Parallel Performance Evaluation Tools: TAU, PAPI, Scalasca and Vampir

Two day tutorial at LLNL
Building 453 R1001 (Armadillo), May 26-27, Livermore, CA

Sameer Shende
sameer@paratools.com
http://www.paratools.com/llnl09

Outline

Outline

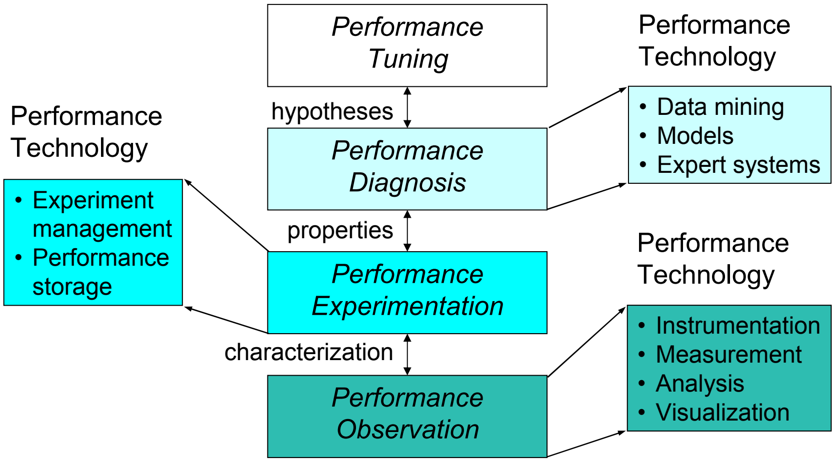
- Day 1:
  - Introduction to performance evaluation tools: TAU, PAPI, Scalasca, and Vampir
  - Hands-on:
    - TAU instrumentation at routine, loop level, PAPI hardware performance counter data collection, derived metrics, analyzing performance using TAU's paraprof profile browser, using Performance database (PerfDMF), memory evaluation, leak detection
- Day 2:
  - Scalasca, TAU PerfExplorer, VampirServer
  - Hands-on:
    - Scalasca bottleneck detection tools, PerfExplorer, trace visualization, workshop examples including the NAS Parallel Benchmarks 3.1

Workshop Goals

- This tutorial is an introduction to portable performance evaluation tools.
- You should leave here with a better understanding of…
  - Concepts and steps involved in performance evaluation
  - Understanding key concepts in improving and understanding code performance
  - How to collect and analyze data from hardware performance counters using PAPI
  - How to instrument your programs with TAU
    - Automatic instrumentation at the routine level and outer loop level
    - Manual instrumentation at the loop/statement level
  - Measurement options provided by TAU
  - Environment variables used for choosing metrics, generating performance data
  - How to use the TAU's profile browser, ParaProf
  - How to use TAU's database for storing and retrieving performance data
  - General familiarity with TAU's use for Fortran, Python, C++,C, MPI for mixed language programming
  - How to generate trace data in different formats
  - How to use Scalasca for detecting performance bottlenecks
  - How to analyze trace data using Vampir, and Jumpshot
  - Facilities provided by the Eclipse PTP integrated development environment for parallel programs

Performance Engineering

- Optimization process
- Effective use of performance technology



Performance Tuning

hypotheses

Performance Diagnosis

properties

Performance Experimentation

characterization

Performance Observation

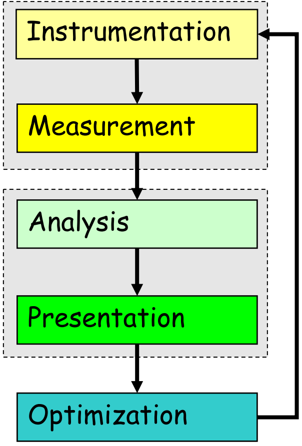
Performance Technology
- Data mining
- Models
- Expert systems

Performance Technology
- Experiment management
- Performance storage

Performance Technology
- Instrumentation
- Measurement
- Analysis
- Visualization

Performance Optimization Cycle

- Expose factors
- Collect performance data
- Calculate metrics
- Analyze results
- Visualize results
- Identify problems
- Tune performance



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.eu/
- Scalasca/KOJAK References
  - Scalasca documentation page
    http://www.scalasca.org/

TAU: A Quick Reference

Part I: TAU: A Tutorial

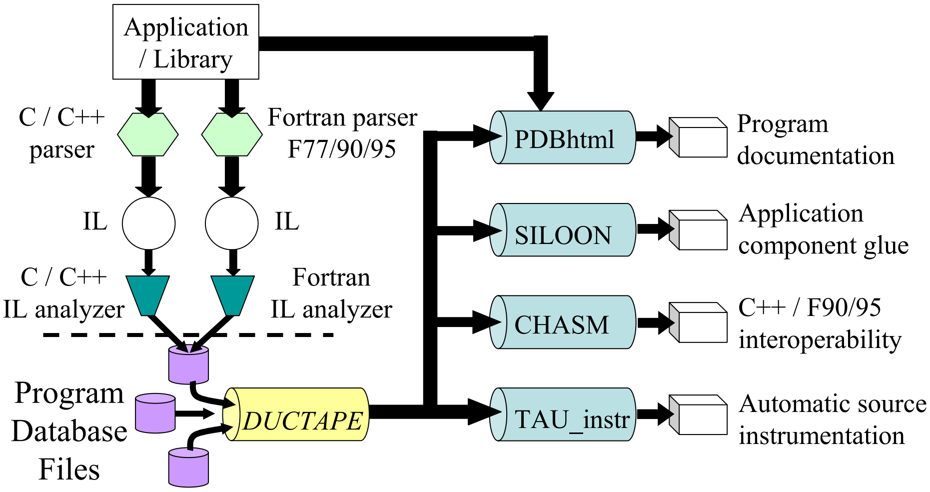
TAU Performance System

- 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

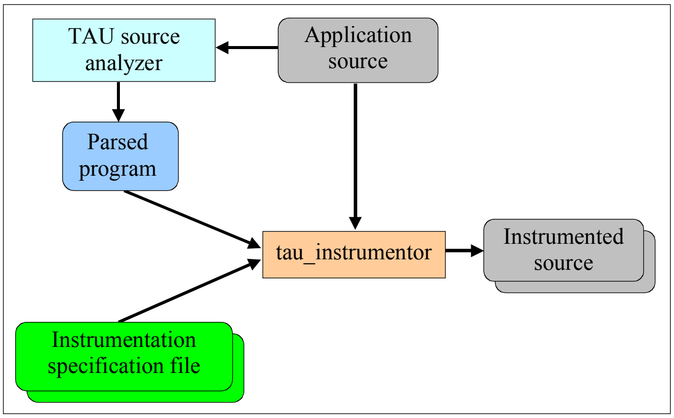
What is TAU?

- 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)

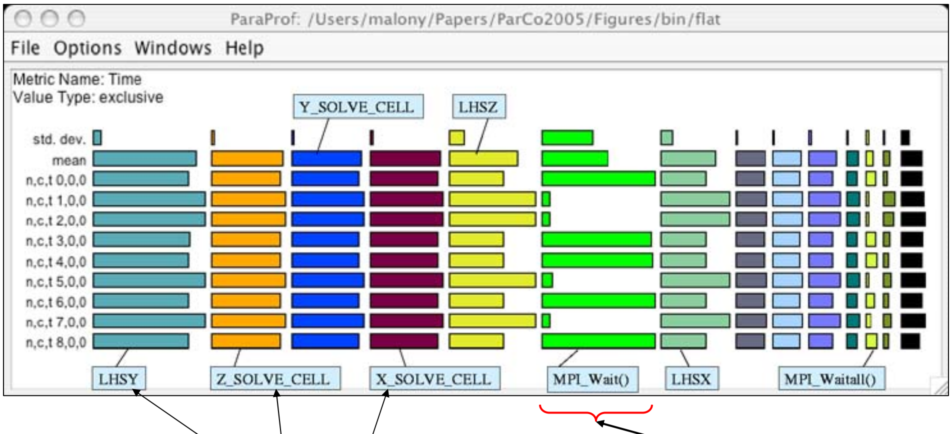
Program Database Toolkit (PDT)



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



ParaProf – Flat Profile (NAS BT)

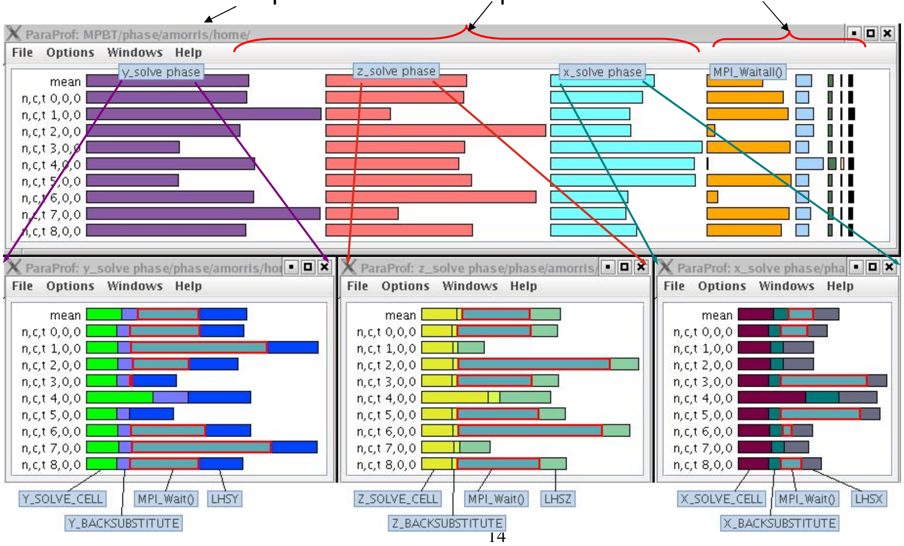


Application routine names reflect phase semantics

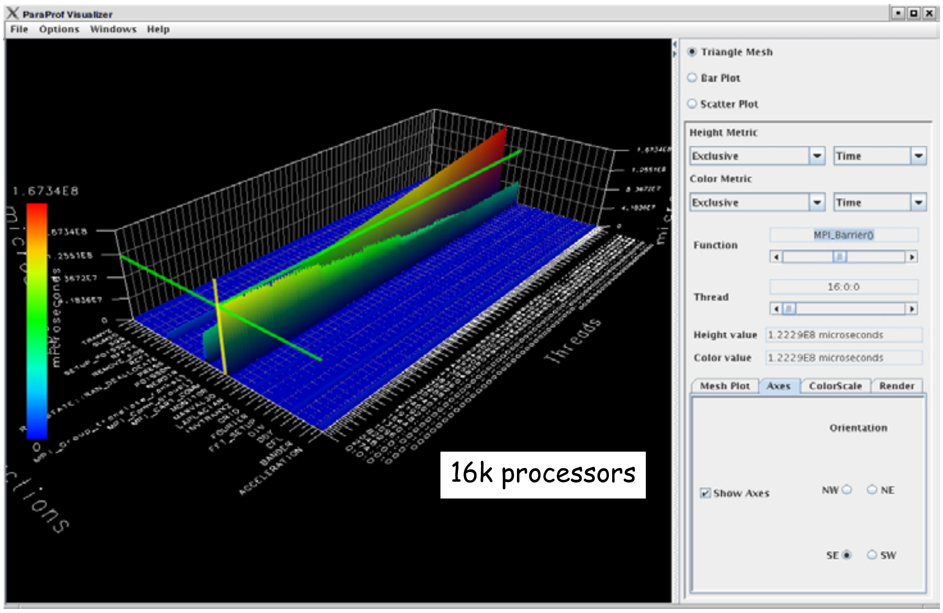
How is MPI_Wait() distributed relative to solver direction?

ParaProf – Phase Profile (NAS BT)

Main phase shows nested phases and immediate events



ParaProf – 3D Full Profile (Miranda, LLNL)



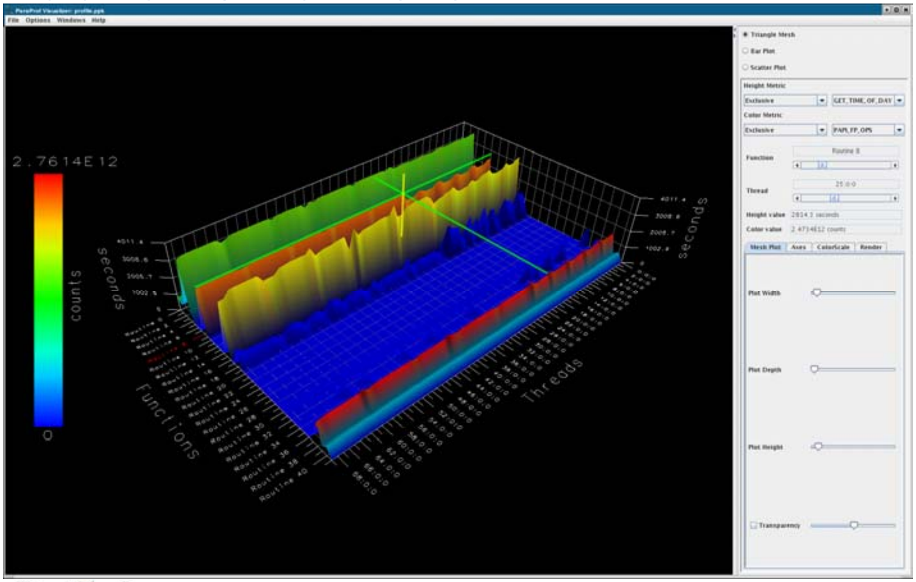
16k processors

● Triangle Mesh
○ Bar Plot
○ Scatter Plot

Height Metric: Exclusive, Time
Color Metric: Exclusive, Time
Function: MPI_Barrier()
Thread: 16:0:0
Height value: 1.2229E8 microseconds
Color value: 1.2229E8 microseconds

Orientation
☑ Show Axes
NW ○ ○ NE
SE ● ○ SW

ParaProf – 3D Full Profile

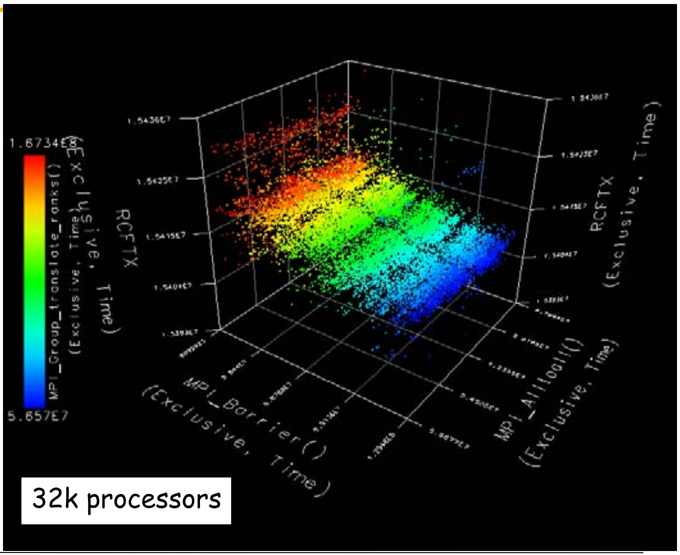


● Triangle Mesh
○ Bar Plot
○ Scatter Plot

Height Metric: Exclusive, GET_TIME_OF_DAY
Color Metric: Exclusive, PAPI_FP_OPS
Thread: 25:0:0
Color value: 2.4734E12 counts

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

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

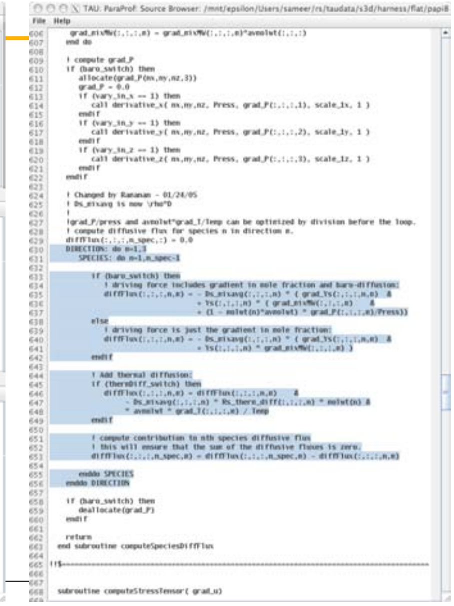
```
                                     1.088  std. dev.
114.979                                     mean
117.62                                      n,c,t 0,0,0
115.134                                     n,c,t 1,0,0
114.709                                     n,c,t 2,0,0
114.615                                     n,c,t 3,0,0
113.547                                     n,c,t 4,0,0
114.581                                     n,c,t 5,0,0
114.837                                     n,c,t 6,0,0
114.789                                     n,c,t 7,0,0
```

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

```
                                     0.91%  std. dev.
12.206%                                     mean
11.931%                                     n,c,t 0,0,0
12.19%                                      n,c,t 1,0,0
12.248%                                     n,c,t 2,0,0
12.258%                                     n,c,t 3,0,0
12.335%                                     n,c,t 4,0,0
12.241%                                     n,c,t 5,0,0
12.221%                                     n,c,t 6,0,0
12.226%                                     n,c,t 7,0,0
```

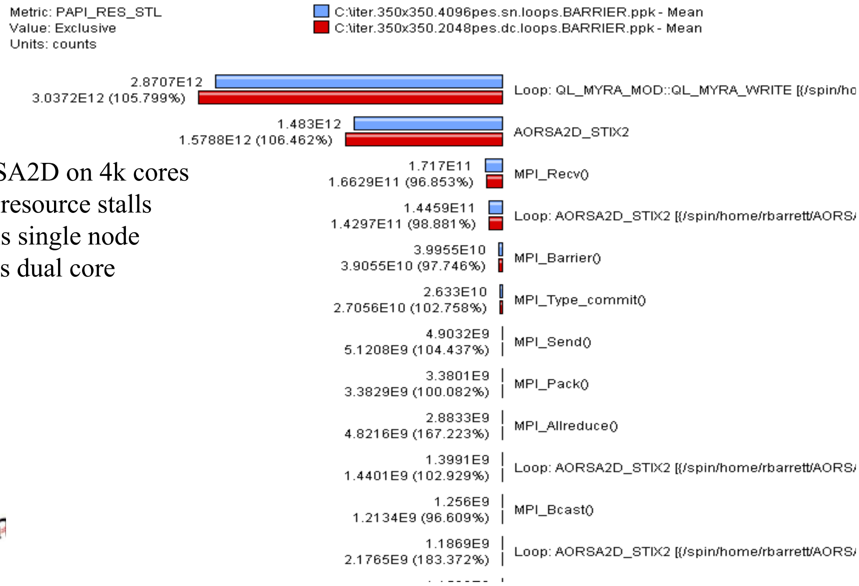
Name: Loop: TRANSPORT_M::COMPUTESPECIESDIFFFLUX [{mixavg_transport_m.pp.f90} {630,5}-{656,19}]
Metric Name: PAPI_L1_DCM
Value: Exclusive
Units: counts

```
                                  836336.1  std. dev.
5.0701E9                                    mean
5.0692E9                                    n,c,t 0,0,0
5.07E9                                      n,c,t 1,0,0
5.069E9                                     n,c,t 2,0,0
5.0701E9                                    n,c,t 3,0,0
5.0708E9                                    n,c,t 4,0,0
5.0711E9                                    n,c,t 5,0,0
5.0712E9                                    n,c,t 6,0,0
5.0692E9                                    n,c,t 7,0,0
```



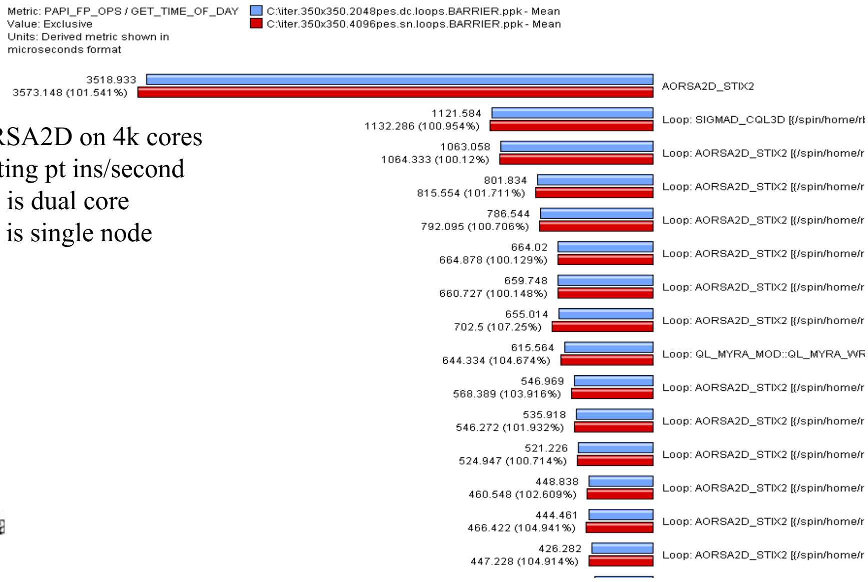
```
606        grad_mixMW(:,:,:,m) = grad_mixMW(:,:,:,m)*avmolwt(:,:,:)
607       end do
608
609       ! compute grad_P
610       if (baro_switch) then
611          allocate(grad_P(nx,ny,nz,3))
612          grad_P = 0.0
613          if (vary_in_x == 1) then
614             call derivative_x( nx,ny,nz, 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



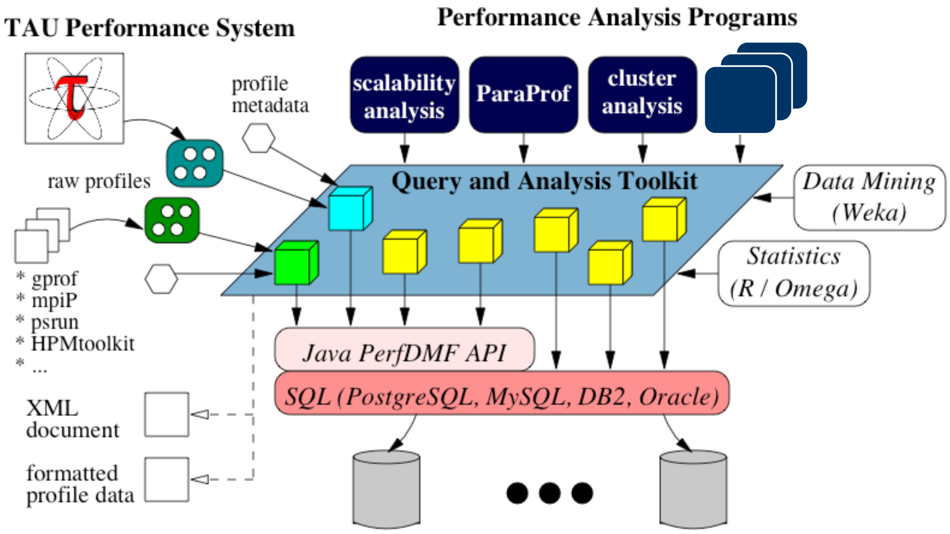
- ❑ AORSA2D on 4k cores
- ❑ PAPI resource stalls
- ❑ Blue is single node
- ❑ Red is dual core

Comparing FLOPS: MultiCore Processors



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

PerfDMF: Performance Data Mgmt. Framework



TAU Performance System

Performance Analysis Programs

scalability analysis

ParaProf

cluster analysis

profile metadata

raw profiles

Query and Analysis Toolkit

Data Mining (Weka)

Statistics (R / Omega)

- gprof
- mpiP
- psrun
- HPMtoolkit
- …

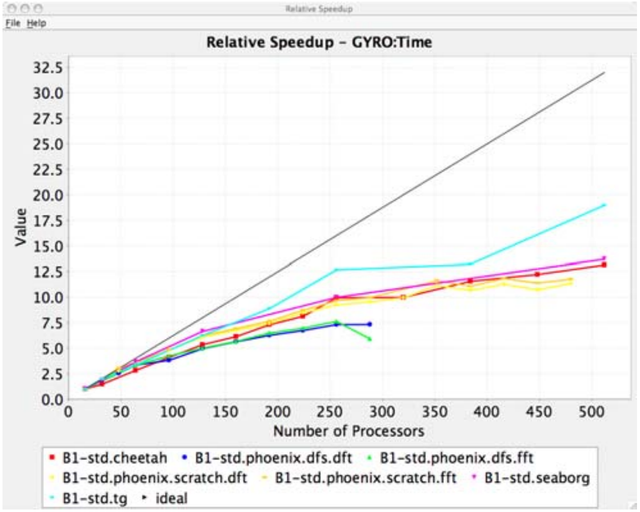
Java PerfDMF API

SQL (PostgreSQL, MySQL, DB2, Oracle)

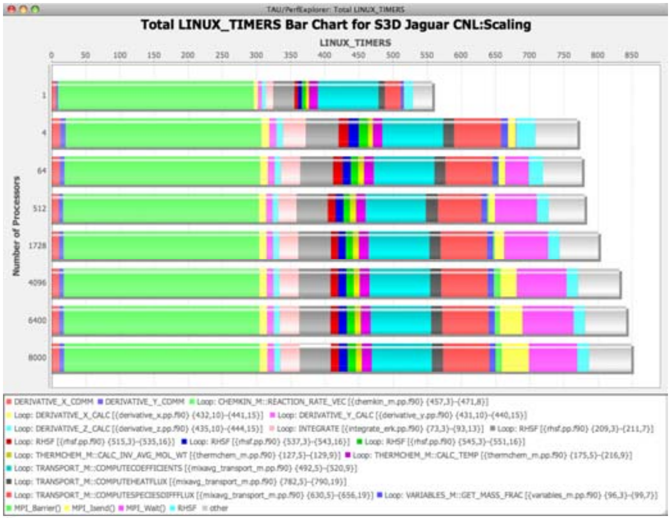
XML document

formatted profile data

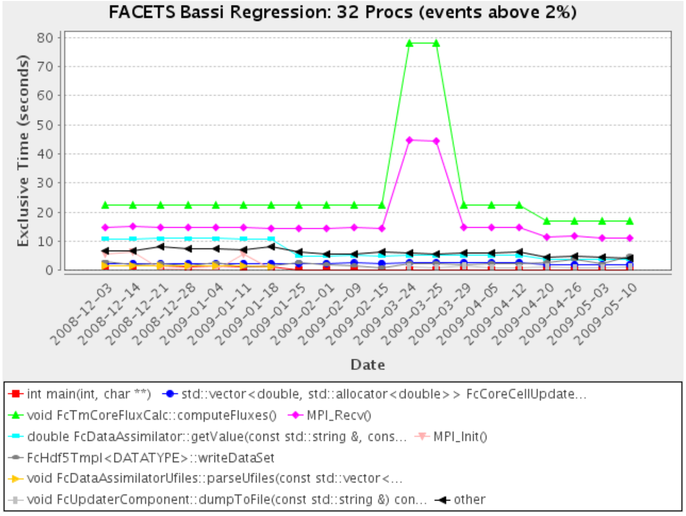
PerfExplorer: Comparing Relative Speedup on Different Architectures



Usage Scenarios: Evaluate Scalability



Performance Regression Testing



Profiling

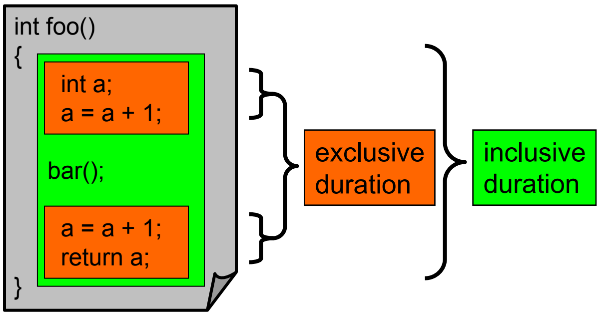
- Recording of aggregated information
  - Counts, time, …
- … about program and system entities
  - Functions, loops, basic blocks, …
  - Processes, threads
- Methods
  - Event-based sampling (indirect, statistical)
    - Open|SpeedShop, PerfSuite, HPCToolkit, gprof,…
  - Direct measurement (deterministic)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



```
int foo()
{
  int a;
  a = a + 1;

  bar();

  a = a + 1;
  return a;
}
```

exclusive duration

inclusive duration

Terminology – Example

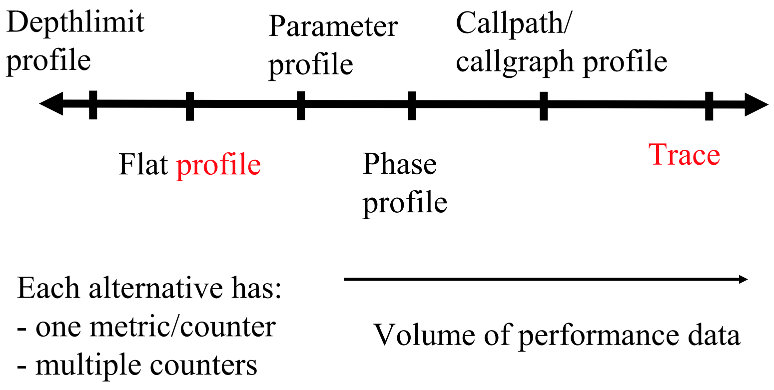
- For routine “int main( )”:
- Exclusive time
  - 100-20-50-20=10 secs
- Inclusive time
  - 100 secs
- Calls
  - 1 call
- Subrs (no. of child routines called)
  - 3
- Inclusive time/call
  - 100secs

```
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



Depthlimit profile

Parameter profile

Callpath/ callgraph profile

Flat profile

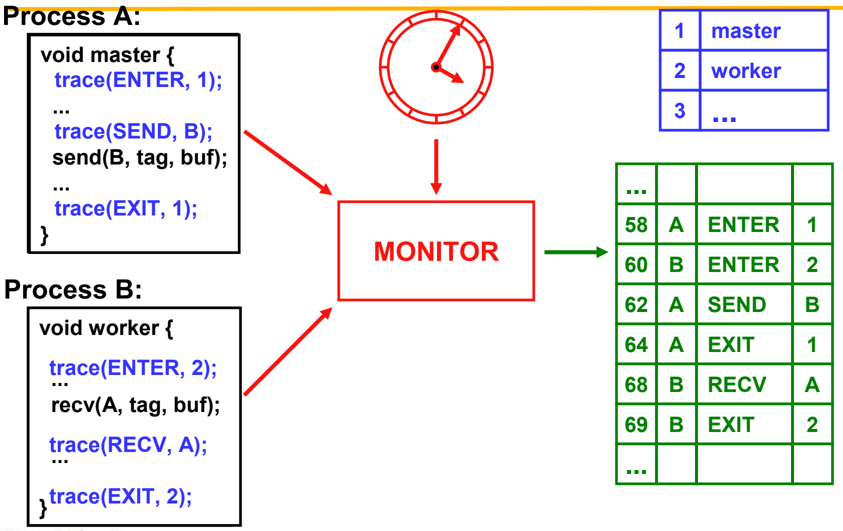
Phase profile

Trace

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:

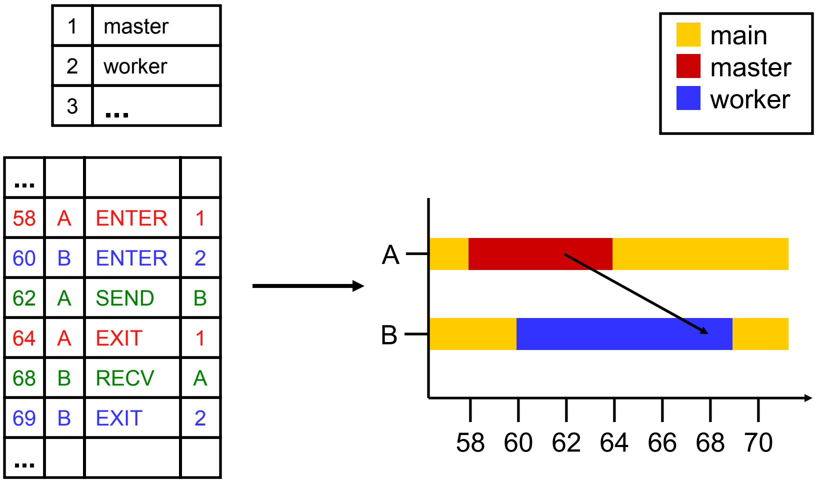
```
void worker {
  trace(ENTER, 2);
  ...
  recv(A, tag, buf);
  trace(RECV, A);
  ...
  trace(EXIT, 2);
}
```

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
...			

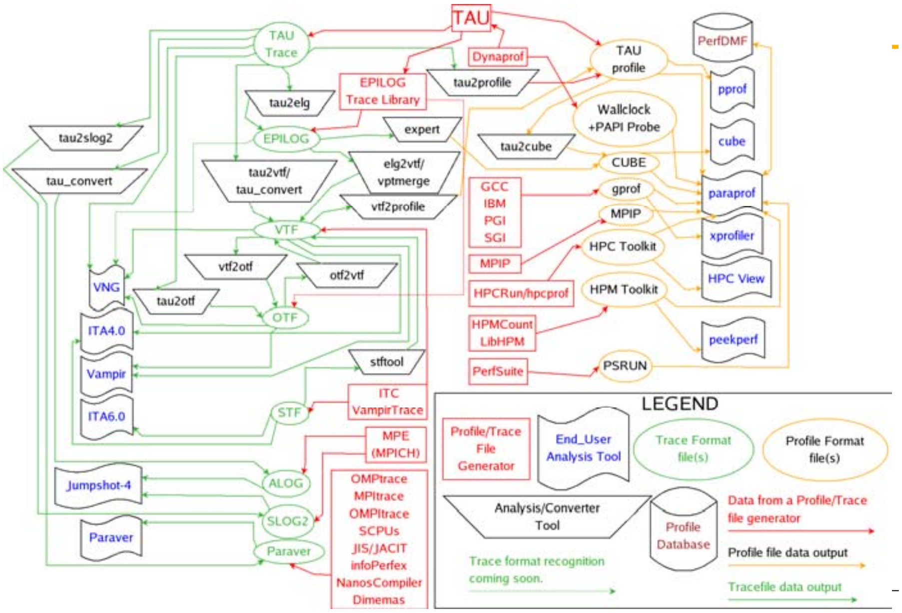
Tracing Analysis and Visualization



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



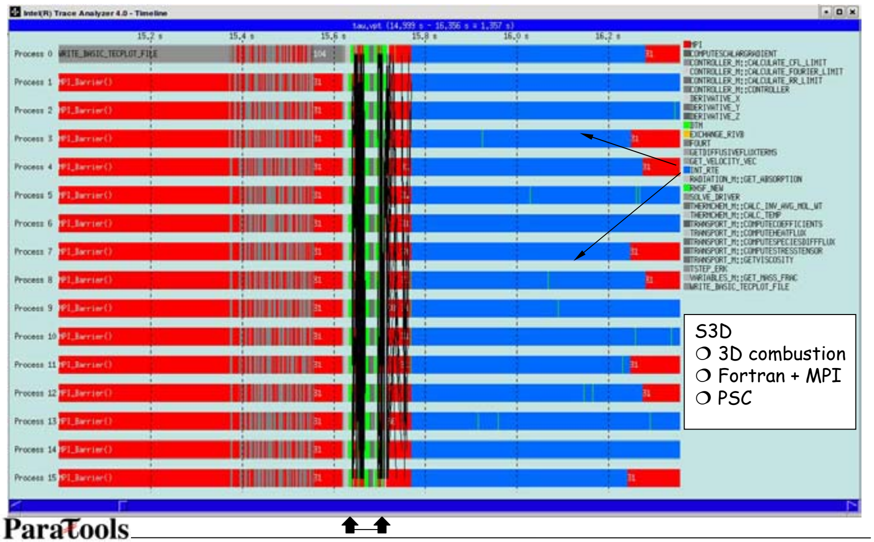
Trace Formats

- 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

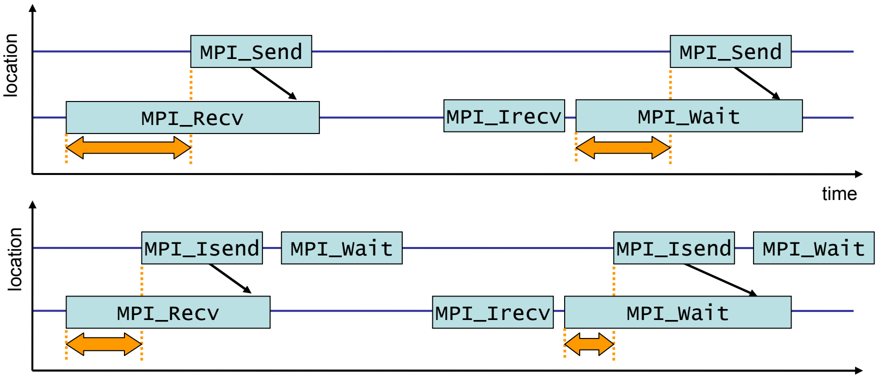
Vampir – Trace Analysis (TAU-to-VTF3) (S3D)



S3D
- ❍ 3D combustion
- ❍ Fortran + MPI
- ❍ PSC

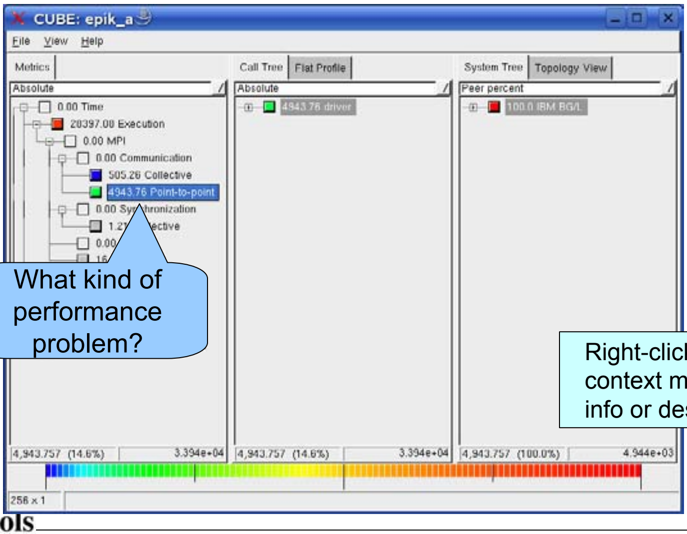
Vampir – Trace Zoomed (S3D)

Scalasca Bottleneck Detection Tool: Late Sender = Early Receive Bottleneck



- Bottleneck: waiting time caused by a blocking receive operation posted earlier than the corresponding send operation. Scalasca has a library of different bottlenecks.
- Scalasca can measure the *yellow* arrows in the trace. It analyzes and generates a profile containing the bottlenecks as metrics (shown in yellow). The profiles may be viewed in TAU's paraprof browser or Scalasca's CUBE browser.

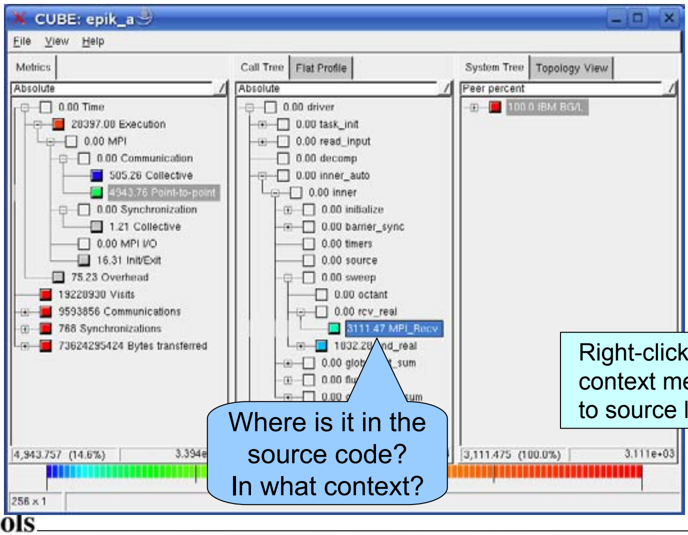
Scalasca: CUBE Profile Browser



What kind of performance problem?

Right-click metric context menu for info or description

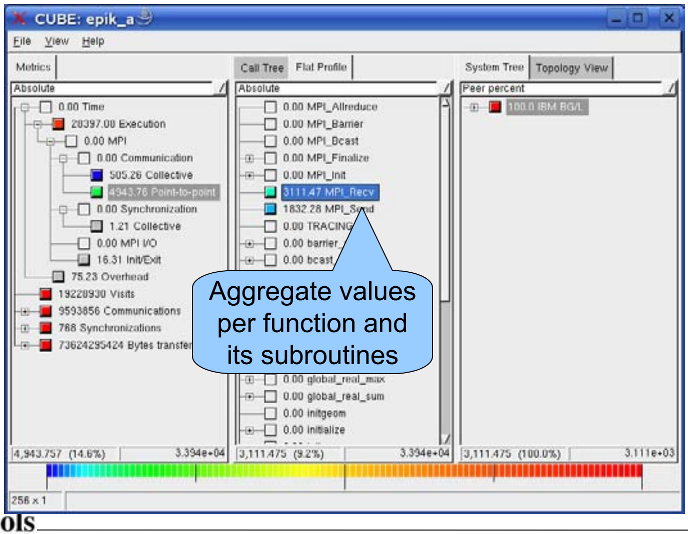
CUBE call tree dimension



Where is it in the source code?
In what context?

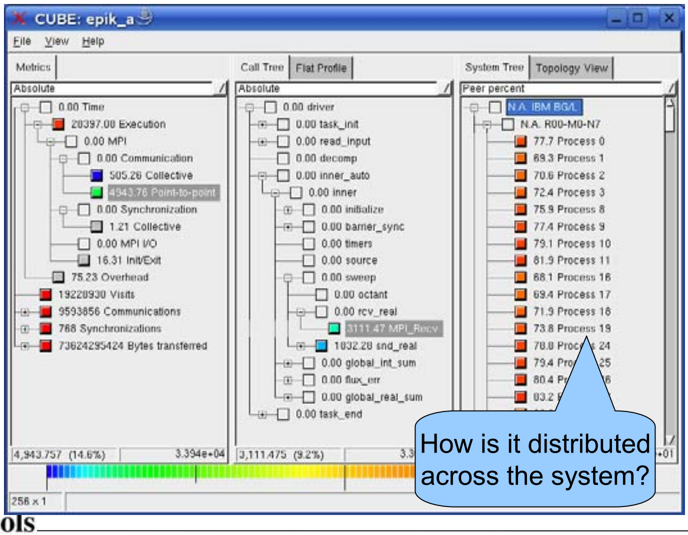
Right-click function context menu to go to source location

Alternative: Flat profile



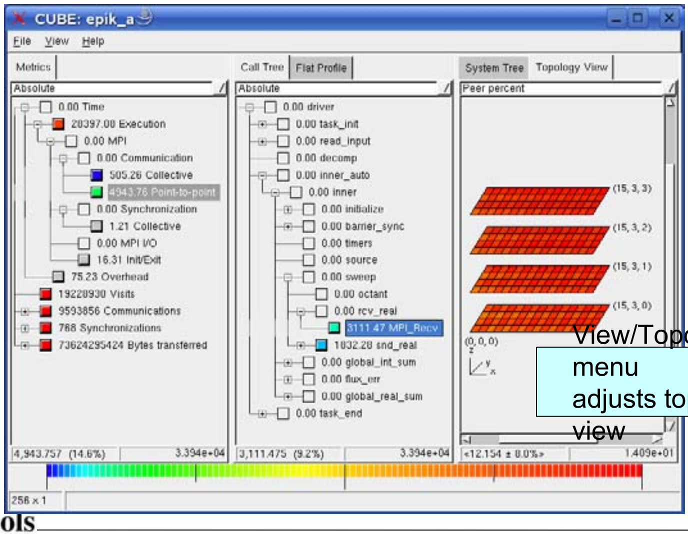
Aggregate values per function and its subroutines

System tree dimension



How is it distributed across the system?

