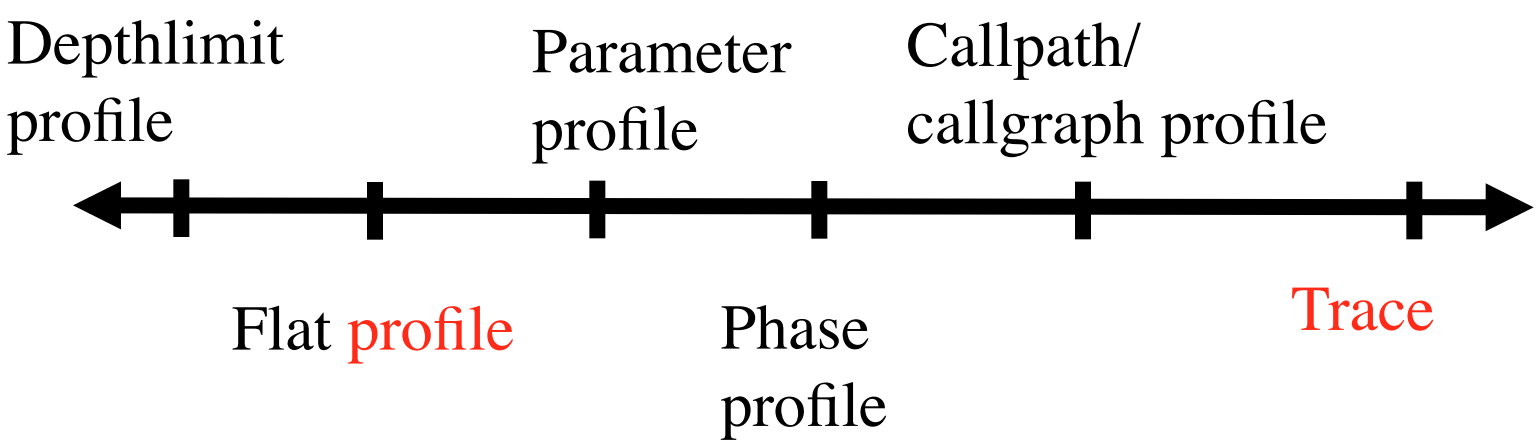
- For routine “int main( )”:
- Exclusive time
  - 100-20-50-20=10 secs
- Inclusive time
  - 100 secs
- Calls
  - 1 call
- Subrs (no. of child routines called)
  - 3
- Inclusive time/call
  - 100secs

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

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

  /* other work */
}

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

Flat and Callpath Profiles

- Static call graph
  - Shows all parent-child calling relationships in a program
- Dynamic call graph
  - Reflects actual execution time calling relationships
- Flat profile
  - Performance metrics for when event is active
  - Exclusive and inclusive
- Callpath profile
  - Performance metrics for calling path (event chain)
  - Differentiate performance with respect to program execution state
  - Exclusive and inclusive

Performance Evaluation Alternatives



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

Volume of performance data

Tracing Measurement

Process A:

```
void master {
  trace(ENTER, 1);
  ...
  trace(SEND, B);
  send(B, tag, buf);
  ...
  trace(EXIT, 1);
}
```

Process B:

```
void worker {
  trace(ENTER, 2);
  ...
  recv(A, tag, buf);
  trace(RECV, A);
  ...
  trace(EXIT, 2);
}
```

MONITOR

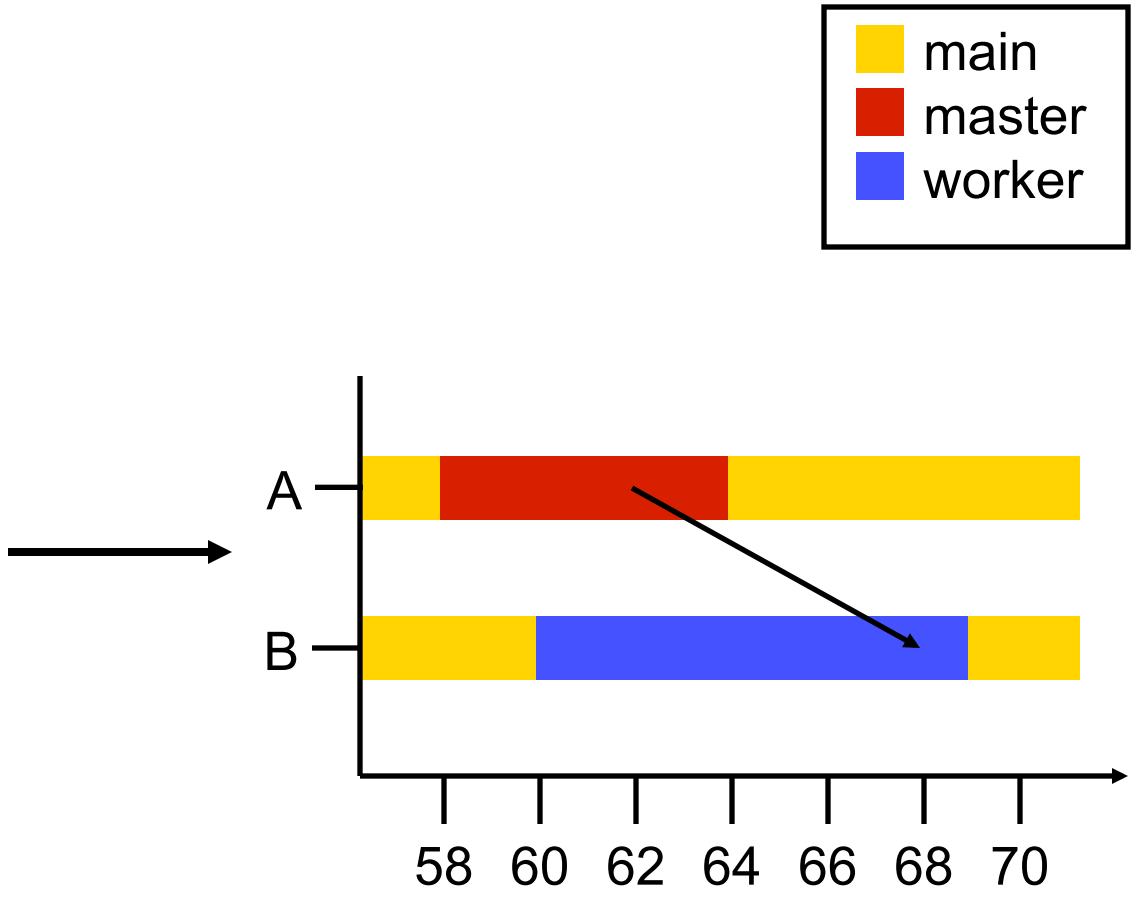
1	master
2	worker
3	...