Alternative: Topology display



- 0.00 Time
  - 20397.00 Execution
    - 0.00 MPI
      - 0.00 Communication
        - 505.26 Collective
        - 4943.76 Point-to-point
      - 0.00 Synchronization
        - 1.21 Collective
      - 0.00 MPI I/O
      - 16.31 Init/Exit
  - 75.23 Overhead
- 19220930 Visits
- 9593856 Communications
- 768 Synchronizations
- 73624295424 Bytes transferred

4,943.757 (14.6%) 3.394e+04

- 0.00 driver
  - 0.00 task_init
  - 0.00 read_input
  - 0.00 decomp
  - 0.00 inner_auto
    - 0.00 inner
      - 0.00 initialize
      - 0.00 barrier_sync
      - 0.00 timers
      - 0.00 source
      - 0.00 sweep
        - 0.00 octant
        - 0.00 rcv_real
          - 3111.47 MPI_Recv
        - 1032.28 snd_real
      - 0.00 global_int_sum
      - 0.00 flux_err
      - 0.00 global_real_sum
  - 0.00 task_end

3,111.475 (9.2%) 3.394e+04

<12.154 ± 0.0%> 1.409e+01

View/Topology menu adjusts topology view

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 /usr/global/tools/tau/training/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 /usr/global/tools/tau/training/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…
```

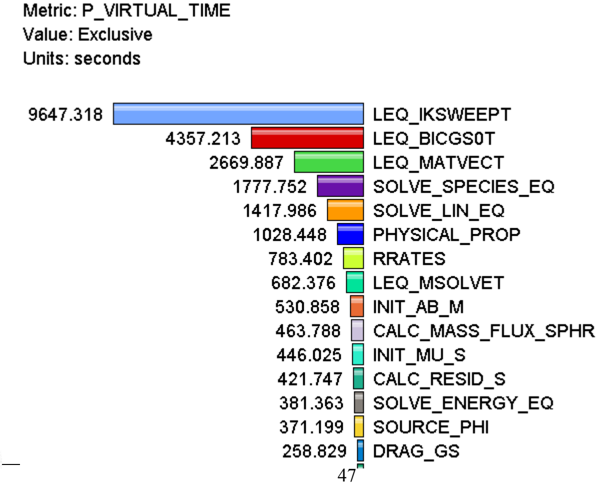
- For an MPI+F90 application, you may want to start with:

Makefile.tau-mpi-pdt

- Supports MPI instrumentation & PDT for automatic source instrumentation
- `% setenv TAU_MAKEFILE /usr/global/tools/tau/training/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:

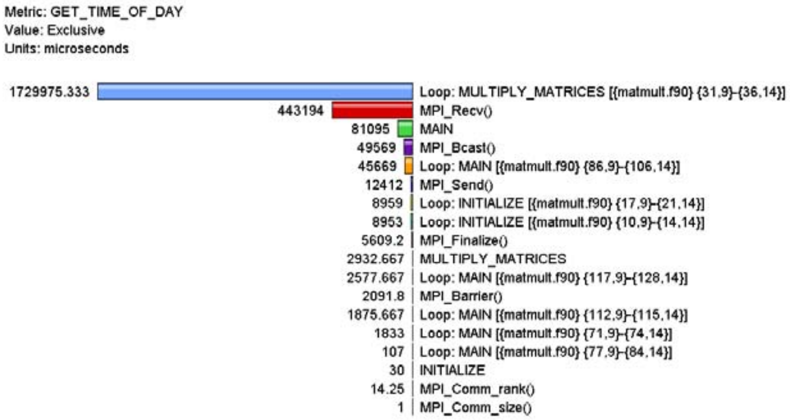


Solution: Generating a flat profile with MPI

```
% setenv TAU_MAKEFILE /usr/global/tools/tau/training/tau-2.18.2/bgp
                              /lib/Makefile.tau-mpi-pdt
% set path=(/usr/global/tools/tau/training/tau-2.18.2/bgp/bin $path)
OR
% source /usr/global/tools/tau/training/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:

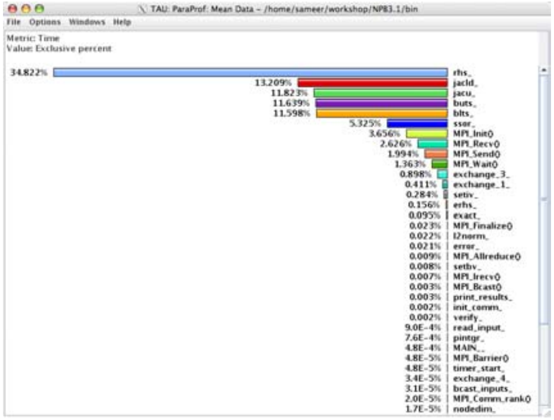


Solution: Generating a loop level profile

```
% setenv TAU_MAKEFILE /usr/global/tools/tau/training/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=(/usr/global/tools/tau/training/tau-2.18.2/bgp/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

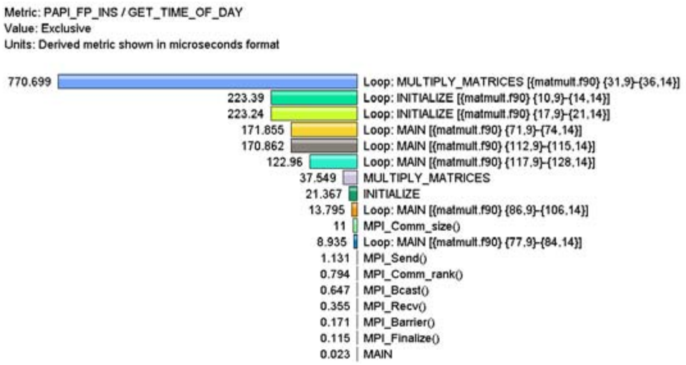


Use Compiler-Based Instrumentation

```
% setenv TAU_MAKEFILE /usr/global/tools/tau/training/tau-2.18.2/bgp
                 /lib/Makefile.tau-mpi
% setenv TAU_OPTIONS '-optCompInst -optVerbose'
% % set path=(/usr/global/tools/tau/training/tau-2.18.2/bgp/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

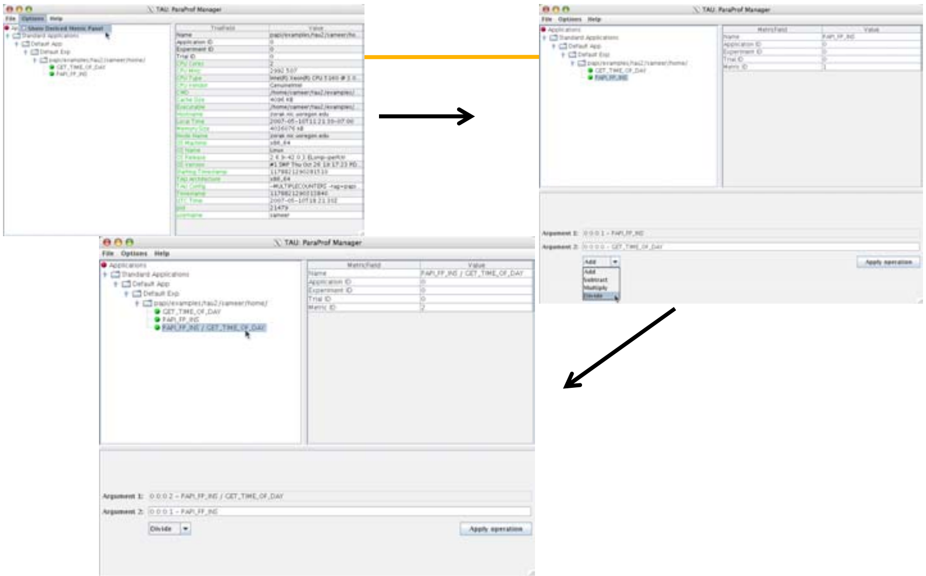
- Goal: What MFlops am I getting in all loops?
- Flat profile with PAPI_FP_INS/OPS and time (-multiplecounters) with loop instrumentation:



Generate a PAPI profile with 2 or more counters

```
% setenv TAU_MAKEFILE /usr/global/tools/tau/training/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=(/usr/global/tools/tau/training/tau-2.18.2/bgp/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
% 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



MetricField	Value
Name	PAPI_FP_INS
Application ID	0
Experiment ID	0
Trial ID	0
Metric ID	1

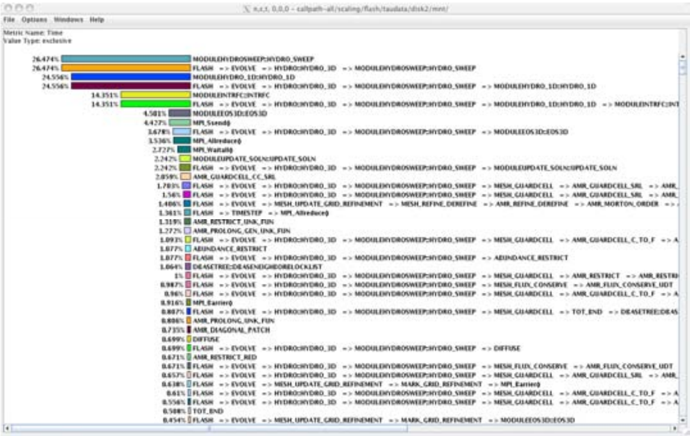
Argument 1: 0:0:0:1 – PAPI_FP_INS
Argument 2: 0:0:0:0 – GET_TIME_OF_DAY

MetricField	Value
Name	PAPI_FP_INS / GET_TIME_OF_DAY
Application ID	0
Experiment ID	0
Trial ID	0
Metric ID	2

Argument 1: 0:0:0:2 – PAPI_FP_INS / GET_TIME_OF_DAY
Argument 2: 0:0:0:1 – PAPI_FP_INS

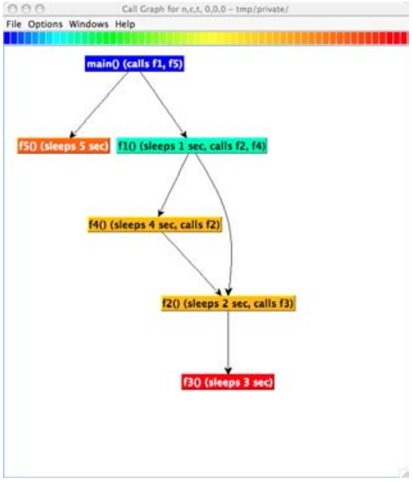
Usage Scenarios: Generating Callpath Profile

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



Callpath Profile

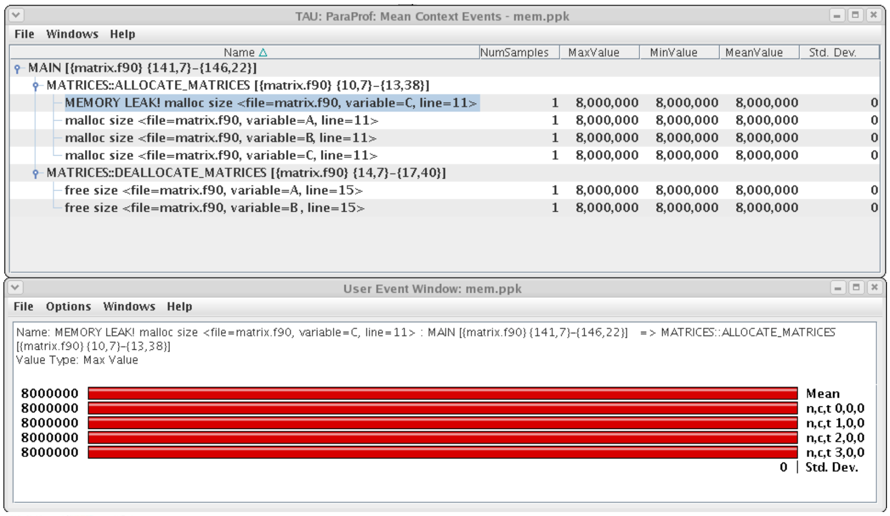
- Generates program callgraph



Generate a Callpath Profile

```
% setenv TAU_MAKEFILE /usr/global/tools/tau/training/tau-2.18.2/bgp
                              /lib/Makefile.tau-callpath-mpi-pdt
% set path=(/usr/global/tools/tau/training/tau-2.18.2/bgp/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



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

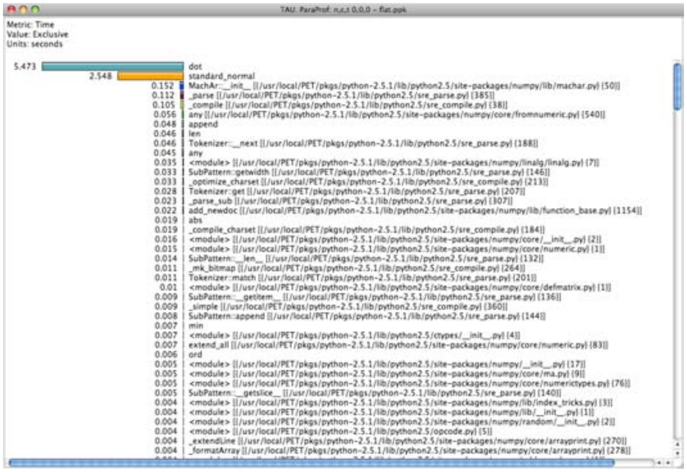
Name: MEMORY LEAK! malloc size <file=matrix.f90, variable=C, line=11> : MAIN [{matrix.f90} {141,7}-{146,22}] => MATRICES::ALLOCATE_MATRICES [{matrix.f90} {10,7}-{13,38}]
Value Type: Max Value

Detect Memory Leaks

```
% setenv TAU_MAKEFILE /usr/global/tools/tau/training/tau-2.18.2/bgp
                          /lib/Makefile.tau-mpi-pdt
% setenv TAU_OPTIONS '-optDetectMemoryLeaks -optVerbose'
% set path=(/usr/global/tools/tau/training/tau-2.18.2/bgp/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:

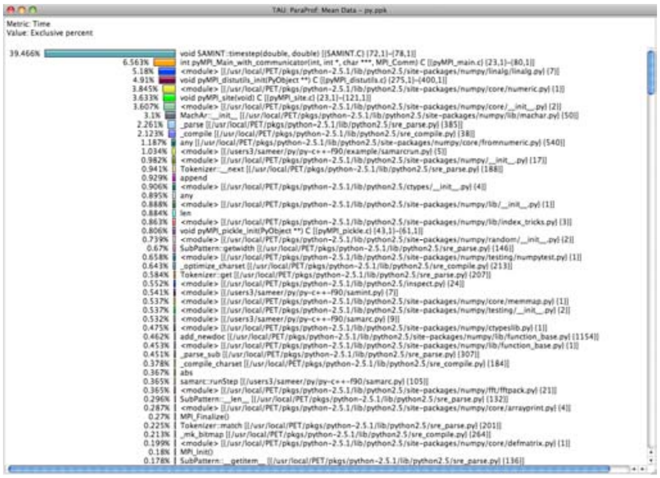
```
% cat wrapper.py
#!/usr/bin/env python

# 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++ ...



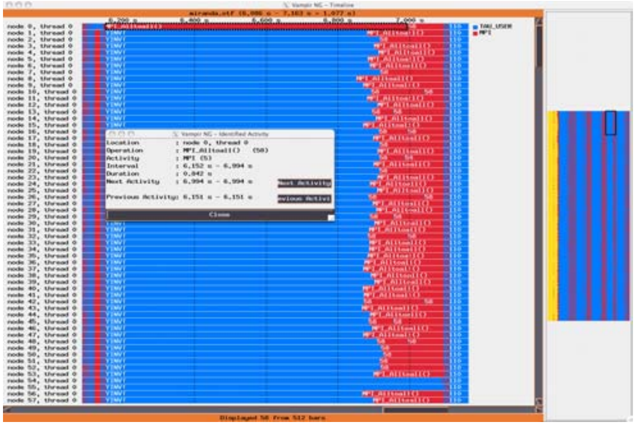
```
Metric: Time
Value: Exclusive percent

 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)
```

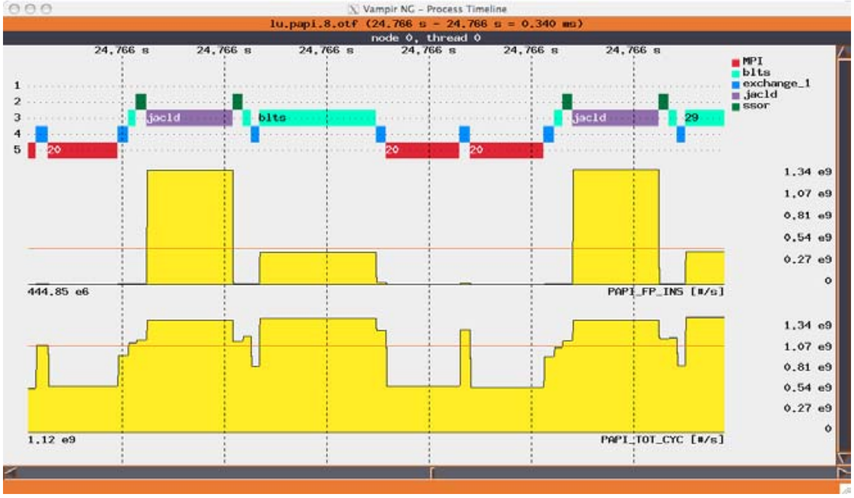
Usage Scenarios: Generating a Trace File

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



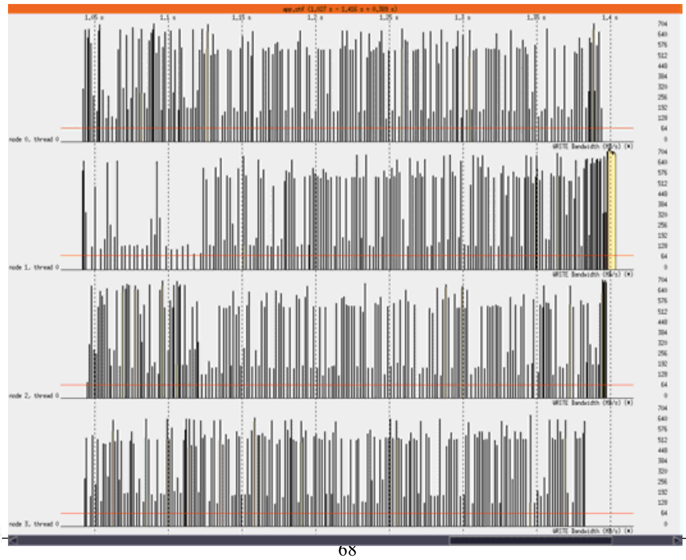
```
Location          : node 0, thread 0
Operation         : MPI_Alltoall()   (58)
Activity          : MPI (5)
Interval          : 6,152 s – 6,994 s
Duration          : 0,842 s
Next Activity     : 6,994 s – 6,994 s

Previous Activity: 6,151 s – 6,151 s
```

VNG Process Timeline with PAPI Counters



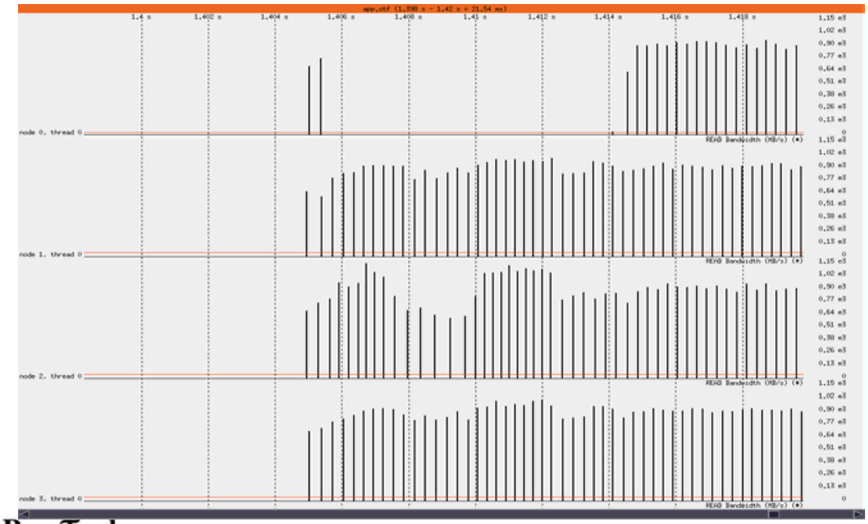
ParaTools

Vampir Counter Timeline Showing I/O BW

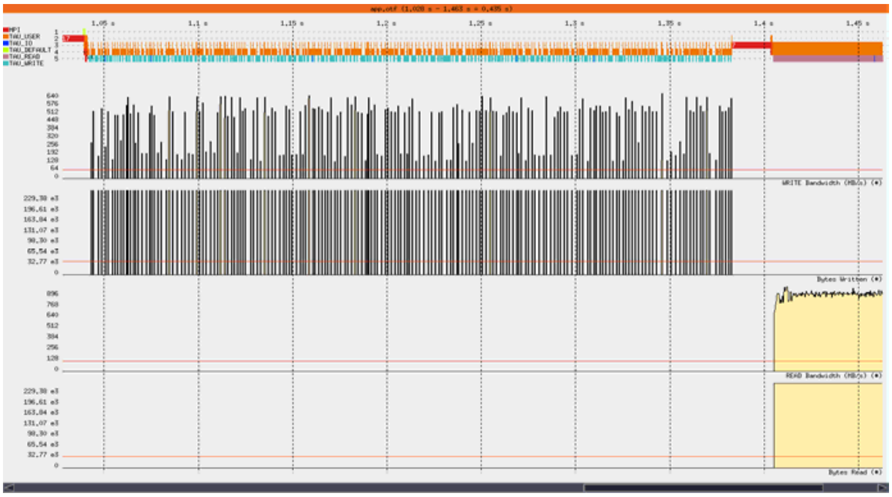


ParaTools

TAU: I/O Instrumentation for Read Bandwidth: Vampir



Vampir Process Timeline for Rank O (IOR, LLNL)

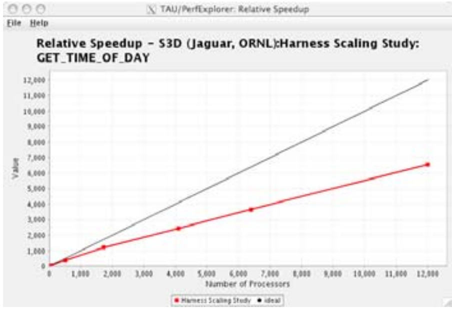


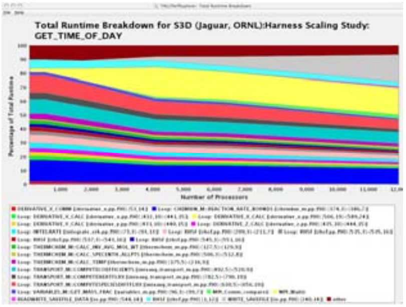
Generate a Trace File

```
% setenv TAU_MAKEFILE /usr/global/tools/tau/training/tau-
2.18.2/bgp
               /lib/Makefile.tau-mpi-pdt-trace
or setenv TAU_TRACE 1 (in TAU v2.18.2+)
% set path=(/usr/global/tools/tau/training/tau-
2.18.2/bgp/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



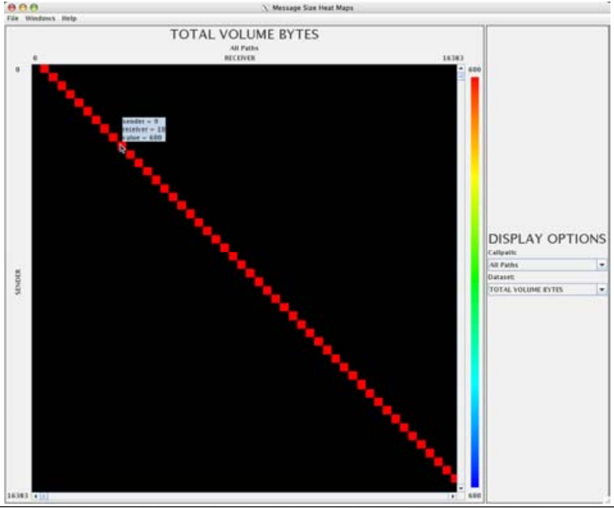


Evaluate Scalability using PerfExplorer Charts

```
% setenv TAU_MAKEFILE /usr/global/tools/tau/training/tau-2.18.2/bgp
                              /lib/Makefile.tau-mpi-pdt
% set path=(/usr/global/tools/tau/training/tau-2.18.2/bgp/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?



Callpath: All Paths

Dataset: TOTAL VOLUME BYTES

Evaluate Scalability using PerfExplorer Charts

```
% setenv TAU_MAKEFILE /usr/global/tools/tau/training/tau-2.18.2/bgp
                              /lib/Makefile.tau-mpi-pdt
% set path=(/usr/global/tools/tau/training/tau-2.18.2/bgp/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

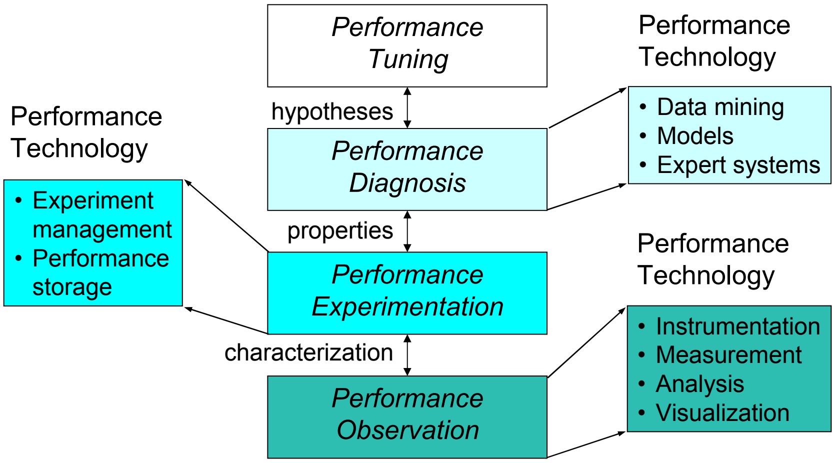
- Add one of
  source /usr/global/tools/tau/training/src/tau.bashrc
  or
  source /usr/global/tools/tau/training/src/tau.cshrc
  to the end of your **.login** file (for bash or csh/tcsh users respectively)
  These files contain LLNL specific location information.

- wget http://www.paratools.com/llnl09/workshop.tar.gz
  or
  cp /usr/global/tools/tau/training/src/workshop.tar.gz .
  and follow the README file.

Part II: Introduction to Performance Engineering

Performance Engineering

- Optimization process
- Effective use of performance technology



Performance Tuning

hypotheses

Performance Diagnosis

properties

Performance Experimentation

characterization

Performance Observation

Performance Technology

- Data mining
- Models
- Expert systems

Performance Technology

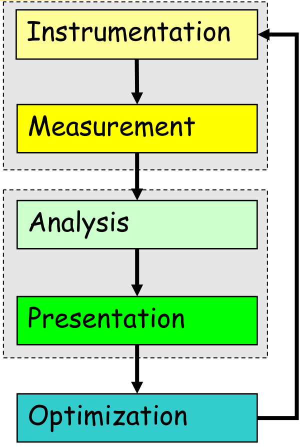
- Experiment management
- Performance storage

Performance Technology

- Instrumentation
- Measurement
- Analysis
- Visualization

Performance Optimization Cycle

- Expose factors
- Collect performance data
- Calculate metrics
- Analyze results
- Visualize results
- Identify problems
- Tune performance



Parallel Performance Properties

- Parallel code performance is influenced by both sequential and parallel factors?
- Sequential factors
  - Computation and memory use
  - Input / output
- Parallel factors
  - Thread / process interactions
  - Communication and synchronization

Performance Observation

- Understanding performance requires observation of performance properties
- Performance tools and methodologies are primarily distinguished by what observations are made and how
  - What aspects of performance factors are seen
  - What performance data is obtained
- Tools and methods cover broad range

Metrics and Measurement

- Observability depends on measurement
- A metric represents a type of measured data
  - Count, time, hardware counters
- A measurement records performance data
  - Associates with program execution aspects
- Derived metrics are computed
  - Rates (e.g., flops)
- Metrics / measurements decided by need

Execution Time

- Wall-clock time
  - Based on realtime clock
- Virtual process time
  - Time when process is executing
    - serial time and system time
  - Does not include time when process is stalled
- Parallel execution time
  - Runs whenever any parallel part is executing
  - Global time basis

Direct Performance Observation

- Execution *actions* exposed as *events*
  - In general, actions reflect some execution state
    - presence at a code location or change in data
    - occurrence in parallelism context (thread of execution)
  - Events encode actions for observation
- Observation is *direct*
  - Direct instrumentation of program code (probes)
  - Instrumentation invokes performance measurement
  - Event measurement = performance data + context
- Performance experiment
  - Actual events + performance measurements

Indirect Performance Observation

- Program code instrumentation is not used
- Performance is observed indirectly
  - Execution is interrupted
    - can be triggered by different events
  - Execution state is queried (sampled)
    - different performance data measured
  - *Event-based sampling* (ESB)
- Performance attribution is inferred
  - Determined by execution context (state)
  - Observation resolution determined by interrupt period
  - Performance data associated with context for period

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

Direct Observation: Instrumentation

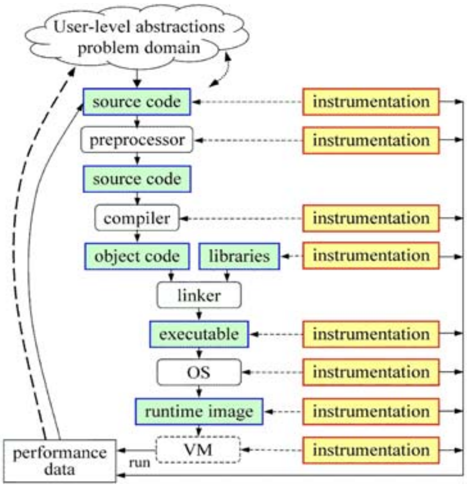
- Events defined by instrumentation access
- Instrumentation levels
  - Source code
  - Object code
  - Runtime system
  - Library code
  - Executable code
  - Operating system
- Different levels provide different information
- Different tools needed for each level
- Levels can have different granularity

Direct Observation: Techniques

- Static instrumentation
  - Program instrumented prior to execution
- Dynamic instrumentation
  - Program instrumented at runtime
- Manual and automatic mechanisms
- Tool required for automatic support
  - Source time: preprocessor, translator, compiler
  - Link time: wrapper library, preload
  - Execution time: binary rewrite, dynamic
- Advantages / disadvantages

Direct Observation: Mapping

- Associate performance data with high-level semantic abstractions
- Abstract events at user-level provide semantic context



Indirect Observation: Events/Triggers

- Events are actions external to program code
  - Timer countdown, HW counter overflow, …
  - Consequence of program execution
  - Event frequency determined by:
    - Type, setup, number enabled (exposed)
- Triggers used to invoke measurement tool
  - Traps when events occur (interrupt)
  - Associated with events
  - May add differentiation to events

Indirect Observation: Context

- When events trigger, execution context determined at time of trap (interrupt)
  - Access to PC from interrupt frame
  - Access to information about process/thread
  - Possible access to call stack
    - requires call stack unwinder
- Assumption is that the context was the same during the preceding period
  - Between successive triggers
  - Statistical approximation valid for long running programs

ParaTools

Direct / Indirect Comparison

- Direct performance observation
  - ☺ Measures performance data exactly
  - ☺ Links performance data with application events
  - 😐 Requires instrumentation of code
  - ☹ Measurement overhead can cause execution intrusion and possibly performance perturbation
- Indirect performance observation
  - ☺ Argued to have less overhead and intrusion
  - ☺ Can observe finer granularity
  - ☺ No code modification required (may need symbols)
  - ☹ Inexact measurement and attribution

ParaTools

Measurement Techniques

- When is measurement triggered?
  - External agent (indirect, asynchronous)
    - interrupts, hardware counter overflow, …
  - Internal agent (direct, synchronous)
    - through code modification
- How are measurements made?
  - Profiling
    - summarizes performance data during execution
    - per process / thread and organized with respect to context
  - Tracing
    - trace record with performance data and timestamp
    - per process / thread

Measured Performance

- Counts
- Durations
- Communication costs
- Synchronization costs
- Memory use
- Hardware counts
- System calls

Critical issues

- Accuracy
  - Timing and counting accuracy depends on resolution
  - Any performance measurement generates overhead
    - Execution on performance measurement code
  - Measurement overhead can lead to intrusion
  - Intrusion can cause perturbation
    - alters program behavior
- Granularity
  - How many measurements are made
  - How much overhead per measurement
- Tradeoff (general wisdom)
  - Accuracy is inversely correlated with granularity

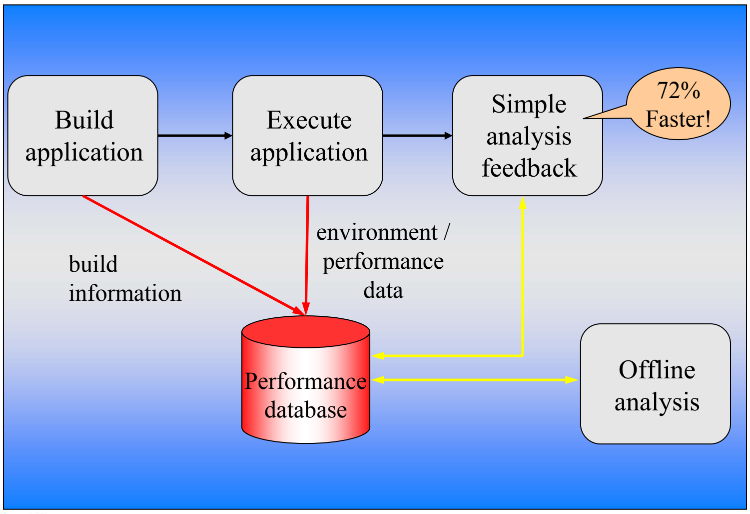
Performance Problem Solving Goals

- Answer questions at multiple levels of interest
  - High-level performance data spanning dimensions
    - machine, applications, code revisions, data sets
    - examine broad performance trends
  - Data from low-level measurements
    - use to predict application performance
- Discover general correlations
  - performance and features of external environment
  - Identify primary performance factors
- Benchmarking analysis for application prediction
- Workload analysis for machine assessment

Performance Analysis Questions

- How does performance vary with different compilers?
- Is poor performance correlated with certain OS features?
- Has a recent change caused unanticipated performance?
- How does performance vary with MPI variants?
- Why is one application version faster than another?
- What is the reason for the observed scaling behavior?
- Did two runs exhibit similar performance?
- How are performance data related to application events?
- Which machines will run my code the fastest and why?
- Which benchmarks predict my code performance best?

Automatic Performance Analysis



Performance Data Management

- Performance diagnosis and optimization involves multiple performance experiments
- Support for common performance data management tasks augments tool use
  - Performance experiment data and metadata storage
  - Performance database and query
- What type of performance data should be stored?
  - Parallel profiles or parallel traces
  - Storage size will dictate
  - Experiment metadata helps in meta analysis tasks
- Serves tool integration objectives

Metadata Collection

- Integration of metadata with each parallel profile
  - Separate information from performance data
- Three ways to incorporate metadata
  - Measured hardware/system information
    - CPU speed, memory in GB, MPI node IDs, …
  - Application instrumentation (application-specific)
    - Application parameters, input data, domain decomposition
    - Capture arbitrary name/value pair and save with experiment
  - Data management tools can read additional metadata
    - Compiler flags, submission scripts, input files, …
    - Before or after execution
- Enhances analysis capabilities

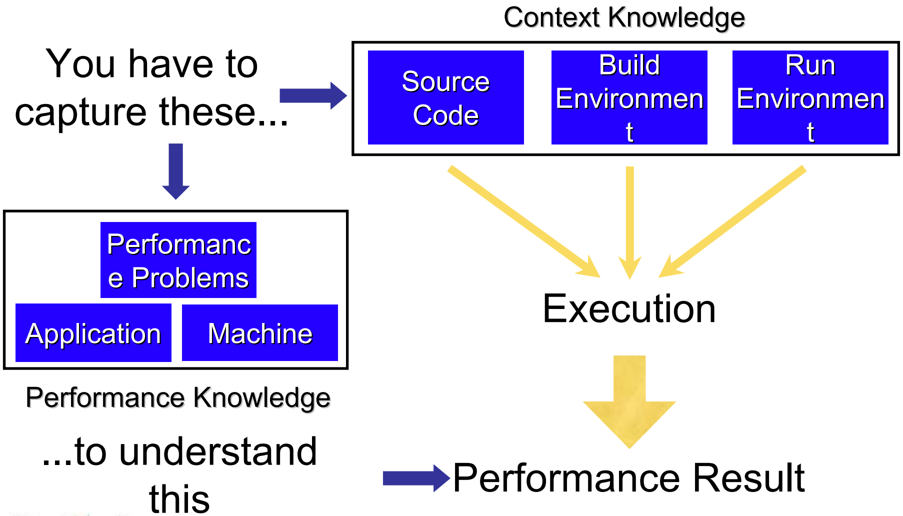
Performance Data Mining

- Conduct parallel performance analysis in a systematic, collaborative and reusable manner
  - Manage performance complexity and automate process
  - Discover performance relationship and properties
  - Multi-experiment performance analysis
- Data mining applied to parallel performance data
  - Comparative, clustering, correlation, characterization, …
  - Large-scale performance data reduction
- Implement extensible analysis framework
  - Abtraction / automation of data mining operations
  - Interface to existing analysis and data mining tools

How to explain performance?

- Should not just redescribed performance results
- Should explain performance phenomena
  - What are the causes for performance observed?
  - What are the factors and how do they interrelate?
  - Performance analytics, forensics, and decision support
- Add *knowledge* to do more intelligent things
  - Automated analysis needs good informed feedback
  - Performance model generation requires interpretation
- Performance knowledge discovery framework
  - Integrating meta-information
  - Knowledge-based performance problem solving

Metadata and Knowledge Role



Context Knowledge

You have to capture these...

Source Code

Build Environment

Run Environment

Performance Problems

Application

Machine

Performance Knowledge

Execution

...to understand this

Performance Result

Performance Optimization Process

- Performance characterization
  - Identify major performance contributors
  - Identify sources of performance inefficiency
  - Utilize timing and hardware measures
- Performance diagnosis (Performance Debugging)
  - Look for conditions of performance problems
  - Determine if conditions are met and their severity
  - What and where are the performance bottlenecks
- Performance tuning
  - Focus on dominant performance contributors
  - Eliminate main performance bottlenecks

Part III:
PAPI

University of Tennessee, Knoxville

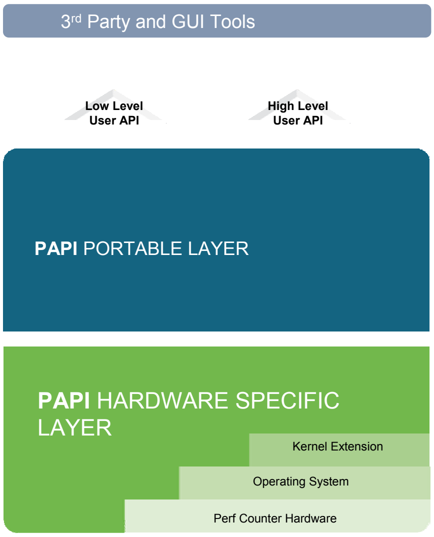
What's PAPI?

- Middleware to provide a consistent programming interface for the performance counter hardware found in most major micro-processors.
- Countable events are defined in two ways:
  - platform-neutral *preset* events
  - Platform-dependent native events
- Presets can be derived from multiple *native events*
- All events are referenced by name and collected in EventSets for sampling
- Events can be multiplexed if counters are limited
- Statistical sampling implemented by:
  - Hardware overflow if supported by the platform
  - Software overflow with timer driven sampling

PAPI Counter Interfaces

PAPI provides 3 interfaces to the underlying counter hardware:

1. A Low Level API manages hardware events in user defined groups called EventSets, and provides access to advanced features.
2. A High Level API provides the ability to start, stop and read the counters for a specified list of events.
3. Graphical and end-user tools provide facile data collection and visualization.



PAPI Preset Events

- Preset Events
  - Standard set of over 100 events for application performance tuning
  - No standardization of the exact definition
  - Mapped to either single or linear combinations of native events on each platform
  - Use papi_avail utility to see what preset events are available on a given platform

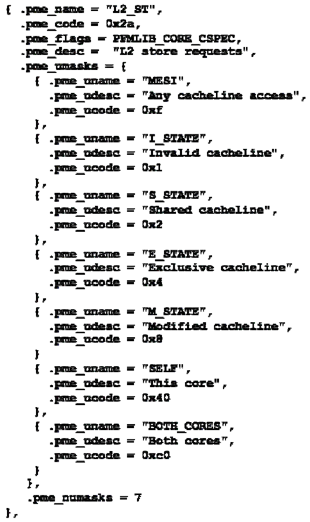
Level 1 Cache

PAPI_L1_DCH:	Level 1 data cache hits
PAPI_L1_DCA:	Level 1 data cache accesses
PAPI_L1_DCR:	Level 1 data cache reads
PAPI_L1_DCW:	Level 1 data cache writes
PAPI_L1_DCM:	Level 1 data cache misses
PAPI_L1_ICH:	Level 1 instruction cache hits
PAPI_L1_ICA:	Level 1 instruction cache accesses
PAPI_L1_ICR:	Level 1 instruction cache reads
PAPI_L1_ICW:	Level 1 instruction cache writes
PAPI_L1_ICM:	Level 1 instruction cache misses
PAPI_L1_TCH:	Level 1 total cache hits
PAPI_L1_TCA:	Level 1 total cache accesses
PAPI_L1_TCR:	Level 1 total cache reads
PAPI_L1_TCW:	Level 1 total cache writes
PAPI_L1_TCM:	Level 1 cache misses
PAPI_L1_LDM:	Level 1 load misses
PAPI_L1_STM:	Level 1 store misses

PAPI Native Events

- Native Events
  - Any event countable by the CPU
  - Same interface as for preset events
  - Use *papi_native_avail* utility to see all available native events
- Use *papi_event_chooser* utility to select a compatible set of events

PRESET,
PAPI_L2_DCA,
DERIVED_ADD,
L2_LD:SELF:ANY:MESI,
L2_ST:SELF:MESI



```
{  .pme_name = "L2_ST",
   .pme_code = 0x2a,
   .pme_flags = PFMLIB_CORE_CSPEC,
   .pme_desc =  "L2 store requests",
   .pme_umasks = {
     {  .pme_uname = "MESI",
        .pme_udesc = "Any cacheline access",
        .pme_ucode = 0xf
     },
     {  .pme_uname = "I_STATE",
        .pme_udesc = "Invalid cacheline",
        .pme_ucode = 0x1
     },
     {  .pme_uname = "S_STATE",
        .pme_udesc = "Shared cacheline",
        .pme_ucode = 0x2
     },
     {  .pme_uname = "E_STATE",
        .pme_udesc = "Exclusive cacheline",
        .pme_ucode = 0x4
     },
     {  .pme_uname = "M_STATE",
        .pme_udesc = "Modified cacheline",
        .pme_ucode = 0x8
     }
     {  .pme_uname = "SELF",
        .pme_udesc = "This core",
        .pme_ucode = 0x40
     },
     {  .pme_uname = "BOTH_CORES",
        .pme_udesc = "Both cores",
        .pme_ucode = 0xc0
     }
    },
    .pme_numasks = 7
},
```

PAPI & Multicore

- Multicore is the (near term) future of Petascale computing
- Minimizing resource contention will be key
  - Memory bandwidth
  - Cache sharing
  - Bus and other resource contention

Multicore counter support

- AMD Barcelona
  - 4 L3 shared cache events:
    - READ_REQUEST_TO_L3_CACHE
    - L3_CACHE_MISSES
    - L3_FILLS_CAUSED_BY_L2_EVICTIONS
    - L3_EVICTIONS
  - First 3 are qualified per core
- Intel Core2 series:
  - SELF/ANY
  - L2 shared cache, bus, snoop
  - 39 events/~140 are core qualified
- Itanium Montecito:
  - SELF/ANY
  - 50 bus events (~1/6) are core qualified

Extending PAPI beyond the CPU

- PAPI has historically targeted on on-processor performance counters
- Several categories of off-processor counters exist
  - network interfaces: Myrinet, Infiniband, GigE
  - memory interfaces: Cray X1, SeaStar
  - thermal and power interfaces: ACPI, lm-sensors
  - accelerators?
- CHALLENGE:
  - Extend the PAPI interface to address multiple counter domains
  - Preserve the PAPI calling semantics, ease of use, and platform independence for existing applications

Motivation

- Performance counters also exist in off-cpu resources
- All information is valuable for performance optimization
- Increasing cpu counts & power demands place greater importance on:
  - Thermal health and management
  - Power consumption
- Multicore systems require careful resource balancing
- Higher processor & core counts make communications metrics more critical:
  - Bandwidth
  - Latency
  - Dropped packets
  - Bytes transferred

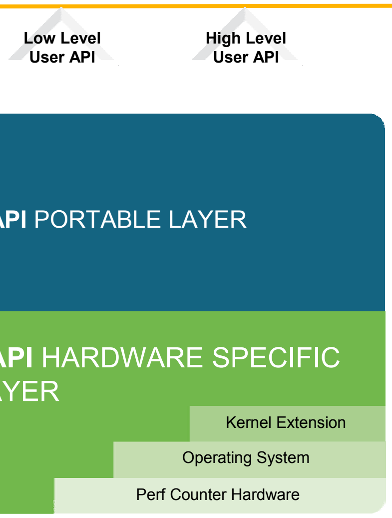
Limitations

- Interfaces are often obscure, unexposed or non-standard
- Performance data (accelerators) can be vastly different than cpus
- Measurements are usually system-wide and asynchronous
  - May not matter on dedicated single-task OS's like Cray Catamount and Blue Gene CNK
  - But matters more for Multicore
- Often very different time scales

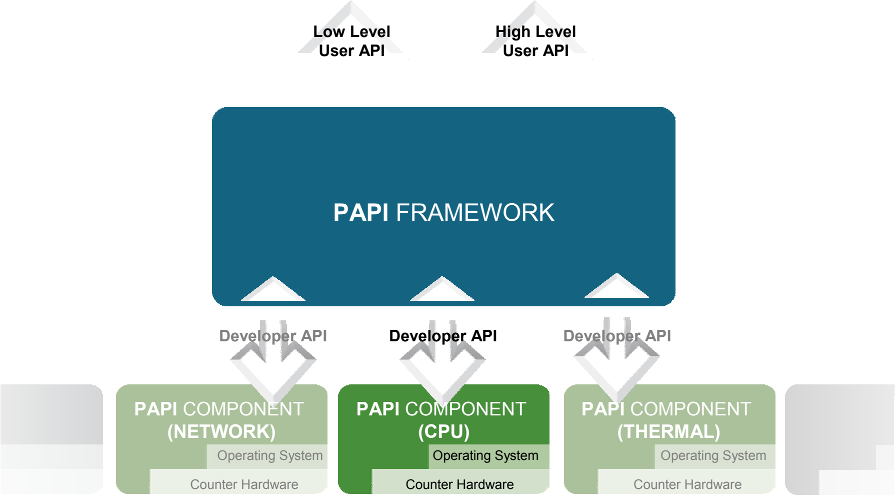
Component PAPI Goals

- Support simultaneous access to on- and off-processor counters
- Isolate hardware dependent code in separable ‘component’ modules
- Extend platform independent code to support multiple simultaneous components
- Add or modify API calls to support access to any of several components
- Modify build environment for easy selection and configuration of multiple available components

Monolithic ‘**PAPI** Classic’



Component **PAPI**



Low Level User API

High Level User API

PAPI FRAMEWORK

Developer API

Developer API

Developer API

PAPI COMPONENT **(NETWORK)**

Operating System

Counter Hardware

PAPI COMPONENT **(CPU)**

Operating System

Counter Hardware

PAPI COMPONENT **(THERMAL)**

Operating System

Counter Hardware

For more information

- PAPI Website: http://icl.cs.utk.edu/papi/
  - Software
  - Release notes
  - Documentation
  - Links to tools that use PAPI
  - Mailing/discussion lists

TAU Performance System®

Part IV: TAU Internals

Performance Tools FAQ/Concerns

- Does it automatically instrument my code? At the routine level? At the outer-loop level?
- Can it show me where time is spent in my code? PAPI Flops? L1 data cache misses? Can I measure more than one quantity in a trial?
- Does the tool support profiling (runtime summarization) as well as tracing (time-line based displays)? What about profile snapshots? Callpath (parent-child) profiles? Can I use it to easily benchmark codes?
- Can I observe the performance data at runtime as the application executes?
- Can it show me memory utilization? Memory leaks? Mallocs/frees? When and where?
- What about I/O? Can I observe bandwidth of reads/writes? Volume of I/O? What about Kernel events? User space+Kernel?
- What is the typical overhead? Can I reduce it to < 5%? < 1%? Can it compensate and remove timer overhead from performance data? Can it throttle away instrumentation in lightweight routines at runtime to reduce overhead?
- I already have profile data from <XYZ> tool. Can it import my legacy data?
- I prefer <XYZ> performance tool for visualization. Can it hook up with this tool? Are there converters?

Performance Tools FAQ/Concerns (contd.)

- Can I use it for multi-core CPUs? Compare the performance of application running on a single vs. multi-core processor? Can I observe multi-core data snoops, invalidates?
- Can I share the performance data with my colleagues in a secure manner (web/database)? Can it automatically track progress of my application over time (~ 6 mos)? Can I use it for scalability studies? Over multiple platforms?
- Are the GUI client tools available under Linux? MS Windows? Apple?
- Does it run on all Cray, IBM, SGI, HP … platforms? CNL? Catamount?
- Does it support MPI? MPI2? Threads? Hybrid MPI+Pthreads/MPI+OpenMP?
- Does it support Fortran? C++, C? Java? Python? Python+MPI+F90+C++…?
- Does it support Intel/PGI/PathScale/IBM/Cray/Sun compilers?
- Are tools available in command-line form & GUI? IDE GUI? Web-based? 3D?
- Is it already installed and supported on my HPC system? What about systems at NERSC? ANL? LLNL? LANL? NASA? DoD? NSF sites?...
- Is there support (phone/e-mail) available for the tool? Professional support? For instrumentation? Analysis?
- Will it work on the new <XYZ> HPC platform scheduled for release six months from now?
- Is it free? BSD license? …

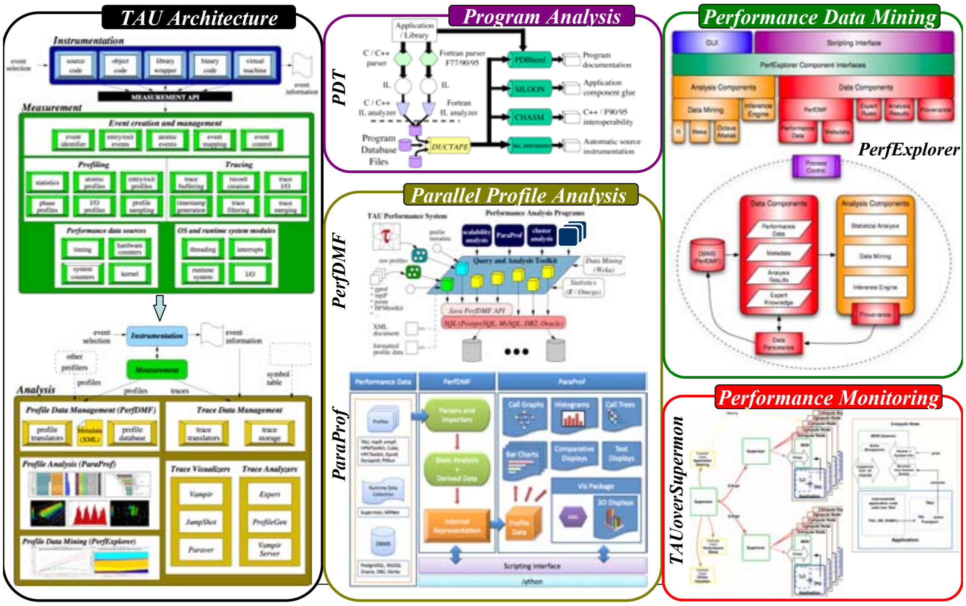
TAU Performance System® Project

- ***T***uning and ***A***nalysis ***U***tilities (15+ year project effort)
- **Performance system framework for HPC systems**
  - Integrated, scalable, and flexible
  - Target parallel programming paradigms
- **Integrated toolkit for performance problem solving**
  - Instrumentation, measurement, analysis, and visualization
  - Portable performance profiling and tracing facility
  - Performance data management and data mining
- ***Partners***
  - LLNL, ANL, LANL
  - Research Centre Jülich, TU Dresden

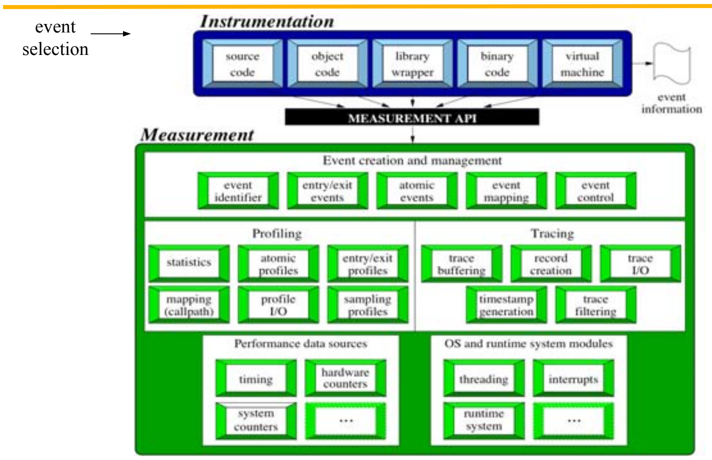
TAU Parallel Performance System Goals

- **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



event selection

Instrumentation

source code

object code

library wrapper

binary code

virtual machine

event information

MEASUREMENT API

Measurement

Event creation and management

event identifier

entry/exit events

atomic events

event mapping

event control

Profiling

statistics

atomic profiles

entry/exit profiles

mapping (callpath)

profile I/O

sampling profiles

Tracing

trace buffering

record creation

trace I/O

timestamp generation

trace filtering

Performance data sources

timing

hardware counters

system counters

...

OS and runtime system modules

threading

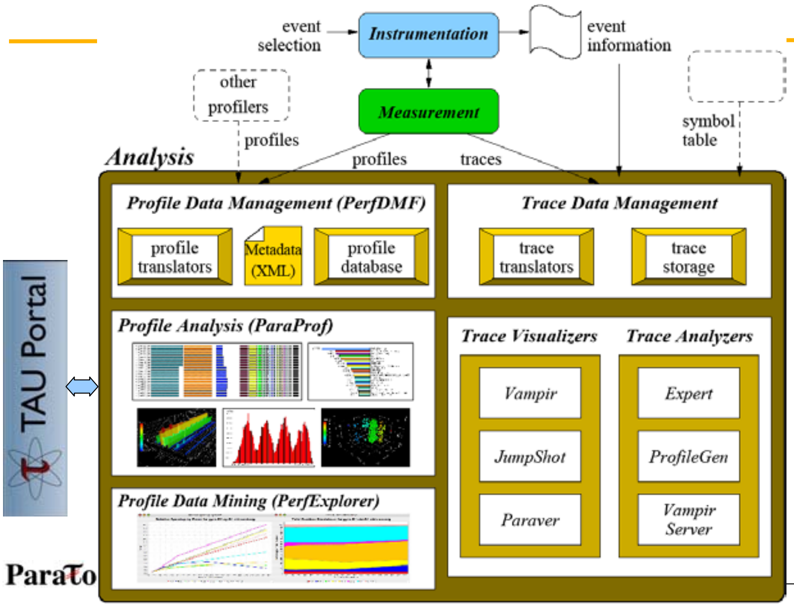
interrupts

runtime system

...

ParaTools

TAU Performance System Architecture



event selection

Instrumentation

event information

other profilers

Measurement

symbol table

profiles

Analysis

profiles

traces

Profile Data Management (PerfDMF)

profile translators

Metadata (XML)

profile database

Trace Data Management

trace translators

trace storage

Profile Analysis (ParaProf)

Trace Visualizers

Vampir

JumpShot

Paraver

Trace Analyzers

Expert

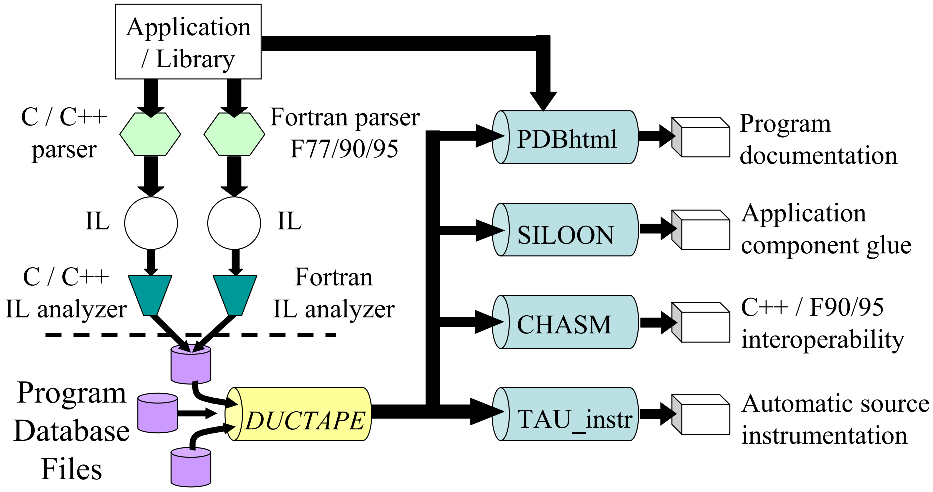
ProfileGen

Vampir Server

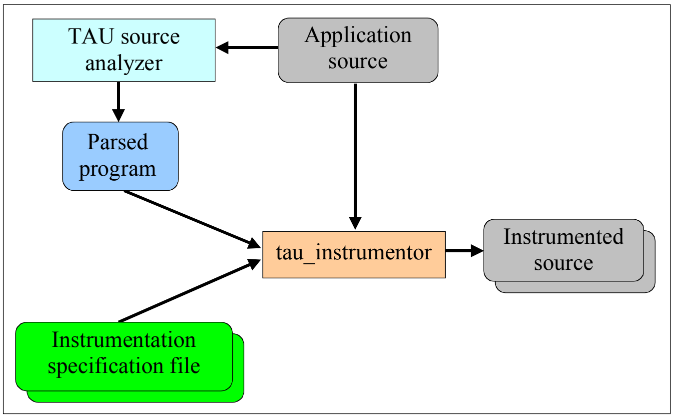
Profile Data Mining (PerfExplorer)

TAU Portal

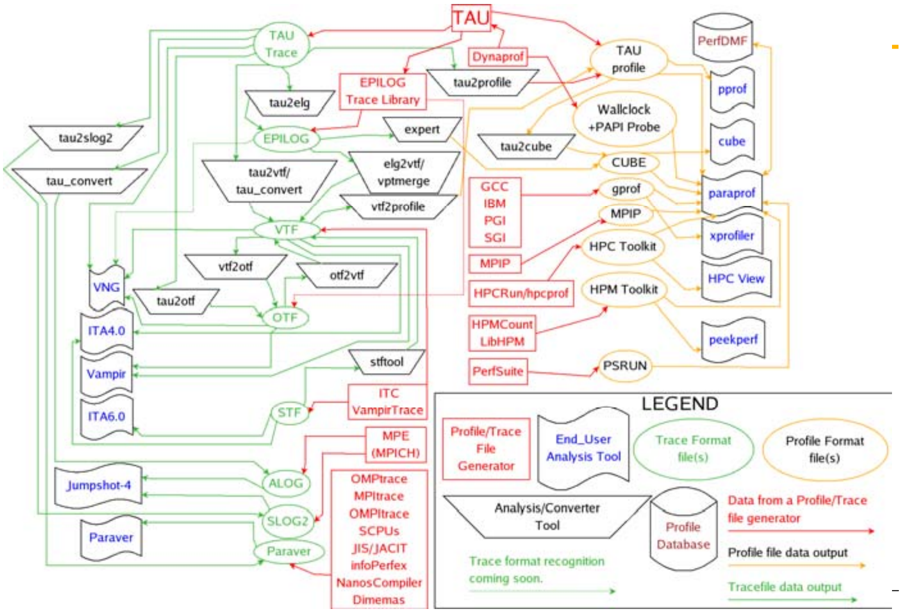
Program Database Toolkit (PDT)



Automatic Source-Level Instrumentation in TAU



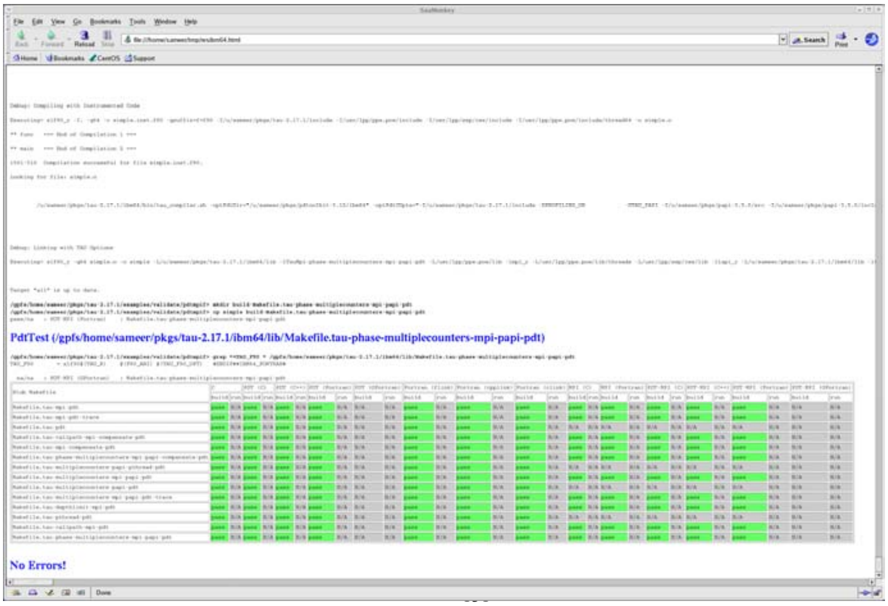
Building Bridges to Other Tools



Installing TAU on 64bit AIX

- Install PAPI and PDT
  - PAPI:
    - ./configure –prefix=$HOME/pkgs/papi-3.5.0;
    - Make ; make install
  - PDT:
    - ./configure –prefix=$HOME/pkgs/pdt-3.13 –PGI
    - make; make install
- Install TAU:
  - ./installtau –pdt=$HOME/pkgs/pdt-3.13 –papi=$HOME/pkgs/papi-3.5.0 -c++=pgCC -cc=pgcc -fortran=pgi –mpiinc=<dir> -mpilib=<dir>
    - Configures multiple typically requested versions for you in ia64/lib/Makefile.tau-* configurations
  - tau_validate –html –build x86_64 >& results.html
  - mozilla results.html

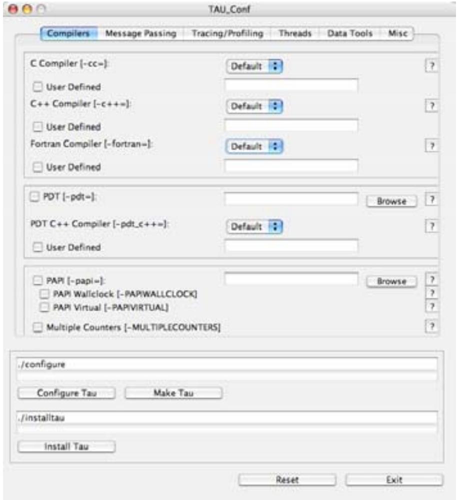
Validating an Install



PdtTest (/gpfs/home/sameer/pkgs/tau-2.17.1/ibm64/lib/Makefile.tau-phase-multiplecounters-mpi-papi-pdt)

No Errors!

TAU_SETUP: A GUI for Installing TAU



C Compiler [-cc=]: Default

☐ User Defined

C++ Compiler [-c++=]: Default

☐ User Defined

Fortran Compiler [-fortran=]: Default

☐ User Defined

☐ PDT [-pdt=]:

PDT C++ Compiler [-pdt_c++=]: Default

☐ User Defined

☐ PAPI [-papi=]:

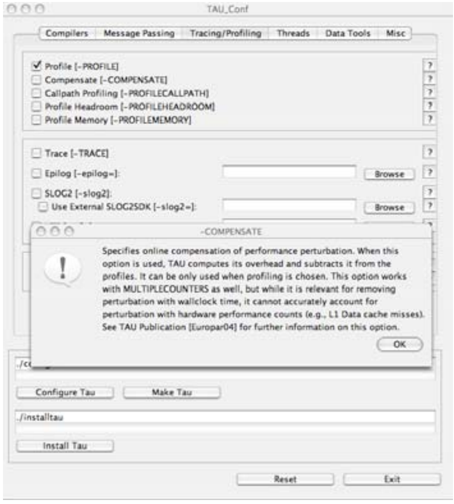
☐ PAPI Wallclock [-PAPIWALLCLOCK]

☐ PAPI Virtual [-PAPIVIRTUAL]

☐ Multiple Counters [-MULTIPLECOUNTERS]

./configure

./installtau



☑ Profile [-PROFILE]

☐ Compensate [-COMPENSATE]

☐ Callpath Profiling [-PROFILECALLPATH]

☐ Profile Headroom [-PROFILEHEADROOM]

☐ Profile Memory [-PROFILEMEMORY]

☐ Trace [-TRACE]

☐ Epilog [-epilog=]:

☐ SLOG2 [-slog2]:

☐ Use External SLOG2SDK [-slog2=]:

Specifies online compensation of performance perturbation. When this option is used, TAU computes its overhead and subtracts it from the profiles. It can be only used when profiling is chosen. This option works with MULTIPLECOUNTERS as well, but while it is relevant for removing perturbation with wallclock time, it cannot accurately account for perturbation with hardware performance counts (e.g., L1 Data cache misses). See TAU Publication [Europar04] for further information on this option.

./installtau

Upgrading TAU v2.18 configurations to 2.18.2

- Upgrade TAU
  - Previous installation in $HOME/pkgs/tau-2.18
    - `cd tau-2.18.2`
    - `./upgradetau   /usr/global/tools/pkgs/tau-2.18`
      - Builds all previous configurations in the current dir
      - You may also upgrade with a new package say PDT 3.14.1
    -
      ```
      ./upgradetau /usr/global/tools/pkgs/tau-2.18 -
      pdt=/usr/global/tools/pkgs/pdtoolkit-3.14.1
      ```

- Validate your new installation
    - `./tau_validate -html -build x86_64 >& results.html`
    - ``mozilla `pwd`/results.html``

Using TAU

- Install TAU
  % ./configure [options]; make clean install
- Replace the names of your compiler with tau_f90.sh, tau_cxx.sh and tau_cc.sh in your makefiles
- Set environment variables
  - Choose the measurement option and compile your code:
    - setenv TAU_MAKEFILE $TAU/Makefile.tau-mpi-pdt
    - setenv TAU_OPTIONS '-optVerbose -optKeepFiles -optPreProcess'
  - At runtime, if more than one metric is measured (-multiplecounters):
    - setenv COUNTER1 GET_TIME_OF_DAY
    - setenv COUNTER2 PAPI_FP_INS
    - setenv COUNTER3 PAPI_NATIVE_<native_name>
      - Use papi_native_avail, papi_avail, and papi_event_chooser to select these preset and native event names
- Build the application, run it, analyze performance data

Using TAU: A brief Introduction

- To instrument source code:
  % setenv TAU_MAKEFILE /usr/global/tools/tau/training/tau-2.18.2/bgp/lib/Makefile.tau-mpi-pdt
  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 then run:
  % pprof (for text based profile display)
  % paraprof (for GUI)
- LABS:
  % source /usr/global/tools/tau/training/src/tau.cshrc
  % cp /usr/global/tools/tau/training/src/workshop.tar.gz .
  and follow instructions in README file

TAU Instrumentation Approach

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

TAU Instrumentation

- Flexible instrumentation mechanisms at multiple levels
  - Source code
    - manual (TAU API, TAU Component API)
    - automatic
      - C, C++, F77/90/95 (Program Database Toolkit (*PDT*))
      - OpenMP (directive rewriting (*Opari*), *POMP spec*)
  - Object code
    - pre-instrumented libraries (e.g., MPI using *PMPI*)
    - statically-linked and dynamically-linked
  - Executable code
    - dynamic instrumentation (pre-execution) (*DynInstAPI*)
    - virtual machine instrumentation (e.g., Java using *JVMPI*)
    - Python interpreter based instrumentation at runtime
  - Proxy Components

TAU Measurement Approach

- Portable and scalable parallel profiling solution
  - Multiple profiling types and options
  - Event selection and control (enabling/disabling, throttling)
  - Online profile access and sampling
  - Online performance profile overhead compensation
- Portable and scalable parallel tracing solution
  - Trace translation to Open Trace Format (OTF)
  - Trace streams and hierarchical trace merging
- Robust timing and hardware performance support
- Multiple counters (hardware, user-defined, system)
- Performance measurement for CCA component software

Using TAU

- Configuration
- Instrumentation
  - Manual
  - MPI – Wrapper interposition library
  - PDT- Source rewriting for C,C++, F77/90/95
  - Compiler-based instrumentation for C, C++, F90
  - OpenMP – Directive rewriting
  - Component based instrumentation – Proxy components
  - Binary Instrumentation
    - DyninstAPI – Runtime Instrumentation/Rewriting binary
    - Java – Runtime instrumentation
    - Python – Runtime instrumentation
- Measurement
- Performance Analysis

TAU Measurement System Configuration

- configure [OPTIONS]

{-c++=<CC>, -cc=<cc>}	Specify C++ and C compilers
-pdt=<dir>	Specify location of PDT
-opari=<dir>	Specify location of Opari OpenMP tool
-papi=<dir>	Specify location of PAPI
-vampirtrace=<dir>	Specify location of VampirTrace
-mpi[inc/lib]=<dir>	Specify MPI library instrumentation
-dyninst=<dir>	Specify location of DynInst Package
-shmem[inc/lib]=<dir>	Specify PSHMEM library instrumentation
-python[inc/lib]=<dir>	Specify Python instrumentation
-tag=<name>	Specify a unique configuration name
-epilog=<dir>	Specify location of EPILOG
-slog2	Build SLOG2/Jumpshot tracing package
-otf=<dir>	Specify location of OTF trace package
-arch=<architecture>	Specify architecture explicitly (bgl, xt3,ibm64,ibm64linux…)
{-pthread, -sproc}	Use pthread or SGI sproc threads
-openmp	Use OpenMP threads
-jdk=<dir>	Specify Java instrumentation (JDK)
-fortran=[vendor]	Specify Fortran compiler

TAU Measurement System Configuration

- configure [OPTIONS]

-TRACE	Generate binary TAU traces
-PROFILE (default)	Generate profiles (summary)
-PROFILECALLPATH	Generate call path profiles
-PROFILEPHASE	Generate phase based profiles
-PROFILEMEMORY	Track heap memory for each routine
-PROFILEHEADROOM	Track memory headroom to grow
-MULTIPLECOUNTERS	Use hardware counters + time
-COMPENSATE	Compensate timer overhead
-CPUTIME	Use usertime+system time
-PAPIWALLCLOCK	Use PAPI’s wallclock time
-PAPIVIRTUAL	Use PAPI’s process virtual time
-SGITIMERS	Use fast IRIX timers
-LINUXTIMERS	Use fast x86 Linux timers

TAU Measurement Configuration – Examples

- ./configure -arch=x86_64 –pdt=/usr/global/tools/pkgs/pdtoolkit-3.14 -mpi Configure using PDT and MPI
- ./configure -arch=x86_64 -papi=/usr/global/tools/pkgs/papi-3.6.2 -pdt=<dir> -mpi -MULTIPLECOUNTERS; make clean install
  - Use PAPI counters (one or more) with C/C++/F90 automatic instrumentation. Also instrument the MPI library.
- Typically configure multiple measurement libraries
- Each configuration creates a unique <arch>/lib/Makefile.tau<options> stub makefile. It corresponds to the configuration options used. e.g.,
  - $(PET_HOME)/tau/x86_64/lib/Makefile.tau-mpi-pdt
  - $(PET_HOME)/tau/x86_64/lib/Makefile.tau-multiplecounters-mpi-papi-pdt

TAU Measurement Configuration – Examples

```
% cd $(PET_HOME)/tau/x86_64/lib; ls Makefile.*pgi
Makefile.tau-pdt
Makefile.tau-mpi-pdt
Makefile.tau-callpath-mpi-pdt
Makefile.tau-mpi-pdt-trace
Makefile.tau-mpi-compensate-pdt
Makefile.tau-multiplecounters-mpi-papi-pdt
Makefile.tau-multiplecounters-mpi-papi-pdt-trace
Makefile.tau-mpi-papi-pdt-epilog-scalasca-trace
Makefile.tau-pdt…
```

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

Makefile.tau-mpi-pdt

  - Supports MPI instrumentation & PDT for automatic source instrumentation for PGI compilers

Configuration Parameters in Stub Makefiles

- Each TAU stub Makefile resides in <tau>/<arch>/lib directory
- Variables:

TAU_CXX	Specify the C++ compiler used by TAU
TAU_CC, TAU_F90	Specify the C, F90 compilers
TAU_DEFS	Defines used by TAU. Add to CFLAGS
TAU_LDFLAGS	Linker options. Add to LDFLAGS
TAU_INCLUDE	Header files include path. Add to CFLAGS
TAU_LIBS	Statically linked TAU library. Add to LIBS
TAU_SHLIBS	Dynamically linked TAU library
TAU_MPI_LIBS	TAU’s MPI wrapper library for C/C++
TAU_MPI_FLIBS	TAU’s MPI wrapper library for F90
TAU_FORTRANLIBS	Must be linked in with C++ linker for F90
TAU_CXXLIBS	Must be linked in with F90 linker
TAU_INCLUDE_MEMORY	Use TAU’s malloc/free wrapper lib
TAU_DISABLE	TAU’s dummy F90 stub library
TAU_COMPILER	Instrument using tau_compiler.sh script

- Each stub makefile encapsulates the parameters that TAU was configured with
- It represents a specific instance of the TAU libraries. TAU scripts use stub makefiles to identify what performance measurements are to be performed.

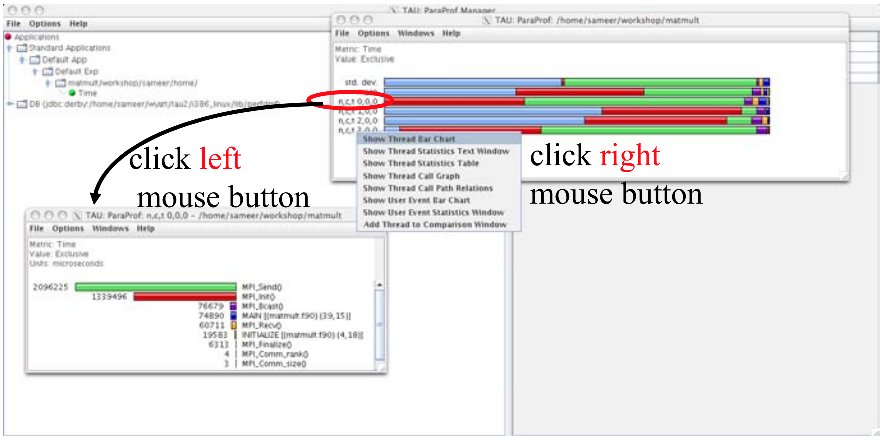
Using TAU

- Install TAU
  % configure [options]; make clean install
- Typically modify application makefile and choose TAU configuration
  - Select TAU's stub makefile, change name of compiler in Makefile

  % setenv TAU_MAKEFILE /usr/global/tools/tau/training/tau-2.18.2/bgp/lib/Makefile.tau-mpi-pdt

  % setenv TAU_OPTIONS '-optVerbose -optKeepFiles ...'
  - F90 = tau_f90.sh CXX = tau_cxx.sh CC = tau_cc.sh
- Set environment variables
  - Directory where profiles/traces are to be stored/counter selection
- Execute application
  % qsub run.cray.job
- Analyze performance data
  - paraprof, vampir, pprof, paraver …

ParaProf Main Window



```
Metric: Time
Value: Exclusive
Units: microseconds

2096225  MPI_Send()
1339496  MPI_Init()
  76679  MPI_Bcast()
  74890  MAIN [{matmult.f90} {39,15}]
  60711  MPI_Recv()
  19583  INITIALIZE [{matmult.f90} {4,18}]
   6313  MPI_Finalize()
      4  MPI_Comm_rank()
      3  MPI_Comm_size()
```

click left
mouse button

click right
mouse button

% paraprof matmult.ppk

TAU’s MPI Wrapper Interposition Library

- Uses standard MPI Profiling Interface
  - Provides name shifted interface
    - MPI_Send = PMPI_Send
    - Weak bindings
- Interpose TAU’s MPI wrapper library between MPI and TAU
  - -lmpi replaced by –lTauMpi –lpmpi –lmpi
- No change to the source code!
  - Just re-link the application to generate performance data
  - setenv TAU_MAKEFILE <dir>/<arch>/lib/Makefile.tau-mpi -[options]
  - Use tau_cxx.sh, tau_f90.sh and tau_cc.sh as compilers

Runtime MPI Shared Library Instrumentation

- We can now interpose the MPI wrapper library for applications that have already been compiled
  - No re-compilation or re-linking necessary!
- Uses LD_PRELOAD for Linux
- On AIX, TAU uses MPI_EUILIB / MPI_EUILIBPATH
- Simply compile TAU with MPI support and prefix your MPI program with tauex

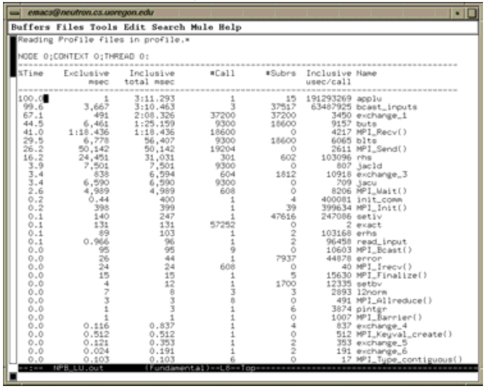
```
% mpirun -np 4 tauex a.out
```

- Requires shared library MPI - does not work on XT3
- Approach will work with other shared libraries

-PROFILE Configuration Option

- Generates flat profiles (one for each MPI process)
  - It is the default option.
- Uses wallclock time (gettimeofday() sys call)
- Calculates exclusive, inclusive time spent in each timer and number of calls

% pprof



```
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            1     3:11.293           1          15  191293269 applu
 99.6        3,667     3:10.463           3       37517   63487925 bcast_inputs
 67.1          491     2:08.326       37200       37200       3450 exchange_1
 44.5        6,461     1:25.159        9300       18600       9157 buts
 41.0     1:18.436     1:18.436       18600           0       4217 MPI_Recv()
 29.5        6,778       56,407        9300       18600       6065 blts
 26.2       50,142       50,142       19204           0       2611 MPI_Send()
 16.2       24,451       31,031         301         602     103096 rhs
  3.9        7,501        7,501        9300           0        807 jacld
  3.4          838        6,594         604        1812      10918 exchange_3
  3.4        6,590        6,590        9300           0        709 jacu
  2.6        4,989        4,989         608           0       8206 MPI_Wait()
  0.2         0.44          400           1           4     400081 init_comm
  0.2          398          399           1          39     399634 MPI_Init()
  0.1          140          247           1       47616     247086 setiv
  0.1          131          131       57252           0          2 exact
  0.1           89          103           1           2     103168 erhs
  0.1        0.966           96           1           2      96458 read_input
  0.0           95           95           9           0      10603 MPI_Bcast()
  0.0           26           44           1        7937      44878 error
  0.0           24           24         608           0         40 MPI_Irecv()
  0.0           15           15           1           5      15630 MPI_Finalize()
  0.0            4           12           1        1700      12335 setbv
  0.0            7            8           3           3       2893 l2norm
  0.0            3            3           8           0        491 MPI_Allreduce()
  0.0            1            3           1           6       3874 pintgr
  0.0            1            1           1           0       1007 MPI_Barrier()
  0.0        0.116        0.837           1           4        837 exchange_4
  0.0        0.512        0.512           1           0        512 MPI_Keyval_create()
  0.0        0.121        0.353           1           2        353 exchange_5
  0.0        0.024        0.191           1           2        191 exchange_6
  0.0        0.103        0.103           6           0         17 MPI_Type_contiguous()
--:--  NPB_LU.out        (Fundamental)--L8--Top-----------------------------------------
```

-MULTIPLECOUNTERS Configuration Option

- Instead of one metric, profile or trace with more than one metric
  - Set environment variables COUNTER[1-25] to specify the metric
    - % setenv COUNTER1 GET_TIME_OF_DAY
    - % setenv COUNTER2 PAPI_L2_DCM
    - % setenv COUNTER3 PAPI_FP_OPS
    - % setenv COUNTER4 PAPI_NATIVE_<native_event>
    - % setenv COUNTER5 P_WALL_CLOCK_TIME …

    OR
    - % setenv TAU_METRICS GET_TIME_OF_DAY:PAPI_L2_DCM:PAPI_FP_OPS...
- When used with –TRACE option, the first counter **must** be GET_TIME_OF_DAY
    - % setenv COUNTER1 GET_TIME_OF_DAY
    - Provides a globally synchronized real time clock for tracing
- -multiplecounters appears in the name of the stub Makefile
- Often used with –papi=<dir> to measure hardware performance counters and time
- papi_native_avail and papi_avail are two useful tools

Papi_avail

```
cfe1.sameer 66> ./papi_avail | more
Available events and hardware information.
-------------------------------------------------------------------------
Vendor string and code     : GenuineIntel (1)
Model string and code      : Itanium 2 (1)
CPU Revision               : 5.000000
CPU Megahertz              : 1500.000000
CPU's in this Node         : 28
Nodes in this System       : 1
Total CPU's                : 28
Number Hardware Counters : 4
Max Multiplex Counters     : 32
-------------------------------------------------------------------------
The following correspond to fields in the PAPI_event_info_t structure.

Name            Code            Avail   Deriv   Description (Note)
PAPI_L1_DCM     0x80000000      Yes     No      Level 1 data cache misses
PAPI_L1_ICM     0x80000001      Yes     No      Level 1 instruction cache misses
PAPI_L2_DCM     0x80000002      Yes     Yes     Level 2 data cache misses
PAPI_L2_ICM     0x80000003      Yes     No      Level 2 instruction cache misses
PAPI_L3_DCM     0x80000004      Yes     Yes     Level 3 data cache misses
PAPI_L3_ICM     0x80000005      Yes     No      Level 3 instruction cache misses
PAPI_L1_TCM     0x80000006      Yes     Yes     Level 1 cache misses
PAPI_L2_TCM     0x80000007      Yes     No      Level 2 cache misses


...
```

Papi_native_avail

```
cfe1.sameer 67> ./papi_native_avail | more
Available native events and hardware information.
-------------------------------------------------------------------------
Vendor string and code     : GenuineIntel (1)
Model string and code      : Itanium 2 (1)
CPU Revision               : 5.000000
CPU Megahertz              : 1500.000000
CPU's in this Node         : 28
Nodes in this System       : 1
Total CPU's                : 28
Number Hardware Counters : 4
Max Multiplex Counters     : 32
-------------------------------------------------------------------------
The following correspond to fields in the PAPI_event_info_t structure.

Symbol                           Event Code   Long Description
 Register Name[n]
 Register Value[n]
ALAT_CAPACITY_MISS_ALL           0x40000000   ALAT Entry Replaced -- both integer and floating point i
nstructions
ALAT_CAPACITY_MISS_FP            0x40000001   ALAT Entry Replaced -- only floating point instructions
ALAT_CAPACITY_MISS_INT           0x40000002   ALAT Entry Replaced -- only integer instructions
...
```

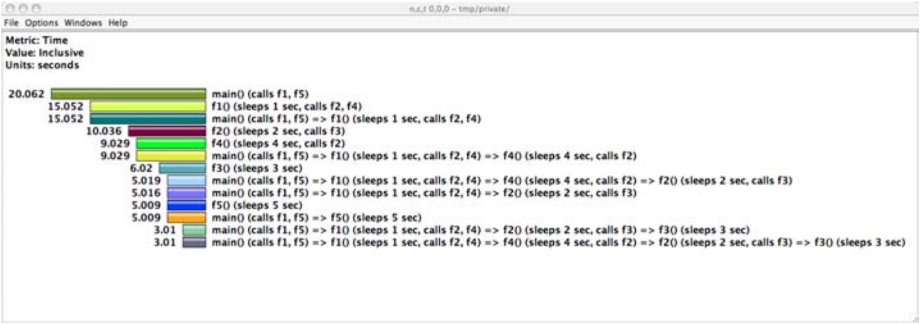
Papi_event_chooser on IA-64

```
cfe1.sameer 68> ./papi_event_chooser
Usage: eventChooser NATIVE|PRESET evt1 evet2 ...

cfe1.sameer 72> ./papi_event_chooser PRESET PAPI_FP_OPS PAPI_L1_DCM PAPI_TOT_CYC
Test case eventChooser: Available events which can be added with given events.
-------------------------------------------------------------------------
Vendor string and code   : GenuineIntel (1)
Model string and code    : Itanium 2 (1)
CPU Revision             : 5.000000
CPU Megahertz            : 1500.000000
CPU's in this Node       : 28
Nodes in this System     : 1
Total CPU's              : 28
Number Hardware Counters : 4
Max Multiplex Counters   : 32
-------------------------------------------------------------------------
Name           Derived Description (Mgr. Note)
PAPI_L1_ICM     No      Level 1 instruction cache misses ()
PAPI_L2_ICM     No      Level 2 instruction cache misses ()
PAPI_L3_ICM     No      Level 3 instruction cache misses

...
```

PAPI_L1_ICM may be counted with these counters on IA64

-PROFILECALLPATH Configuration Option

- Generates profiles that show the calling order (edges & nodes in callgraph)
  - A=>B=>C shows the time spent in C when it was called by B and B was called by A
  - Control the depth of callpath using TAU_CALLPATH_DEPTH env. Variable
  - -callpath in the name of the stub Makefile name
  - In TAU 2.18.2+, any executable can generate callpath profiles using
  - **`% setenv TAU_CALLPATH 1`**

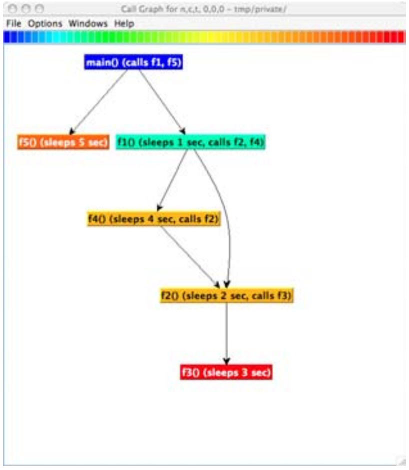


Metric: Time
Value: Inclusive
Units: seconds

```
20.062  main() (calls f1, f5)
15.052  f1() (sleeps 1 sec, calls f2, f4)
15.052  main() (calls f1, f5) => f1() (sleeps 1 sec, calls f2, f4)
10.036  f2() (sleeps 2 sec, calls f3)
9.029   f4() (sleeps 4 sec, calls f2)
9.029   main() (calls f1, f5) => f1() (sleeps 1 sec, calls f2, f4) => f4() (sleeps 4 sec, calls f2)
6.02    f3() (sleeps 3 sec)
5.019   main() (calls f1, f5) => f1() (sleeps 1 sec, calls f2, f4) => f4() (sleeps 4 sec, calls f2) => f2() (sleeps 2 sec, calls f3)
5.016   main() (calls f1, f5) => f1() (sleeps 1 sec, calls f2, f4) => f2() (sleeps 2 sec, calls f3)
5.009   f5() (sleeps 5 sec)
5.009   main() (calls f1, f5) => f5() (sleeps 5 sec)
3.01    main() (calls f1, f5) => f1() (sleeps 1 sec, calls f2, f4) => f2() (sleeps 2 sec, calls f3) => f3() (sleeps 3 sec)
3.01    main() (calls f1, f5) => f1() (sleeps 1 sec, calls f2, f4) => f4() (sleeps 4 sec, calls f2) => f2() (sleeps 2 sec, calls f3) => f3() (sleeps 3 sec)
```

-PROFILECALLPATH Configuration Option

- Generates program callgraph



Profile Measurement – Three Flavors

- Flat profiles
  - Time (or counts) spent in each routine (nodes in callgraph).
  - Exclusive/inclusive time, no. of calls, child calls
  - E.g,: MPI_Send, foo, …
- Callpath Profiles
  - Flat profiles, **plus**
  - Sequence of actions that led to poor performance
  - Time spent along a calling path (edges in callgraph)
  - E.g., “main=> f1 => f2 => MPI_Send” shows the time spent in MPI_Send when called by f2, when f2 is called by f1, when it is called by main. Depth of this callpath = 4 (TAU_CALLPATH_DEPTH environment variable)
- Phase based profiles
  - Flat profiles, **plus**
  - Flat profiles under a phase (nested phases are allowed)
  - Default “main” phase has all phases and routines invoked outside phases
  - Supports static or dynamic (per-iteration) phases
  - E.g., “IO => MPI_Send” is time spent in MPI_Send in IO phase

-DEPTHLIMIT Configuration Option

- Allows users to enable instrumentation at runtime based on the depth of a calling routine on a callstack.
  - Disables instrumentation in all routines a certain depth away from the root in a callgraph
- TAU_DEPTH_LIMIT environment variable specifies depth

  % setenv TAU_DEPTH_LIMIT 1
  enables instrumentation in only “main”
  % setenv TAU_DEPTH_LIMIT 2
  enables instrumentation in main and routines that are directly called by main

- Stub makefile has -depthlimit in its name:

  setenv TAU_MAKEFILE <taudir>/<arch>/lib/Makefile.tau-mpi-depthlimit-pdt

-COMPENSATE Configuration Option

- Specifies online compensation of performance perturbation
- TAU computes its timer overhead and subtracts it from the profiles
- Works well with time or instructions based metrics
- Does not work with level 1/2 data cache misses
- setenv TAU_COMPENSATE 1 (in TAU v2.18.2+)

-TRACE Configuration Option

- Generates event-trace logs, rather than summary profiles
- Traces show when and where an event occurred in terms of location and the process that executed it
- Traces from multiple processes are merged:

```
% tau_treemerge.pl
```

  - generates tau.trc and tau.edf as merged trace and event definition file
- TAU traces can be converted to Vampir's OTF/VTF3, Jumpshot SLOG2, Paraver trace formats:

```
% tau2otf tau.trc tau.edf app.otf
% tau2vtf tau.trc tau.edf app.vpt.gz
% tau2slog2 tau.trc tau.edf -o app.slog2
% tau_convert -paraver tau.trc tau.edf app.prv
```

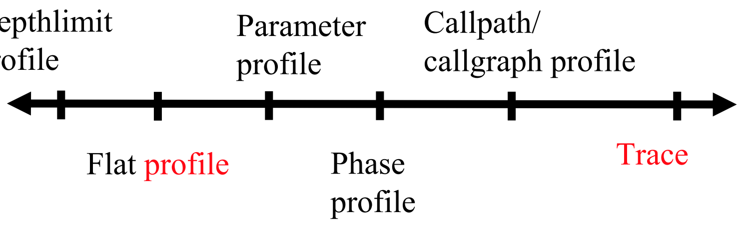
- Stub Makefile has -trace in its name

```
% setenv TAU_MAKEFILE <taudir>/<arch>/lib/
      Makefile.tau-mpi-pdt-trace-pdt
```

  In TAU 2.18.2+ you may simply use with any configuration, at runtime:

```
% setenv TAU_TRACE 1
```

Performance Evaluation Alternatives



Depthlimit profile

Parameter profile

Callpath/ callgraph profile

Flat profile

Phase profile

Trace

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

Volume of performance data

-PROFILEPARAM Configuration Option

- Idea: partition performance data for individual functions based on runtime parameters
- Enable by configuring with –PROFILEPARAM
- TAU call: TAU_PROFILE_PARAM1L (value, “name”)
- Simple example:

```
void foo(long input) {
   TAU_PROFILE("foo", "", TAU_DEFAULT);
   TAU_PROFILE_PARAM1L(input, "input");
   ... }
```

Workload Characterization

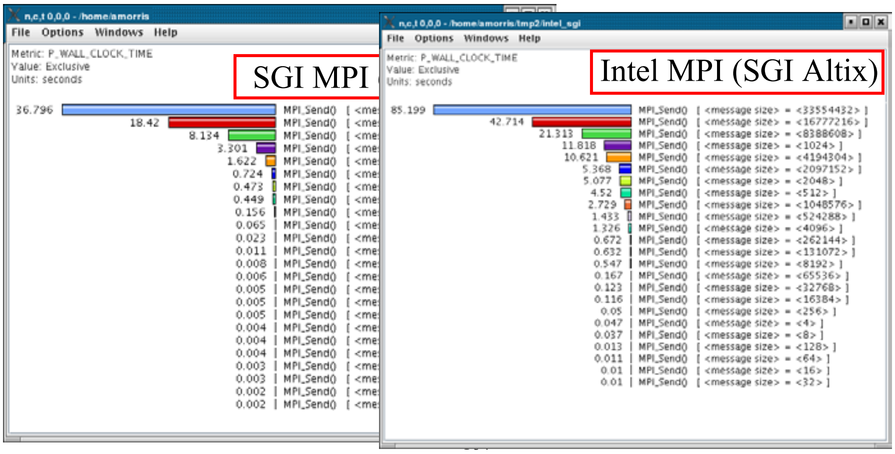
- 5 seconds spent in function “`foo`” becomes
  - 2 seconds for “`foo [ <input> = <25> ]`”
  - 1 seconds for “`foo [ <input> = <5> ]`”
  - …
- Currently used in MPI wrapper library
  - Allows for partitioning of time spent in MPI routines based on parameters (message size, message tag, destination node)
  - Can be extrapolated to infer specifics about the MPI subsystem and system as a whole

Workload Characterization

```
#include <stdio.h>
#include <mpi.h>
int buffer[8*1024*1024];

int main(int argc, char **argv) {
  int rank, size, i, j;
  MPI_Init(&argc, &argv);
  MPI_Comm_size( MPI_COMM_WORLD, &size );
  MPI_Comm_rank( MPI_COMM_WORLD, &rank );
  for (i=0;i<1000;i++)
    for (j=1;j<=8*1024*1024;j*=2) {
      if (rank == 0) {
       MPI_Send(buffer,j,MPI_INT,1,42,MPI_COMM_WORLD);
      } else {
       MPI_Status status;
       MPI_Recv(buffer,j,MPI_INT,0,42,MPI_COMM_WORLD,&status);
      }
    }
  MPI_Finalize();
}
```

Workload Characterization

```
% icc mpi.c -lmpi
% mpirun -np 2 tauex a.out
```

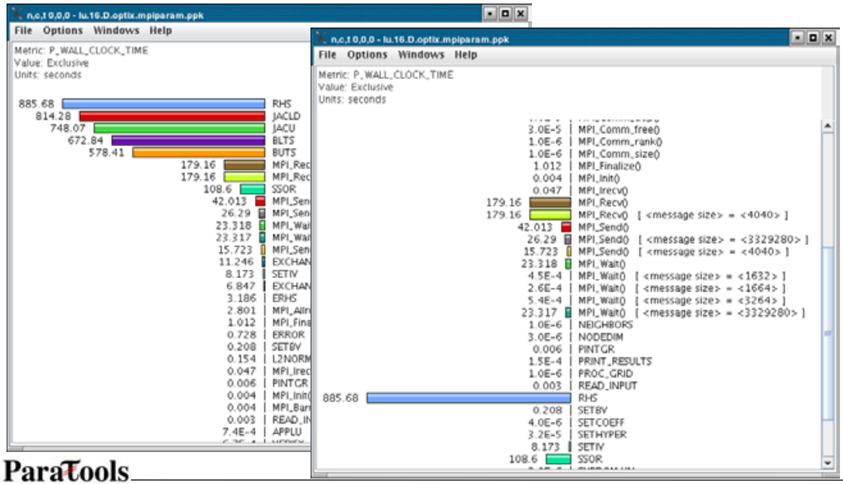


```
Metric: P_WALL_CLOCK_TIME
Value: Exclusive
Units: seconds

36.796  MPI_Send()  [ <me
 18.42  MPI_Send()  [ <me
 8.134  MPI_Send()  [ <me
 3.301  MPI_Send()  [ <me
 1.622  MPI_Send()  [ <me
 0.724  MPI_Send()  [ <me
 0.473  MPI_Send()  [ <me
 0.449  MPI_Send()  [ <me
 0.156  MPI_Send()  [ <me
 0.065  MPI_Send()  [ <me
 0.023  MPI_Send()  [ <me
 0.011  MPI_Send()  [ <me
 0.008  MPI_Send()  [ <me
 0.006  MPI_Send()  [ <me
 0.005  MPI_Send()  [ <me
 0.005  MPI_Send()  [ <me
 0.005  MPI_Send()  [ <me
 0.004  MPI_Send()  [ <me
 0.004  MPI_Send()  [ <me
 0.004  MPI_Send()  [ <me
 0.003  MPI_Send()  [ <me
 0.003  MPI_Send()  [ <me
 0.002  MPI_Send()  [ <me
 0.002  MPI_Send()  [ <me
```

```
Metric: P_WALL_CLOCK_TIME
Value: Exclusive
Units: seconds

85.199  MPI_Send()  [ <message size> = <33554432> ]
42.714  MPI_Send()  [ <message size> = <16777216> ]
21.313  MPI_Send()  [ <message size> = <8388608> ]
11.818  MPI_Send()  [ <message size> = <1024> ]
10.621  MPI_Send()  [ <message size> = <4194304> ]
 5.368  MPI_Send()  [ <message size> = <2097152> ]
 5.077  MPI_Send()  [ <message size> = <2048> ]
  4.52  MPI_Send()  [ <message size> = <512> ]
 2.729  MPI_Send()  [ <message size> = <1048576> ]
 1.433  MPI_Send()  [ <message size> = <524288> ]
 1.326  MPI_Send()  [ <message size> = <4096> ]
 0.672  MPI_Send()  [ <message size> = <262144> ]
 0.632  MPI_Send()  [ <message size> = <131072> ]
 0.547  MPI_Send()  [ <message size> = <8192> ]
 0.167  MPI_Send()  [ <message size> = <65536> ]
 0.123  MPI_Send()  [ <message size> = <32768> ]
 0.116  MPI_Send()  [ <message size> = <16384> ]
  0.05  MPI_Send()  [ <message size> = <256> ]
 0.047  MPI_Send()  [ <message size> = <4> ]
 0.037  MPI_Send()  [ <message size> = <8> ]
 0.013  MPI_Send()  [ <message size> = <128> ]
 0.011  MPI_Send()  [ <message size> = <64> ]
  0.01  MPI_Send()  [ <message size> = <16> ]
  0.01  MPI_Send()  [ <message size> = <32> ]
```

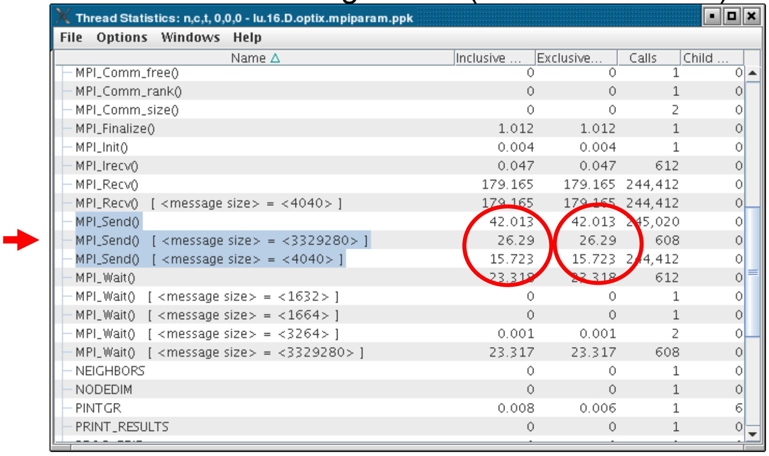
Workload Characterization

- MPI Results (NAS Parallel Benchmark 3.1, LU class D on



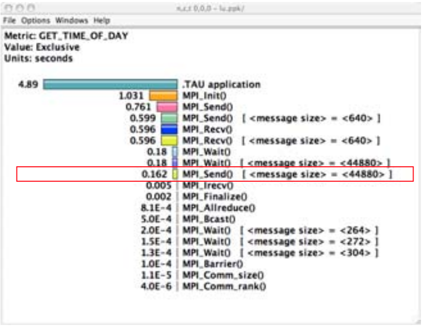
Workload Characterization

- Two different message sizes (~3.3MB and ~4K)



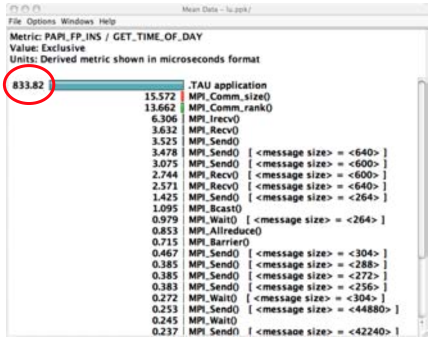
Name △	Inclusive ...	Exclusive...	Calls	Child ...
MPI_Comm_free()	0	0	1	0
MPI_Comm_rank()	0	0	1	0
MPI_Comm_size()	0	0	2	0
MPI_Finalize()	1.012	1.012	1	0
MPI_Init()	0.004	0.004	1	0
MPI_Irecv()	0.047	0.047	612	0
MPI_Recv()	179.165	179.165	244,412	0
MPI_Recv() [<message size> = <4040>]	179.165	179.165	244,412	0
MPI_Send()	42.013	42.013	245,020	0
MPI_Send() [<message size> = <3329280>]	26.29	26.29	608	0
MPI_Send() [<message size> = <4040>]	15.723	15.723	244,412	0
MPI_Wait()	23.318	23.318	612	0
MPI_Wait() [<message size> = <1632>]	0	0	1	0
MPI_Wait() [<message size> = <1664>]	0	0	1	0
MPI_Wait() [<message size> = <3264>]	0.001	0.001	2	0
MPI_Wait() [<message size> = <3329280>]	23.317	23.317	608	0
NEIGHBORS	0	0	1	0
NODEDIM	0	0	1	0
PINTGR	0.008	0.006	1	6
PRINT_RESULTS	0	0	1	0

Job Tracking: ParaProf profile browser



```
Metric: GET_TIME_OF_DAY
Value: Exclusive
Units: seconds

   4.89  .TAU application
  1.031  MPI_Init()
  0.761  MPI_Send()
  0.599  MPI_Send()  [ <message size> = <640> ]
  0.596  MPI_Recv()
  0.596  MPI_Recv()  [ <message size> = <640> ]
   0.18  MPI_Wait()
   0.18  MPI_Wait()  [ <message size> = <44880> ]
  0.162  MPI_Send()  [ <message size> = <44880> ]
  0.005  MPI_Irecv()
  0.002  MPI_Finalize()
 8.1E-4  MPI_Allreduce()
 5.0E-4  MPI_Bcast()
 2.0E-4  MPI_Wait()  [ <message size> = <264> ]
 1.5E-4  MPI_Wait()  [ <message size> = <272> ]
 1.3E-4  MPI_Wait()  [ <message size> = <304> ]
 1.0E-4  MPI_Barrier()
 1.1E-5  MPI_Comm_size()
 4.0E-6  MPI_Comm_rank()
```

LU spent 0.162 seconds sending messages of size 44880



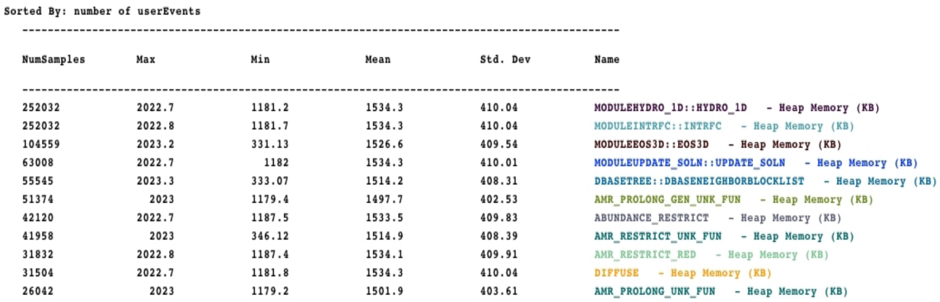
```
Metric: PAPI_FP_INS / GET_TIME_OF_DAY
Value: Exclusive
Units: Derived metric shown in microseconds format

833.82  .TAU application
15.572  MPI_Comm_size()
13.662  MPI_Comm_rank()
 6.306  MPI_Irecv()
 3.632  MPI_Recv()
 3.525  MPI_Send()
 3.478  MPI_Send()  [ <message size> = <640> ]
 3.075  MPI_Send()  [ <message size> = <600> ]
 2.744  MPI_Recv()  [ <message size> = <600> ]
 2.571  MPI_Recv()  [ <message size> = <640> ]
 1.425  MPI_Send()  [ <message size> = <264> ]
 1.095  MPI_Bcast()
 0.979  MPI_Wait()  [ <message size> = <264> ]
 0.853  MPI_Allreduce()
 0.715  MPI_Barrier()
 0.467  MPI_Send()  [ <message size> = <304> ]
 0.385  MPI_Send()  [ <message size> = <288> ]
 0.385  MPI_Send()  [ <message size> = <272> ]
 0.383  MPI_Send()  [ <message size> = <256> ]
 0.272  MPI_Wait()  [ <message size> = <304> ]
 0.253  MPI_Send()  [ <message size> = <44880> ]
 0.245  MPI_Wait()
 0.237  MPI_Send()  [ <message size> = <42240> ]
```

It got 833.82 Mflops!

Memory Profiling in TAU

- Configuration option –PROFILEMEMORY
  - Records global heap memory utilization for each function
  - Takes one sample at beginning of each function and associates the sample with function name
- Configuration option -PROFILEHEADROOM
  - Records headroom (amount of free memory to grow) for each function
  - Takes one sample at beginning of each function and associates it with the callstack [TAU_CALLPATH_DEPTH env variable]
  - Useful for debugging memory usage on IBM BG/L.
- Independent of instrumentation/measurement options selected
- No need to insert macros/calls in the source code
- User defined atomic events appear in profiles/traces

Memory Profiling in TAU (Atomic events)



```
Sorted By: number of userEvents
  ---------------------------------------------------------------------------------------------

  NumSamples        Max             Min             Mean            Std. Dev        Name

  ---------------------------------------------------------------------------------------------
  252032            2022.7          1181.2          1534.3          410.04          MODULEHYDRO_1D::HYDRO_1D   - Heap Memory (KB)
  252032            2022.8          1181.7          1534.3          410.04          MODULEINTRFC::INTRFC   - Heap Memory (KB)
  104559            2023.2          331.13          1526.6          409.54          MODULEEOS3D::EOS3D   - Heap Memory (KB)
  63008             2022.7            1182          1534.3          410.01          MODULEUPDATE_SOLN::UPDATE_SOLN   - Heap Memory (KB)
  55545             2023.3          333.07          1514.2          408.31          DBASETREE::DBASENEIGHBORBLOCKLIST   - Heap Memory (KB)
  51374               2023          1179.4          1497.7          402.53          AMR_PROLONG_GEN_UNK_FUN   - Heap Memory (KB)
  42120             2022.7          1187.5          1533.5          409.83          ABUNDANCE_RESTRICT   - Heap Memory (KB)
  41958               2023          346.12          1514.9          408.39          AMR_RESTRICT_UNK_FUN   - Heap Memory (KB)
  31832             2022.8          1187.4          1534.1          409.91          AMR_RESTRICT_RED   - Heap Memory (KB)
  31504             2022.7          1181.8          1534.3          410.04          DIFFUSE   - Heap Memory (KB)
  26042               2023          1179.2          1501.9          403.61          AMR_PROLONG_UNK_FUN   - Heap Memory (KB)
```

Flash2 code profile (-PROFILEMEMORY) on IBM BlueGene/L [MPI rank 0]

Memory Profiling in TAU

- Instrumentation based observation of global heap memory (not per function)
  - call TAU_TRACK_MEMORY()
  - call TAU_TRACK_MEMORY_HEADROOM()
    - Triggers one sample every 10 secs
  - call TAU_TRACK_MEMORY_HERE()
  - call TAU_TRACK_MEMORY_HEADROOM_HERE()
    - Triggers sample at a specific location in source code
  - call TAU_SET_INTERRUPT_INTERVAL(seconds)
    - To set inter-interrupt interval for sampling
  - call TAU_DISABLE_TRACKING_MEMORY()
  - call TAU_DISABLE_TRACKING_MEMORY_HEADROOM()
    - To turn off recording memory utilization
  - call TAU_ENABLE_TRACKING_MEMORY()
  - call TAU_ENABLE_TRACKING_MEMORY_HEADROOM()
    - To re-enable tracking memory utilization

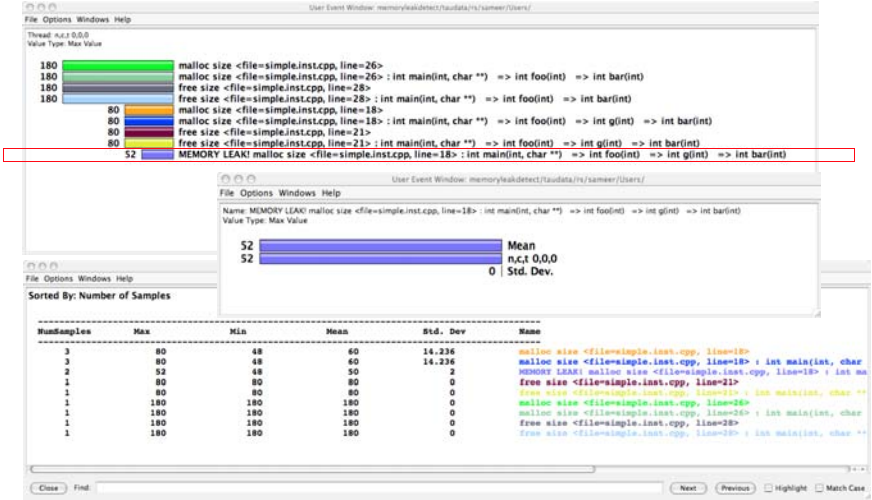
Detecting Memory Leaks in C/C++

- TAU wrapper library for malloc/realloc/free
- During instrumentation, specify
  - -optDetectMemoryLeaks option to TAU_COMPILER

```
% setenv TAU_OPTIONS '-optVerbose -optDetectMemoryLeaks'
% setenv TAU_MAKEFILE <taudir>/<arch>/lib/Makefile.tau-mpi-pdt...
% tau_cxx.sh foo.cpp ...
```

- Tracks each memory allocation/de-allocation in parsed files
- Correlates each memory event with the executing callstack
- At the end of execution, TAU detects memory leaks
- TAU reports leaks based on allocations and the executing callstack
- Set **TAU_CALLPATH_DEPTH** environment variable to limit callpath data
  - default is 2
- Future work
  - Support for C++ new/delete planned
  - Support for Fortran 90/95 allocate/deallocate planned

Memory Leak Detection



```
Thread: n,c,t 0,0,0
Value Type: Max Value

180  malloc size <file=simple.inst.cpp, line=26>
180  malloc size <file=simple.inst.cpp, line=26> : int main(int, char **)   => int foo(int)   => int bar(int)
180  free size <file=simple.inst.cpp, line=28>
180  free size <file=simple.inst.cpp, line=28> : int main(int, char **)   => int foo(int)   => int bar(int)
 80  malloc size <file=simple.inst.cpp, line=18>
 80  malloc size <file=simple.inst.cpp, line=18> : int main(int, char **)   => int foo(int)   => int g(int)   => int bar(int)
 80  free size <file=simple.inst.cpp, line=21>
 80  free size <file=simple.inst.cpp, line=21> : int main(int, char **)   => int foo(int)   => int g(int)   => int bar(int)
 52  MEMORY LEAK! malloc size <file=simple.inst.cpp, line=18> : int main(int, char **)   => int foo(int)   => int g(int)   => int bar(int)
```

```
Name: MEMORY LEAK! malloc size <file=simple.inst.cpp, line=18> : int main(int, char **)   => int foo(int)   => int g(int)   => int bar(int)
Value Type: Max Value

52  Mean
52  n,c,t 0,0,0
 0  Std. Dev.
```

```
Sorted By: Number of Samples

---------------------------------------------------------------------------------------------------------
NumSamples        Max           Min           Mean          Std. Dev      Name
---------------------------------------------------------------------------------------------------------
     3             80            48            60           14.236        malloc size <file=simple.inst.cpp, line=18>
     3             80            48            60           14.236        malloc size <file=simple.inst.cpp, line=18> : int main(int, char
     2             52            48            50                2        MEMORY LEAK! malloc size <file=simple.inst.cpp, line=18> : int ma
     1             80            80            80                0        free size <file=simple.inst.cpp, line=21>
     1             80            80            80                0        free size <file=simple.inst.cpp, line=21> : int main(int, char **
     1            180           180           180                0        malloc size <file=simple.inst.cpp, line=26>
     1            180           180           180                0        malloc size <file=simple.inst.cpp, line=26> : int main(int, char
     1            180           180           180                0        free size <file=simple.inst.cpp, line=28>
     1            180           180           180                0        free size <file=simple.inst.cpp, line=28> : int main(int, char **
```

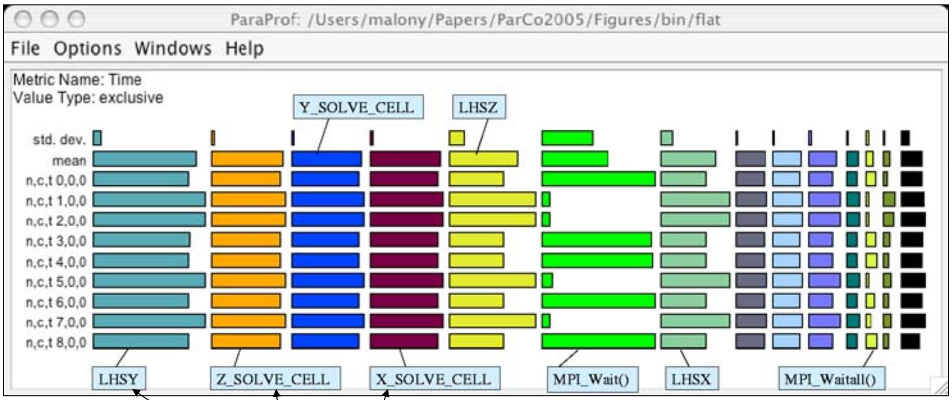
Detecting Memory Leaks in Fortran

```
subroutine foo(x)
integer:: x
integer, allocatable :: A(:), B(:), C(:)

print *, "inside foo"
allocate(A(x), B(x), C(x))
deallocate(A, C)
print *, "exiting foo"

end subroutine foo

program main
call foo(5)
end program main
```

Detecting Memory Leaks in Fortran

```
USER EVENTS Profile :NODE 0, CONTEXT 0, THREAD 0
---------------------------------------------------------------------------------------
NumSamples  MaxValue  MinValue  MeanValue  Std. Dev.  Event Name
---------------------------------------------------------------------------------------
      1           5         5      5       0  MEMORY LEAK! malloc size <file=simple.f, variable=B, line=6> : MAIN => FOO
      1           5         5      5       0  free size <file=simple.f, variable=A, line=7>
      1           5         5      5       0  free size <file=simple.f, variable=A, line=7> : MAIN => FOO
      1           5         5      5       0  free size <file=simple.f, variable=C, line=7>
      1           5         5      5       0  free size <file=simple.f, variable=C, line=7> : MAIN => FOO
      1           5         5      5       0  malloc size <file=simple.f, variable=A, line=6>
      1           5         5      5       0  malloc size <file=simple.f, variable=A, line=6> : MAIN => FOO
      1           5         5      5       0  malloc size <file=simple.f, variable=B, line=6>
      1           5         5      5       0  malloc size <file=simple.f, variable=B, line=6> : MAIN => FOO
      1           5         5      5       0  malloc size <file=simple.f, variable=C, line=6>
      1           5         5      5       0  malloc size <file=simple.f, variable=C, line=6> : MAIN => FOO
---------------------------------------------------------------------------------------
```

Phase Profiling (NAS BT, Flat Profile)



Metric Name: Time
Value Type: exclusive

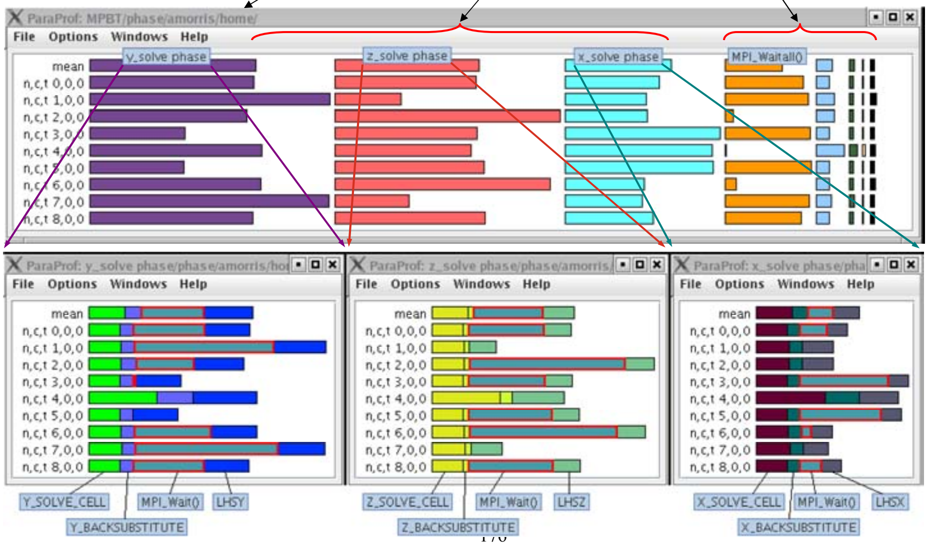
Application routine names reflect phase semantics

How is MPI_Wait() distributed relative to solver direction?

ParaTools

NAS BT – Phase Profile (Main and X, Y, Z)

Main phase shows nested phases and immediate events

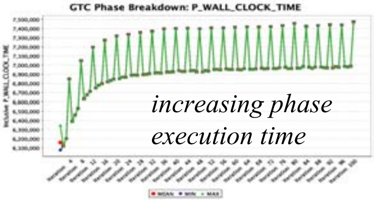


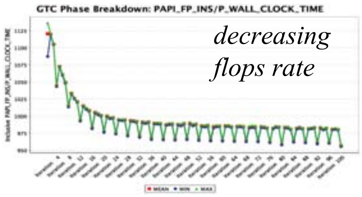
TAU Timers and Phases

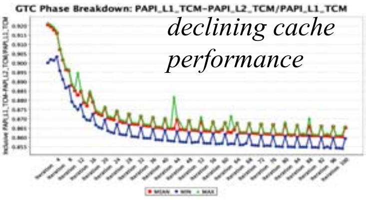
- Static timer
  - Shows time spent in all invocations of a routine (foo)
  - E.g., “foo()” 100 secs, 100 calls
- Dynamic timer
  - Shows time spent in each invocation of a routine
  - E.g., “foo() 3” 4.5 secs, “foo 10” 2 secs (invocations 3 and 10 respectively)
- Static phase
  - Shows time spent in all routines called (directly/indirectly) by a given routine (foo)
  - E.g., “foo() => MPI_Send()” 100 secs, 10 calls shows that a total of 100 secs were spent in MPI_Send() when it was called by foo.
- Dynamic phase
  - Shows time spent in all routines called by a given invocation of a routine.
  - E.g., “foo() 4 => MPI_Send()” 12 secs, shows that 12 secs were spent in MPI_Send when it was called by the 4th invocation of foo.

Performance Dynamics: Phase-Based Profiling

- Profile phases capture performance with respect to application-defined ‘phases’ of execution
  - Separate full profile produce for each phase
- GTC particle-in-cell simulation of fusion turbulence
- Phases assigned to iterations
- Data change affects cache







TAU’s MPI Wrapper Interposition Library

- Uses standard MPI Profiling Interface
  - Provides name shifted interface
    - MPI_Send = PMPI_Send
    - Weak bindings
- Interpose TAU’s MPI wrapper library between MPI and TAU
  - -lmpi replaced by –lTauMpi –lpmpi –lmpi
- No change to the source code! Just re-link the application to generate performance data
  - `setenv TAU_MAKEFILE <dir>/<arch>/lib/Makefile.tau-mpi-[options]`
  - Use tau_cxx.sh, tau_f90.sh and tau_cc.sh as compilers

Using TAU

- Install TAU
  - Configuration
  - Measurement library creation
- Instrument application
  - Manual or automatic source instrumentation
  - Instrumented library (e.g., MPI – wrapper interposition library)
  - Binary instrumentation
- Create performance experiments
  - Integrate with application build environment
  - Set experiment variables
- Execute application
- Analyze performance

Integration with Application Build Environment

- Try to minimize impact on user's application build procedures
- Handle process of parsing, instrumentation, compilation, linking
- Dealing with Makefiles
  - Minimal change to application Makefile
  - Avoid changing compilation rules in application Makefile
  - No explicit inclusion of rules for process stages
- Some applications do not use Makefiles
  - Facilitate integration in whatever procedures used
- Two techniques:
  - TAU shell scripts (tau_<compiler>.sh)
    - Invokes all PDT parser, TAU instrumenter, and compiler
  - TAU_COMPILER

ParaTools

Using Program Database Toolkit (PDT)

1. **Parse the Program to create foo.pdb:**

```
% cxxparse foo.cpp –I/usr/local/mydir –DMYFLAGS …
```

or

```
% cparse foo.c –I/usr/local/mydir –DMYFLAGS …
```

or

```
% f95parse foo.f90 –I/usr/local/mydir …
% f95parse *.f –omerged.pdb –I/usr/local/mydir –R free
```

2. **Instrument the program:**

```
% tau_instrumentor foo.pdb   foo.f90 –o foo.inst.f90
    –f select.tau
```

3. **Compile the instrumented program:**

```
% ifort foo.inst.f90 –c –I/usr/local/mpi/include –o foo.o
```

Tau_[cxx,cc,f90].sh – Improves Integration in Makefiles

```
# set TAU_MAKEFILE and TAU_OPTIONS env vars
CC = tau_cc.sh
F90 = tau_f90.sh
CFLAGS =
LIBS = -lm
OBJS = f1.o f2.o f3.o … fn.o

app: $(OBJS)
    $(F90) $(LDFLAGS) $(OBJS) -o $@ $(LIBS)
.c.o:
    $(CC) $(CFLAGS) -c $<
.f90.o:
    $(F90) $(FFLAGS) -c $<
```

Automatic Instrumentation

- We now provide compiler wrapper scripts
  - Simply replace `mpxlf90` with `tau_f90.sh`
  - Automatically instruments Fortran source code, links with TAU MPI Wrapper libraries.
- Use `tau_cc.sh` and `tau_cxx.sh` for C/C++

Before

```
CXX = mpCC
F90 = mpxlf90_r
CFLAGS =
LIBS = -lm
OBJS = f1.o f2.o f3.o … fn.o

app: $(OBJS)
     $(CXX) $(LDFLAGS) $(OBJS) -o $@
     $(LIBS)
.cpp.o:
     $(CC) $(CFLAGS) -c $<
```

After

```
CXX = tau_cxx.sh
F90 = tau_f90.sh
CFLAGS =
LIBS = -lm
OBJS = f1.o f2.o f3.o … fn.o

app: $(OBJS)
     $(CXX) $(LDFLAGS) $(OBJS) -o $@
     $(LIBS)
.cpp.o:
     $(CC) $(CFLAGS) -c $<
```

TAU_COMPILER Commandline Options

- See `<taudir>/<arch>/bin/tau_compiler.sh –help`
- Compilation:

```
% mpxlf90 -c foo.f90
```

Changes to

```
% f95parse foo.f90 $(OPT1)
% tau_instrumentor foo.pdb foo.f90 –o foo.inst.f90 $(OPT2)
% ftn –c foo.f90 $(OPT3)
```

- Linking:

```
% ftn foo.o bar.o –o app
```

Changes to

```
% ftn foo.o bar.o –o app $(OPT4)
```

- Where options OPT[1-4] default values may be overridden by the user:

```
F90 = tau_f90.sh
```

TAU_COMPILER Options

- Optional parameters for $(TAU_COMPILER): [tau_compiler.sh –help]

-optVerbose	Turn on verbose debugging messages
-optCompInst	Use compiler based instrumentation
-optDetectMemoryLeaks	Turn on debugging memory allocations/ de-allocations to track leaks
-optKeepFiles	Does not remove intermediate .pdb and .inst.* files
-optPreProcess	Preprocess Fortran sources before instrumentation
-optTauSelectFile=""	Specify selective instrumentation file for tau_instrumentor
-optLinking=""	Options passed to the linker. Typically $(TAU_MPI_FLIBS) $(TAU_LIBS) $(TAU_CXXLIBS)
-optCompile=""	Options passed to the compiler. Typically $(TAU_MPI_INCLUDE) $(TAU_INCLUDE) $(TAU_DEFS)
-optPdtF95Opts=""	Add options for Fortran parser in PDT (f95parse/gfparse)
-optPdtF95Reset=""	Reset options for Fortran parser in PDT (f95parse/gfparse)
-optPdtCOpts=""	Options for C parser in PDT (cparse). Typically $(TAU_MPI_INCLUDE) $(TAU_INCLUDE) $(TAU_DEFS)
-optPdtCxxOpts=""	Options for C++ parser in PDT (cxxparse). Typically $(TAU_MPI_INCLUDE) $(TAU_INCLUDE) $(TAU_DEFS)
...	

Compiling Fortran Codes with TAU

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

- To use the compiler based instrumentation instead of PDT (source-based):
  % setenv TAU_OPTIONS '-optCompInst -optVerbose'

- If your Fortran code uses C preprocessor directives (#include, #ifdef, #endif):
  % setenv TAU_OPTIONS '-optPreProcess -optVerbose -optDetectMemoryLeaks'

- To use an instrumentation specification file:
  % setenv TAU_OPTIONS '-optTauSelectFile=mycmd.tau -optVerbose -optPreProcess'
  % cat mycmd.tau

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

Overriding Default Options:TAU_COMPILER

```
% cat Makefile
F90 = tau_f90.sh
OBJS = f1.o f2.o f3.o …
LIBS = -Lappdir –lapplib1 –lapplib2 …

app: $(OBJS)
    $(F90) $(OBJS) –o app $(LIBS)
.f90.o:
    $(F90) –c $<
% setenv TAU_OPTIONS '-optVerbose
      -optTauSelectFile=select.tau -optKeepFiles'
% setenv TAU_MAKEFILE <taudir>/x86_64/lib/Makefile.tau-mpi-pdt
```

Optimization of Program Instrumentation

- Need to eliminate instrumentation in frequently executing lightweight routines
- Throttling of events at runtime (default in tau-2.17.2+):

```
% setenv TAU_THROTTLE 1
```

Turns off instrumentation in routines that execute over 100000 times (TAU_THROTTLE_NUMCALLS) and take less than 10 microseconds of inclusive time per call (TAU_THROTTLE_PERCALL). Use TAU_THROTTLE=0 to disable.

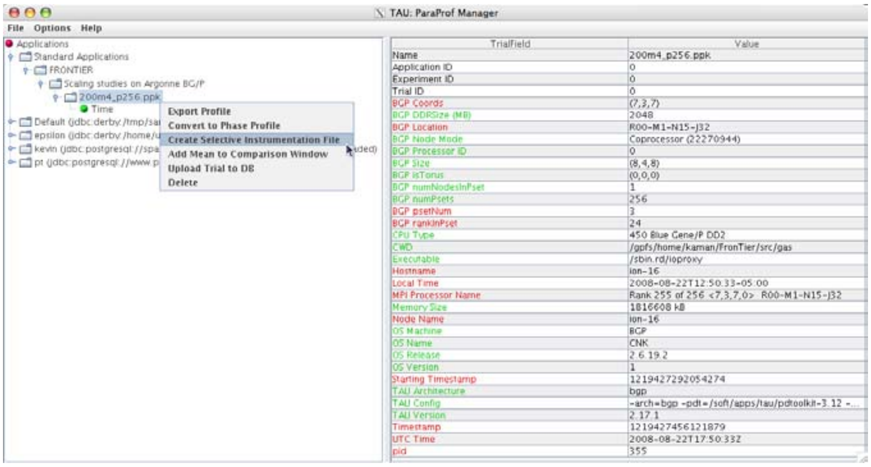
- Selective instrumentation file to filter events

```
% tau_instrumentor [options] –f <file>   OR
% setenv TAU_OPTIONS '-optTauSelectFile=tau.txt'
```

- Compensation of local instrumentation overhead

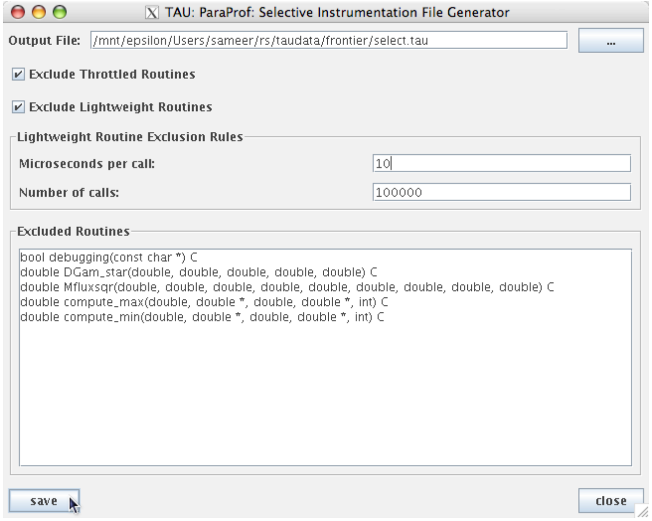
```
% configure -COMPENSATE
or
% setenv TAU_COMPENSATE 1  (in tau-2.18.2+)
```

ParaProf: Creating Selective Instrumentation File



TrialField	Value
Name	200m4_p256.ppk
Application ID	0
Experiment ID	0
Trial ID	0
BGP Coords	(7,3,7)
BGP DDRSize (MB)	2048
BGP Location	R00-M1-N15-J32
BGP Node Mode	Coprocessor (22270944)
BGP Processor ID	0
BGP Size	(8,4,8)
BGP isTorus	(0,0,0)
BGP numNodesInPset	1
BGP numPsets	256
BGP psetNum	3
BGP rankInPset	24
CPU Type	450 Blue Gene/P DD2
CWD	/gpfs/home/kaman/FronTier/src/gas
Executable	/sbin.rd/ioproxy
Hostname	ion-16
Local Time	2008-08-22T12:50:33-05:00
MPI Processor Name	Rank 255 of 256 <7,3,7,0> R00-M1-N15-J32
Memory Size	1816608 kB
Node Name	ion-16
OS Machine	BGP
OS Name	CNK
OS Release	2.6.19.2
OS Version	1
Starting Timestamp	1219427292054274
TAU Architecture	bgp
TAU Config	-arch=bgp -pdt=/soft/apps/tau/pdtoolkit-3.12 -...
TAU Version	2.17.1
Timestamp	1219427456121879
UTC Time	2008-08-22T17:50:33Z
pid	355

Choosing Rules for Excluding Routines



TAU: ParaProf: Selective Instrumentation File Generator

Output File: /mnt/epsilon/Users/sameer/rs/taudata/frontier/select.tau

☑ Exclude Throttled Routines

☑ Exclude Lightweight Routines

Lightweight Routine Exclusion Rules

Microseconds per call: 10

Number of calls: 100000

Excluded Routines

```
bool debugging(const char *) C
double DGam_star(double, double, double, double, double) C
double Mfluxsqr(double, double, double, double, double, double, double, double, double) C
double compute_max(double, double *, double, double *, int) C
double compute_min(double, double *, double, double *, int) C
```

Selective Instrumentation File

- Specify a list of routines to exclude or include (case sensitive)
- # is a wildcard in a routine name. It cannot appear in the first column.