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

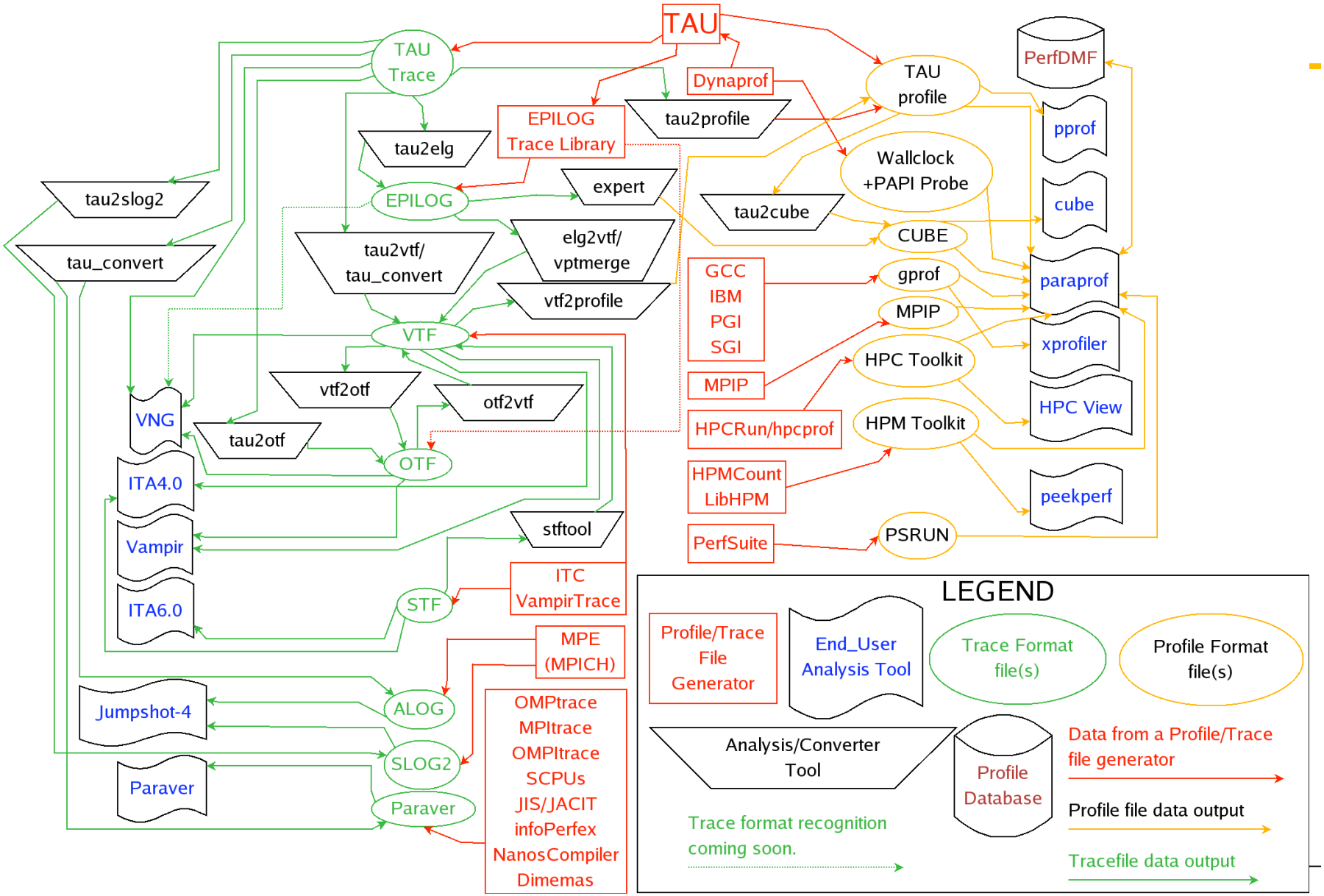
Tracing Analysis and Visualization

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
...			



Building Bridges to Other Tools



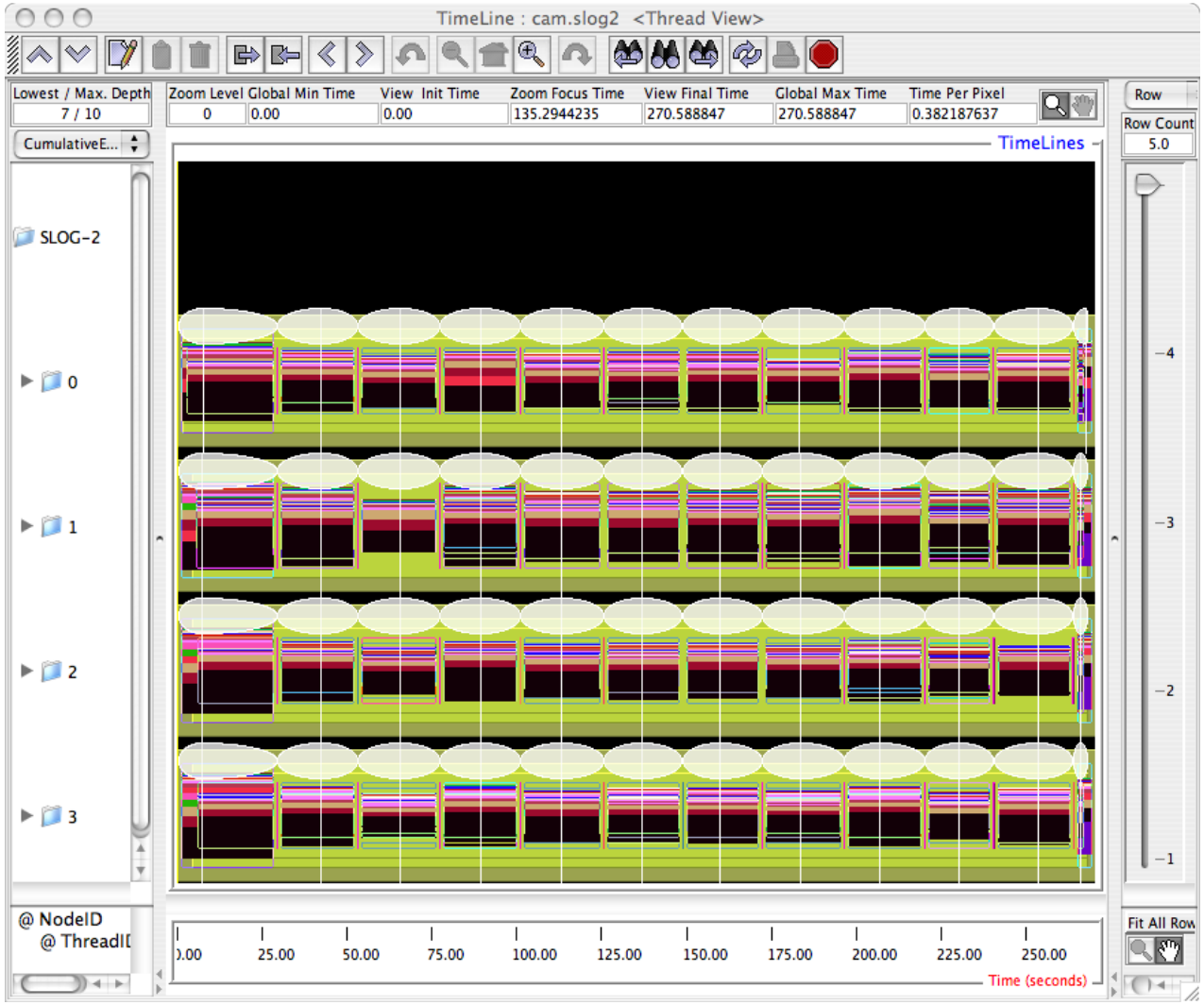
Trace Formats

- Different tools produce different formats
  - Differ by event types supported
  - Differ by ASCII and binary representations
    - Vampir Trace Format (VTF)
    - KOJAK (EPILOG)
    - Jumpshot (SLOG-2)
    - Paraver
- Open Trace Format (OTF)
  - Supports interoperation between tracing tools

Profiling / Tracing Comparison

- Profiling
  - ☺ Finite, bounded performance data size
  - ☺ Applicable to both direct and indirect methods
  - 😐 Loses time dimension (not entirely)
  - ☹ Lacks ability to fully describe process interaction
- Tracing
  - ☺ Temporal and spatial dimension to performance data
  - ☺ Capture parallel dynamics and process interaction
  - 😐 Some inconsistencies with indirect methods
  - ☹ Unbounded performance data size (large)
  - ☹ Complex event buffering and clock synchronization

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

Vampir – Trace Zoomed (S3D)

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:

    % setenv TAU_MAKEFILE /soft/apps/tau/tau-2.18.2/bgp/lib/Makefile.tau-mpi-pdt
    % setenv 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:

    % mpif90 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-2.18.2/bgp/lib; ls Makefile.*
Makefile.tau-pdt
Makefile.tau-mpi-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:

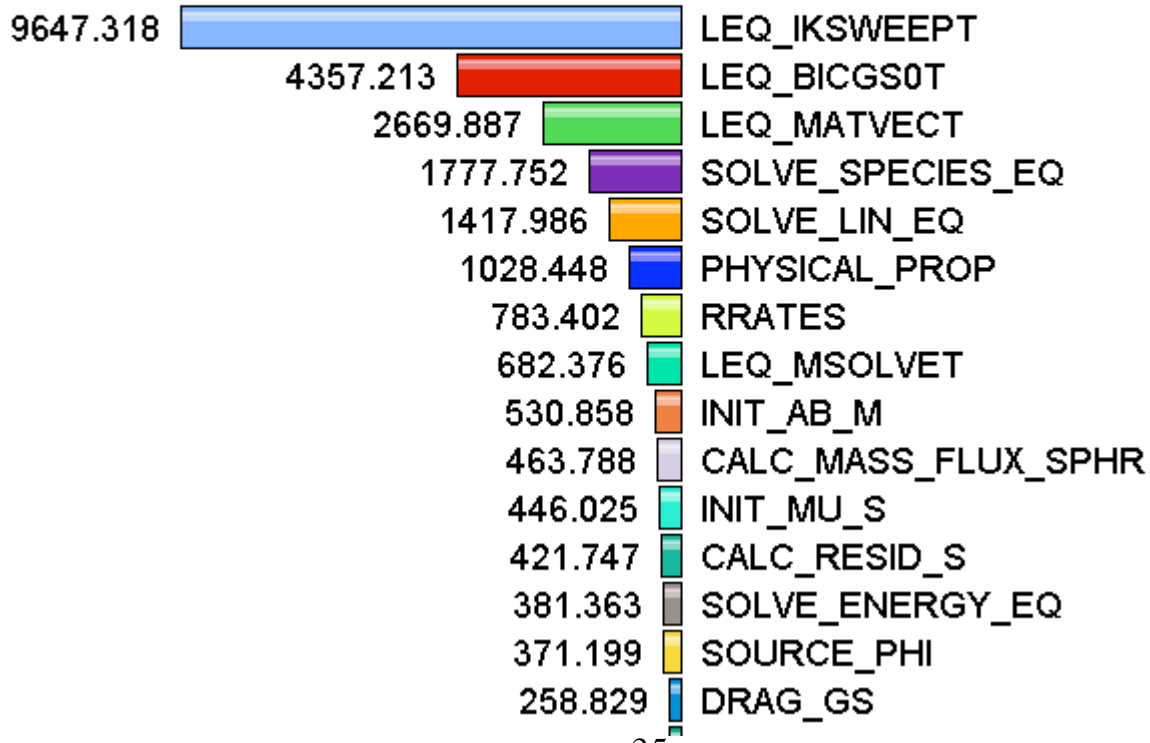
Makefile.tau-mpi-pdt

- Supports MPI instrumentation & PDT for automatic source instrumentation
- `% setenv TAU_MAKEFILE /soft/apps/tau/tau-2.18.2/bgp/lib/Makefile.tau-mpi-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

```
% setenv TAU_MAKEFILE /soft/apps/tau/tau-2.18.2/bgp
                                        /lib/Makefile.tau-mpi-pdt
% set path=(/soft/apps/tau/tau-2.18.2/ppc64/bin $path)
OR
% source /soft/apps/tau/src/tau.cshrc  [ or tau.bashrc]
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)

% qsub run.job
% 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

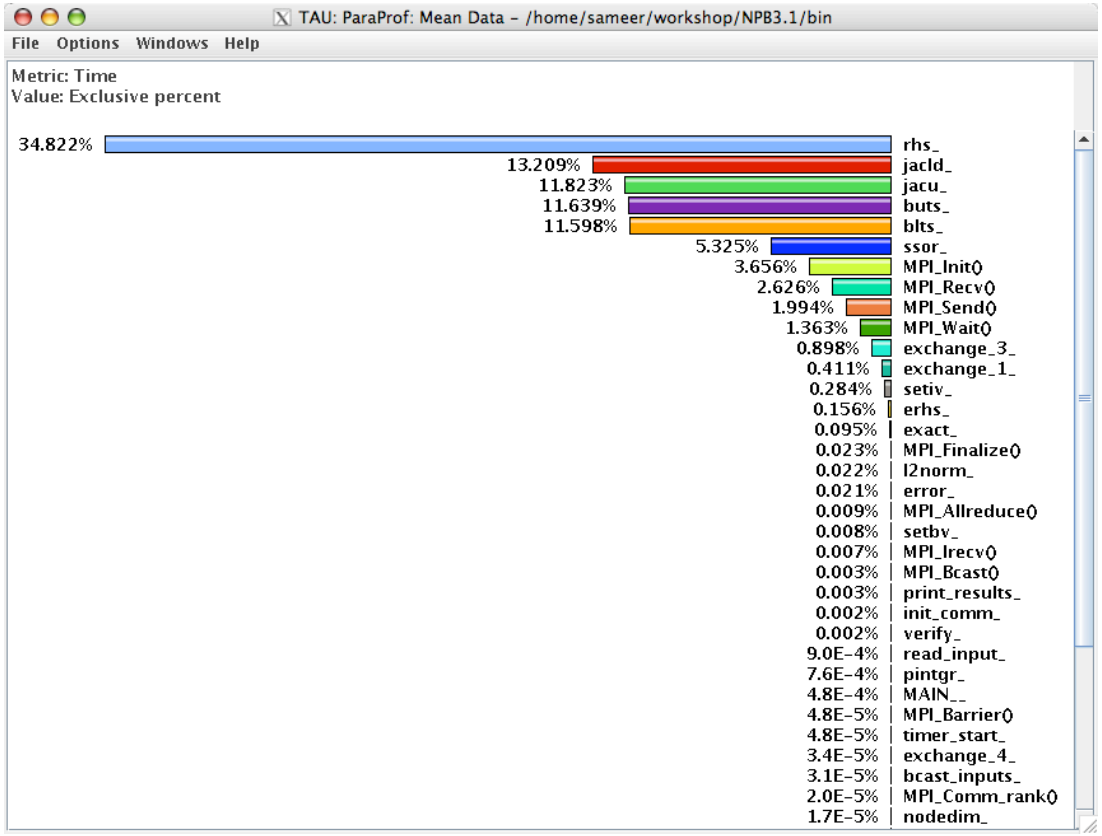
1729975.333	Loop: MULTIPLY_MATRICES [{matmult.f90} {31,9}-{36,14}]
443194	MPI_Recv()
81095	MAIN
49569	MPI_Bcast()
45669	Loop: MAIN [{matmult.f90} {86,9}-{106,14}]
12412	MPI_Send()
8959	Loop: INITIALIZE [{matmult.f90} {17,9}-{21,14}]
8953	Loop: INITIALIZE [{matmult.f90} {10,9}-{14,14}]
5609.2	MPI_Finalize()
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()

Solution: Generating a loop level profile

```
% setenv TAU_MAKEFILE /soft/apps/tau/tau-2.18.2/bgp
                                        /lib/Makefile.tau-mpi-pdt
% setenv TAU_OPTIONS '-optTauSelectFile=select.tau -optVerbose'
% cat select.tau
  BEGIN_INSTRUMENT_SECTION
  loops routine="#"
  END_INSTRUMENT_SECTION

% set path=(/soft/apps/tau/tau-2.18.2/ppc64/bin $path)
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)
% qsub  run.job
% 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