```
BEGIN_EXCLUDE_LIST
Foo
Bar
D#EMM
END_EXCLUDE_LIST
```

- Specify a list of routines to include for instrumentation

```
BEGIN_INCLUDE_LIST
int main(int, char **)
F1
F3
END_INCLUDE_LIST
```

- Specify either an include list or an exclude list!

Selective Instrumentation File

- Optionally specify a list of files to exclude or include (case sensitive)
- * and ? may be used as wildcard characters in a file name

```
BEGIN_FILE_EXCLUDE_LIST
f*.f90
Foo?.cpp
END_FILE_EXCLUDE_LIST
```

- Specify a list of routines to include for instrumentation

```
BEGIN_FILE_INCLUDE_LIST
main.cpp
foo.f90
END_FILE_INCLUDE_LIST
```

Selective Instrumentation File

- User instrumentation commands are placed in INSTRUMENT section
- ? and * used as wildcard characters for file name, # for routine name
- \ as escape character for quotes
- Routine entry/exit, arbitrary code insertion
- Outer-loop level instrumentation

```
BEGIN_INSTRUMENT_SECTION
loops file=“foo.f90” routine=“matrix#”
memory file=“foo.f90” routine=“#”
io routine=“matrix#”
[static/dynamic] phase routine=“MULTIPLY”
dynamic [phase/timer] name=“foo” file=“foo.cpp” line=22 to line=35
file=“foo.f90” line = 123 code = "  print *, \" Inside foo\""
exit routine = “int foo()” code = "cout <<\"exiting foo\"<<endl;"
END_INSTRUMENT_SECTION
```

Instrumentation Specification

```
% tau_instrumentor
Usage : tau_instrumentor <pdbfile> <sourcefile> [-o <outputfile>] [-noinline]
[-g groupname] [-i headerfile] [-c|-c++|-fortran] [-f <instr_req_file> ]
For selective instrumentation, use -f option
% tau_instrumentor foo.pdb foo.cpp -o foo.inst.cpp -f selective.dat
% cat selective.dat
# Selective instrumentation: Specify an exclude/include list of routines/files.
BEGIN_EXCLUDE_LIST
void quicksort(int *, int, int)
void sort_5elements(int *)
void interchange(int *, int *)
END_EXCLUDE_LIST

BEGIN_FILE_INCLUDE_LIST
Main.cpp
Foo?.c
*.C
END_FILE_INCLUDE_LIST
# Instruments routines in Main.cpp, Foo?.c and *.C files only
# Use BEGIN_[FILE]_INCLUDE_LIST with END_[FILE]_INCLUDE_LIST
```

Instrumentation of OpenMP Constructs

- **O**penMP **P**ragma **A**nd **R**egion **I**nstrumentor [UTK, FZJ]
- Source-to-Source translator to insert POMP calls around OpenMP constructs and API functions
- Done: Supports
  - Fortran77 and Fortran90, OpenMP 2.0
  - C and C++, OpenMP 1.0
  - POMP Extensions
  - EPILOG and TAU POMP implementations
  - Preserves source code information (**`#line`** ***line file***)
- tau_ompcheck
  - Balances OpenMP constructs (DO/END DO) and detects errors
  - Invoked by tau_compiler.sh prior to invoking Opari
- KOJAK Project website http://icl.cs.utk.edu/kojak

OpenMP API Instrumentation

- Transform
  - `omp_#_lock()` → `pomp_#_lock()`
  - `omp_#_nest_lock()` → `pomp_#_nest_lock()`

  `[# = init|destroy|set|unset|test]`

- POMP version
  - Calls omp version internally
  - Can do extra stuff before and after call

ParaTools

Example: `!$OMP PARALLEL DO` Instrumentation

```
call pomp_parallel_fork(d)
!$OMP PARALLEL other-clauses...
      call pomp_parallel_begin(d)
      call pomp_do_enter(d)
      !$OMP DO schedule-clauses, ordered-clauses,
               lastprivate-clauses
            do loop
      !$OMP END DO NOWAIT
      call pomp_barrier_enter(d)
      !$OMP BARRIER
      call pomp_barrier_exit(d)
      call pomp_do_exit(d)
      call pomp_parallel_end(d)
!$OMP END PARALLEL DO
call pomp_parallel_join(d)
```

ParaTools

Opari Instrumentation: Example

```
pomp_for_enter(&omp_rd_2);
#line 252 "stommel.c"
#pragma omp for schedule(static) reduction(+: diff) private(j)
  firstprivate (a1,a2,a3,a4,a5) nowait
for( i=i1;i<=i2;i++) {
  for(j=j1;j<=j2;j++){
    new_psi[i][j]=a1*psi[i+1][j] + a2*psi[i-1][j] + a3*psi[i][j+1]
      + a4*psi[i][j-1] - a5*the_for[i][j];
    diff=diff+fabs(new_psi[i][j]-psi[i][j]);
  }
}
pomp_barrier_enter(&omp_rd_2);
#pragma omp barrier
pomp_barrier_exit(&omp_rd_2);
pomp_for_exit(&omp_rd_2);
```

Using Opari with TAU

Configure TAU with Opari (used here with MPI and PDT)

```
% configure -opari -arch=x86_64 -mpi  -
pdt=/usr/contrib/TAU/pdtoolkit-3.14.1
% make clean; make install
% setenv TAU_MAKEFILE /tau/<arch>/lib/Makefile.tau-…opari-…
% tau_cxx.sh -c foo.cpp
% tau_cxx.sh -c bar.f90
% tau_cxx.sh *.o -o app
```

Dynamic Instrumentation

- TAU uses DyninstAPI for runtime code patching
- Developed by U. Wisconsin and U. Maryland
- http://www.dyninst.org
- ***tau_run*** (mutator) loads measurement library
- Instruments mutatee
- MPI issues:
  - one mutator per executable image [TAU, DynaProf]
  - one mutator for several executables [Paradyn, DPCL]

Using DyninstAPI with TAU

Step I: Install DyninstAPI[Download from http://www.dyninst.org]

```
% cd dyninstAPI-6/core; make
```

Set DyninstAPI environment variables (including LD_LIBRARY_PATH)

Step II: Configure TAU with Dyninst

```
% configure -dyninst=/usr/local/dyninstAPI-6
% make clean; make install
```

Builds <taudir>/<arch>/bin/tau_run

```
% tau_run [<-o outfile>] [-Xrun<libname>][-f <select_inst_file>] [-v] <infile>
% tau_run -o a.inst.out a.out
```

Rewrites a.out

```
% tau_run klargest
```

Instruments klargest with TAU calls and executes it

```
% tau_run -XrunTAUsh-papi a.out
```

Loads libTAUsh-papi.so instead of libTAU.so for measurements

Virtual Machine Performance Instrumentation

- Integrate performance system with VM
  - Captures robust performance data (e.g., thread events)
  - Maintain features of environment
    - portability, concurrency, extensibility, interoperation
  - Allow use in optimization methods
- JVM Profiling Interface (JVMPI)
  - Generation of JVM events and hooks into JVM
  - Profiler agent (TAU) loaded as shared object
    - registers events of interest and address of callback routine
  - Access to information on dynamically loaded classes
  - No need to modify Java source, bytecode, or JVM

Using TAU with Java Applications

Step I: Sun JDK 1.4+ [download from www.javasoft.com]

Step II: Configure TAU with JDK (v 1.2 or better)

```
% configure -jdk=/usr/java2 -TRACE -PROFILE
% make clean; make install
```

Builds <taudir>/<arch>/lib/libTAU.so

For Java (without instrumentation):

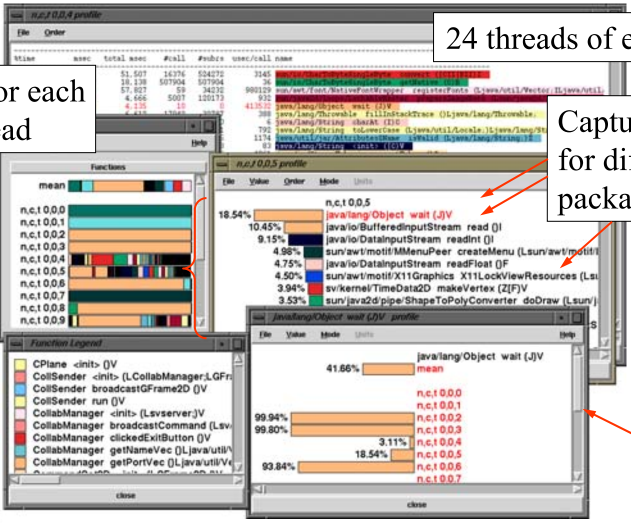
```
% java application
```

With instrumentation:

```
% java -XrunTAU application
% java -XrunTAU:exclude=sun/io,java application
```

Excludes sun/io/* and java/* classes

TAU Profiling of Java Application (SciVis)



24 threads of execution!

Profile for each Java thread

Captures events for different Java packages

global routine profile

Using TAU with Python Applications

Step I: Configure TAU with Python

```
% configure -pythoninc=/usr/include/python2.5/include
% make clean; make install
```

Builds <taudir>/<arch>/lib/<bindings>/pytau.py and tau.py packages for manual and automatic instrumentation respectively

```
% setenv PYTHONPATH $PYTHONPATH\:<taudir>/<arch>/lib/[<dir>]
```

Python Automatic Instrumentation Example

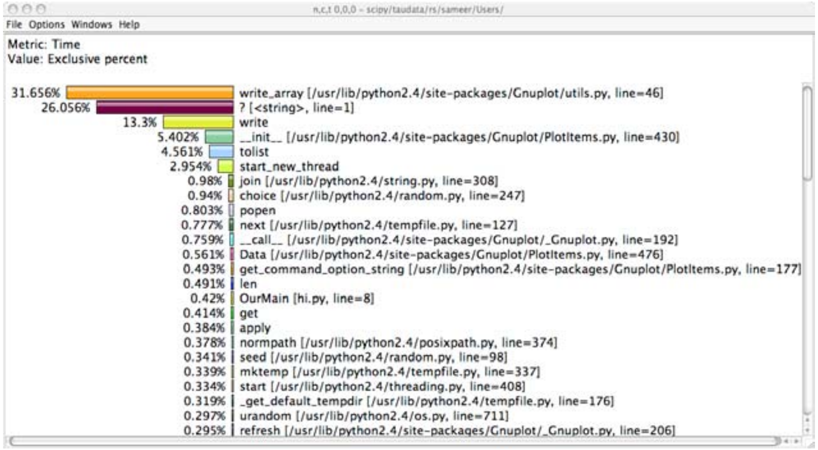
```
#!/usr/bin/env/python

import tau
from time import sleep

def f2():
    print “ In f2: Sleeping for 2 seconds ”
    sleep(2)
def f1():
    print “ In f1: Sleeping for 3 seconds ”
    sleep(3)

def OurMain():
    f1()
tau.run(‘OurMain()’)
```

Running:

```
% setenv PYTHONPATH <tau>/<arch>/lib/bindings-python
% ./auto.py
```

`Instruments OurMain, f1, f2, print…`

Python Instrumentation: SciPy

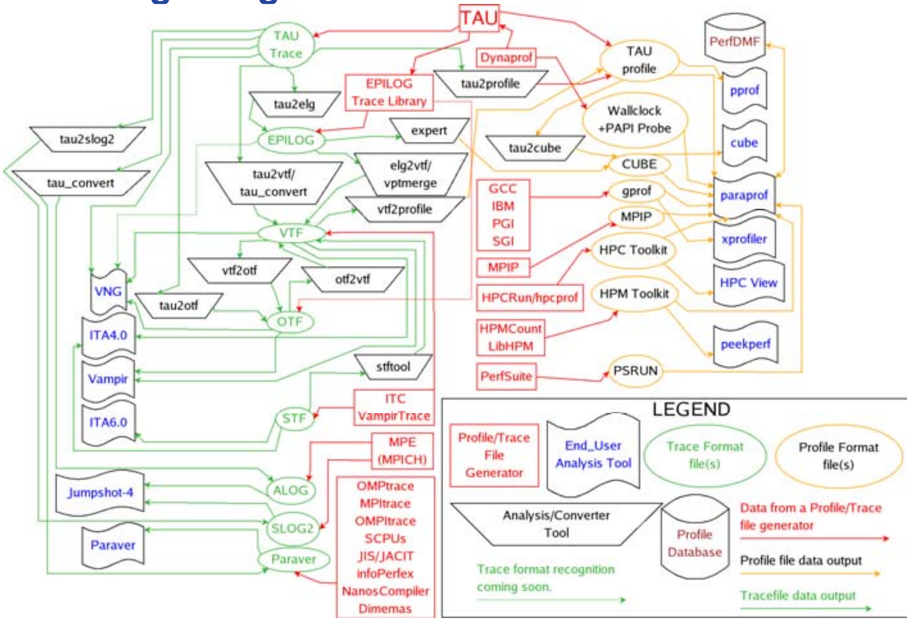


```
Metric: Time
Value: Exclusive percent

31.656%  write_array [/usr/lib/python2.4/site-packages/Gnuplot/utils.py, line=46]
26.056%  ? [<string>, line=1]
13.3%    write
5.402%   __init__ [/usr/lib/python2.4/site-packages/Gnuplot/PlotItems.py, line=430]
4.561%   tolist
2.954%   start_new_thread
0.98%    join [/usr/lib/python2.4/string.py, line=308]
0.94%    choice [/usr/lib/python2.4/random.py, line=247]
0.803%   popen
0.777%   next [/usr/lib/python2.4/tempfile.py, line=127]
0.759%   __call__ [/usr/lib/python2.4/site-packages/Gnuplot/_Gnuplot.py, line=192]
0.561%   Data [/usr/lib/python2.4/site-packages/Gnuplot/PlotItems.py, line=476]
0.493%   get_command_option_string [/usr/lib/python2.4/site-packages/Gnuplot/PlotItems.py, line=177]
0.491%   len
0.42%    OurMain [hi.py, line=8]
0.414%   get
0.384%   apply
0.378%   normpath [/usr/lib/python2.4/posixpath.py, line=374]
0.341%   seed [/usr/lib/python2.4/random.py, line=98]
0.339%   mktemp [/usr/lib/python2.4/tempfile.py, line=337]
0.334%   start [/usr/lib/python2.4/threading.py, line=408]
0.319%   _get_default_tempdir [/usr/lib/python2.4/tempfile.py, line=176]
0.297%   urandom [/usr/lib/python2.4/os.py, line=711]
0.295%   refresh [/usr/lib/python2.4/site-packages/Gnuplot/_Gnuplot.py, line=206]
```

Performance Analysis

- paraprof profile browser (GUI)
- pprof (text based profile browser)
- TAU traces can be exported to many different tools
  - Vampir/VNG [T.U. Dresden] (formerly Intel (R) Trace Analyzer)
  - EXPERT [FZJ]
  - Jumpshot (bundled with TAU) [Argonne National Lab] ...

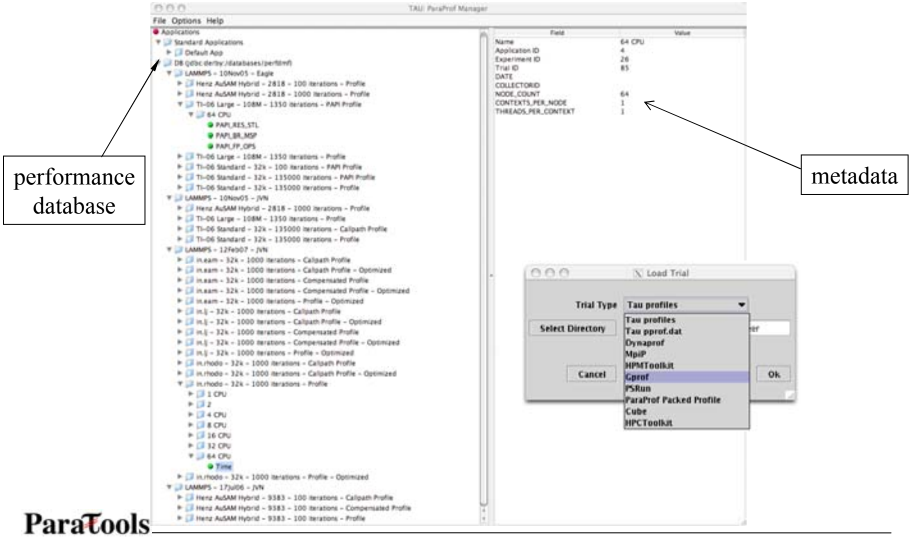
Building Bridges to Other Tools: TAU



TAU Performance System Interfaces

- PDT [U. Oregon, LANL, FZJ] for instrumentation of C++, C99, F95 source code
- PAPI [UTK] for accessing hardware performance counters data
- DyninstAPI [U. Maryland, U. Wisconsin] for runtime instrumentation
- KOJAK [FZJ, UTK]
  - Epilog trace generation library
  - CUBE callgraph visualizer
  - Opari OpenMP directive rewriting tool
- Vampir/VNG Trace Analyzer [TU Dresden]
- VTF3/OTF trace generation library [TU Dresden] (available from TAU website)
- Paraver trace visualizer [CEPBA]
- Jumpshot-4 trace visualizer [MPICH, ANL]
- JVMPI from JDK for Java program instrumentation [Sun]
- Paraprof profile browser/PerfDMF database supports:
  - TAU format
  - Gprof [GNU]
  - HPM Toolkit [IBM]
  - MpiP [ORNL, LLNL]
  - Dynaprof [UTK]
  - PSRun [NCSA]

ParaProf – Manager Window

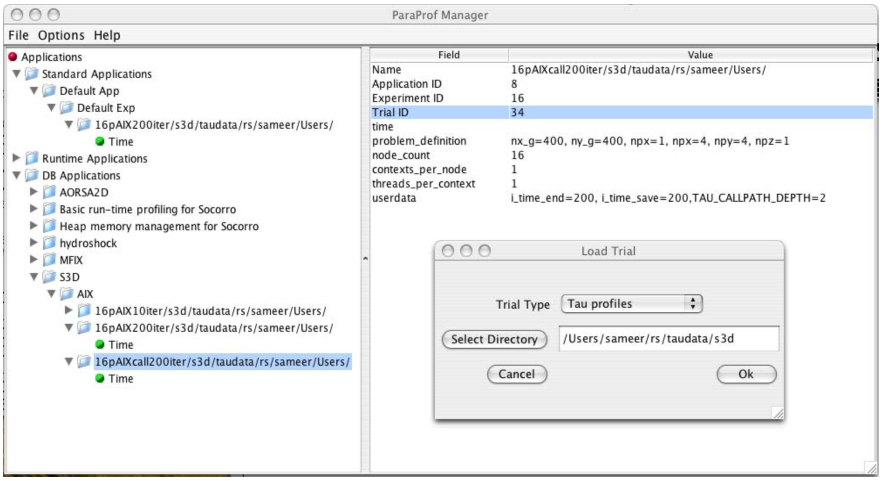


- Applications
  - Standard Applications
    - Default App
  - DB (jdbc:derby:/databases/perfdmf)
    - LAMMPS – 10Nov05 – Eagle
      - Henz AuSAM Hybrid – 2818 – 100 iterations – Profile
      - Henz AuSAM Hybrid – 2818 – 1000 iterations – Profile
      - TI-06 Large – 108M – 1350 iterations – PAPI Profile
        - 64 CPU
          - PAPI_RES_STL
          - PAPI_BR_MSP
          - PAPI_FP_OPS
      - TI-06 Large – 108M – 1350 iterations – Profile
      - TI-06 Standard – 32k – 100 iterations – PAPI Profile
      - TI-06 Standard – 32k – 135000 iterations – PAPI Profile
      - TI-06 Standard – 32k – 135000 iterations – Profile
    - LAMMPS – 10Nov05 – JVN
      - Henz AuSAM Hybrid – 2818 – 1000 iterations – Profile
      - TI-06 Large – 108M – 1350 iterations – Profile
      - TI-06 Standard – 32k – 135000 iterations – Callpath Profile
      - TI-06 Standard – 32k – 135000 iterations – Profile
    - LAMMPS – 12Feb07 – JVN
      - in.eam – 32k – 1000 iterations – Callpath Profile
      - in.eam – 32k – 1000 iterations – Callpath Profile – Optimized
      - in.eam – 32k – 1000 iterations – Compensated Profile
      - in.eam – 32k – 1000 iterations – Compensated Profile – Optimized
      - in.eam – 32k – 1000 iterations – Profile – Optimized
      - in.lj – 32k – 1000 iterations – Callpath Profile
      - in.lj – 32k – 1000 iterations – Callpath Profile – Optimized
      - in.lj – 32k – 1000 iterations – Compensated Profile
      - in.lj – 32k – 1000 iterations – Compensated Profile – Optimized
      - in.lj – 32k – 1000 iterations – Profile – Optimized
      - in.rhodo – 32k – 1000 iterations – Callpath Profile
      - in.rhodo – 32k – 1000 iterations – Callpath Profile – Optimized
      - in.rhodo – 32k – 1000 iterations – Profile
        - 1 CPU
        - 2
        - 4 CPU
        - 8 CPU
        - 16 CPU
        - 32 CPU
        - 64 CPU
          - Time
      - in.rhodo – 32k – 1000 iterations – Profile – Optimized
    - LAMMPS – 17Jul06 – JVN
      - Henz AuSAM Hybrid – 9383 – 100 iterations – Callpath Profile
      - Henz AuSAM Hybrid – 9383 – 100 iterations – Compensated Profile
      - Henz AuSAM Hybrid – 9383 – 100 iterations – Profile

Field	Value
Name	64 CPU
Application ID	4
Experiment ID	26
Trial ID	85
DATE	
COLLECTORID	
NODE_COUNT	64
CONTEXTS_PER_NODE	1
THREADS_PER_CONTEXT	1

Trial Type: Tau profiles

Performance Database: Storage of MetaData



- Applications
  - Standard Applications
    - Default App
      - Default Exp
        - 16pAIX200iter/s3d/taudata/rs/sameer/Users/
          - Time
  - Runtime Applications
  - DB Applications
    - AORSA2D
    - Basic run-time profiling for Socorro
    - Heap memory management for Socorro
    - hydroshock
    - MFIX
    - S3D
      - AIX
        - 16pAIX10iter/s3d/taudata/rs/sameer/Users/
        - 16pAIX200iter/s3d/taudata/rs/sameer/Users/
          - Time
        - 16pAIXcall200iter/s3d/taudata/rs/sameer/Users/
          - Time

Field	Value
Name	16pAIXcall200iter/s3d/taudata/rs/sameer/Users/
Application ID	8
Experiment ID	16
Trial ID	34
time	
problem_definition	nx_g=400, ny_g=400, npx=1, npx=4, npy=4, npz=1
node_count	16
contexts_per_node	1
threads_per_context	1
userdata	i_time_end=200, i_time_save=200,TAU_CALLPATH_DEPTH=2

Load Trial

Trial Type: Tau profiles

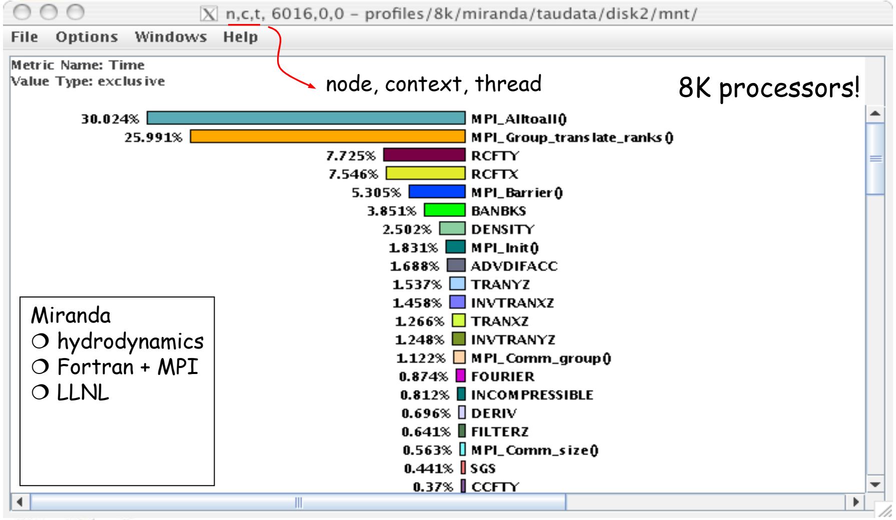
Select Directory: /Users/sameer/rs/taudata/s3d

ParaProf Main Window (Lammps)

Metric: Time
Value: Exclusive

void LAMMPS_NS::PairLJCharmmCoulLong::compute(int, int) [{pair_lj_charmm_coul_long.cpp} {81,1}-{231,1}]
Exclusive Time: 1.0445368E7
Inclusive Time: 1.0617319E7
Calls: 1001.0
SubCalls: 1001.0

ParaProf – Flat Profile (Miranda)



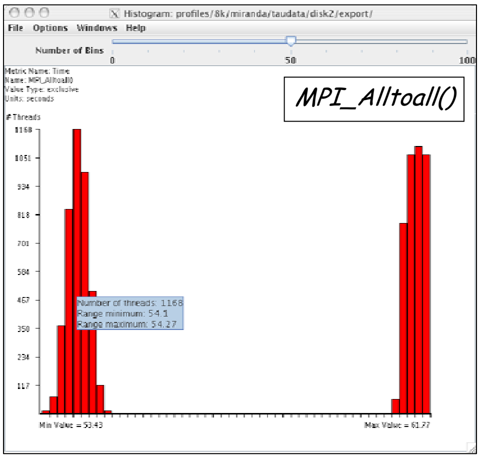
Metric Name: Time
Value Type: exclusive

node, context, thread

8K processors!

Miranda
- hydrodynamics
- Fortran + MPI
- LLNL

ParaProf – Histogram View (Miranda)

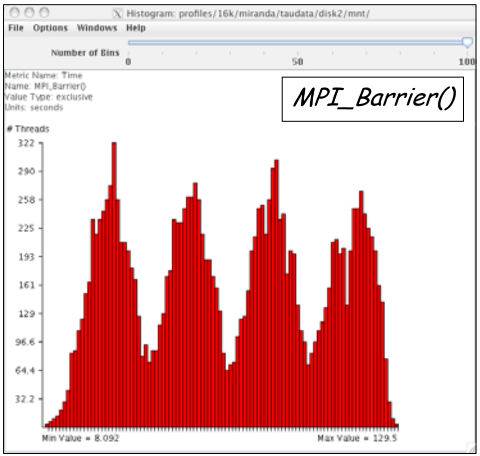


Metric Name: Time
Name: MPI_Alltoall()
Value Type: exclusive
Units: seconds

Number of threads: 1168
Range minimum: 54.1
Range maximum: 54.27

MPI_Alltoall()

8k processors

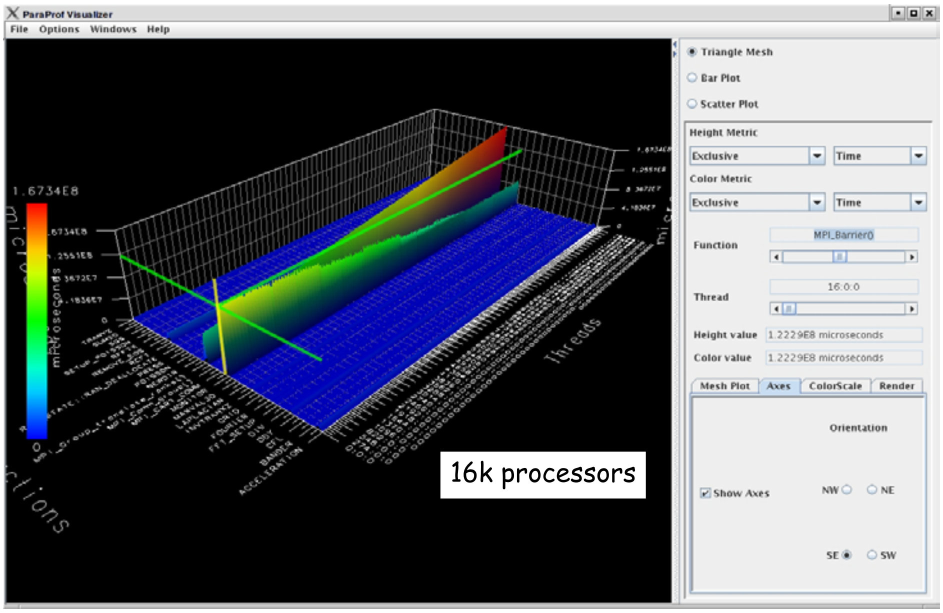


Metric Name: Time
Name: MPI_Barrier()
Value Type: exclusive
Units: seconds

MPI_Barrier()

16k processors

ParaProf – 3D Full Profile (Miranda)



16k processors

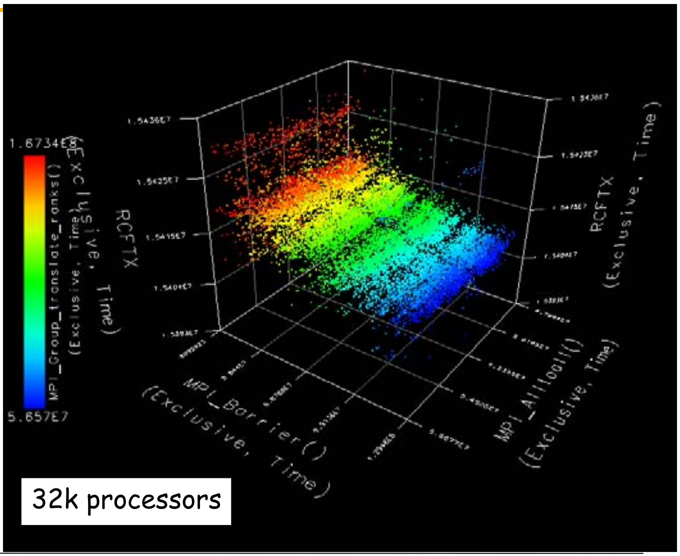
● Triangle Mesh
○ Bar Plot
○ Scatter Plot

Height Metric: Exclusive, Time
Color Metric: Exclusive, Time
Function: MPI_Barrier()
Thread: 16:0:0
Height value: 1.2229E8 microseconds
Color value: 1.2229E8 microseconds

Orientation
☑ Show Axes
NW ○ ○ NE
SE ● ○ SW

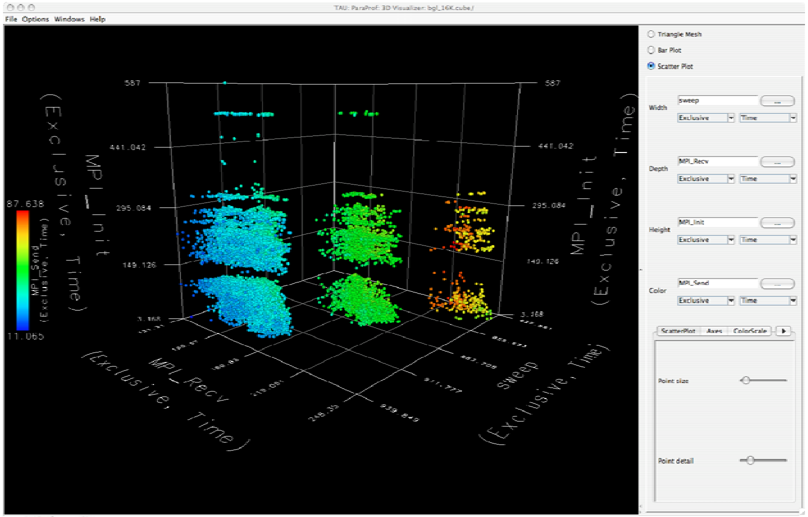
ParaProf – 3D Scatterplot (Miranda)

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



32k processors

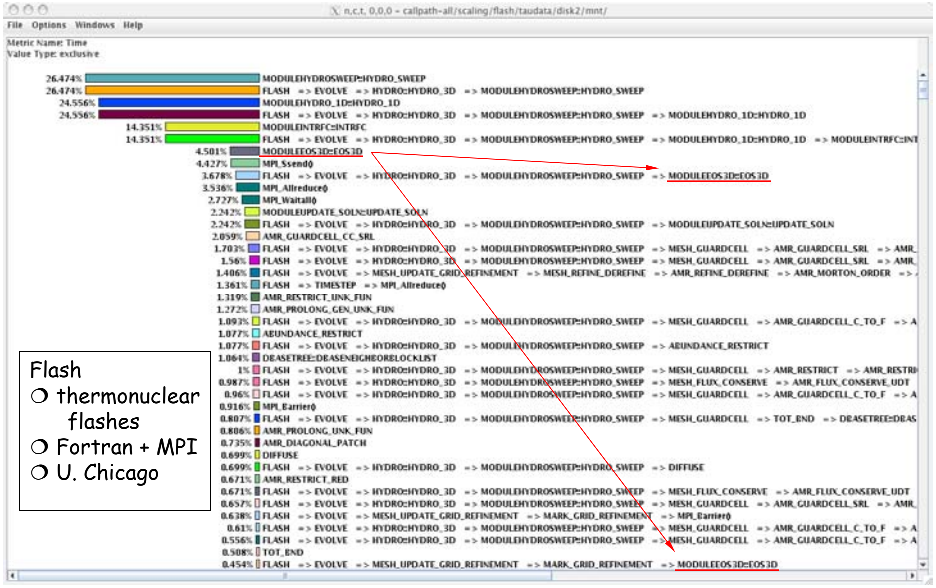
ParaProf – 3D Scatterplot (SWEEP3D CUBE)



○ Triangle Mesh
○ Bar Plot
● Scatter Plot

Width: sweep — Exclusive — Time
Depth: MPI_Recv — Exclusive — Time
Height: MPI_Init — Exclusive — Time
Color: MPI_Send — Exclusive — Time

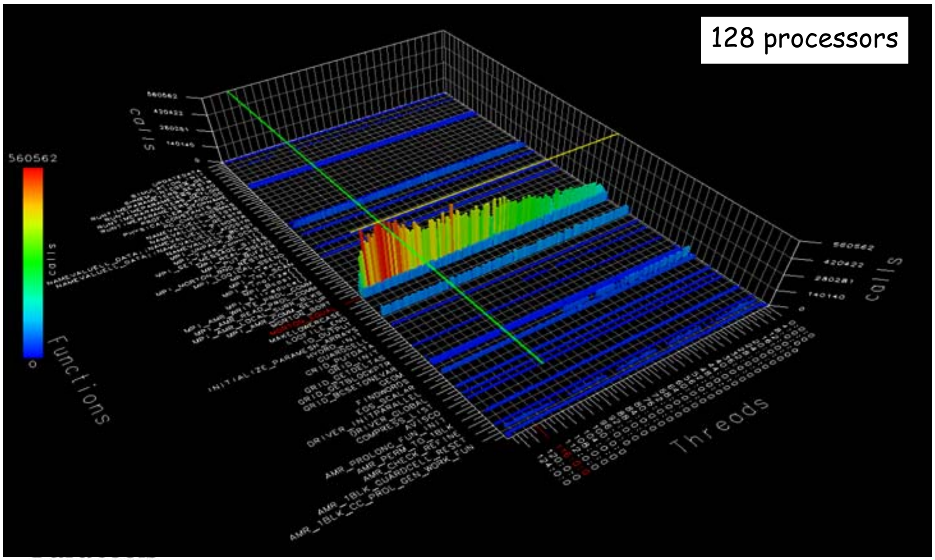
ParaProf – Callpath Profile (Flash)



```
Metric Name: Time
Value Type: exclusive

26.474% MODULEHYDROSWEEP::HYDRO_SWEEP
26.474% FLASH => EVOLVE => HYDRO::HYDRO_3D => MODULEHYDROSWEEP::HYDRO_SWEEP
24.556% MODULEHYDRO_1D::HYDRO_1D
24.556% FLASH => EVOLVE => HYDRO::HYDRO_3D => MODULEHYDROSWEEP::HYDRO_SWEEP => MODULEHYDRO_1D::HYDRO_1D
14.351% MODULEINTRFC::INTRFC
14.351% FLASH => EVOLVE => HYDRO::HYDRO_3D => MODULEHYDROSWEEP::HYDRO_SWEEP => MODULEHYDRO_1D::HYDRO_1D => MODULEINTRFC::INT
 4.501% MODULEEOS3D::EOS3D
 4.427% MPI_Ssend()
 3.678% FLASH => EVOLVE => HYDRO::HYDRO_3D => MODULEHYDROSWEEP::HYDRO_SWEEP => MODULEEOS3D::EOS3D
 3.536% MPI_Allreduce()
 2.727% MPI_Waitall()
 2.242% MODULEUPDATE_SOLN::UPDATE_SOLN
 2.242% FLASH => EVOLVE => HYDRO::HYDRO_3D => MODULEHYDROSWEEP::HYDRO_SWEEP => MODULEUPDATE_SOLN::UPDATE_SOLN
 2.059% AMR_GUARDCELL_CC_SRL
 1.703% FLASH => EVOLVE => HYDRO::HYDRO_3D => MODULEHYDROSWEEP::HYDRO_SWEEP => MESH_GUARDCELL => AMR_GUARDCELL_SRL => AMR_
  1.56% FLASH => EVOLVE => HYDRO::HYDRO_3D => MODULEHYDROSWEEP::HYDRO_SWEEP => MESH_GUARDCELL => AMR_GUARDCELL_SRL => AMR_
 1.406% FLASH => EVOLVE => MESH_UPDATE_GRID_REFINEMENT => MESH_REFINE_DEREFINE => AMR_REFINE_DEREFINE => AMR_MORTON_ORDER =>
 1.361% FLASH => TIMESTEP => MPI_Allreduce()
 1.319% AMR_RESTRICT_UNK_FUN
 1.272% AMR_PROLONG_GEN_UNK_FUN
 1.093% FLASH => EVOLVE => HYDRO::HYDRO_3D => MODULEHYDROSWEEP::HYDRO_SWEEP => MESH_GUARDCELL => AMR_GUARDCELL_C_TO_F => A
 1.077% ABUNDANCE_RESTRICT
 1.077% FLASH => EVOLVE => HYDRO::HYDRO_3D => MODULEHYDROSWEEP::HYDRO_SWEEP => ABUNDANCE_RESTRICT
 1.064% DBASETREE::DBASENEIGHBORBLOCKLIST
    1% FLASH => EVOLVE => HYDRO::HYDRO_3D => MODULEHYDROSWEEP::HYDRO_SWEEP => MESH_GUARDCELL => AMR_RESTRICT => AMR_RESTRI
0.987% FLASH => EVOLVE => HYDRO::HYDRO_3D => MODULEHYDROSWEEP::HYDRO_SWEEP => MESH_FLUX_CONSERVE => AMR_FLUX_CONSERVE_UDT
 0.96% FLASH => EVOLVE => HYDRO::HYDRO_3D => MODULEHYDROSWEEP::HYDRO_SWEEP => MESH_GUARDCELL => AMR_GUARDCELL_C_TO_F => A
0.916% MPI_Barrier()
0.807% FLASH => EVOLVE => HYDRO::HYDRO_3D => MODULEHYDROSWEEP::HYDRO_SWEEP => MESH_GUARDCELL => TOT_BND => DBASETREE::DBAS
0.806% AMR_PROLONG_UNK_FUN
0.735% AMR_DIAGONAL_PATCH
0.699% DIFFUSE
0.699% FLASH => EVOLVE => HYDRO::HYDRO_3D => MODULEHYDROSWEEP::HYDRO_SWEEP => DIFFUSE
0.671% AMR_RESTRICT_RED
0.671% FLASH => EVOLVE => HYDRO::HYDRO_3D => MODULEHYDROSWEEP::HYDRO_SWEEP => MESH_FLUX_CONSERVE => AMR_FLUX_CONSERVE_UDT
0.657% FLASH => EVOLVE => HYDRO::HYDRO_3D => MODULEHYDROSWEEP::HYDRO_SWEEP => MESH_GUARDCELL => AMR_GUARDCELL_SRL => AMR_
0.638% FLASH => EVOLVE => MESH_UPDATE_GRID_REFINEMENT => MARK_GRID_REFINEMENT => MPI_Barrier()
 0.61% FLASH => EVOLVE => HYDRO::HYDRO_3D => MODULEHYDROSWEEP::HYDRO_SWEEP => MESH_GUARDCELL => AMR_GUARDCELL_C_TO_F => A
0.556% FLASH => EVOLVE => HYDRO::HYDRO_3D => MODULEHYDROSWEEP::HYDRO_SWEEP => MESH_GUARDCELL => AMR_GUARDCELL_C_TO_F => A
0.508% TOT_BND
0.454% FLASH => EVOLVE => MESH_UPDATE_GRID_REFINEMENT => MARK_GRID_REFINEMENT => MODULEEOS3D::EOS3D
```

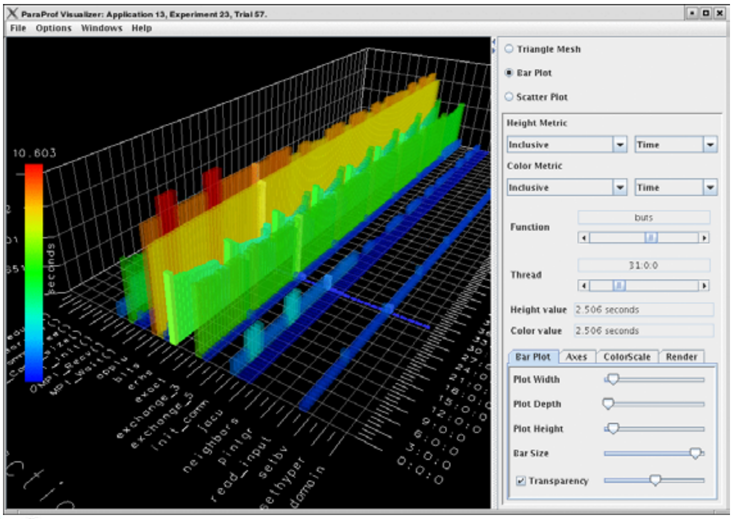
Flash
- thermonuclear flashes
- Fortran + MPI
- U. Chicago

ParaProf – 3D Full Profile Bar Plot (Flash)



128 processors

ParaProf Bar Plot (Zoom in/out +/-)



ParaProf Visualizer: Application 13, Experiment 23, Trial 57.

○ Triangle Mesh
● Bar Plot
○ Scatter Plot

Height Metric: Inclusive, Time

Color Metric: Inclusive, Time

Function: buts

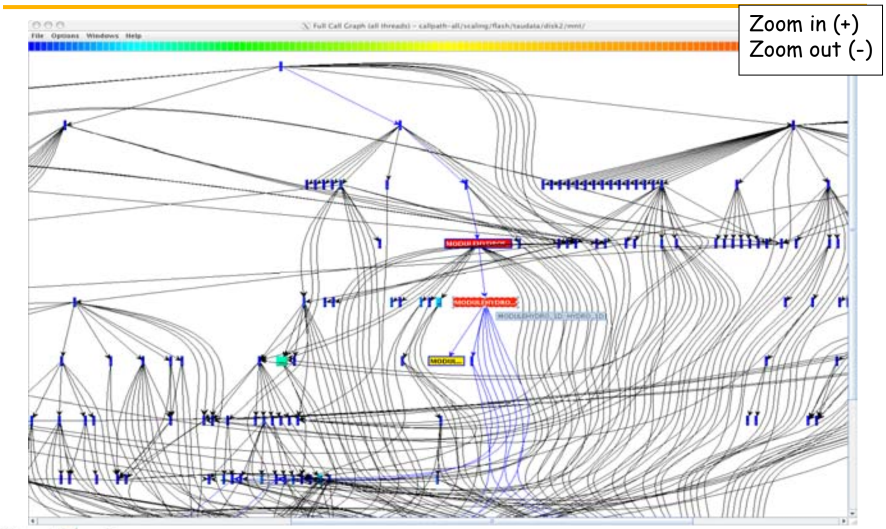
Thread: 31:0:0

Height value: 2.506 seconds

Color value: 2.506 seconds

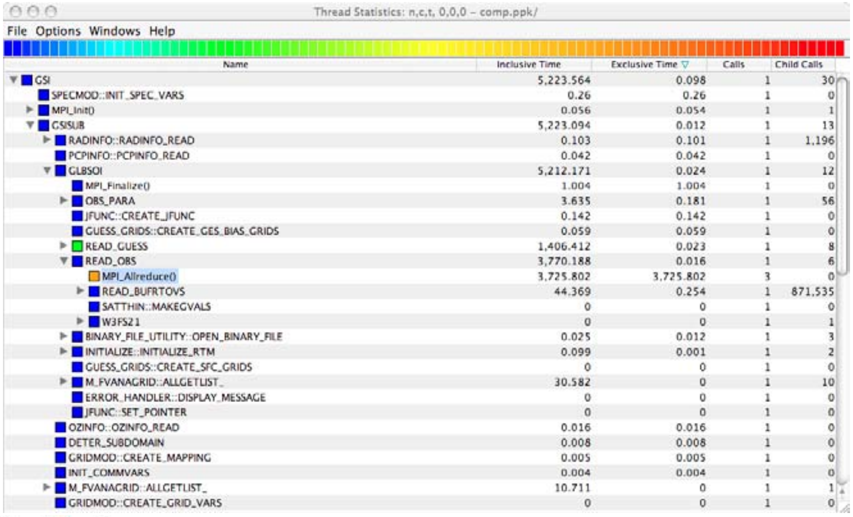
☑ Transparency

ParaProf – Callgraph Zoomed (Flash)



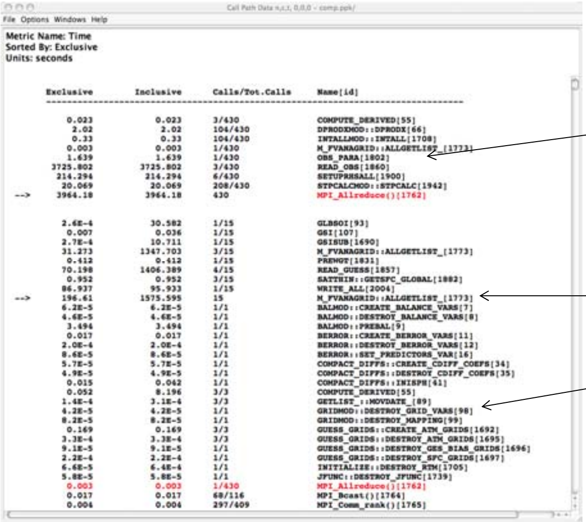
Zoom in (+)
Zoom out (-)

ParaProf - Thread Statistics Table (GSI)



Name	Inclusive Time	Exclusive Time ▽	Calls	Child Calls
▼ GSI	5,223.564	0.098	1	30
SPECMOD::INIT_SPEC_VARS	0.26	0.26	1	0
▶ MPI_Init()	0.056	0.054	1	1
▼ GSISUB	5,223.094	0.012	1	13
▶ RADINFO::RADINFO_READ	0.103	0.101	1	1,196
PCPINFO::PCPINFO_READ	0.042	0.042	1	0
▼ GLBSOI	5,212.171	0.024	1	12
MPI_Finalize()	1.004	1.004	1	0
▶ OBS_PARA	3.635	0.181	1	56
JFUNC::CREATE_JFUNC	0.142	0.142	1	0
GUESS_GRIDS::CREATE_GES_BIAS_GRIDS	0.059	0.059	1	0
▶ READ_GUESS	1,406.412	0.023	1	8
▼ READ_OBS	3,770.188	0.016	1	6
MPI_Allreduce()	3,725.802	3,725.802	3	0
▶ READ_BUFRTOVS	44.369	0.254	1	871,535
SATTHIN::MAKEGVALS	0	0	1	0
▶ W3FS21	0	0	1	1
▶ BINARY_FILE_UTILITY::OPEN_BINARY_FILE	0.025	0.012	1	3
▶ INITIALIZE::INITIALIZE_RTM	0.099	0.001	1	2
GUESS_GRIDS::CREATE_SFC_GRIDS	0	0	1	0
▶ M_FVANAGRID::ALLGETLIST_	30.582	0	1	10
ERROR_HANDLER::DISPLAY_MESSAGE	0	0	1	0
JFUNC::SET_POINTER	0	0	1	0
OZINFO::OZINFO_READ	0.016	0.016	1	0
DETER_SUBDOMAIN	0.008	0.008	1	0
GRIDMOD::CREATE_MAPPING	0.005	0.005	1	0
INIT_COMMVARS	0.004	0.004	1	0
▶ M_FVANAGRID::ALLGETLIST_	10.711	0	1	1
GRIDMOD::CREATE_GRID_VARS	0	0	1	0

ParaProf - Callpath Thread Relations Window



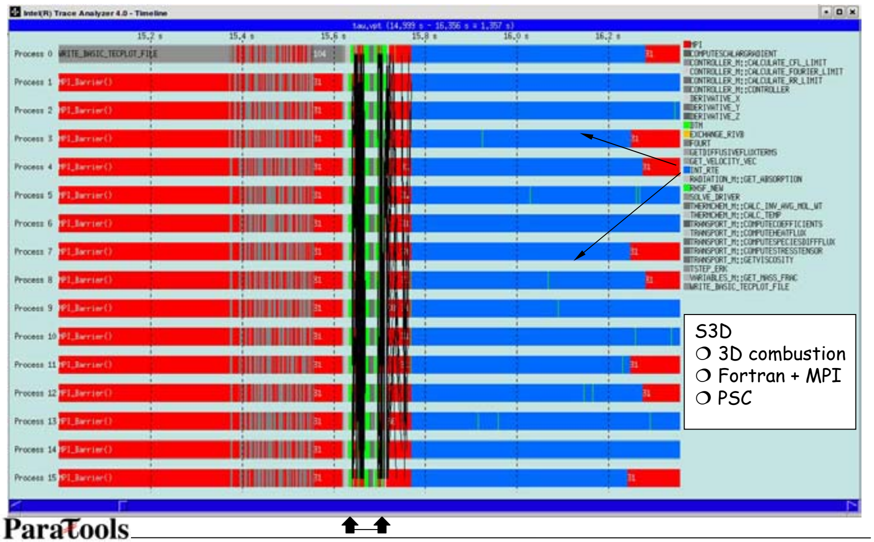
```
Metric Name: Time
Sorted By: Exclusive
Units: seconds

        Exclusive       Inclusive     Calls/Tot.Calls     Name[id]
        ----------------------------------------------------------------------------

            0.023           0.023     3/430               COMPUTE_DERIVED[55]
             2.02            2.02     104/430             DPRODXMOD::DPRODX[66]
             0.33            0.33     104/430             INTALLMOD::INTALL[1708]
            0.003           0.003     1/430               M_FVANAGRID::ALLGETLIST_[1773]
            1.639           1.639     1/430               OBS_PARA[1802]
         3725.802        3725.802     3/430               READ_OBS[1860]
          214.294         214.294     6/430               SETUPRHSALL[1900]
           20.069          20.069     208/430             STPCALCMOD::STPCALC[1942]
-->       3964.18         3964.18     430                 MPI_Allreduce()[1762]

           2.6E-4          30.582     1/15                GLBSOI[93]
            0.007           0.036     1/15                GSI[107]
           2.7E-4          10.711     1/15                GSISUB[1690]
           31.273        1347.703     3/15                M_FVANAGRID::ALLGETLIST_[1773]
            0.412           0.412     1/15                PREWGT[1831]
           70.198        1406.389     4/15                READ_GUESS[1857]
            0.952           0.952     3/15                SATTHIN::GETSFC_GLOBAL[1882]
           86.937          95.933     1/15                WRITE_ALL[2004]
-->        196.61        1575.595     15                  M_FVANAGRID::ALLGETLIST_[1773]
           6.2E-5          6.2E-5     1/1                 BALMOD::CREATE_BALANCE_VARS[7]
           4.6E-5          4.6E-5     1/1                 BALMOD::DESTROY_BALANCE_VARS[8]
            3.494           3.494     1/1                 BALMOD::PREBAL[9]
            0.017           0.017     1/1                 BERROR::CREATE_BERROR_VARS[11]
           2.0E-4          2.0E-4     1/1                 BERROR::DESTROY_BERROR_VARS[12]
           8.6E-5          8.6E-5     1/1                 BERROR::SET_PREDICTORS_VAR[16]
           5.7E-5          5.7E-5     1/1                 COMPACT_DIFFS::CREATE_CDIFF_COEFS[34]
           4.9E-5          4.9E-5     1/1                 COMPACT_DIFFS::DESTROY_CDIFF_COEFS[35]
            0.015           0.042     1/1                 COMPACT_DIFFS::INISPH[41]
            0.052           8.196     3/3                 COMPUTE_DERIVED[55]
           1.4E-4          3.1E-4     3/3                 GETLIST_::MOVDATE_[89]
           4.2E-5          4.2E-5     1/1                 GRIDMOD::DESTROY_GRID_VARS[98]
           8.2E-5          8.2E-5     1/1                 GRIDMOD::DESTROY_MAPPING[99]
            0.169           0.169     3/3                 GUESS_GRIDS::CREATE_ATM_GRIDS[1692]
           3.3E-4          3.3E-4     3/3                 GUESS_GRIDS::DESTROY_ATM_GRIDS[1695]
           9.1E-5          9.1E-5     1/1                 GUESS_GRIDS::DESTROY_GES_BIAS_GRIDS[1696]
           2.2E-4          2.2E-4     1/1                 GUESS_GRIDS::DESTROY_SFC_GRIDS[1697]
           6.6E-5          6.4E-4     1/1                 INITIALIZE::DESTROY_RTM[1705]
           5.8E-5          5.8E-5     1/1                 JFUNC::DESTROY_JFUNC[1739]
            0.003           0.003     1/430               MPI_Allreduce()[1762]
            0.017           0.017     68/116              MPI_Bcast()[1764]
            0.004           0.004     297/409             MPI_Comm_rank()[1765]
```

Parent

Routine

Children

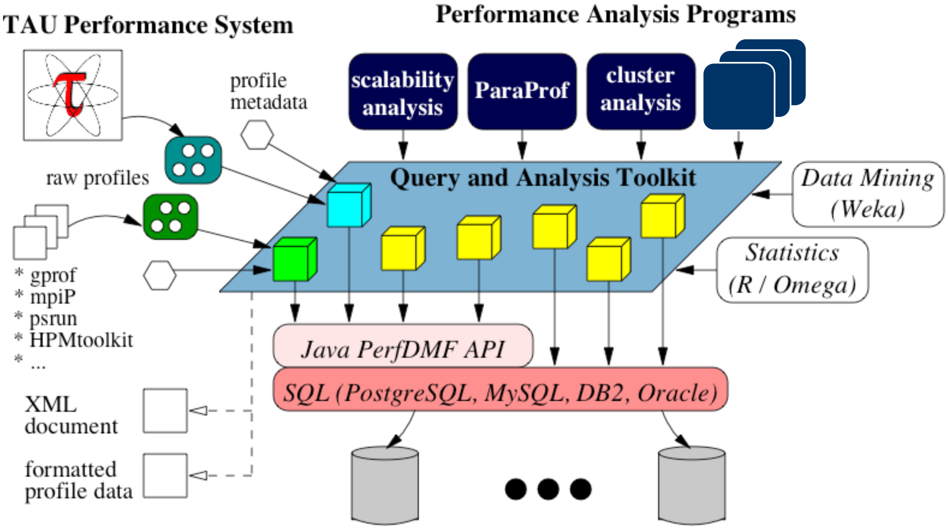
Vampir – Trace Analysis (TAU-to-VTF3) (S3D)



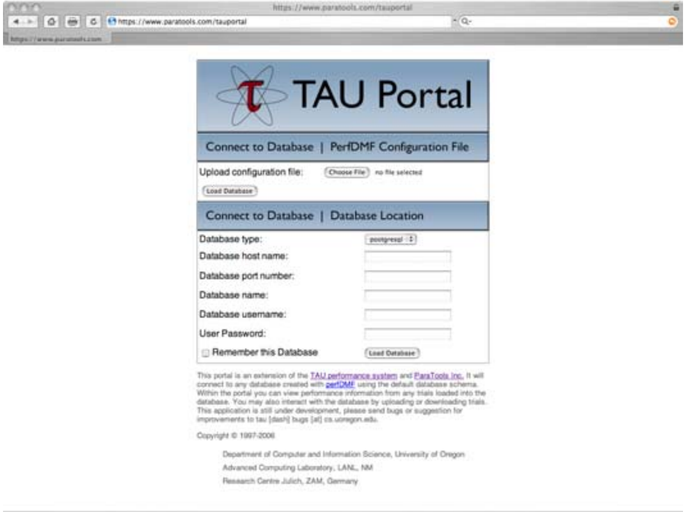
S3D
- 3D combustion
- Fortran + MPI
- PSC

Vampir – Trace Zoomed (S3D)

PerfDMF: Performance Data Mgmt. Framework



TAU Portal - www.paratools.com/tauportal



Connect to Database | PerfDMF Configuration File

Upload configuration file:

Connect to Database | Database Location

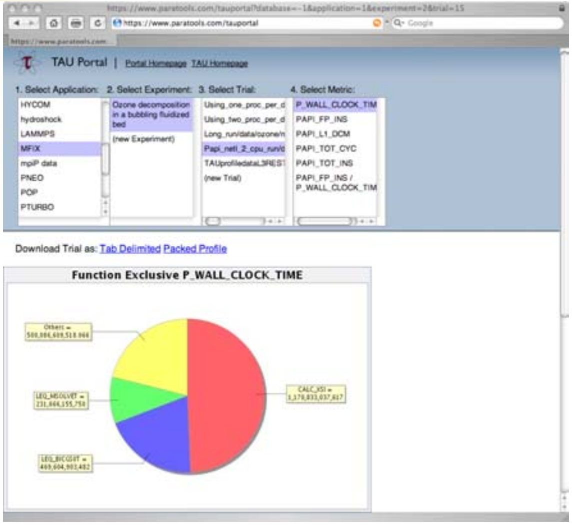
Database type:
Database host name:
Database port number:
Database name:
Database username:
User Password:
☐ Remember this Database

This portal is an extension of the TAU performance system and ParaTools Inc. It will connect to any database created with perfDMF using the default database schema. Within the portal you can view performance information from any trials loaded into the database. You may also interact with the database by uploading or downloading trials. This application is still under development, please send bugs or suggestion for improvements to tau [dash] bugs [at] cs.uoregon.edu.

Copyright © 1997-2006

Department of Computer and Information Science, University of Oregon
Advanced Computing Laboratory, LANL, NM
Research Centre Julich, ZAM, Germany

TAU Portal



1. Select Application:
HYCOM
hydroshock
LAMMPS
MFIX
mpiP data
PNEO
POP
PTURBO

2. Select Experiment:
Ozone decomposition in a bubbling fluidized bed
(new Experiment)

3. Select Trial:
Using_one_proc_per_d
Using_two_proc_per_d
Long_run/data/ozone/n
Papi_nell_2_cpu_run/d
TAUprofiledataL3REST
(new Trial)

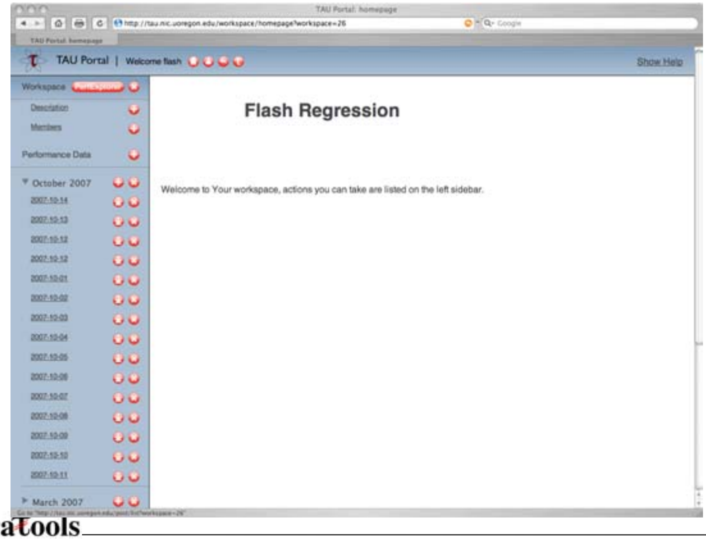
4. Select Metric:
P_WALL_CLOCK_TIM
PAPI_FP_INS
PAPI_L1_DCM
PAPI_TOT_CYC
PAPI_TOT_INS
PAPI_FP_INS / P_WALL_CLOCK_TIM

Download Trial as: Tab Delimited Packed Profile

TAU Portal

- Web-based access to TAU
- Support collaborative performance study
  - Secure performance data sharing
  - Does not require TAU installation
  - Launch TAU performance tools with Java WebStart
    - ParaProf, PerfExplorer
- FLASH regression testing
  - Nightly regression testcases
  - Uploaded to the database automatically
  - Interactive review of performance through TAU portal
  - Multi-experiment analysis

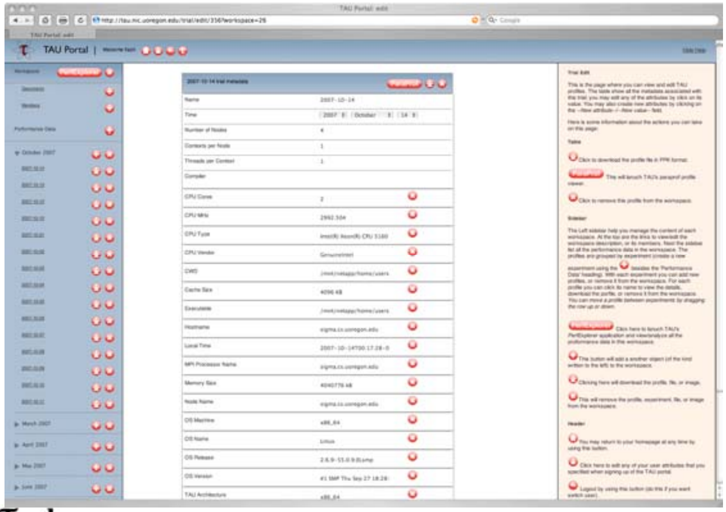
Portal: Nightly Performance Regression Testing



Flash Regression

Welcome to Your workspace, actions you can take are listed on the left sidebar.

TAU Portal: Launch ParaProf/PerfExplorer



PerfExplorer: Regression Testing

- Performance Data
  - jdbc:mysql://www.paratools.com:3306/paratool_tau
  - jdbc:mysql://icl.cs.utk.edu:3306/perfdmf
  - jdbc:mysql://www.nic.uoregon.edu:3306/Flash_Regression_production
    - Portal
      - April 2007
      - August 2007
      - July 2007
      - June 2007
      - March 2007
      - May 2007
      - October 2007
      - September 2007
    - Views

Chart Title:
Series Name/Value: experiment.name
X Axis Value: trial.XML_METADATA
X Axis Name:
Y Axis Value: mean.inclusive
Y Axis Name:
Dimension reduction: None
Cutoff (0<x<100):
Metric: Time
Units: seconds
Event:
XML Field: UTC Time

PerfExplorer: Limiting Events (> 3%), Oct 2007

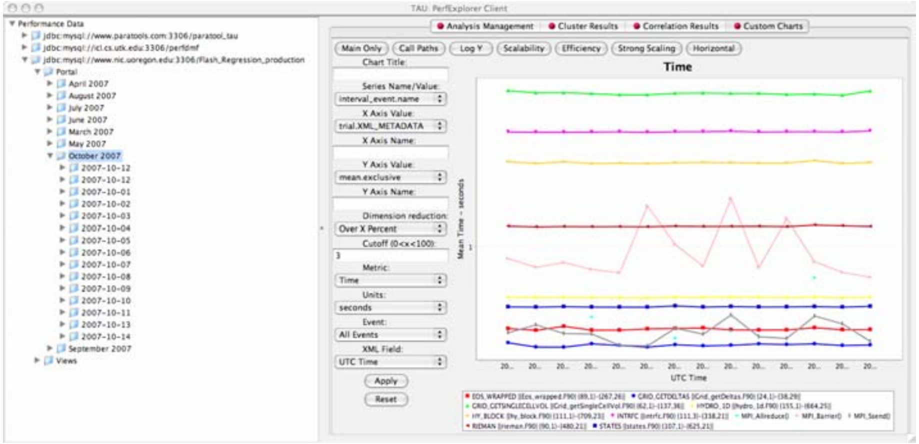


Chart Title:
Series Name/Value: interval_event.name
X Axis Value: trial.XML_METADATA
X Axis Name:
Y Axis Value: mean.exclusive
Y Axis Name:
Dimension reduction: Over X Percent
Cutoff (0<x<100): 3
Metric: Time
Units: seconds
Event: All Events
XML Field: UTC Time

PerfExplorer: Exclusive Time for Events (2007)

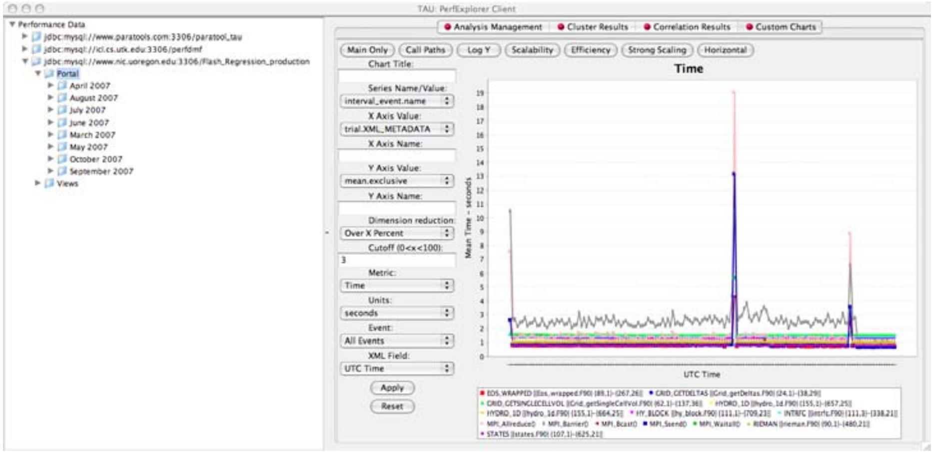


Chart Title:
Series Name/Value: interval_event.name
X Axis Value: trial.XML_METADATA
X Axis Name:
Y Axis Value: mean.exclusive
Y Axis Name:
Dimension reduction: Over X Percent
Cutoff (0<x<100): 3
Metric: Time
Units: seconds
Event: All Events
XML Field: UTC Time

Using Performance Database (PerfDMF)

- Configure PerfDMF (Done by each user)

```
% perfdmf_configure --create-defaults
```

  - Choose derby, PostgreSQL, MySQL, Oracle or DB2
  - Hostname
  - Username
  - Password
  - Say yes to downloading required drivers (we are not allowed to distribute these)
  - Stores parameters in your ~/.ParaProf/perfdmf.cfg file
- Configure PerfExplorer (Done by each user)

```
% perfexplorer_configure
```

- Execute PerfExplorer

```
% perfexplorer
```

PerfDMF and the TAU Portal

- Development of the TAU portal
  - Common repository for collaborative data sharing
  - Profile uploading, downloading, user management
  - Paraprof, PerfExplorer can be launched from the portal using Java Web Start (no TAU installation required)
- Portal URL

  http://tau.nic.uoregon.edu

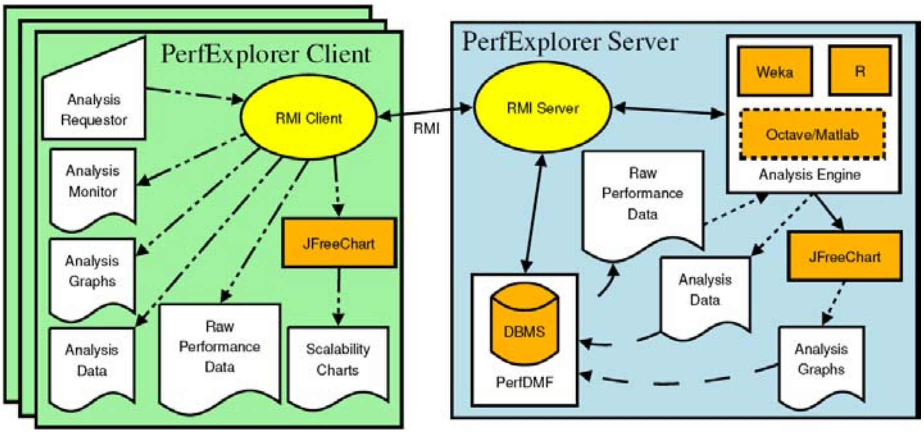
Performance Data Mining (Objectives)

- Conduct parallel performance analysis process
  - In a systematic, collaborative and reusable manner
  - Manage performance complexity
  - Discover performance relationship and properties
  - Automate process
- Multi-experiment performance analysis
- Large-scale performance data reduction
  - Summarize characteristics of large processor runs
- Implement extensible analysis framework
  - Abstraction / automation of data mining operations
  - Interface to existing analysis and data mining tools

Performance Data Mining (PerfExplorer)

- Performance knowledge discovery framework
  - Data mining analysis applied to parallel performance data
    - comparative, clustering, correlation, dimension reduction, …
  - Use the existing TAU infrastructure
    - TAU performance profiles, PerfDMF
  - Client-server based system architecture
- Technology integration
  - Java API and toolkit for portability
  - PerfDMF
  - R-project/Omegahat, Octave/Matlab statistical analysis
  - WEKA data mining package
  - JFreeChart for visualization, vector output (EPS, SVG)

Performance Data Mining (PerfExplorer)



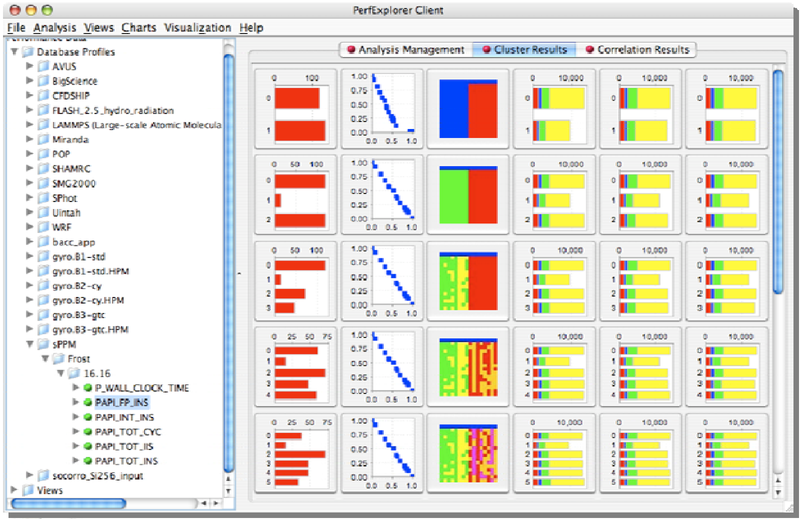
PerfExplorer - Analysis Methods

- Data summaries, distributions, scatter plots
- Clustering
  - *k*-means
  - Hierarchical
- Correlation analysis
- Dimension reduction
  - PCA
  - Random linear projection
  - Thresholds
- Comparative analysis
- Data management views

PerfExplorer - Cluster Analysis

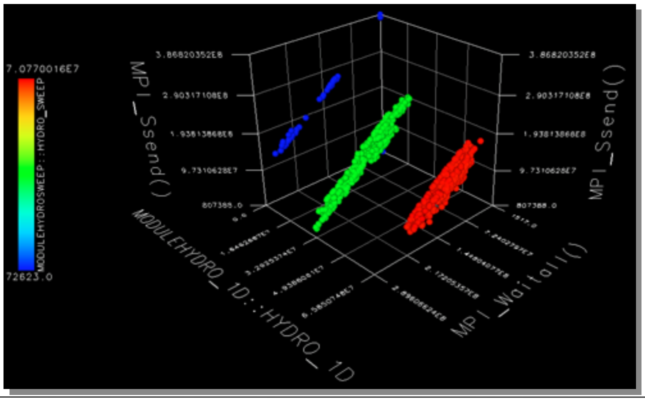
- Performance data represented as vectors - each dimension is the cumulative time for an event
- *k*-means: *k* random centers are selected and instances are grouped with the "closest" (Euclidean) center
- New centers are calculated and the process repeated until stabilization or max iterations
- Dimension reduction necessary for meaningful results
- Virtual topology, summaries constructed

PerfExplorer - Cluster Analysis (sPPM)



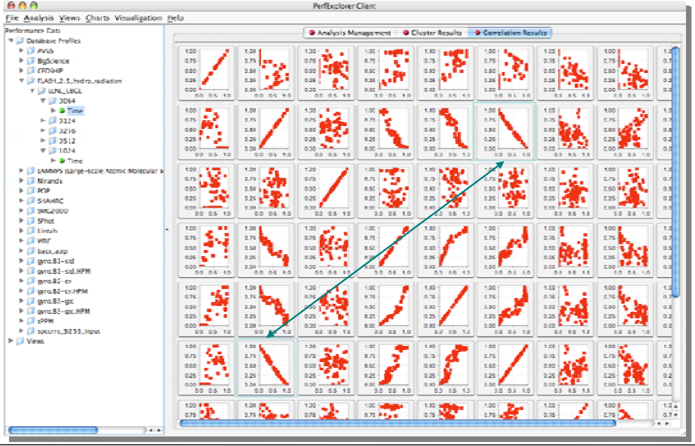
PerfExplorer - Cluster Analysis

- Four significant events automatically selected (from 16K processors)
- Clusters and correlations are visible



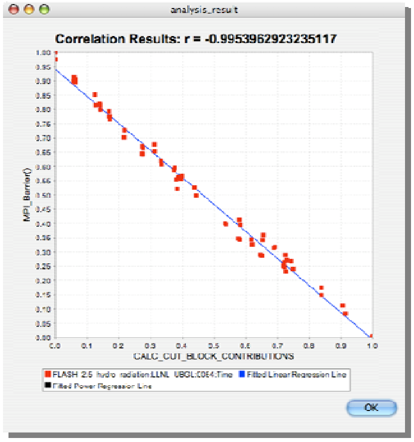
PerfExplorer - Correlation Analysis (Flash)

- Describes strength and direction of a linear relationship between two variables (events) in the data



PerfExplorer - Correlation Analysis (Flash)

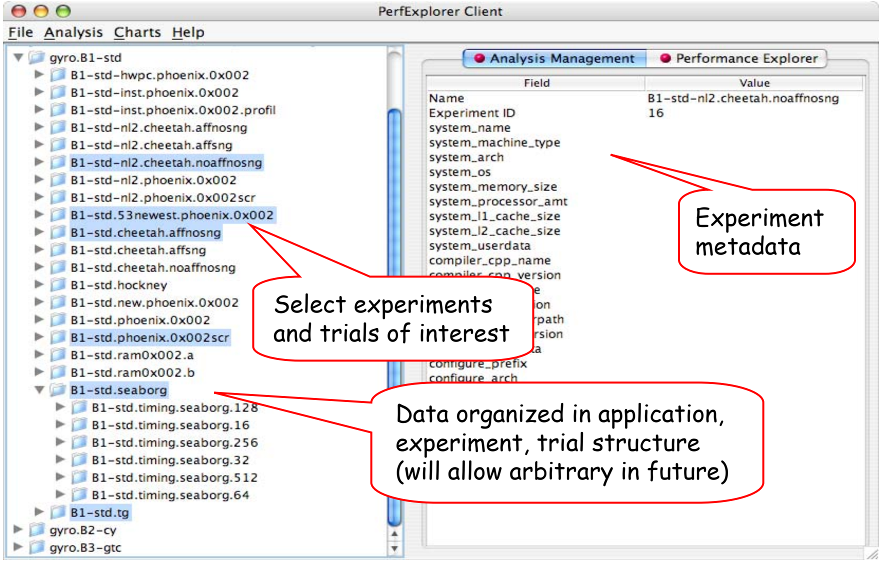
- -0.995 indicates strong, negative relationship
- As CALC_CUT_BLOCK_CONTRIBUTIONS() increases in execution time, MPI_Barrier() decreases



PerfExplorer - Comparative Analysis

- Relative speedup, efficiency
  - total runtime, by event, one event, by phase
- Breakdown of total runtime
- Group fraction of total runtime
- Correlating events to total runtime
- Timesteps per second
- Performance Evaluation Research Center (PERC)
  - PERC tools study (led by ORNL, Pat Worley)
  - In-depth performance analysis of select applications
  - Evaluation performance analysis requirements
  - Test tool functionality and ease of use

PerfExplorer - Interface



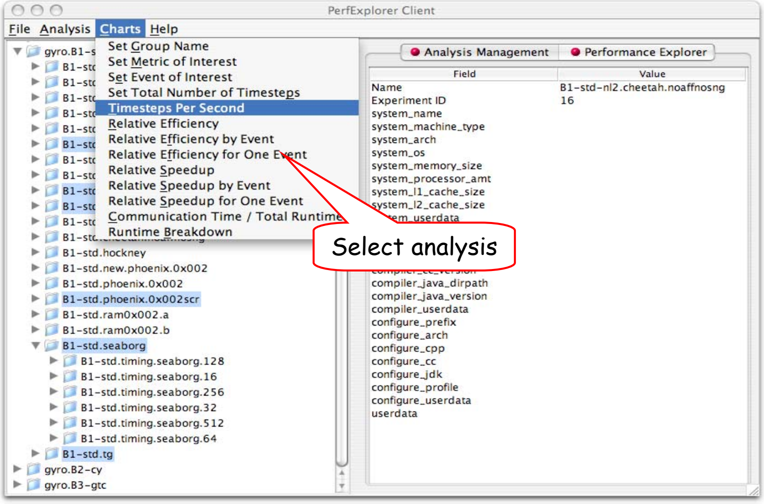
Field	Value
Name	B1-std-nl2.cheetah.noaffnosng
Experiment ID	16
system_name	
system_machine_type	
system_arch	
system_os	
system_memory_size	
system_processor_amt	
system_l1_cache_size	
system_l2_cache_size	
system_userdata	
compiler_cpp_name	
configure_prefix	
configure_arch	

Experiment metadata

Select experiments and trials of interest

Data organized in application, experiment, trial structure (will allow arbitrary in future)

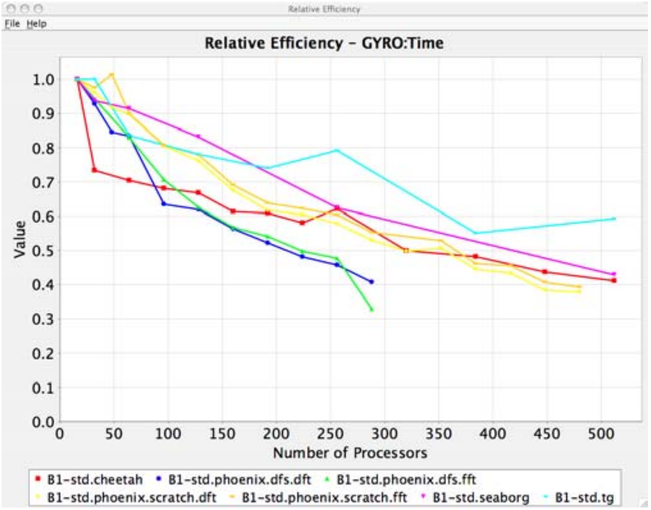
PerfExplorer - Interface



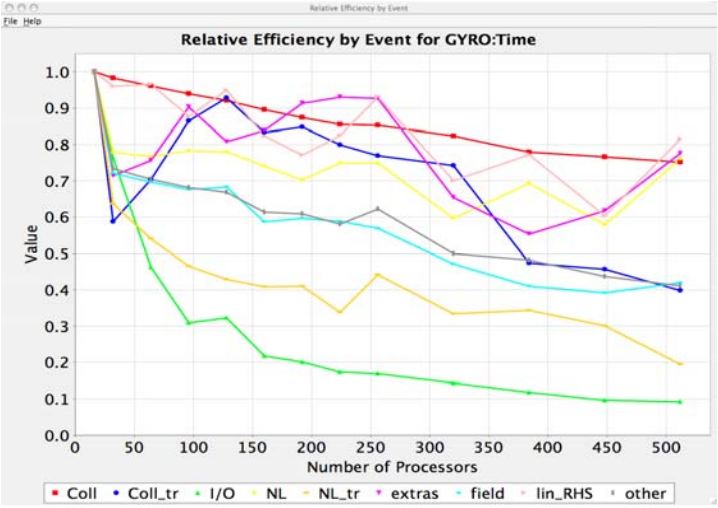
Field	Value
Name	B1-std-nl2.cheetah.noaffnosng
Experiment ID	16
system_name	
system_machine_type	
system_arch	
system_os	
system_memory_size	
system_processor_amt	
system_l1_cache_size	
system_l2_cache_size	
compiler_java_dirpath	
compiler_java_version	
compiler_userdata	
configure_prefix	
configure_arch	
configure_cpp	
configure_cc	
configure_jdk	
configure_profile	
configure_userdata	
userdata	

Select analysis

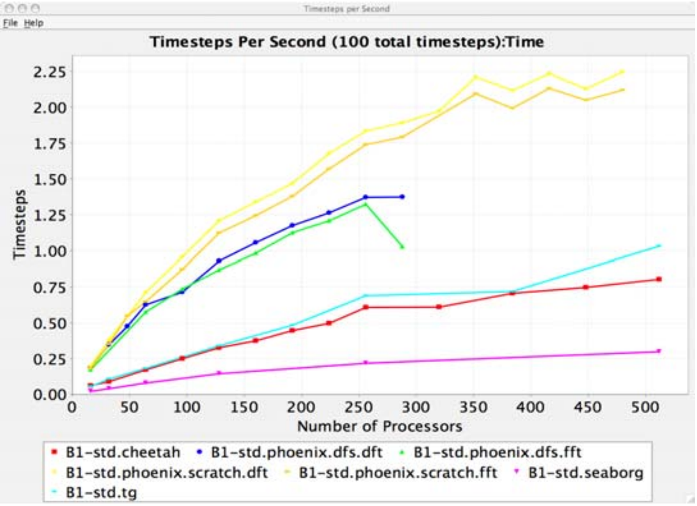
PerfExplorer - Relative Efficiency Plots



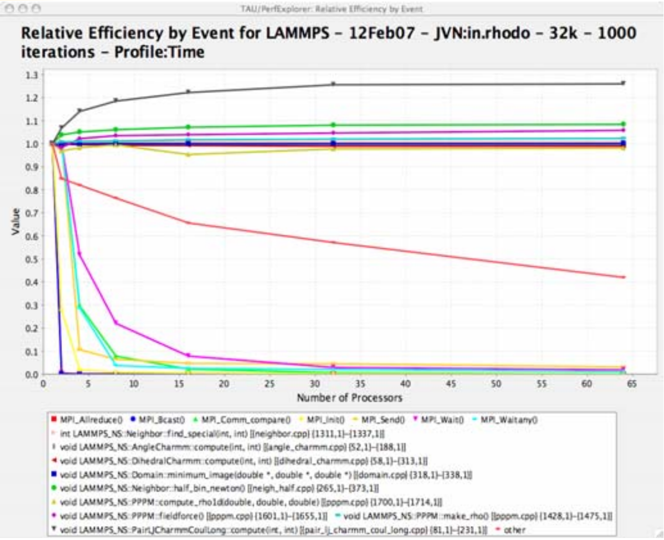
PerfExplorer - Relative Efficiency by Routine



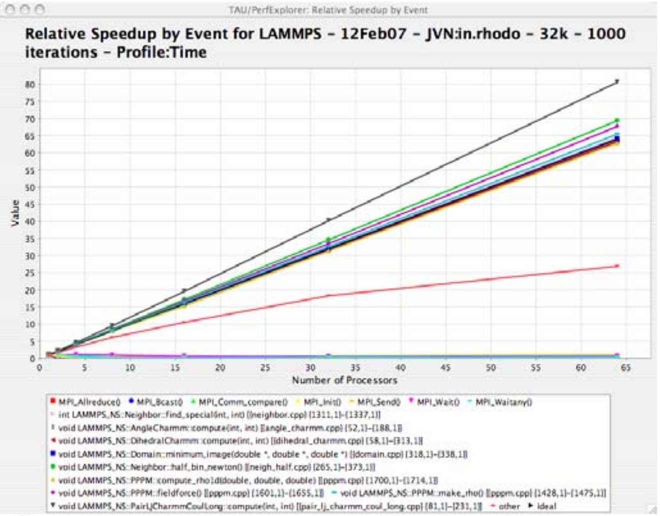
PerfExplorer - Timesteps Per Second



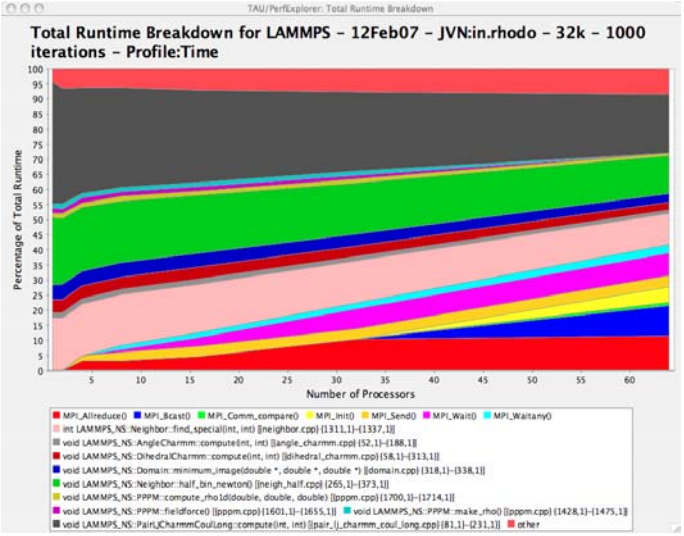
PerfExplorer - Relative Efficiency



PerfExplorer - Relative Speedup by Event

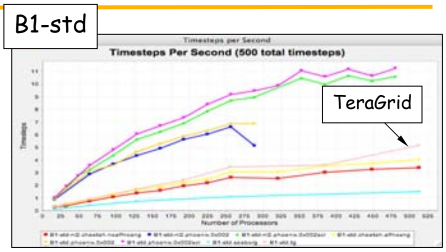


PerfExplorer - Runtime Breakdown

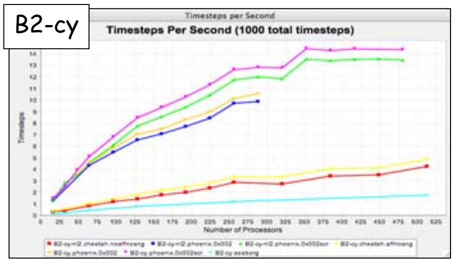


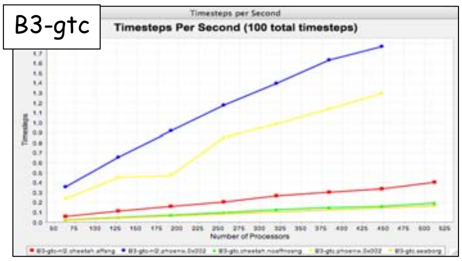
PerfExplorer - Timesteps per Second for GYRO

- Cray X1 is the fastest to solution
  - In all 3 tests
- FFT (nl2) improves time
  - B3-gtc only
- TeraGrid faster than p690
  - For B1-std?
- All plots generated automatically


B1-std

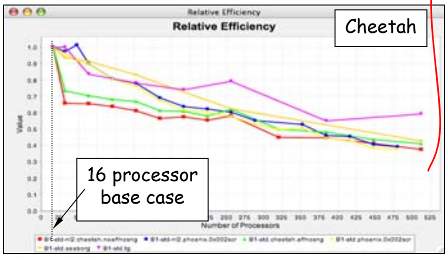
TeraGrid


B2-cy

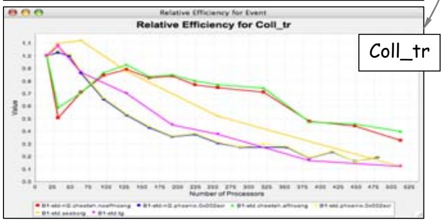

B3-gtc

PerfExplorer - Relative Efficiency (B1-std)

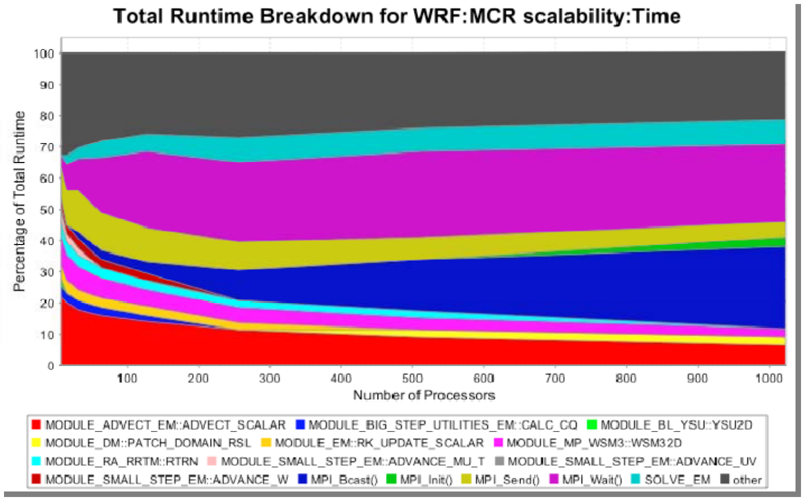
- By experiment (B1-std)
  - Total runtime (Cheetah (red))
- By event for one experiment
  - Coll_tr (blue) is significant
- By experiment for one event
  - Shows how Coll_tr behaves for all experiments


Cheetah

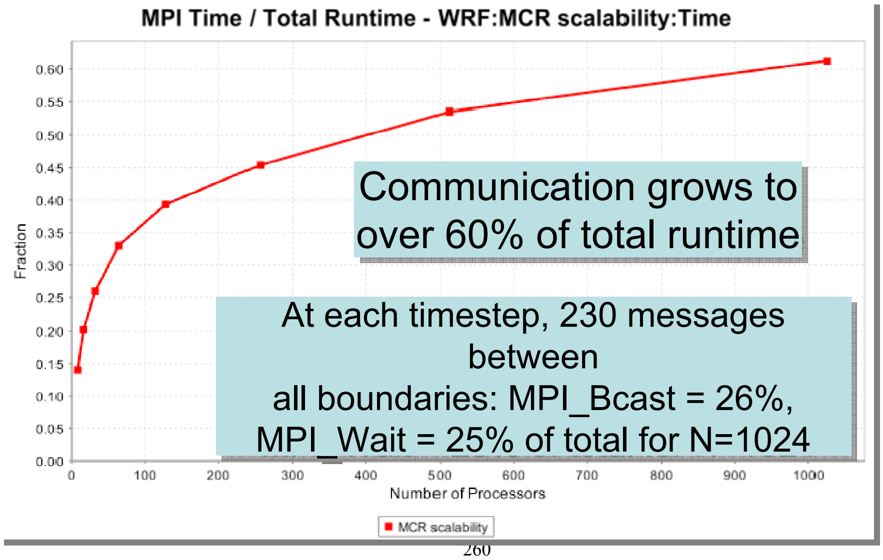
16 processor base case


Coll_tr

PerfExplorer - Runtime Breakdown



Group % of Total



Communication grows to over 60% of total runtime

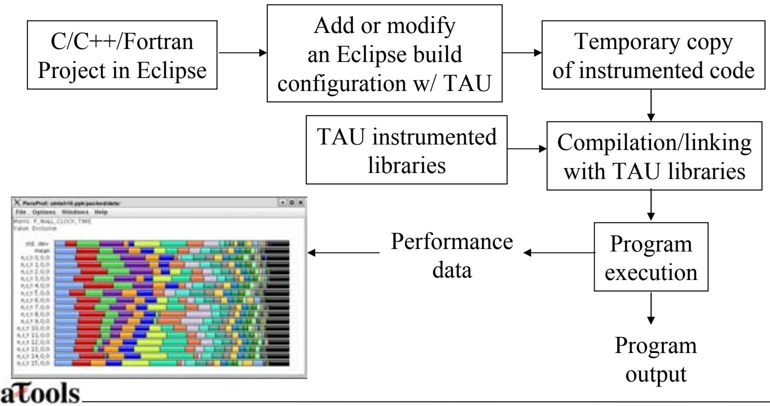
At each timestep, 230 messages between
all boundaries: MPI_Bcast = 26%,
MPI_Wait = 25% of total for N=1024

TAU Integration with IDEs

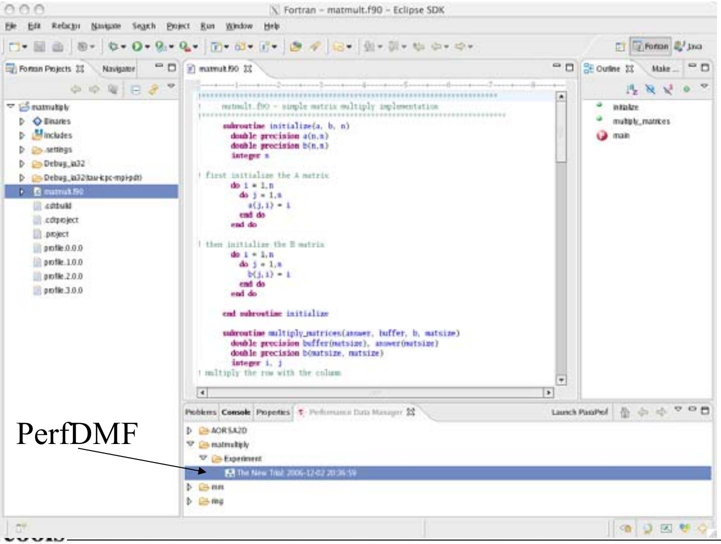
- High performance software development environments
  - Tools may be complicated to use
  - Interfaces and mechanisms differ between platforms / OS
- Integrated development environments
  - Consistent development environment
  - Numerous enhancements to development process
  - Standard in industrial 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

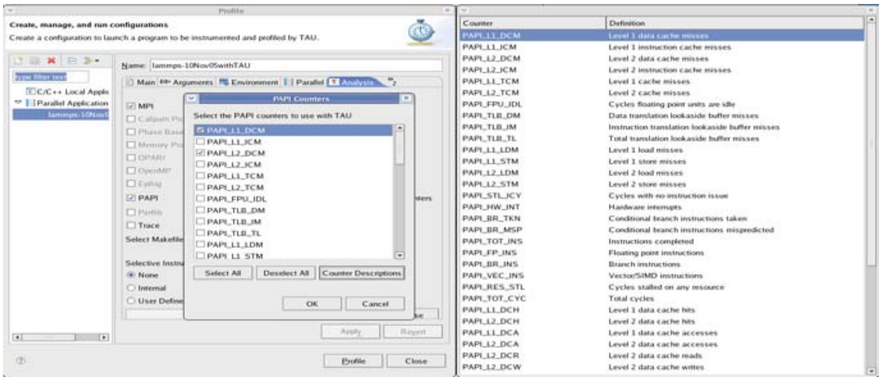


Fortran - matmult.f90 - Eclipse SDK

```
!****************************************************************************
!    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
```

PerfDMF

Choosing PAPI Counters with TAU in Eclipse



Profile

Create, manage, and run configurations

Create a configuration to launch a program to be instrumented and profiled by TAU.

Name: lammps-10Nov05withTAU

☑ MPI
☐ Callpath Pr
☐ Phase Base
☐ Memory Pr
☐ OPARI
☐ OpenMP
☐ Epilog
☑ PAPI
☐ Perflib
☐ Trace

Select Makefile

Selective Instru
● None
○ Internal
○ User Define

PAPI Counters

Select the PAPI counters to use with TAU

☑ PAPI_L1_DCM
☐ PAPI_L1_ICM
☑ PAPI_L2_DCM
☐ PAPI_L2_ICM
☐ PAPI_L1_TCM
☐ PAPI_L2_TCM
☐ PAPI_FPU_IDL
☐ PAPI_TLB_DM
☐ PAPI_TLB_IM
☐ PAPI_TLB_TL
☐ PAPI_L1_LDM
☐ PAPI_L1_STM

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