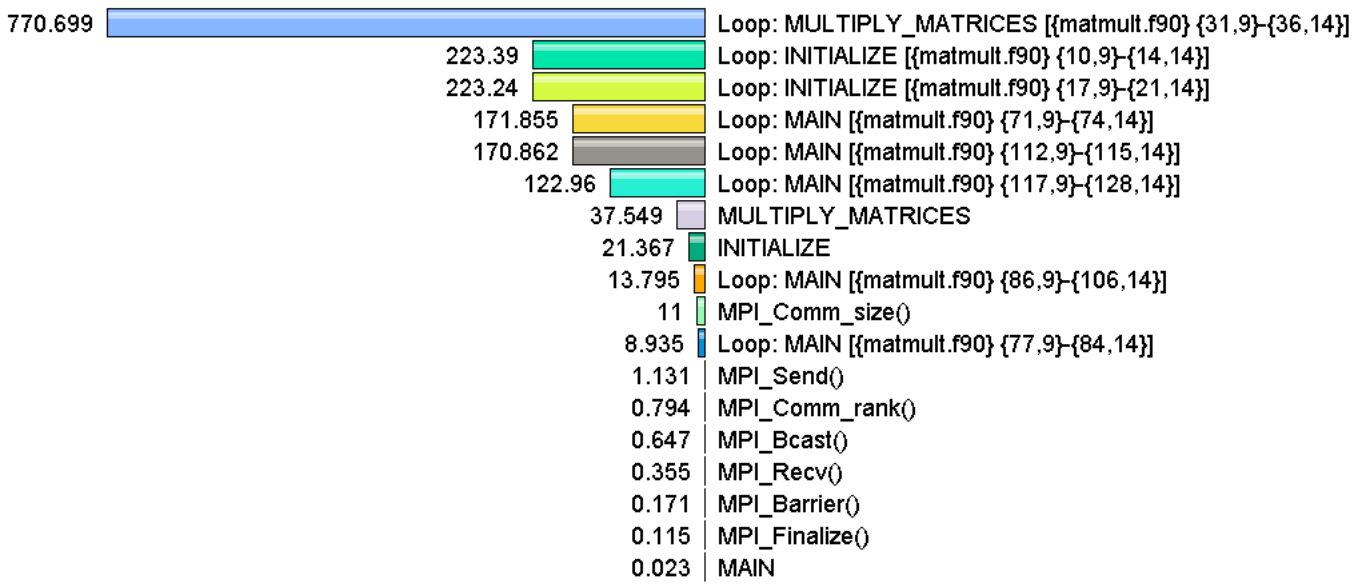
Use Compiler-Based Instrumentation

```
% setenv TAU_MAKEFILE /soft/apps/tau/tau-2.18.2/bgp
         /lib/Makefile.tau-mpi
% setenv TAU_OPTIONS '-optCompInst -optVerbose'
% % set path=(/soft/apps/tau/tau-2.18.2/ppc64/bin $path)
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)

% qsub  run.job
% 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 (-multiplecounters) with loop instrumentation:

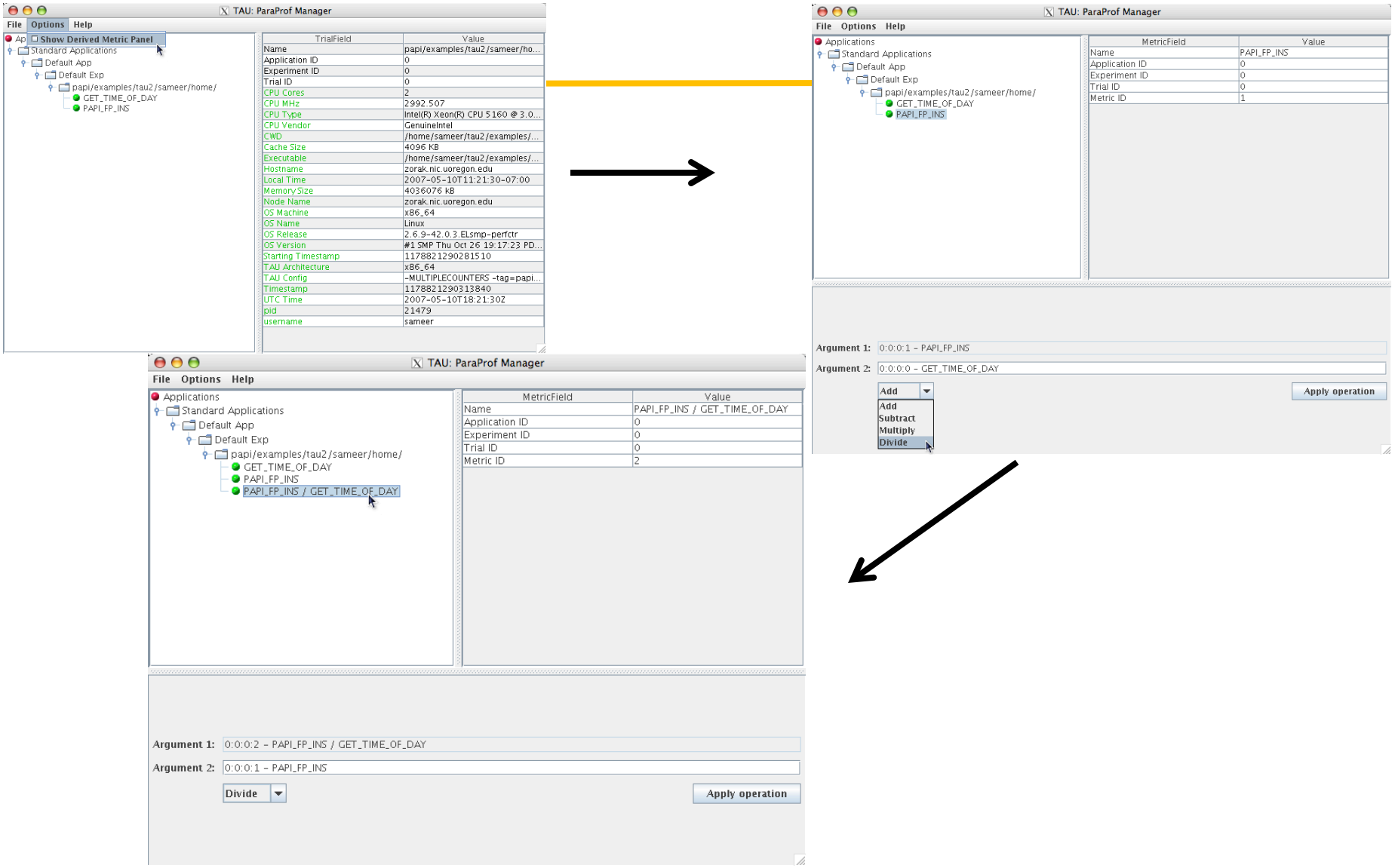
Metric: PAPI_FP_INS / GET_TIME_OF_DAY
Value: Exclusive
Units: Derived metric shown in microseconds format



Generate a PAPI profile with 2 or more counters

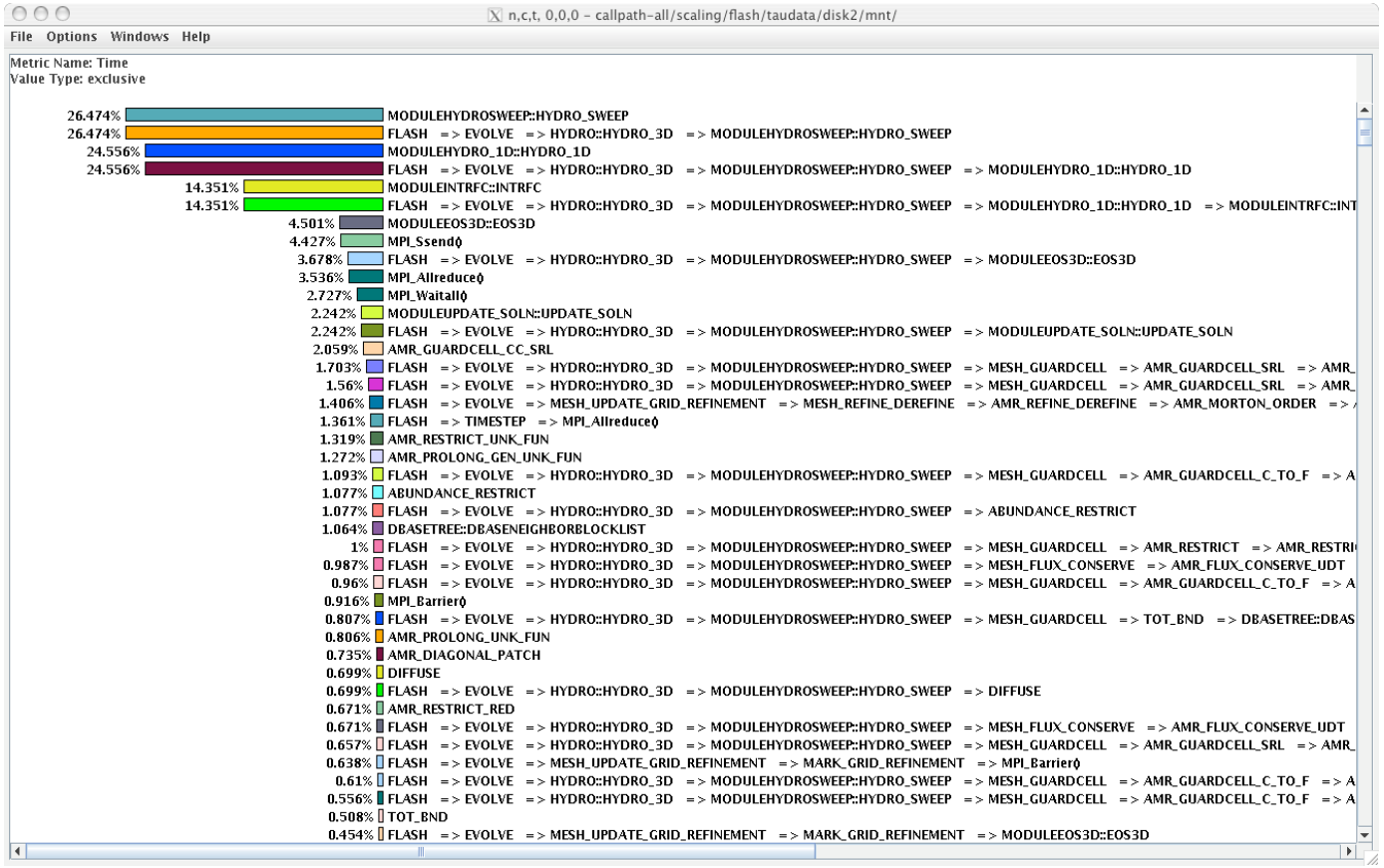
```
% setenv TAU_MAKEFILE /soft/apps/tau/tau-2.18.2/bgp
                      /lib/Makefile.tau-papi-mpi-pdt
% setenv TAU_OPTIONS '-optTauSelectFile=select.tau -optVerbose'
% cat select.tau
  BEGIN_INSTRUMENT_SECTION
  loops routine="#"
  END_INSTRUMENT_SECTION

% set path=(/soft/apps/tau/tau-2.18.2/ppc64/bin $path)
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)
% setenv COUNTER1 GET_TIME_OF_DAY
% setenv COUNTER2 PAPI_FP_INS
OR
% setenv TAU_METRICS TIME:PAPI_FP_INS
% setenv TAU_METRICS TIME:PAPI_NATIVE_<event_name>:….
% qsub  run.job
% 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



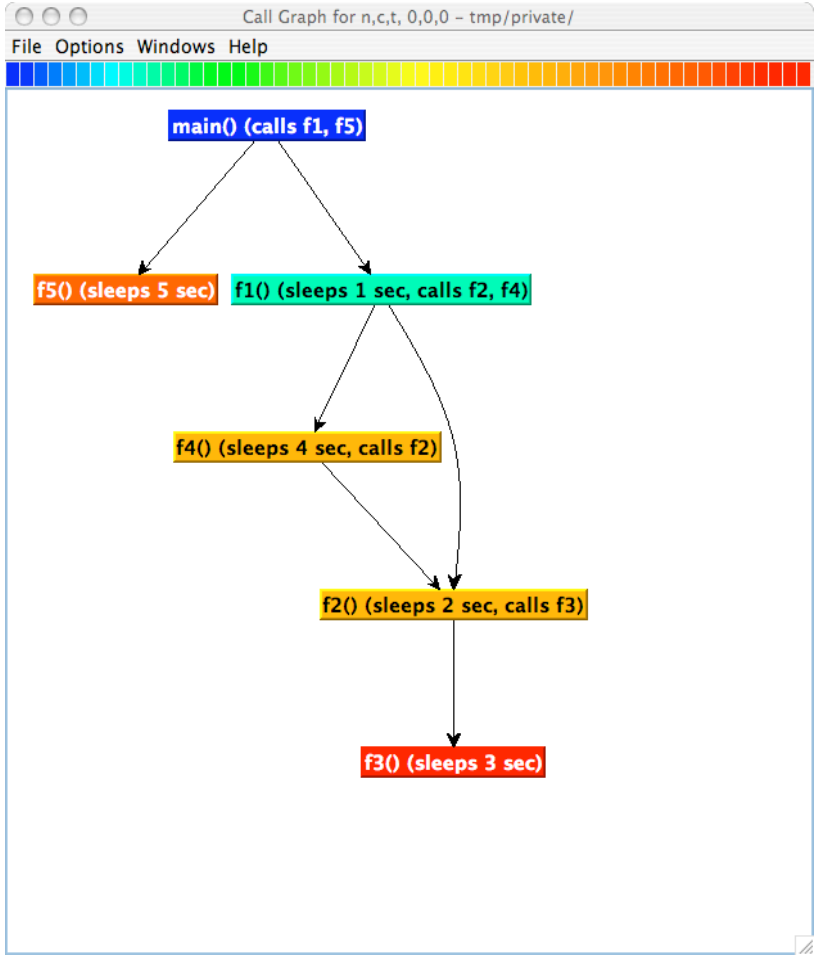
Usage Scenarios: Generating Callpath Profile

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



Callpath Profile

- Generates program callgraph



Generate a Callpath Profile

```
% setenv TAU_MAKEFILE /soft/apps/tau/tau-2.18.2/bgp
                                /lib/Makefile.tau-callpath-mpi-pdt
% set path=(/soft/apps/tau/tau-2.18.2/ppc64/bin $path)
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)
% setenv TAU_CALLPATH_DEPTH 100

% qsub  run.job
% 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:
% setenv 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

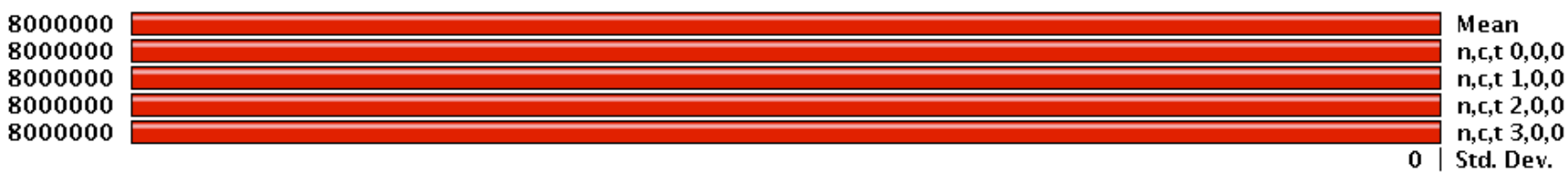
File Windows Help

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

```
% setenv TAU_MAKEFILE /soft/apps/tau/tau-2.18.2/bgp
                        /lib/Makefile.tau-mpi-pdt
% setenv TAU_OPTIONS '-optDetectMemoryLeaks -optVerbose'
% set path=(/soft/apps/tau/tau-2.18.2/ppc64/bin $path)
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)
% setenv TAU_CALLPATH_DEPTH 100