```
% /usr/global/tools/pkgs/eclipse/eclipse
```

TAU Performance System Status

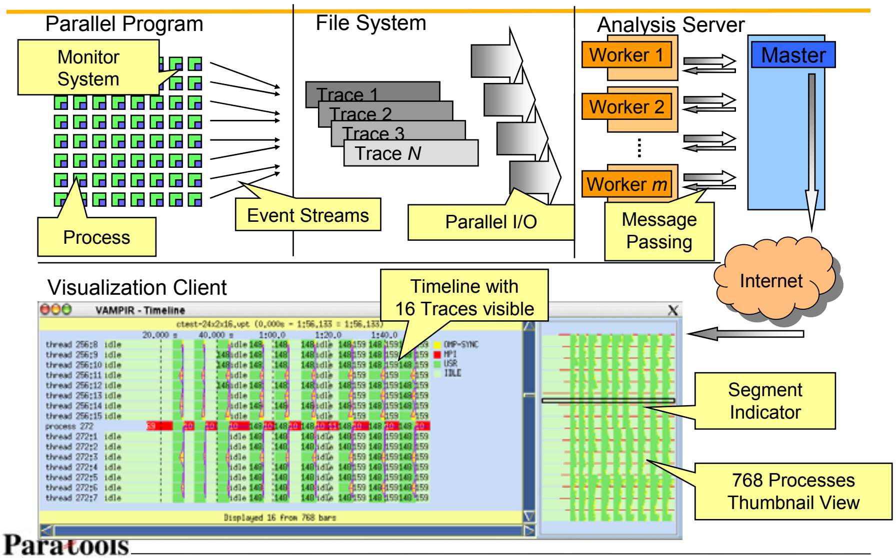
- Computing platforms (selected)
  - IBM SP/pSeries/BGL/Cell PPE, SGI Altix/Origin, Cray T3E/SV-1/X1/XT3, HP (Compaq) SC (Tru64), Sun, Linux clusters (IA-32/64, Alpha, PPC, PA-RISC, Power, Opteron), Apple (G4/5, OS X), Hitachi SR8000, NEC SX Series, Windows …
- Programming languages
  - C, C++, Fortran 77/90/95, HPF, Java, Python
- Thread libraries (selected)
  - pthreads, OpenMP, SGI sproc, Java,Windows, Charm++
- Compilers (selected)
  - Intel, PGI, GNU, Fujitsu, Sun, PathScale, SGI, Cray, IBM, HP, NEC, Absoft, Lahey, Nagware, ...

Part V: 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



Parallel Program

Monitor System

Process

Event Streams

File System

Trace 1
Trace 2
Trace 3
Trace *N*

Parallel I/O

Analysis Server

Worker 1
Worker 2
⋮
Worker *m*

Master

Message Passing

Internet

Visualization Client

Timeline with 16 Traces visible

Segment Indicator

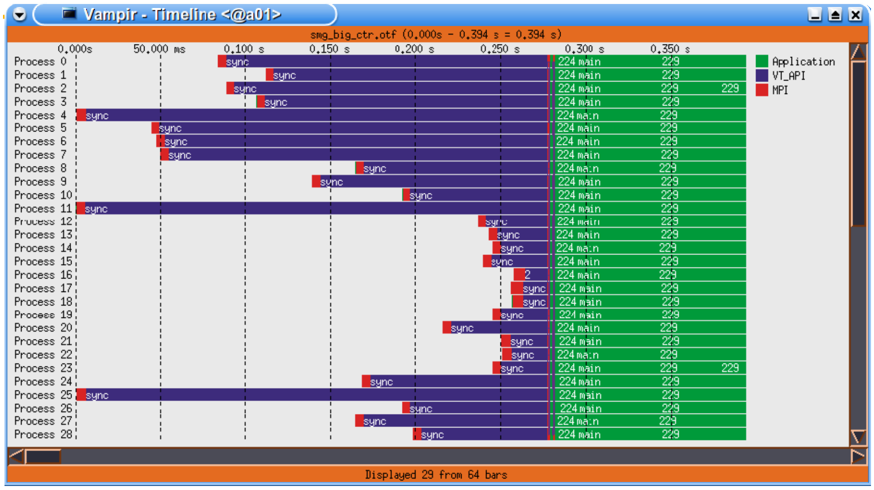
768 Processes Thumbnail View

Vampir Displays

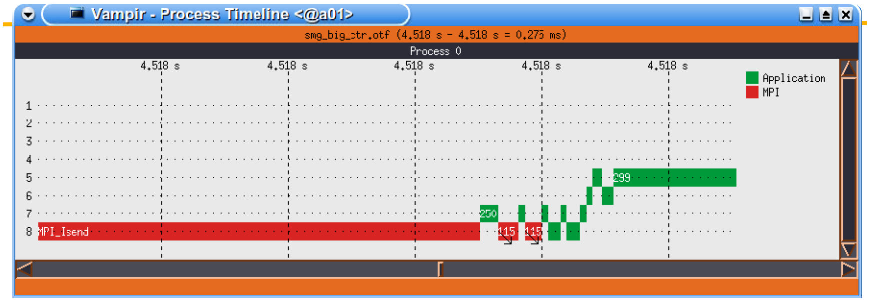
The main displays of Vampir:

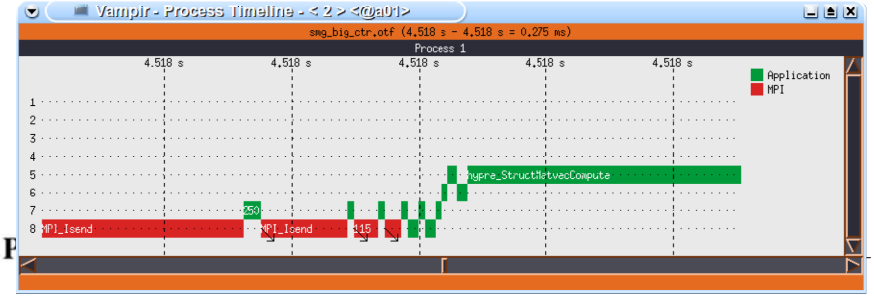
- Global Timeline
- Process Timeline w/o Counters
- Statistic Summary
- Summary Timeline
- Message Statistics
- Collective Operation Statistics
- Counter Timeline
- Call Tree

Vampir Global Timeline Display

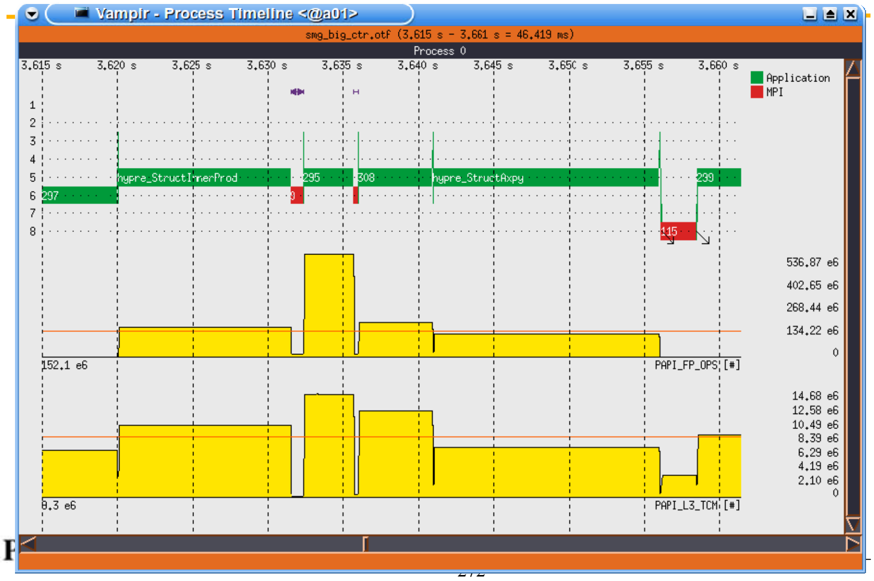


Process Timeline Display

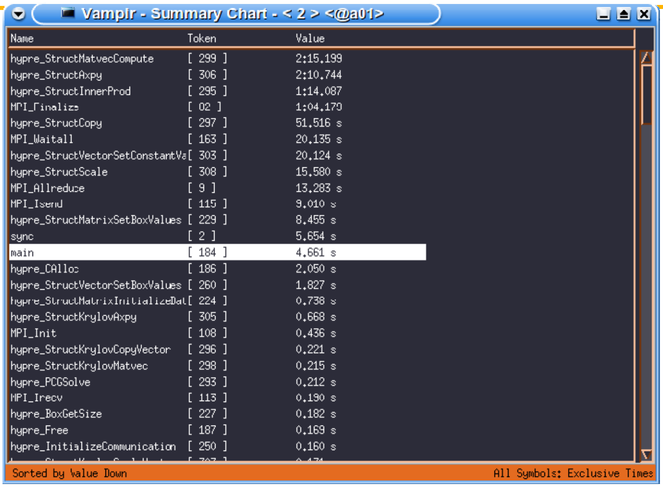




Process Timeline with Counters

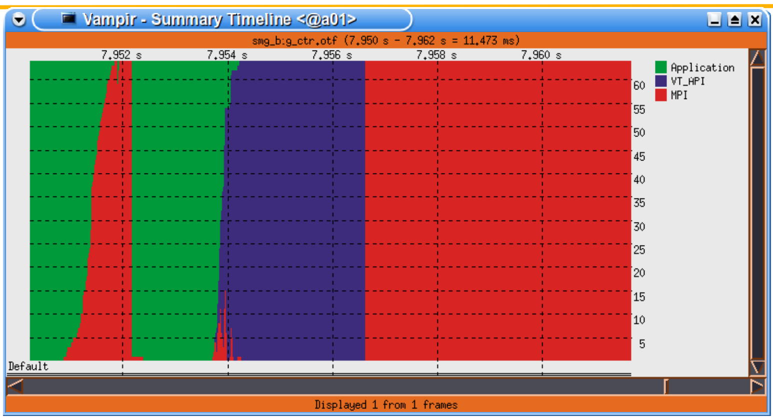


Statistic Summary Display

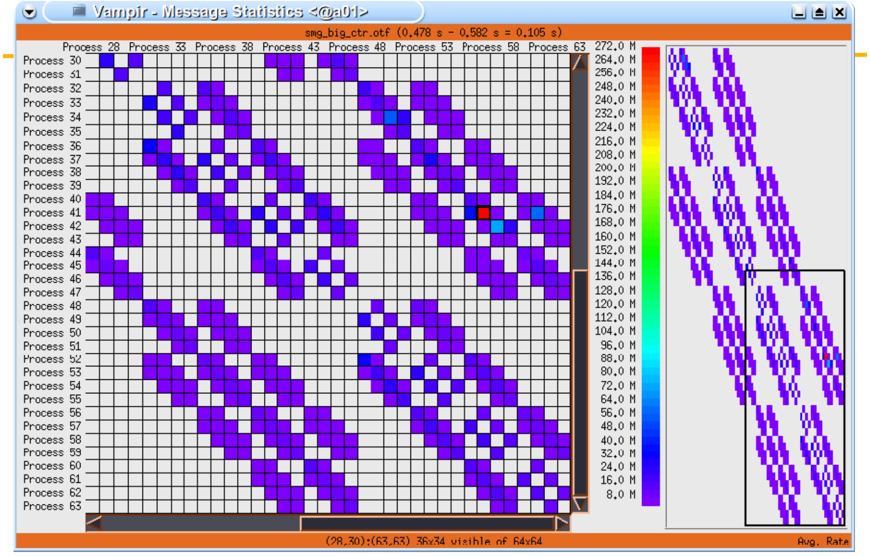


```
Name                            Token     Value
hypre_StructMatvecCompute       [ 299 ]   2:15.199
hypre_StructAxpy                [ 306 ]   2:10.744
hypre_StructInnerProd           [ 295 ]   1:14.087
MPI_Finalize                    [ 02 ]    1:04.170
hypre_StructCopy                [ 297 ]   51.516 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
```

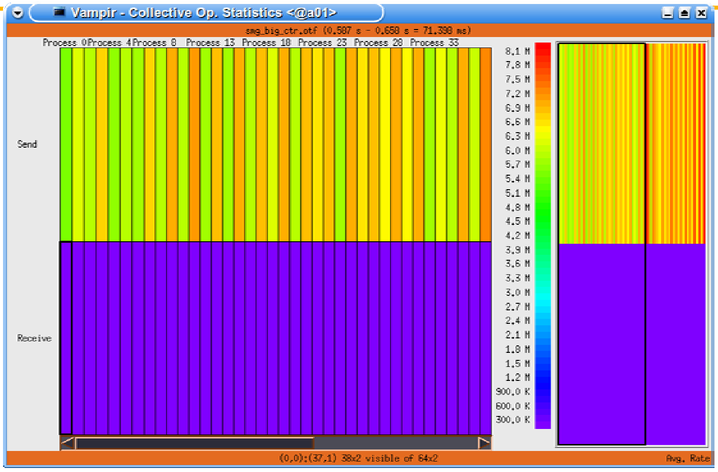
Summary Timeline Display



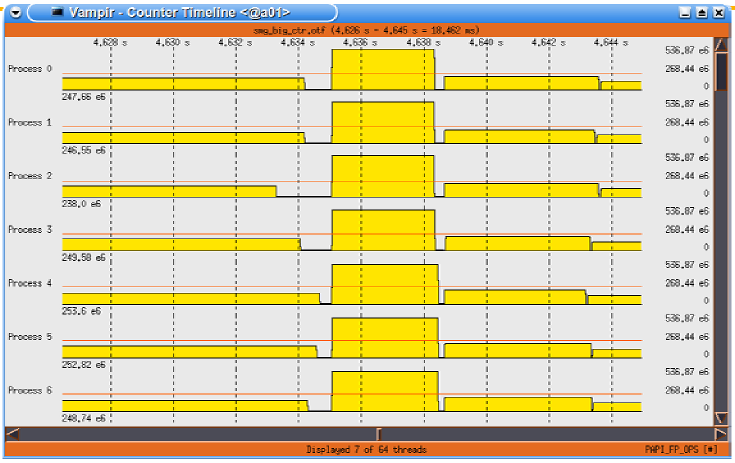
Message Statistics Display



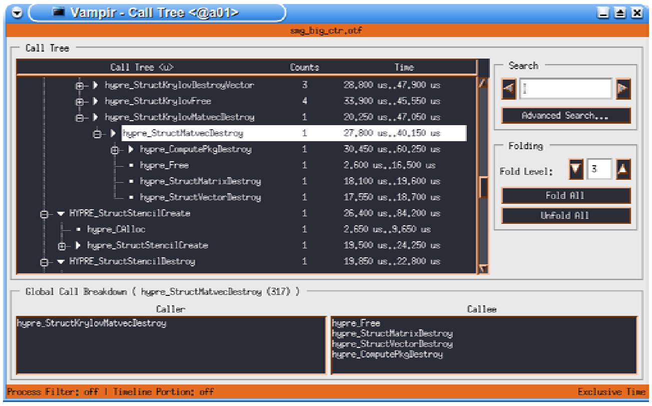
Collective Operation Statistics



Counter Timeline Display



Call Tree Display



Call Tree <u>	Counts	Time
hypre_StructKrylovDestroyVector	3	28,800 us..47,900 us
hypre_StructKrylovFree	4	33,900 us..45,550 us
hypre_StructKrylovMatvecDestroy	1	20,250 us..47,050 us
hypre_StructMatvecDestroy	1	27,800 us..40,150 us
hypre_ComputePkgDestroy	1	30,450 us..60,250 us
hypre_Free	1	2,600 us..16,500 us
hypre_StructMatrixDestroy	1	18,100 us..19,600 us
hypre_StructVectorDestroy	1	17,550 us..18,700 us
HYPRE_StructStencilCreate	1	26,400 us..84,200 us
hypre_CAlloc	1	2,650 us..9,650 us
hypre_StructStencilCreate	1	19,500 us..24,250 us
HYPRE_StructStencilDestroy	1	19,850 us..22,800 us

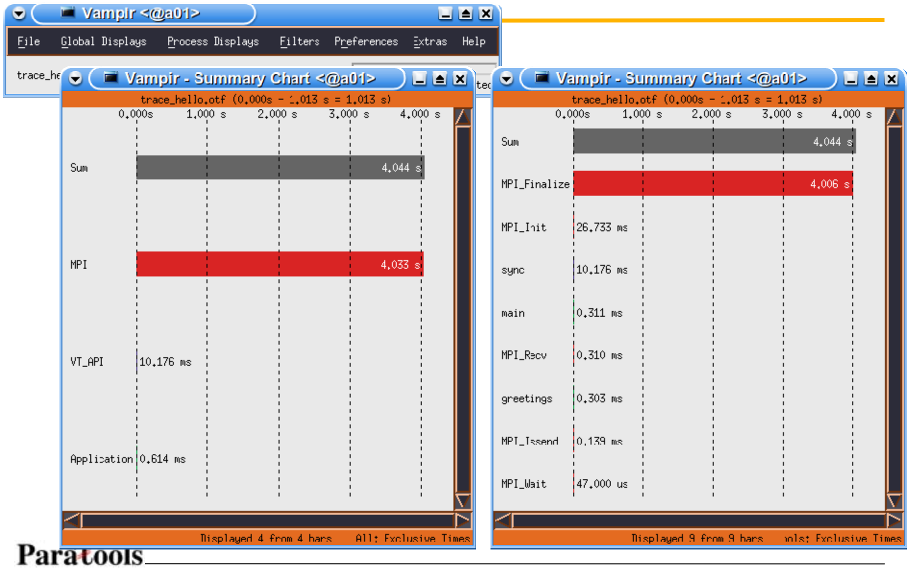
Global Call Breakdown (hypre_StructMatvecDestroy (317))

Caller	Callee
hypre_StructKrylovMatvecDestroy	hypre_Free hypre_StructMatrixDestroy hypre_StructVectorDestroy hypre_ComputePkgDestroy

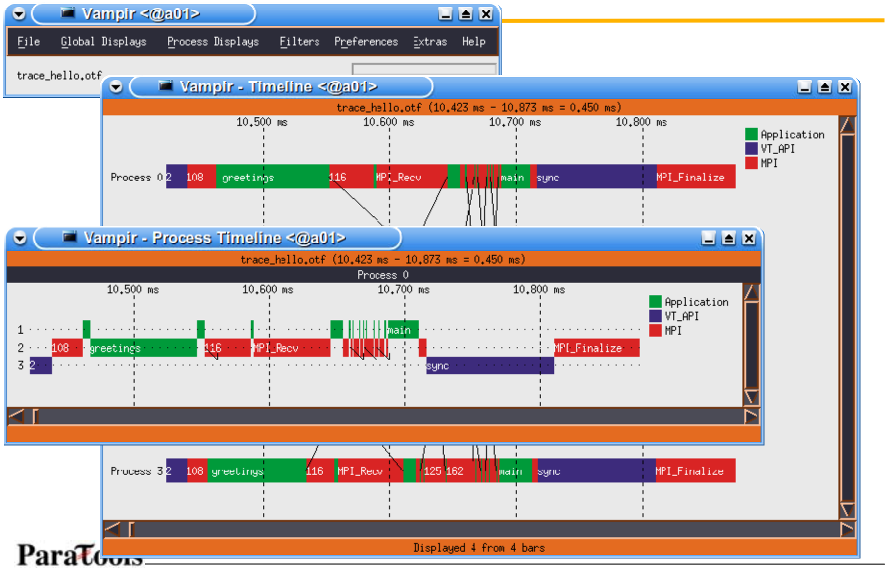
Open Trace Format (OTF)

- Open source trace file format
- Available at http://www.tu-dresden.de/zih/otf/
- Includes powerful libotf for reading/parsing/writing in custom applications
- multi-level API:
  - High level interface for analysis tools
  - Low level interface for trace libraries
- Actively developed in cooperation with the University of Oregon and the Lawrence Livermore National Laboratory

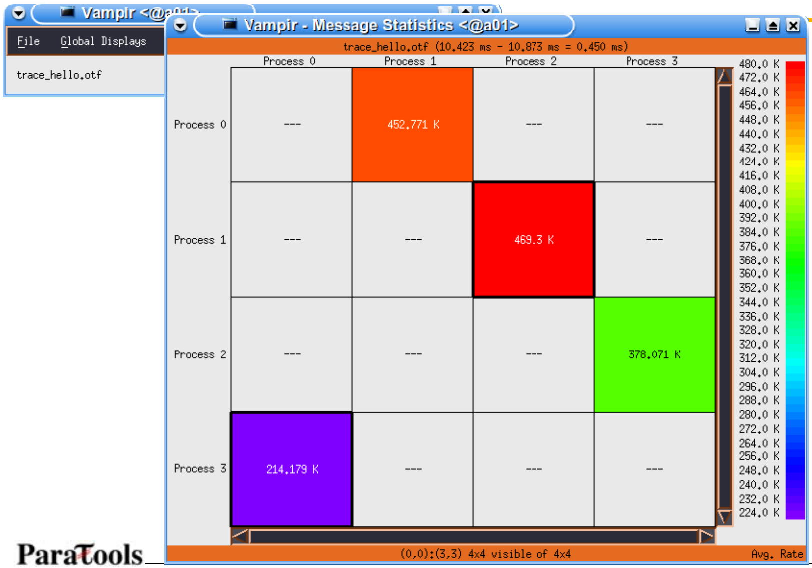
Hands-on: VampirServer



Hands-on: More Displays



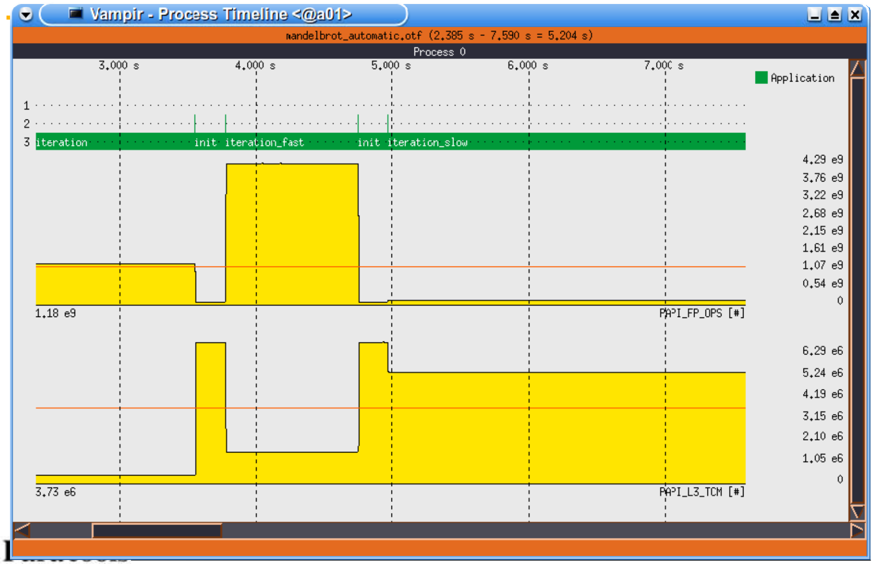
Hands-on: More Displays



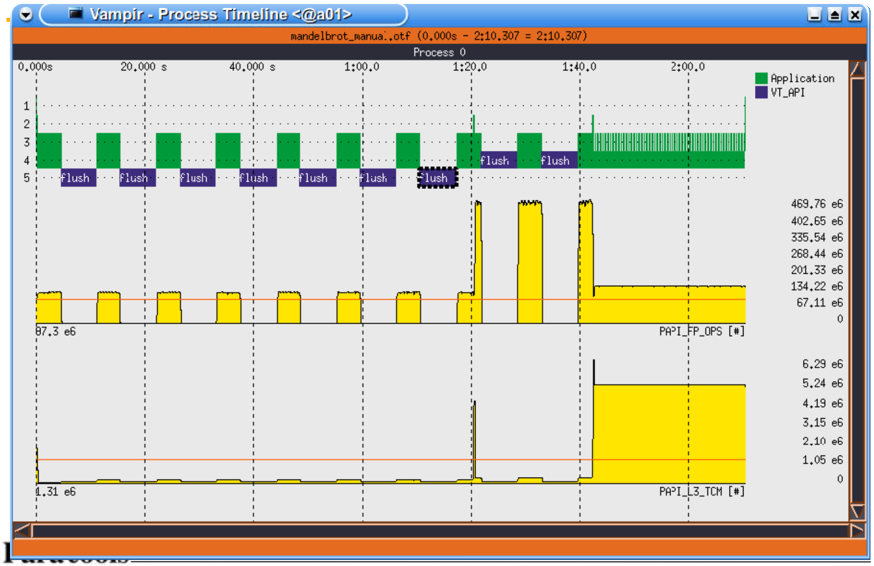
trace_hello.otf (10.423 ms - 10.873 ms = 0.450 ms)

	Process 0	Process 1	Process 2	Process 3
Process 0	---	452.771 K	---	---
Process 1	---	---	469.3 K	---
Process 2	---	---	---	378.071 K
Process 3	214.179 K	---	---	---

Hands-on: Performance Counters



Extra Manual Instrumentation



Finding Performance Bottlenecks

Finding Bottlenecks

- Trace Visualization
  - Vampir provides a number of display types
  - each allows many different options
- Advice
  - identify essential parts of an application (initialization, main iteration, I/O, finalization)
  - identify important components of the code (serial computation, MPI P2P, collective MPI, OpenMP)
  - make a hypothesis about performance problems
  - consider application's internal workings if known
  - select the appropriate displays
  - use statistic displays in conjunction with timelines

Finding Bottlenecks

- Communication
- Computation
- Memory, I/O, etc
- Tracing itself

Bottlenecks in Communication

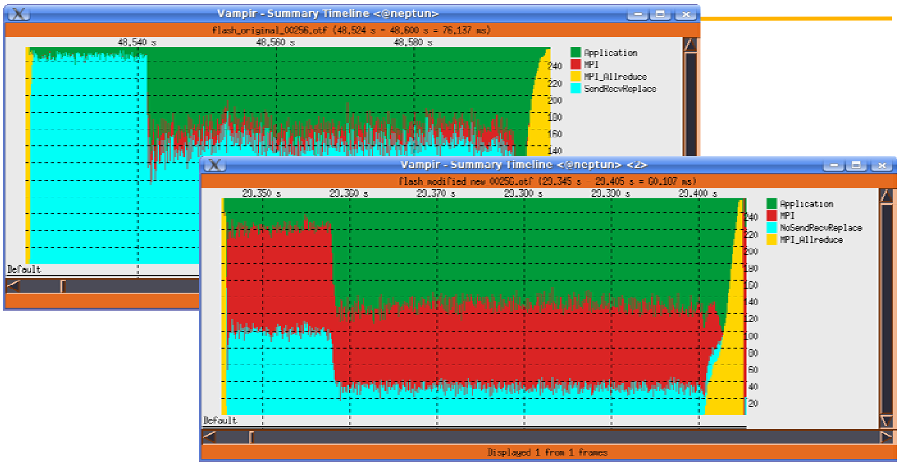
- communication as such
  (dominating over computation)
- late sender, late receiver
- point-to-point messages instead of collective communication
- unmatched messages
- overcharge of MPI's buffers
- bursts of large messages (bandwidth)
- frequent short messages (latency)
- unnecessary synchronization (barrier)

all of the above usually result in high MPI time share

Bottlenecks in Communication

Example: prevalent communication

Bottlenecks in Communication

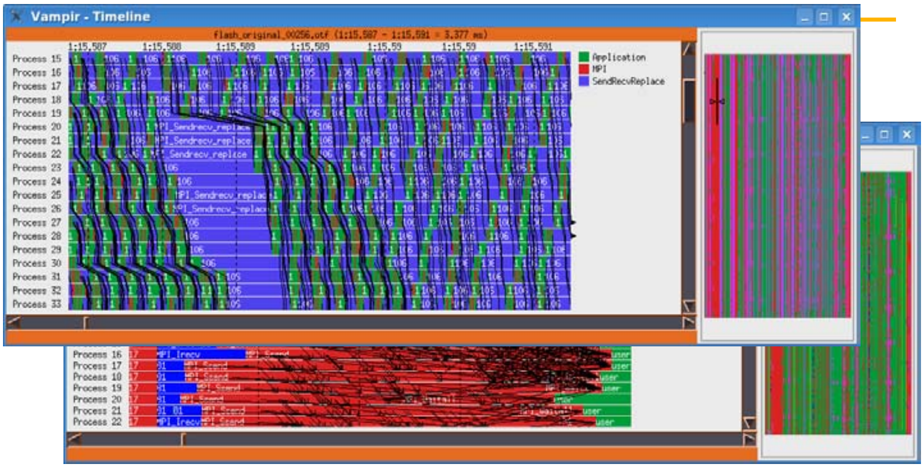


prevalent communication: MPI_Allreduce

Bottlenecks in Communication

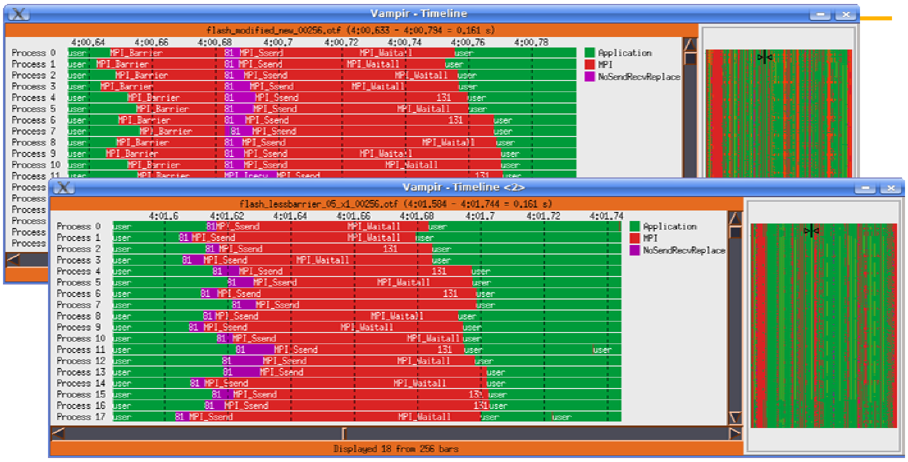
prevalent communication: timeline view

Bottlenecks in Communication



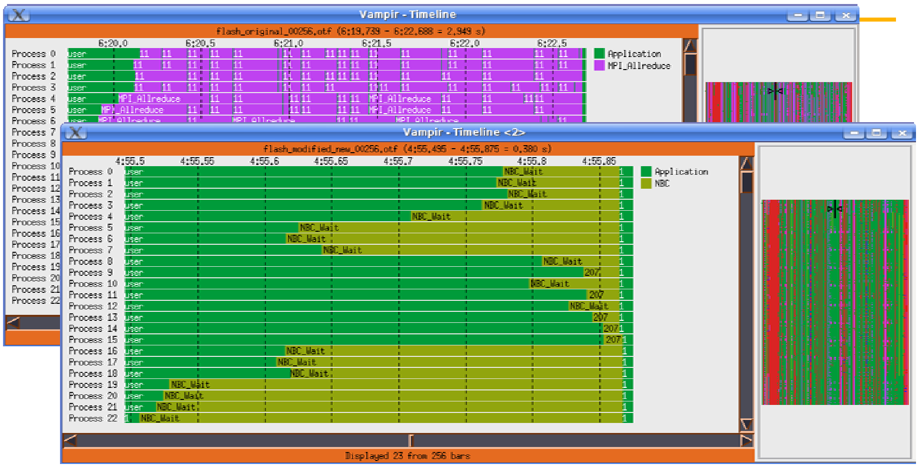
Propagated Delays in MPI_SendReceiveReplace

Bottlenecks in Communication



unnecessary MPI_Barriers

Bottlenecks in Communication



Patterns of Successive MPI_Allreduce Calls

Further Bottlenecks

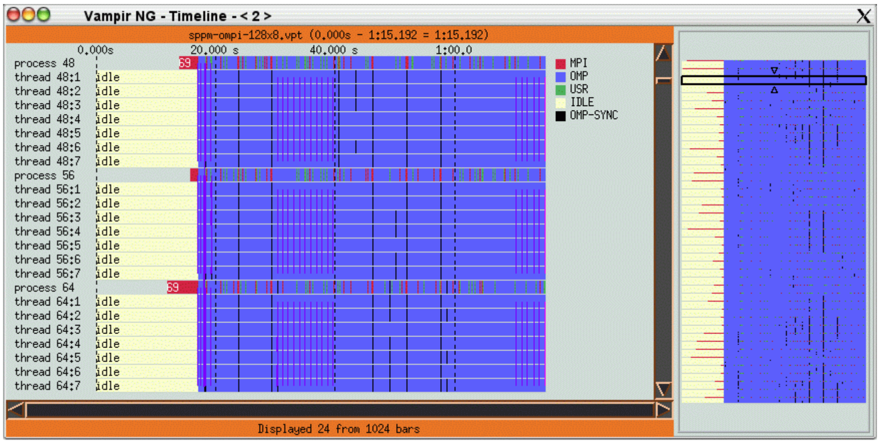
- unbalanced computation
  - single late comer

- strictly serial parts of program
  - idle processes/threads

- very frequent tiny function calls
- sparse loops

ParaTools

Further Bottlenecks



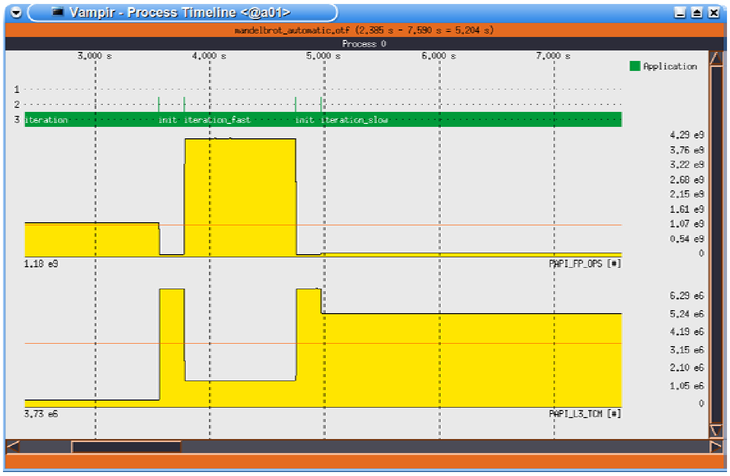
Example: Idle OpenMP threads

ParaTools

Bottlenecks in Computation

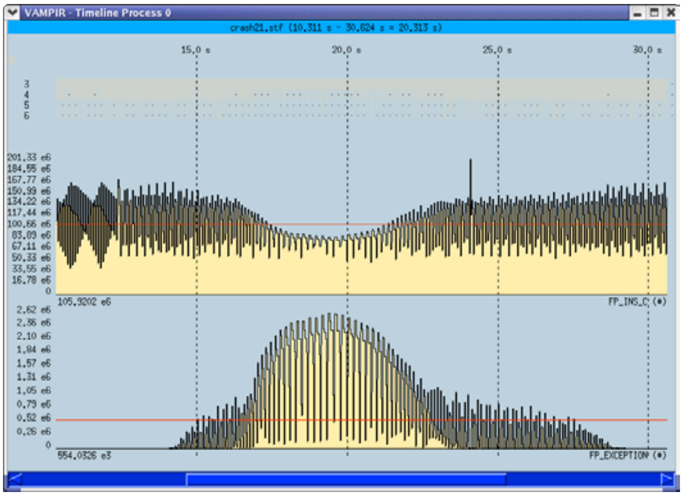
- memory bound computation
  - inefficient L1/L2/L3 cache usage
  - TLB misses
  - detectable via HW performance counters
- I/O bound computation
  - slow input/output
  - sequential I/O on single process
  - I/O load imbalance
- exception handling

Bottlenecks in Communication



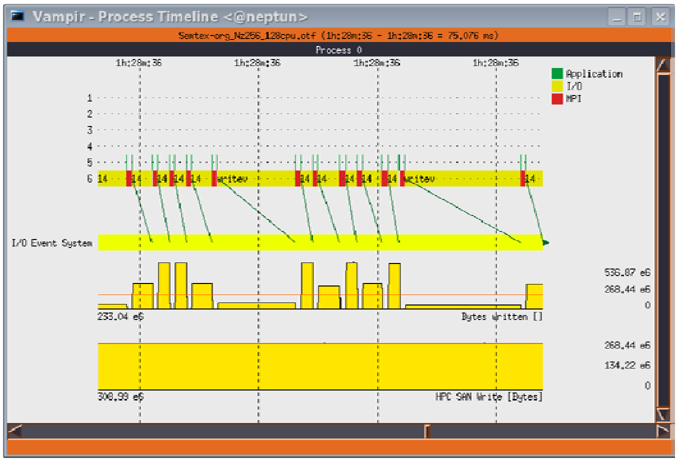
low FP rate due to heavy cache misses

Bottlenecks in Communication



low FP rate due to heavy FP exceptions

Bottlenecks in Communication



irregular slow I/O operations

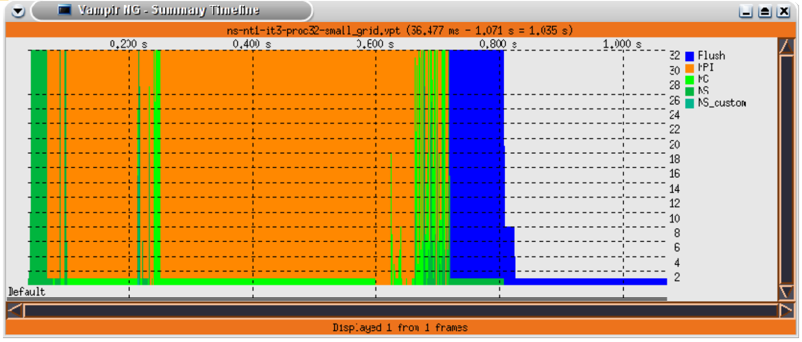
Effects due to Tracing Itself

- measurement overhead
  - esp. grave for tiny function calls
  - solve with selective instrumentation

- long/frequent/asynchronous trace buffer flushes
- too many concurrent counters

- heisenbugs

Effects due to Tracing Itself



Trace buffer flushes are explicitly marked in the trace.
It is rather harmless at the end of a trace as shown here.

Conclusion and Outlook

- performance analysis very important in HPC

- use performance analysis tools for profiling and tracing
- do not spend effort in DIY solutions, e.g. like printf-debugging

- use tracing tools with some precautions
    - overhead
    - data volume

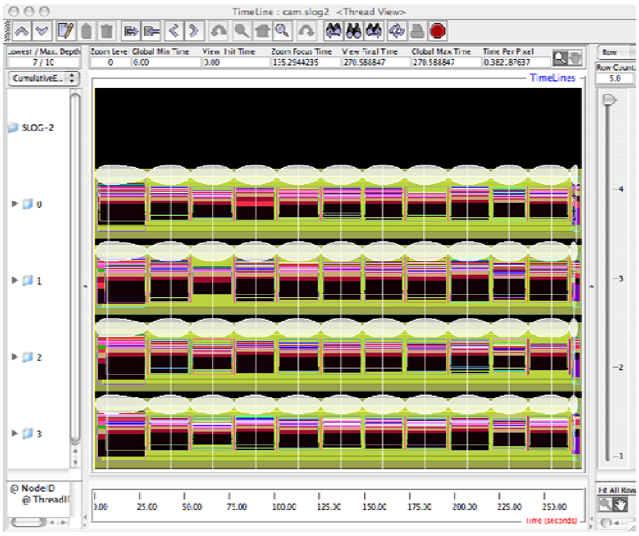
- let us know about problems and about feature wishes
- vampirsupport@zih.tu-dresden.de

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 NAVO IBM and ERDC XT3/4
  - 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



Part VI: KOJAK/Scalasca

scalasca

Overview

- Introduction
  - Motivation for automatic trace analysis
- Scalasca components and usage
  - instrumentation
  - measurement collection & automated analysis
  - analysis report exploration
- Demonstration
- Summary

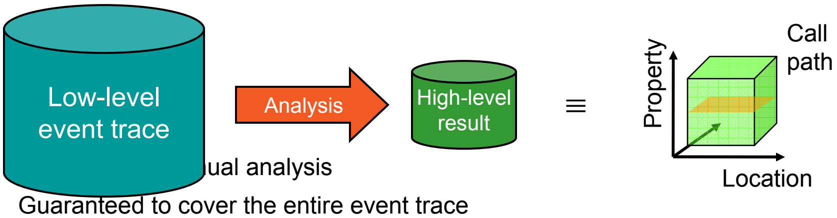
Motivation

- Tracing offers critical insight into temporal behaviour of parallel execution unavailable from summarization
  - Inefficiencies manifest as wait states and imbalance
- Trace sizes proportional to number of processes/threads
  - as well as length of measurement and depth of detail
- Large-scale parallel traces must be carefully managed
  - minimization/elimination of disruptive file I/O
  - efficient parallel analysis of traces
  - effective hierarchical/graphical analysis presentation
- Simplification and ease-of-use
  - Automation of search for and classification of event patterns
  - Integration with trace visualizers to examine key instances

---

Automatic Trace Analysis

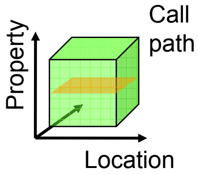
- Idea:
  - Automatic search for *patterns* of inefficient behaviour
  - Classification of behaviour
  - Quantification of significance



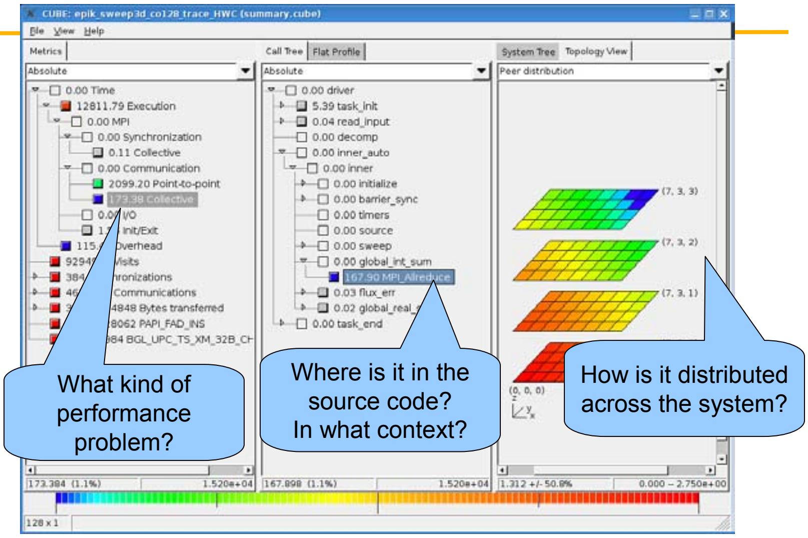
- ual analysis
- Guaranteed to cover the entire event trace

CUBE Result Browser

- Representation of results (severity matrix) along three hierarchical axes
  - Performance property
  - Call tree path
  - System location
- Three coupled tree browsers
- Each node displays severity
  - As colour: for easy identification of hotspots
  - As value: for precise comparison
  - Inclusive value when closed or exclusive when expanded
  - Customizable via display mode



Basic Analysis Presentation



```
0.00 Time
  12811.79 Execution
    0.00 MPI
      0.00 Synchronization
        0.11 Collective
      0.00 Communication
        2099.20 Point-to-point
        173.38 Collective
      0.0  I/O
      1.   Init/Exit
    115.  Overhead
  9294   Visits
  384    ronizations
  46     Communications
  3      4848 Bytes transferred
         8062 PAPI_FAD_INS
         984 BGL_UPC_TS_XM_32B_CH
173.384 (1.1%)          1.520e+04
```

```
0.00 driver
  5.39 task_init
  0.04 read_input
  0.00 decomp
  0.00 inner_auto
    0.00 inner
      0.00 initialize
      0.00 barrier_sync
      0.00 timers
      0.00 source
      0.00 sweep
      0.00 global_int_sum
        167.90 MPI_Allreduce
      0.03 flux_err
      0.02 global_real_
  0.00 task_end
167.898 (1.1%)          1.520e+04
```

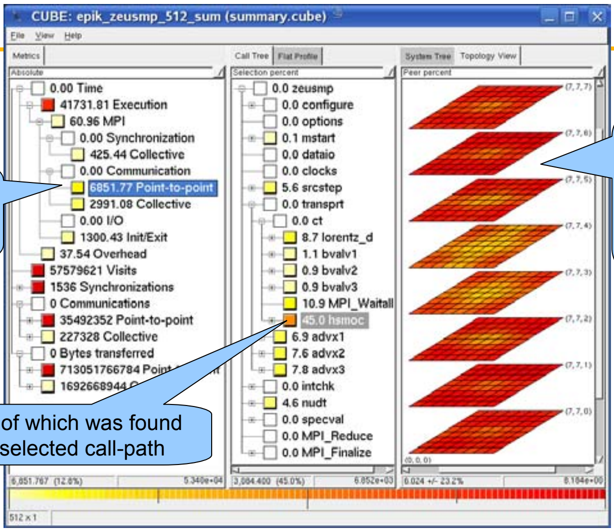
```
1.312 +/- 50.8%          0.000 – 2.750e+00
```

What kind of performance problem?

Where is it in the source code? In what context?

How is it distributed across the system?

Summary Profile Analysis

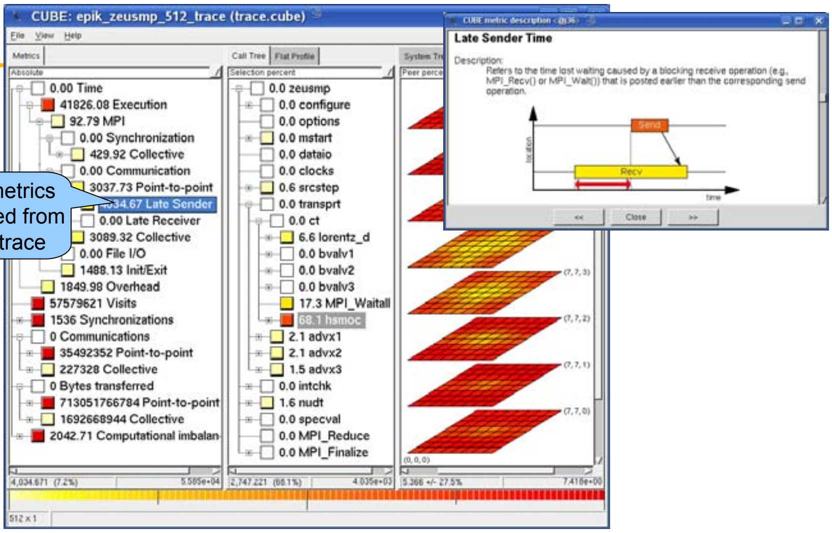


Selected metric has 13% of total CPU time

distributed on processes according to application's Cartesian topology

45% of which was found on selected call-path

Trace Pattern Analysis



Late Sender Time

Description:
Refers to the time lost waiting caused by a blocking receive operation (e.g. MPI_Recv() or MPI_Wait()) that is posted earlier than the corresponding send operation.

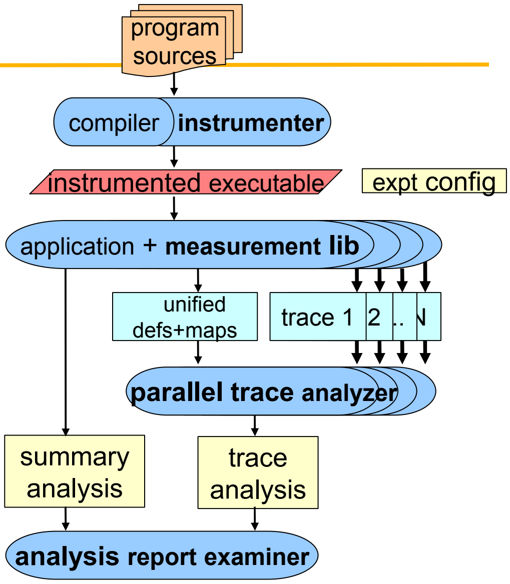
Extra metrics calculated from event trace

Analysis Methodology

- Instrumentation of application executable and libraries
  - automatic MPI, OpenMP and function instrumentation
  - complementary manual region and phase instrumentation
- Execution of instrumented executable under control of configurable measurement collection & analysis nexus
  - commence from scalable runtime summary
    - identify excess instrumentation and trace buffer requirements
  - target tracing where it is most productive (and practical)
    - analyze traces using same resources as measurement
- Interactive analysis report exploration and algebra
  - examine severities and their locations
  - combine, compare and process reports
- Refine and repeat as necessary

Scalasca Components

- Scalasca instrumenter
  = SKIN
- Scalasca measurement
  collector & analyzer
  = SCAN
- Scalasca analysis
  report examiner
  = SQUARE



program sources

compiler **instrumenter**

instrumented executable

expt config

application + **measurement lib**

unified defs+maps

trace 1 2 .. N

parallel trace analyzer

summary analysis

trace analysis

analysis report examiner

Scalasca unified command: scalasca

- Run without action argument for basic usage info