% qsub  run.job
% 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

```
% setenv TAU_MAKEFILE /usr/global/tools/tau/ibm64
                        /lib/Makefile.tau-python-pdt
% set path=(/usr/global/tools/tau/ibm64/bin $path)
% cat wrapper.py
  import tau
  def OurMain():
      import foo
  tau.run('OurMain()')
Uninstrumented:
% ./foo.py
Instrumented:
% setenv PYTHONPATH <taudir>/ibm64/<lib>/bindings-python-pdt
(same options string as TAU_MAKEFILE)
% setenv LD_LIBRARY_PATH <taudir>/x86_64/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

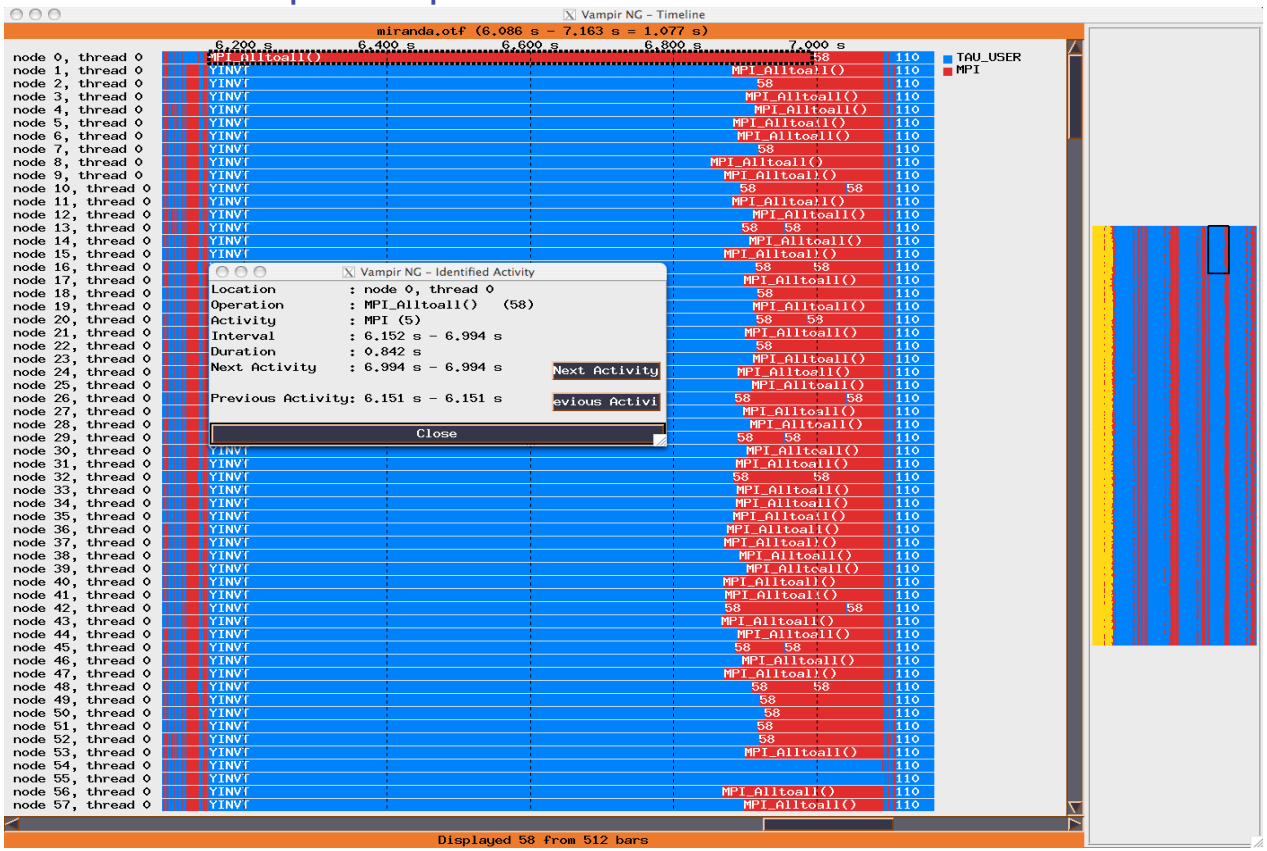
39.466%	void SAMINT::timestep(double, double) [{SAMINT.C} {72,1}-{78,1}]
6.563%	int pyMPI_Main_with_communicator(int, int *, char ***, MPI_Comm) C [{pyMPI_main.c} {23,1}-{80,1}]
5.18%	<module> [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/site-packages/numpy/linalg/linalg.py} {7}]
4.91%	void pyMPI_distutils_init(PyObject **) 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

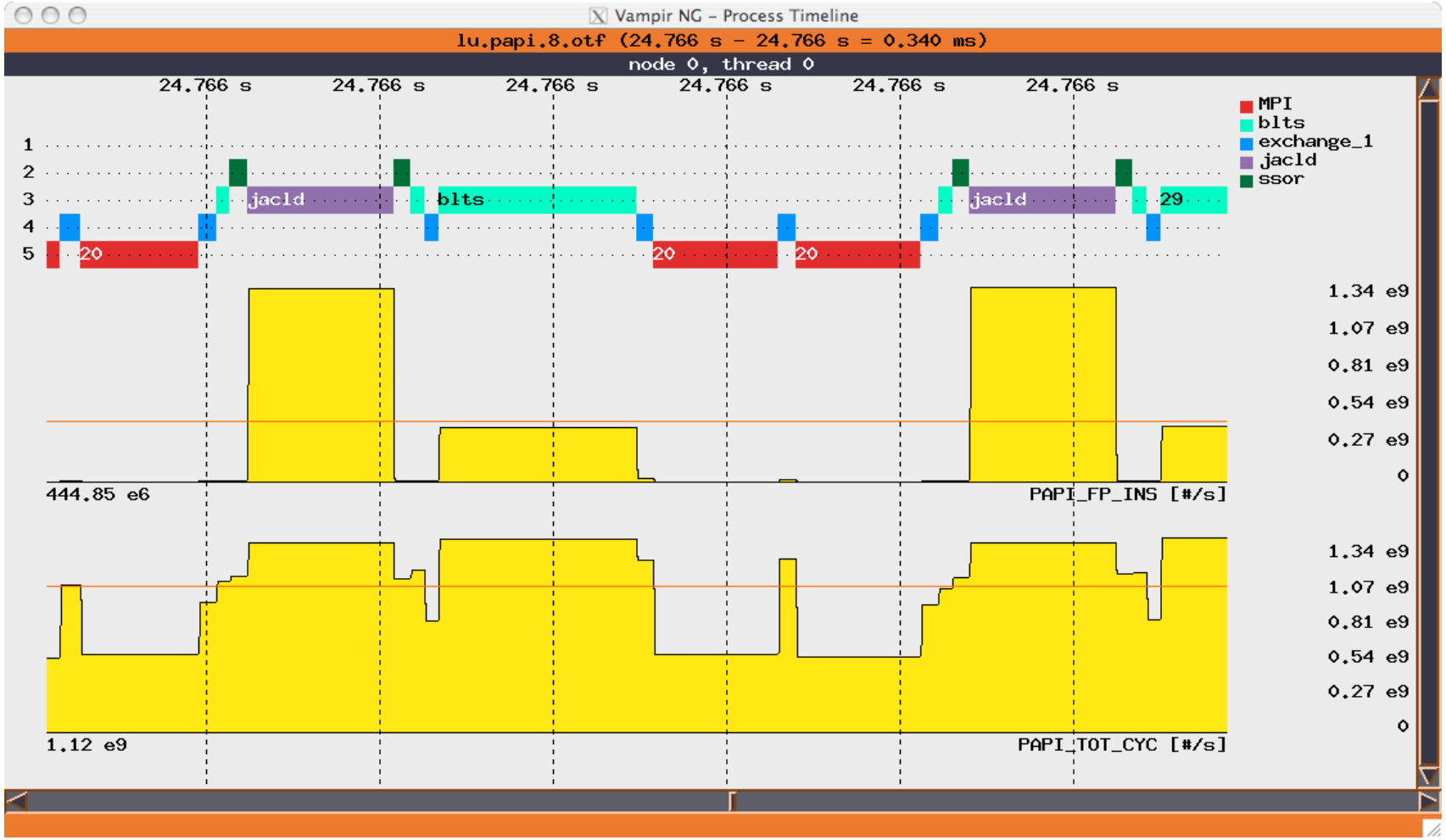
```
% setenv TAU_MAKEFILE /usr/global/tools/tau/ibm64
                        /lib/Makefile.tau-python-mpi-pdt
% set path=(/usr/global/tools/tau/ibm64/bin $path)
% setenv TAU_OPTIONS '-optShared -optVerbose…'
(Python needs shared object based TAU library)
% make F90=tau_f90.sh CXX=tau_cxx.sh CC=tau_cc.sh  (build libs, pyMPI w/TAU)
% cat wrapper.py
  import tau
  def OurMain():
      import App
  tau.run('OurMain()')
Uninstrumented:
% mpirun.lsf /usr/global/tools/.unsupported/pyMPI-2.5b0/bin/pyMPI ./App.py
Instrumented:
% setenv PYTHONPATH <taudir>/x86_64/<lib>/bindings-python-mpi-pdt
(same options string as TAU_MAKEFILE)
% setenv LD_LIBRARY_PATH <taudir>/x86_64/lib/bindings-python-mpi-pdt\:
$LD_LIBRARY_PATH
% mpirun -np 4 /usr/global/tools/.unsupported/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

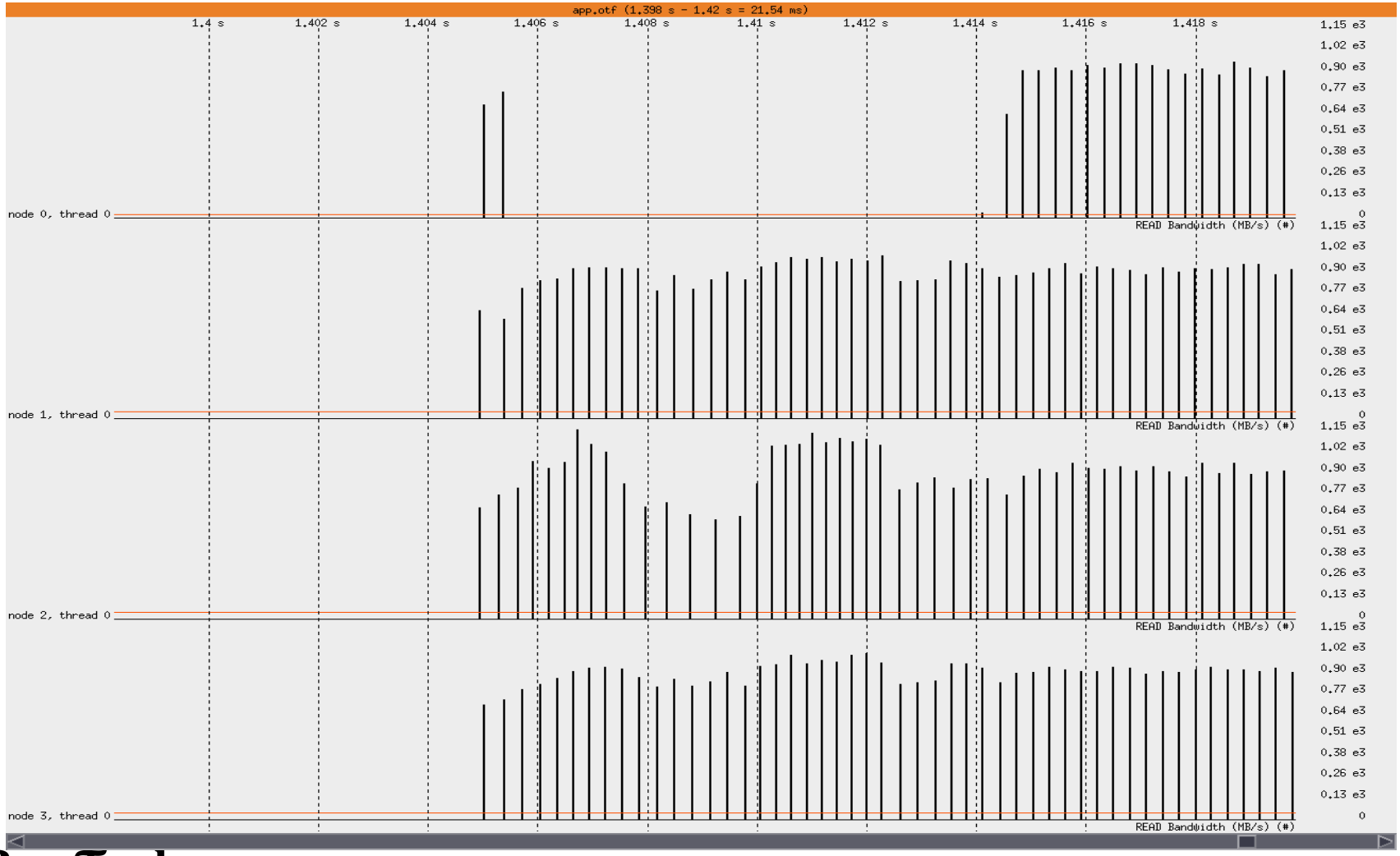


VNG Process Timeline with PAPI Counters

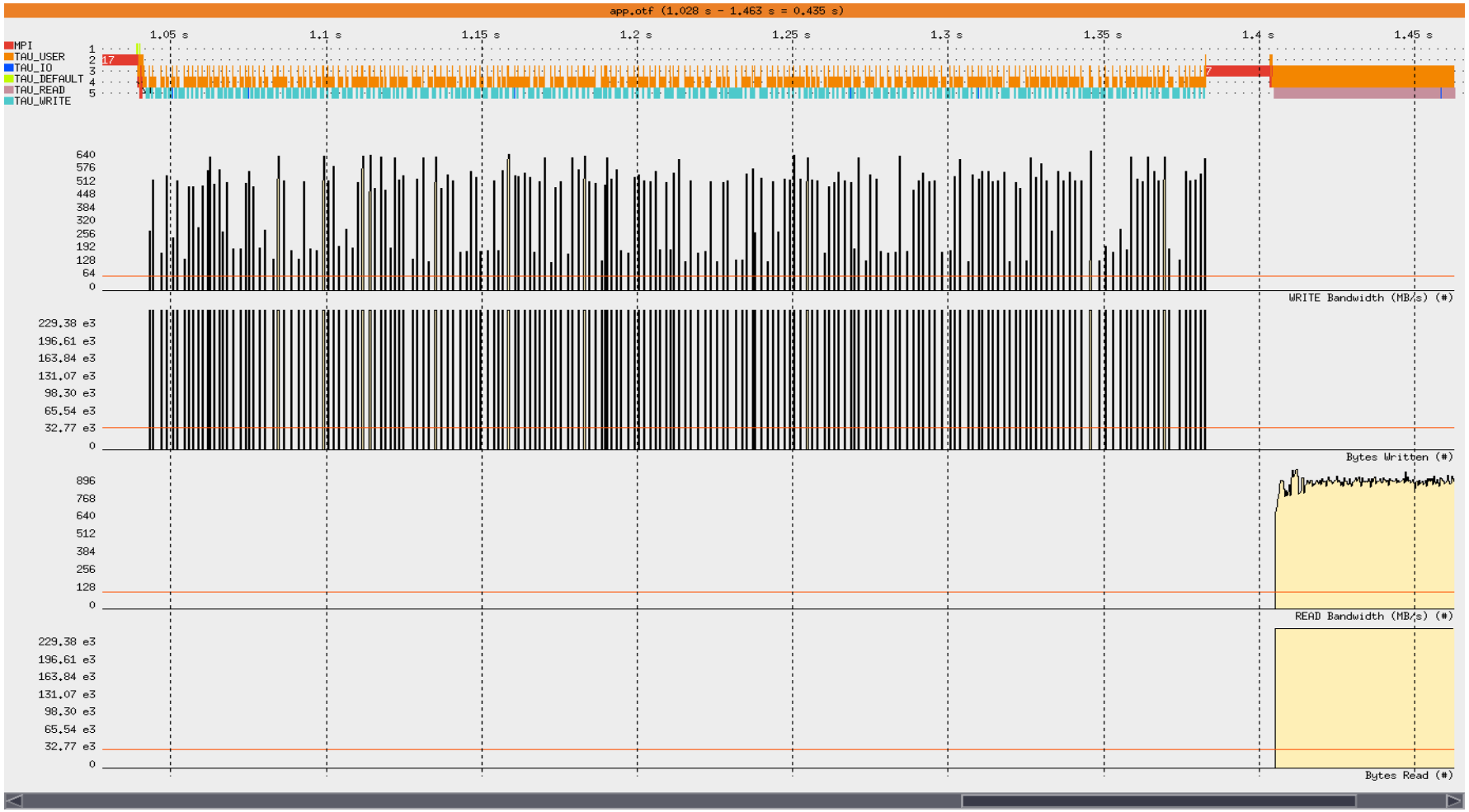


Vampir Counter Timeline Showing I/O BW

TAU: I/O Instrumentation for Read Bandwidth: Vampir



Vampir Process Timeline for Rank O (IOR, LLNL)

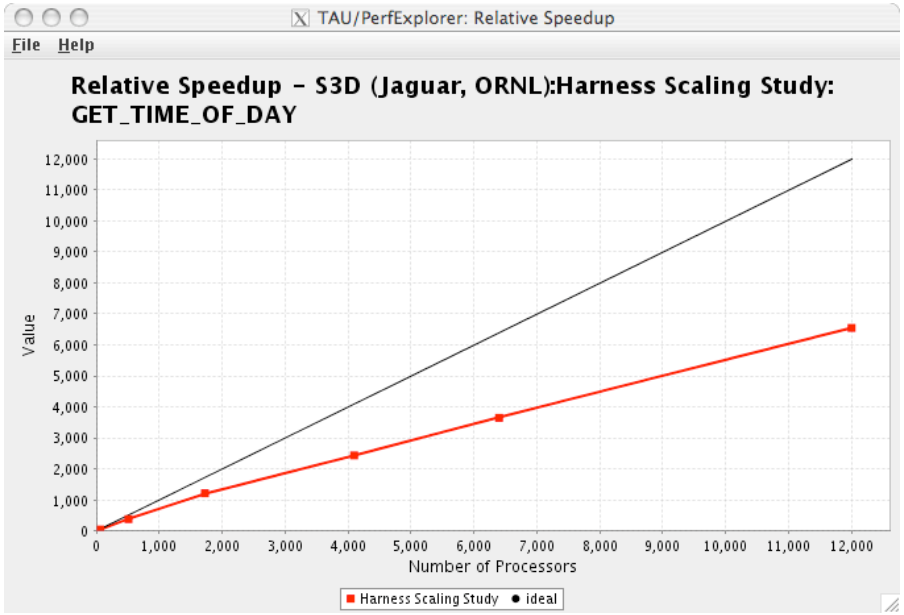


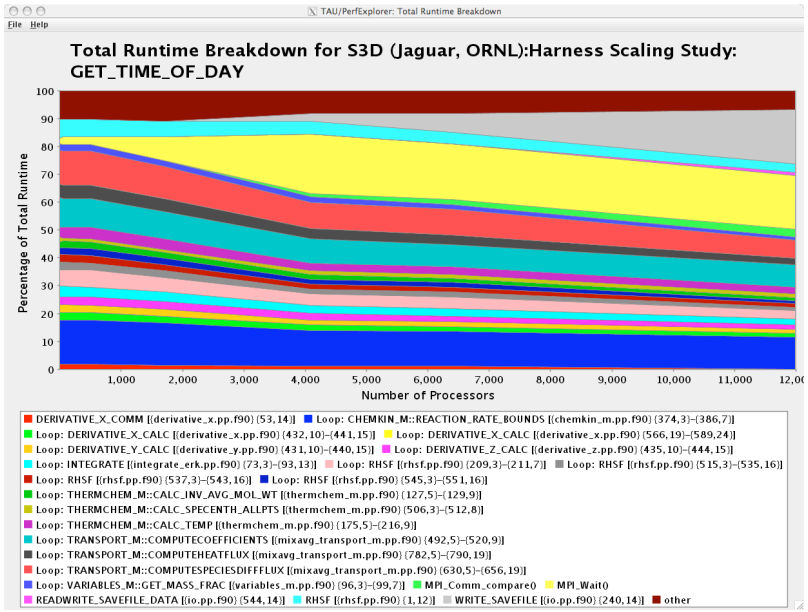
Generate a Trace File

```
% setenv TAU_MAKEFILE /soft/apps/tau/tau-2.18.2/bgp
                      /lib/Makefile.tau-mpi-pdt-trace
or setenv TAU_TRACE 1 (in TAU v2.18.2+)
% set path=(/soft/apps/tau/tau-2.18.2/ppc64/bin $path)
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)
% qsub  run.job
% 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).
```

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



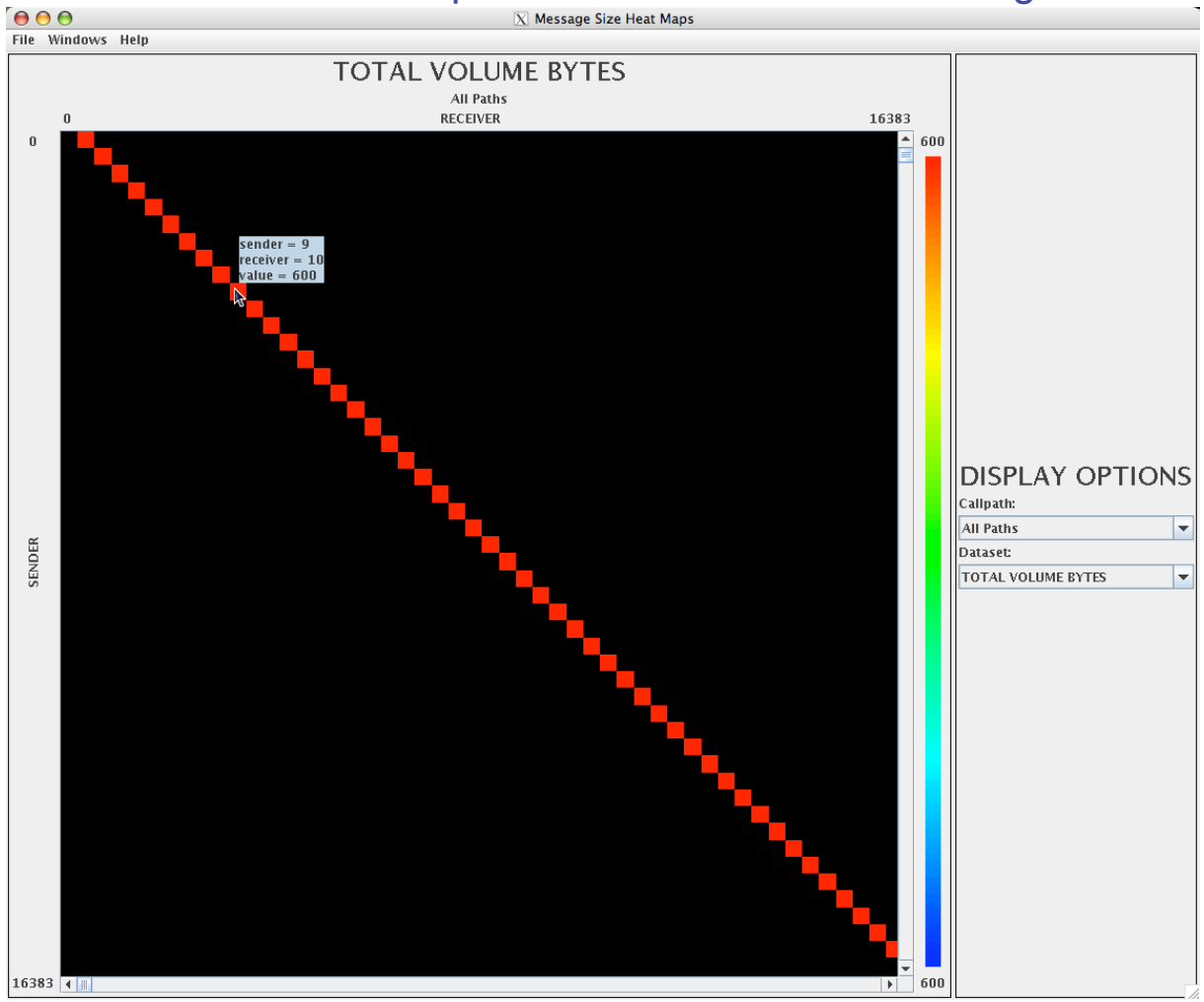


Evaluate Scalability using PerfExplorer Charts

```
% setenv TAU_MAKEFILE /soft/apps/tau/tau-2.18.2/bgp
                              /lib/Makefile.tau-mpi-pdt
% set path=(/soft/apps/tau/tau-2.18.2/ppc64/bin $path)
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)
% qsub  run1p.job
% paraprof --pack 1p.ppk
% qsub  run2p.job …
% 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

```
% setenv TAU_MAKEFILE /soft/apps/tau/tau-2.18.2/bgp
                              /lib/Makefile.tau-mpi-pdt
% set path=(/soft/apps/tau/tau-2.18.2/ppc64/bin $path)
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)
% setenv TAU_COMM_MATRIX 1

% qsub run.job (setting the environment variables)

% paraprof
(Windows -> Communication Matrix)
```

Labs

- Add one of
  source /soft/apps/tau/src/tau.bashrc
  or
  source /soft/apps/tau/src/tau.cshrc
  to the end of your **.login** file (for bash or csh/tcsh users respectively)
  These files contain ANL specific location information.

Support Acknowledgements

- Department of Energy (DOE)
  - Office of Science
    - MICS, Argonne National Lab
  - ASC/NNSA
    - University of Utah ASC/NNSA Level 1
    - ASC/NNSA, LLNL
- Department of Defense (DoD)
  - HPC Modernization Office (HPCMO)
- NSF SDCI
- Research Centre Juelich
- LBL, ORNL, ANL, LANL, LLNL
- TU Dresden
- Univesity of Oregon