```
% scalasca
usage: scalasca [-v][-n] {action}
1. prepare application objects and executable for measurement:
   scalasca -instrument <compile-or-link-command>    # skin
2. run application under control of measurement system:
   scalasca -analyze <application-launch-command>   # scan
3. interactively explore measurement analysis report:
   scalasca -examine <experiment-archive|report>       # square
```

- Simply a convenience wrapper for action commands

Scalasca instrumenter: skin

- Usage: scalasca -instrument [opts] $CC ...
  - **scalasca -instrument -user** mpicc -fast -c bar.c
  - ***skin*** mpif90 -Openmp -o foobar -fast foo.c bar.o -lm
- Processes source modules during compile & augments link with measurement library
  - Configures automatic function instrumentation capability of native compiler (if available)
    - All functions in source module(s) are instrumented
  - **[–pomp]** option enables processing of POMP directives
    - Optional manual source annotation of functions & regions
    - Replaces automatic function instrumentation
  - [**–**user] activates EPIK user-annotation API

Scalasca collector & analyzer: scan

- Usage: scalasca -analyze [opts] <launch command>
  - ***scan*** *[opts]* [launcher [args]] [target [target-args]]
- Prepares & runs measurement collection,
  with follow-on trace analysis (if appropriate)
  - [-n] preview without executing launches
  - [-s] enables runtime summarization [default]
  - [-t] enables trace collection & automatic pattern analysis
  - determines NP and/or NT (number of processes & threads) and MODE=vn|co|dual|smp (where appropriate)
  - names default measurement experiment archive `epik_$(TARGET)_$(MODE)$(NP)x$(NT)_[sum|trace]`
  - [-f filter] specifies file listing functions not to be measured
  - [-m metric1:metric2:...] includes hardware counter metrics

Scalasca analysis report explorer: square

- Usage: scalasca -examine <epik_archive | cubefile >
  - **scalasca -examine** epik_sweep3d_co32_trace
  - ***square*** epik_sweep3d_co32_trace/summary.cube
- Prepares & presents final analysis report
  - Checks EPIK archive directory for cubefiles
  - Remaps primitive initial analysis report(s) into refined formal report(s) with enriched metrics & metric hierarchies
    - epitome.cube -> summary.cube
    - scout.cube -> trace.cube
  - Presents refined report in CUBE3 browser
    - Trace analysis shown in preference to summary analysis
    - Additional reports can be loaded via File/Open menu

EPIK experiment archive

- Directory created by measurement library
  - Measurement aborts if archive already exists!
- Contains all files related to measurement
  - Measurement & analysis logs (epik.log, scout.log, etc.)
  - Primitive analysis reports (epitome.cube, scout.cube)
  - Refined analysis reports (summary.cube, trace.cube)
  - Process trace datafiles (ELG/*)
  - Unified definitions & map data (epik.esd, epik.map)
  - Miscellaneous (epik.conf, epik.filt, epik.path)

EPIK measurement configuration

- ***epik_conf*** reports current configuration
  - logged in measurement archive as epik.conf
- Read from EPIK.CONF file(s)
  - System default: $SCALASCA_DIR/doc
  - Directory specified with EPIK_CONF environment variable [defaults to “.”]
- Over-ridden by environment variables
  - with same names as configuration file variables
- Over-ridden by ***scan*** command-line settings

Trace collection & analysis issues

- Process rank trace too large for trace collection buffers
  - Results in intermediate trace buffer flushes
    (with remainder flushed at measurement finalization)
  - Serious measurement perturbation!
- Irrelevant functions encumber analysis
  - Undesirable complexity and processing slowdown
  - Parallel trace analyzer requires memory more than twice largest rank
    (uncompressed) trace size to load entire trace
- Options
  - enlarge trace buffer size: ELG_BUFFER_SIZE
    - ***cube3_score*** utility provides estimate from summary
  - remove selected function instrumentation
  - specific function measurement filter (if supported!)

Selective instrumentation/measurement

- Unimportant functions can be determined from summary analysis report
  - form leaves of callpath-tree (w/o MPI)
  - negligible proportion of (exclusive) execution time
  - high proportion of (exclusive) visit count
  - ***cube3_score -r*** provides region breakdown & classification
    - MPI, USR (no MPI), COM (combined/intermediate)
- Eliminating pure user (USR) regions reduces overheads
  - runtime processing, storage & analysis
- Makes them “invisible” in the analysis
  - logically become part of their calling functions
    (as if they were in-lined by an optimizing compiler!)

Scalasca runtime summarization

- Event measurements accumulated and summarized for each call-path during runtime execution
- Summary report produced at finalization
- Provides overview of measured execution
  - contains call-path Visit frequency, Time, and MPI message statistics
  - *plus* optional hardware counter metrics
  - size independent of length of execution
- Scales to long execution measurements

Scalasca trace analysis

- Trace analysis based on parallel replay
  - enables scalability to thousands of processes
  - however, only suited to relatively brief measurements!
- Extends summary metric analysis
  - Summary can help configure selective tracing
- Allows execution performance properties to be more accurately determined and refined
- Can be combined with complementary runtime summary analysis
  - avoiding storage/processing overhead of hardware counter metrics in traces via direct summarization

Measurement support

- OpenMP compilers
  - GCC
  - IBM XL
  - Intel
  - Pathscale
  - PGI
  - Sun Studio
  - ...
- Supported functionality varies by language, version & system

- MPI libraries
  - MPICH 1 & 2
  - OpenMPI
  - Intel-MPI
  - IBM POE & BlueGene
  - Cray XT
  - Sun HPC ClusterTools
  - SGI MPToolkit
  - SiCortex MPI
  - Scali-MPI
  - HP-MPI
  - LAM

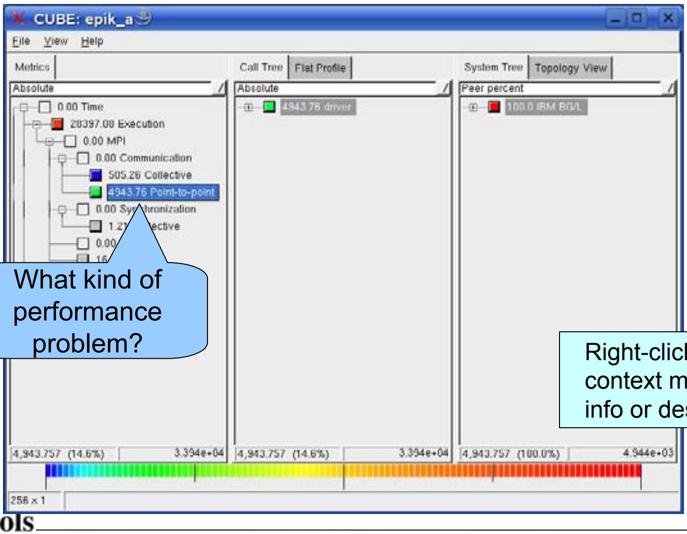
Basic use of Scalasca

- Automatic function instrumentation
  - Supported by most but not all compilers!

- Summary measurement experiment
- Summary analysis report exploration

- Trace collection & analysis experiment
- Trace pattern analysis report exploration

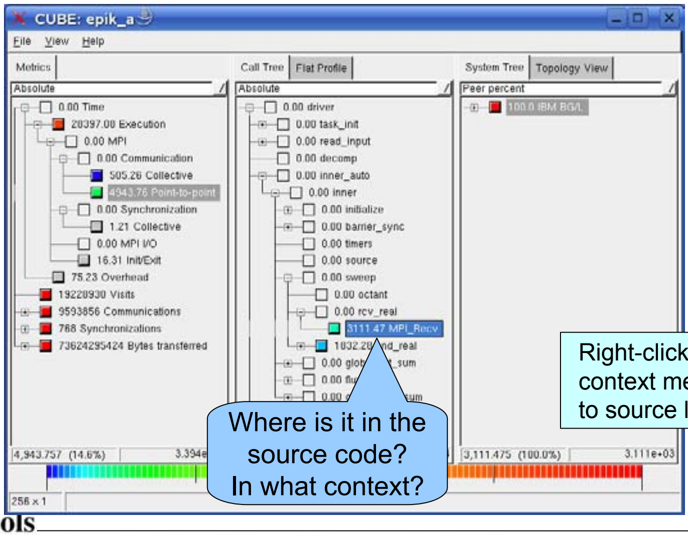
CUBE metrics dimension



What kind of performance problem?

Right-click metric context menu for info or description

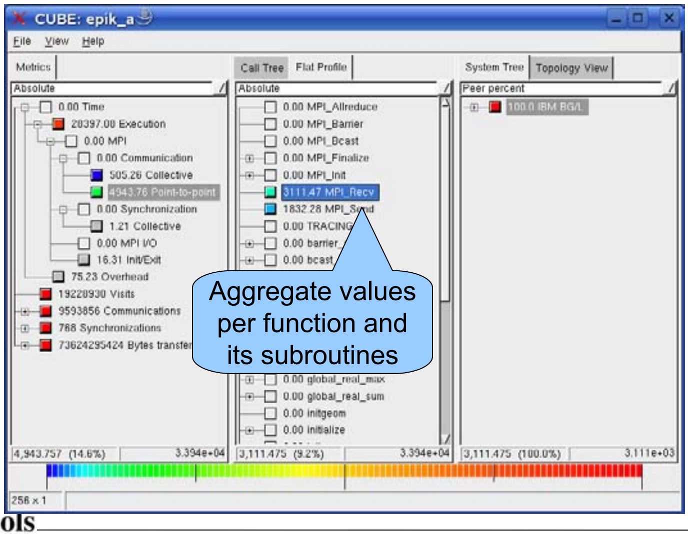
CUBE call tree dimension



Where is it in the source code?
In what context?

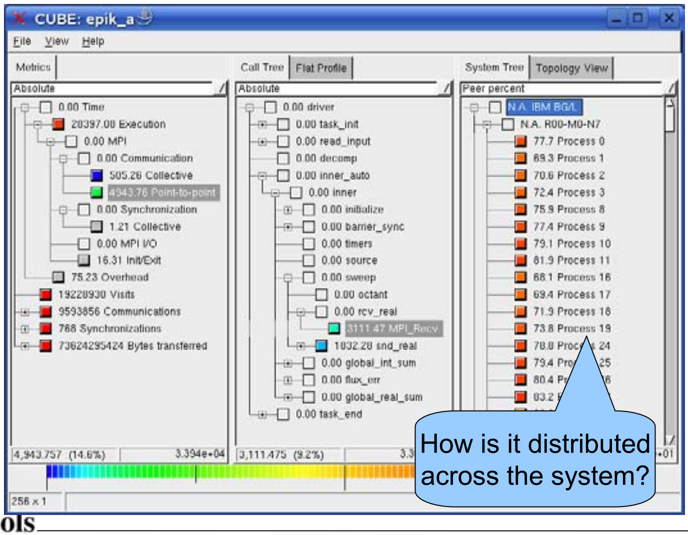
Right-click function context menu to go to source location

Alternative: Flat profile



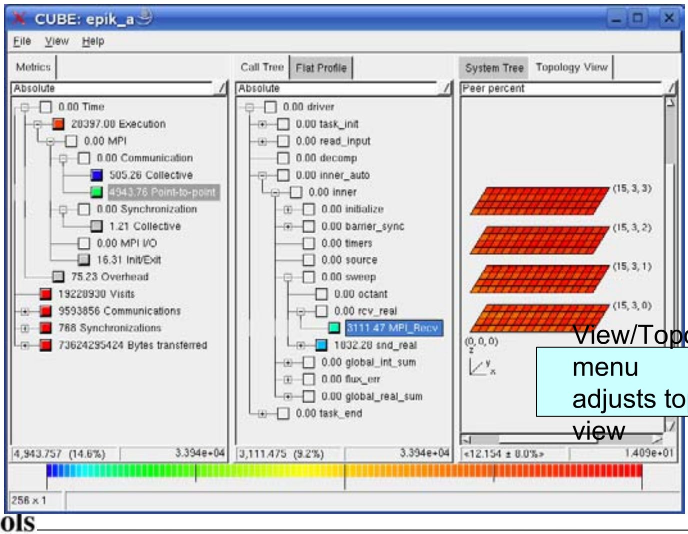
Aggregate values per function and its subroutines

System tree dimension



How is it distributed across the system?

Alternative: Topology display



View/Topology menu adjusts topology view

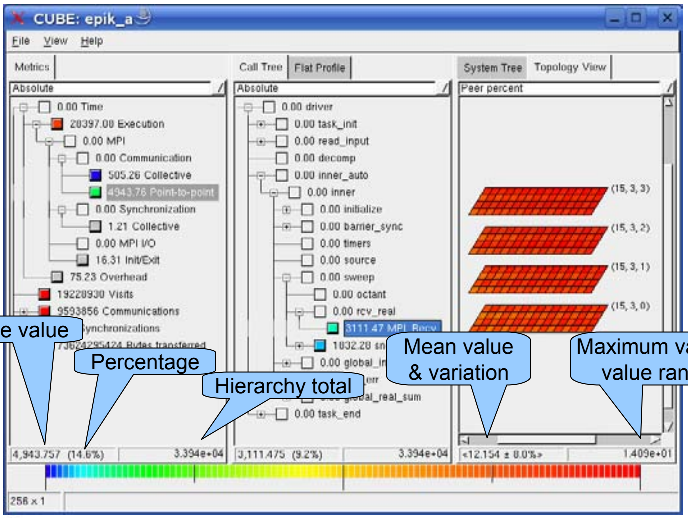
Topology display

- Topology information is recorded for
  - the hardware (supported on some systems)
  - MPI topologies (e.g., `MPI_Cart_create`())
  - user-defined virtual topologies (under construction)

- Advantage
  - Better scalability than text-based system tree

- Restriction
  - Currently supports only 1D, 2D and 3D Cartesian topologies

Status fields



Absolute value

Percentage

Hierarchy total

Mean value & variation

Maximum value / value range

Display modes

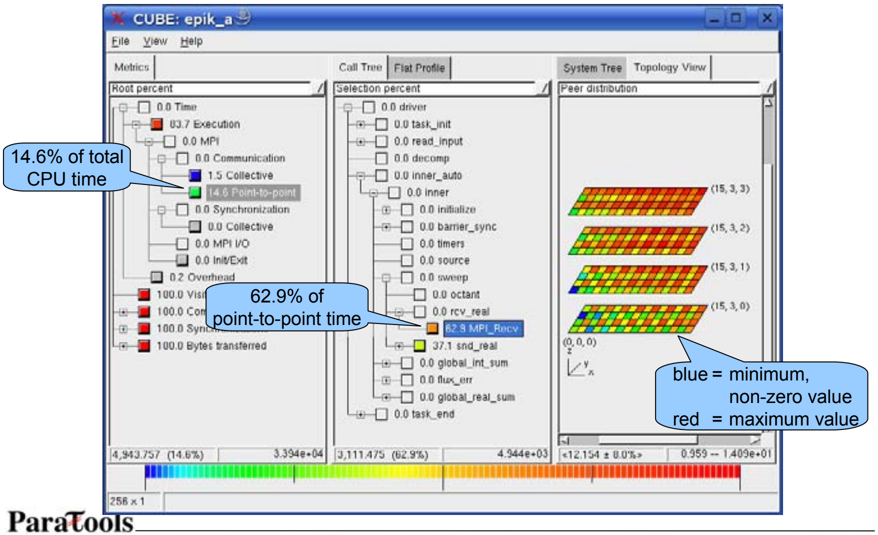
- Absolute
  - Absolute values in seconds/number of occurrences
- Root percent
  - Percentage relative to the root node of the hierarchy
- External percent
  - Similar to “Root percent”, but relative to another data set
- Selection percent
  - Percentage relative to the node selected in the neighbouring column on the left

Display modes (system tree/topology only)

- Peer percent
  - Percentage relative to maximum of peer values (all values of the current leaf level)
- Peer distribution
  - Percentage relative to maximum and non-zero minimum of peer values

---

Display mode example

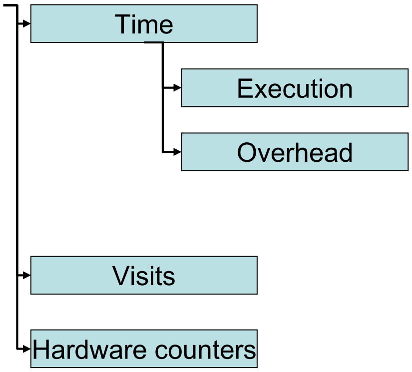


14.6% of total CPU time

62.9% of point-to-point time

blue = minimum, non-zero value
red = maximum value

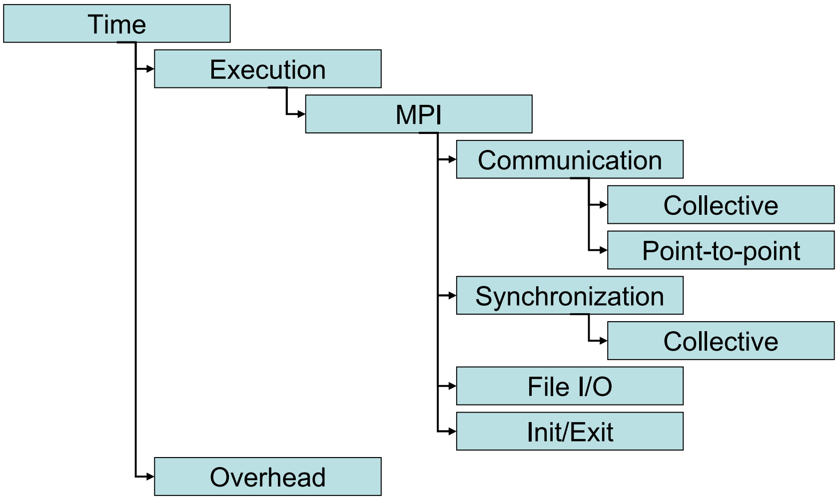
Generic metrics



Time	Total CPU allocation time
Execution	Execution time without overhead
Overhead	Time spent in activities related to measurement (not including dilation per instrumented routine/region!)
Visits	Number of times a routine/region was executed
Hardware counters	Aggregated counter values for each routine/region

---

MPI Time hierarchy

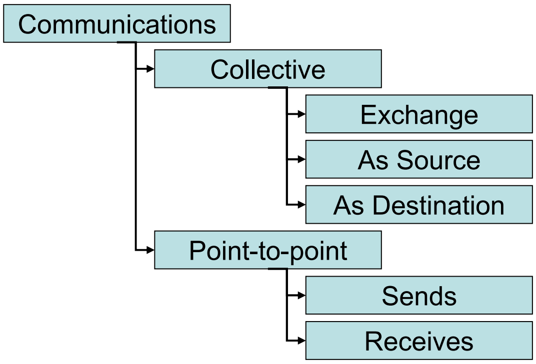


- Time
  - Execution
    - MPI
      - Communication
        - Collective
        - Point-to-point
      - Synchronization
        - Collective
      - File I/O
      - Init/Exit
  - Overhead

MPI Time hierarchy (cont.)

Time	Total CPU allocation time
Execution	Execution time without overhead
Overhead	Time spent in tasks related to measurement (not including dilation from instrumentation!)
MPI	Time spent in pre-instrumented MPI functions
Communication	Time spent in MPI communication calls, subdivided into collective and point-to-point
Synchronization	Time spent in MPI synchronization calls
File I/O	Time spent in MPI file I/O functions
Init/Exit	Time spent in `MPI_Init()` and `MPI_Finalize()`

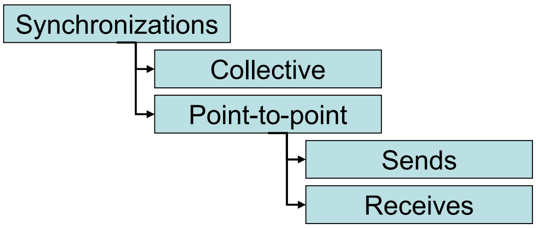
MPI Communications hierarchy



- Communications
  - Collective
    - Exchange
    - As Source
    - As Destination
  - Point-to-point
    - Sends
    - Receives

- Provides the number of calls to an MPI communication function of the corresponding class
- Zero-sized message transfers are considered *synchronization*!

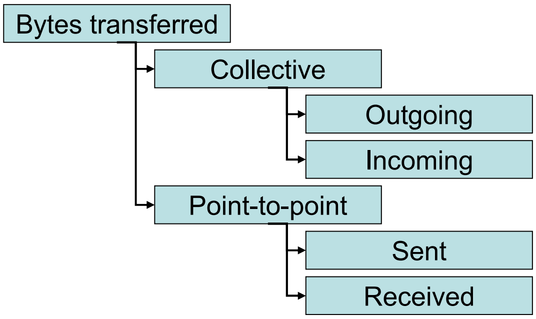
MPI Synchronizations hierarchy



- Synchronizations
  - Collective
  - Point-to-point
    - Sends
    - Receives

- Provides the number of calls to an MPI synchronization function of the corresponding class
- MPI synchronizations include zero-sized message transfers!

MPI Bytes transferred hierarchy



- Bytes transferred
  - Collective
    - Outgoing
    - Incoming
  - Point-to-point
    - Sent
    - Received

- Provides the number of bytes transferred by an MPI communication function of the corresponding class

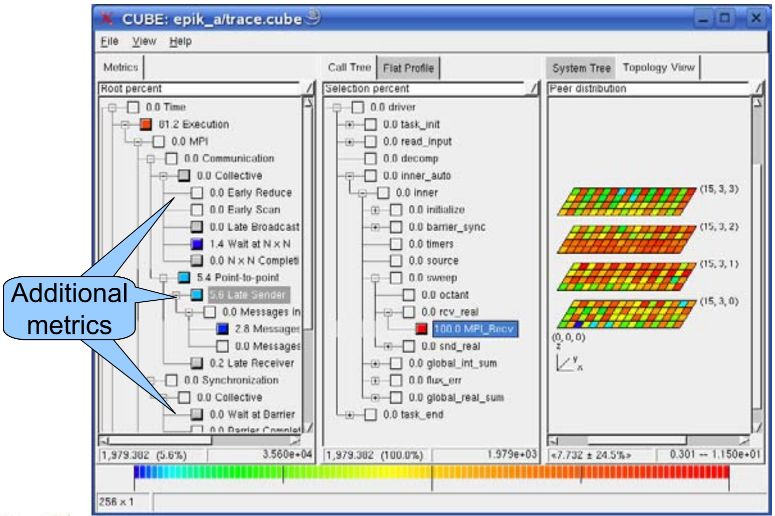
Combined trace collection & analysis

- Modify jobscript
  - Use “scan **-t**” (or set EPK_TRACE=1)
  - Trace experiment EPK_TITLE set to `$(TARGET)_$(MODE)$(NP)_trace`
  - Creates new experiment archive directory `./epik_$(EPK_TITLE)`
  - Trace unified & buffers flushed at measurement finalization
  - Automatic trace pattern analysis immediately follows
- Explore trace pattern analysis report using CUBE

Trace analysis output example

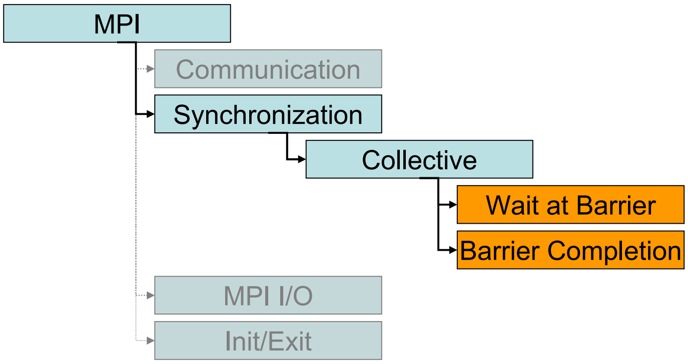
```
SCOUT

Analyzing experiment archive ./epik_sweep3d_co32_trace

Reading definition files  ... done
Reading event trace files ... done
Preprocessing             ... done
Analyzing event traces    ... done
Writing report            ... done

Total processing time: 4.083s
Total number of events: 5206596
Max. memory usage: 15.453 MB
```

Trace analysis result



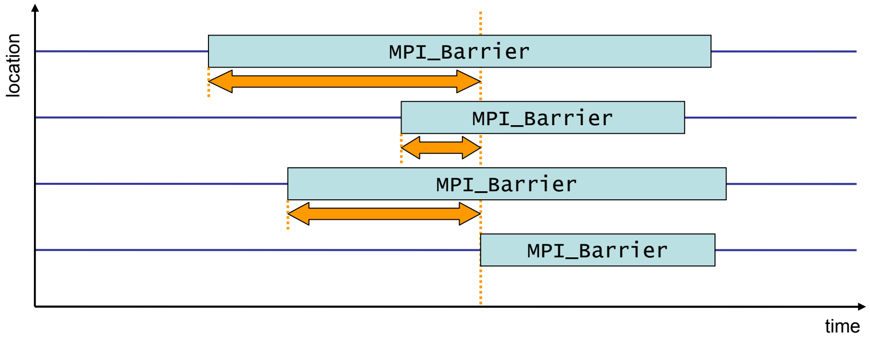
Additional metrics

MPI collective synchronization time



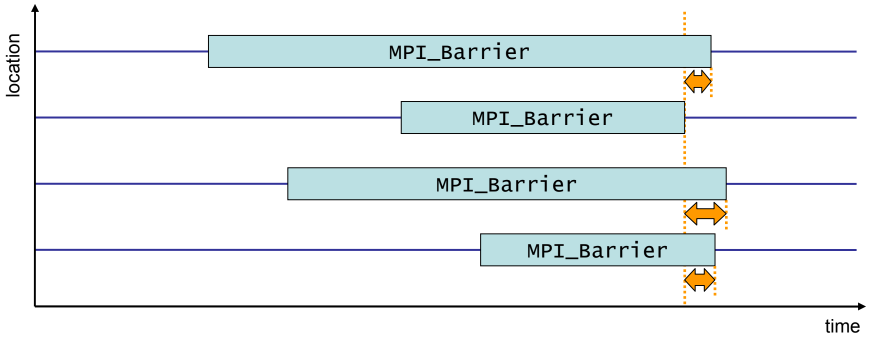
- MPI
  - Communication
  - Synchronization
    - Collective
      - Wait at Barrier
      - Barrier Completion
  - MPI I/O
  - Init/Exit

Wait at Barrier = Early Barrier



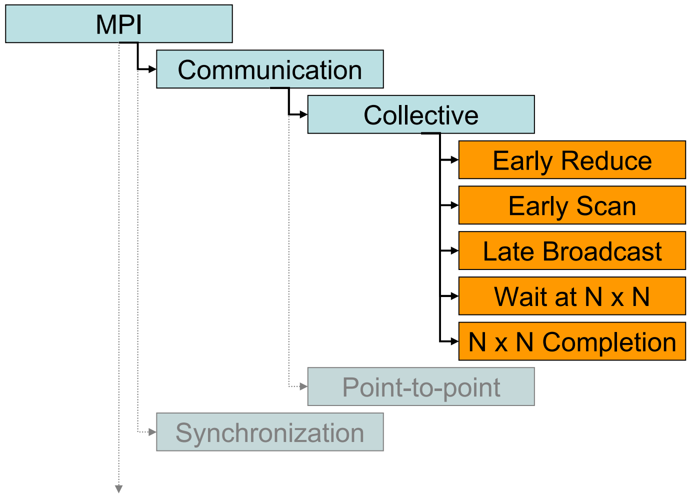
- Time spent waiting in front of a barrier call until the last process reaches the barrier operation
- Applies to: `MPI_Barrier`()

Barrier Completion



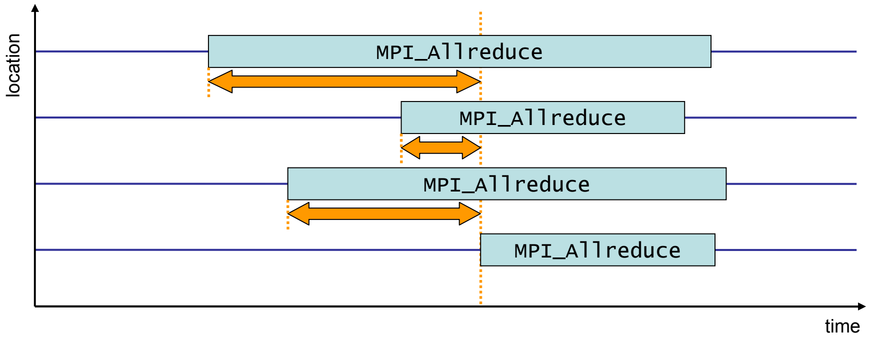
- Time spent in barrier after the first process has left the operation
- Applies to: `MPI_Barrier`()

MPI collective communication time



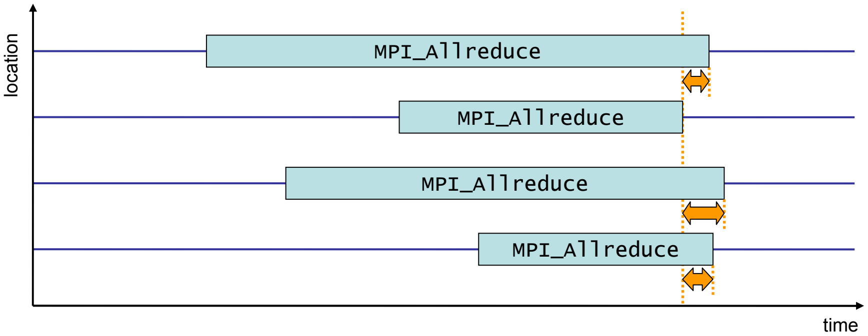
- MPI
  - Communication
    - Collective
      - Early Reduce
      - Early Scan
      - Late Broadcast
      - Wait at N x N
      - N x N Completion
    - Point-to-point
  - Synchronization

Wait at N x N = Early N x N



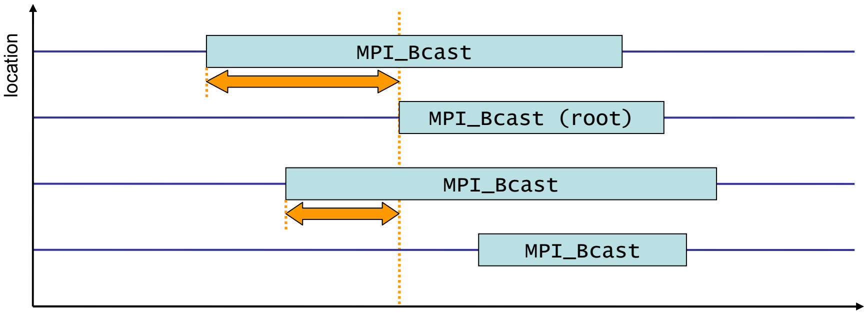
- Time spent waiting in front of a synchronizing collective operation call until the last process reaches the operation
- Applies to: `MPI_Allreduce`(), `MPI_Alltoall`(), `MPI_Alltoallv`(), `MPI_Allgather`(), `MPI_Allgatherv`(), `MPI_Reduce_scatter`()

N x N Completion



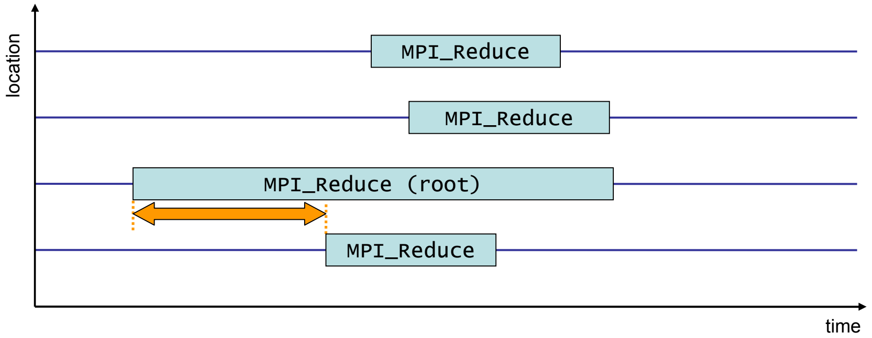
- Time spent in synchronizing collective operations after the first process has left the operation
- Applies to: `MPI_Allreduce`(), `MPI_Alltoall`(), `MPI_Alltoallv`(), `MPI_Allgather`(), `MPI_Allgatherv`(), `MPI_Reduce_scatter`()

Late Broadcast = Early Broadcast



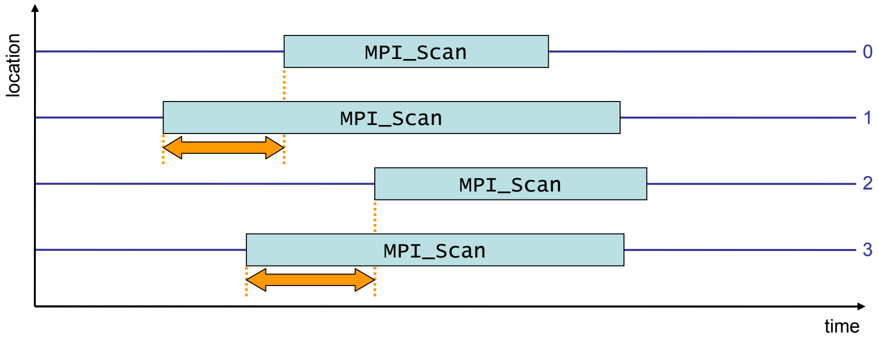
- Waiting times of the destination processes of a collective 1-to-N communication operation which enter the operation earlier than the source process (root)
  - Late Broadcast by source = Early Broadcast by destinations
- Applies to: `MPI_Bcast`(), `MPI_Scatter`(), `MPI_Scatterv`()

Early Reduce



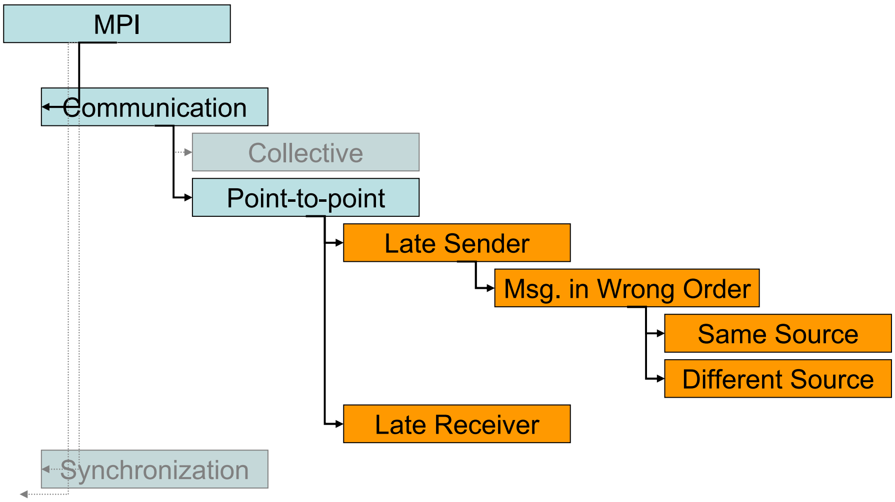
- Waiting time if the destination process (root) of a collective N-to-1 communication operation enters the operation earlier than its sending counterparts
- Applies to: `MPI_Reduce`(), `MPI_Gather`(), `MPI_Gatherv`()

Early Scan



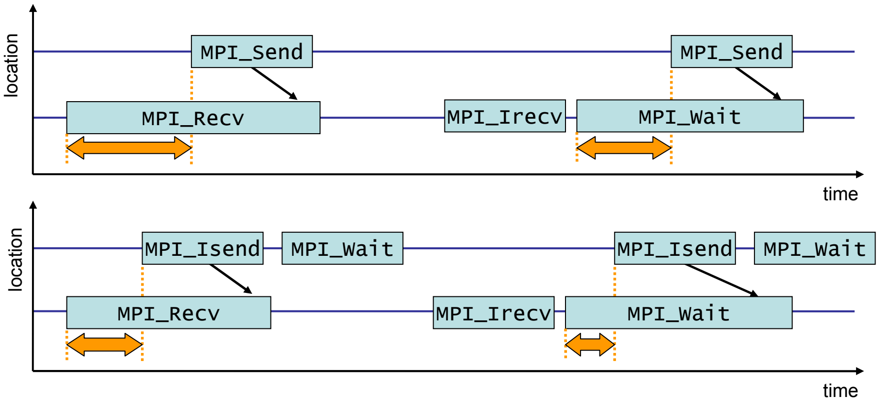
- Waiting time if process *n* enters a prefix reduction operation earlier than its sending counterparts (i.e., ranks 0..*n*-1)
- Applies to: `MPI_Scan`()

MPI point-to-point communication time



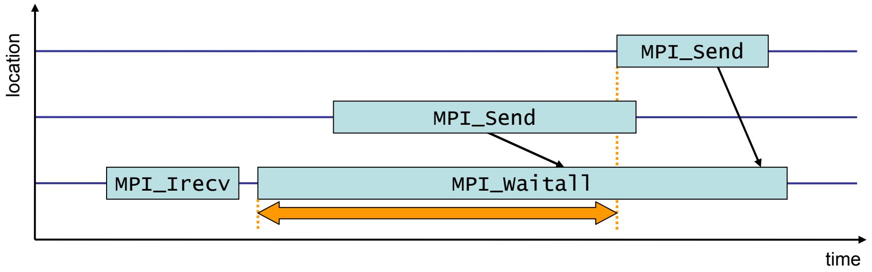
- MPI
  - Communication
    - Collective
    - Point-to-point
      - Late Sender
        - Msg. in Wrong Order
          - Same Source
          - Different Source
      - Late Receiver
  - Synchronization

Late Sender = Early Receive



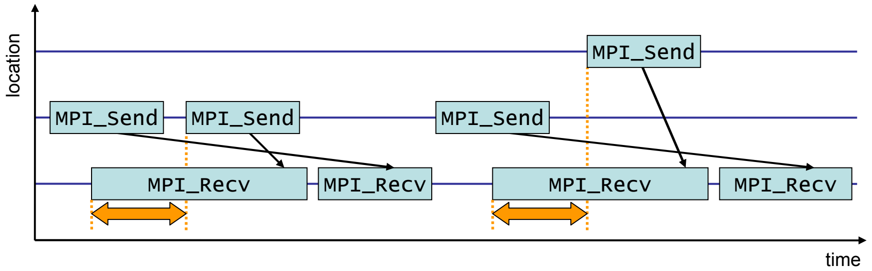
- Waiting time caused by a blocking receive operation posted earlier than the corresponding send operation
- Applies to blocking as well as non-blocking communication

Late Sender = Early Receive (cont.)



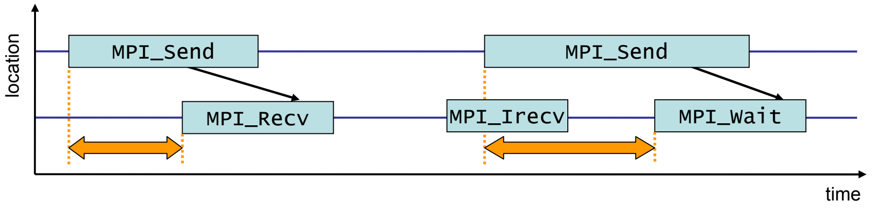
- While waiting for several messages, the maximum waiting time is accounted
- Applies to: `MPI_Waitall`(), `MPI_Waitsome`()

Late Sender, Messages in Wrong Order



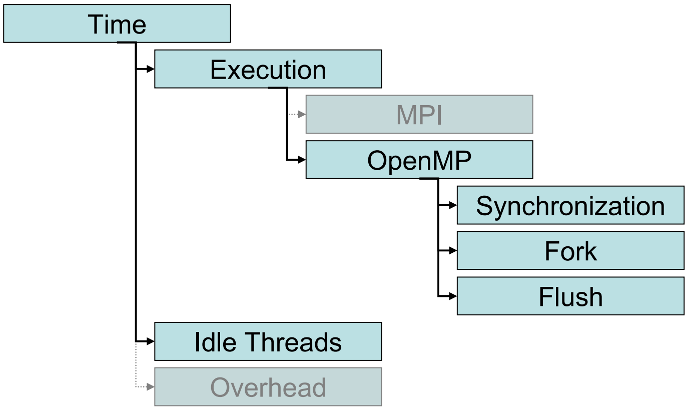
- Refers to Late Sender situations which are caused by messages received in wrong order
  - Early receive of message out of order
- Comes in two flavours:
  - Messages sent from same source location
  - Messages sent from different source locations

Late Receiver = Early Send



- Waiting time caused by a blocking send operation posted earlier than the corresponding receive operation
- Does not apply to non-blocking sends

OpenMP Time hierarchy



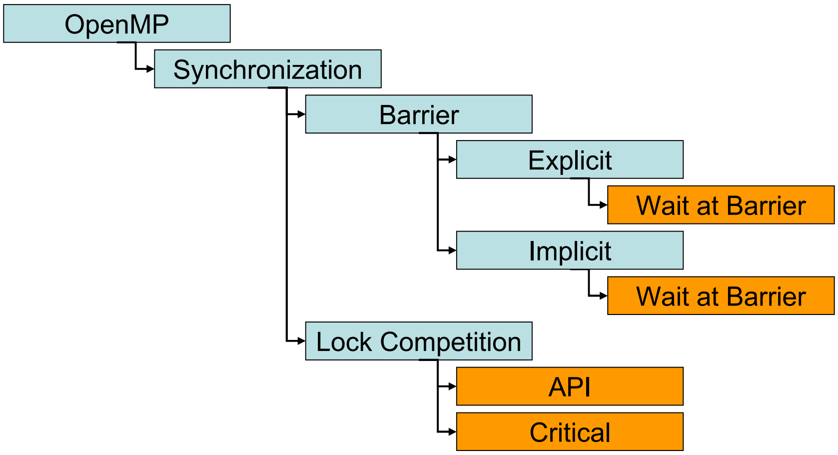
- Time
  - Execution
    - MPI
    - OpenMP
      - Synchronization
      - Fork
      - Flush
  - Idle Threads
  - Overhead

OpenMP Time hierarchy details

OpenMP	Time spent for all OpenMP-related tasks
Synchronization	Time spent synchronizing OpenMP threads
Fork	Time spent by master thread to create thread teams
Flush	Time spent in OpenMP flush directives
Idle Threads	Time spent idle on CPUs reserved for slave threads

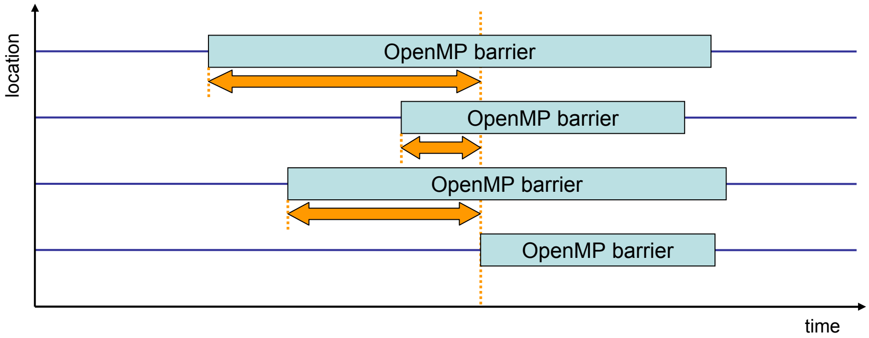
---

OpenMP synchronization time



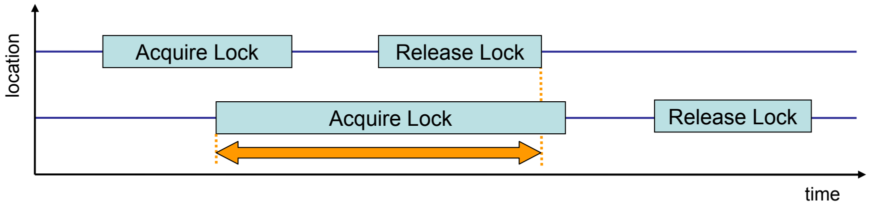
- OpenMP
  - Synchronization
    - Barrier
      - Explicit
        - Wait at Barrier
      - Implicit
        - Wait at Barrier
    - Lock Competition
      - API
      - Critical

Wait at Barrier = Early Barrier



- Time threads spend waiting in front of a barrier call until the last thread reaches the barrier operation
- Applies to: Implicit/explicit barriers

Lock competition



- Time a thread spends waiting for a lock that is held by other threads until it is released and can be acquired by this thread
- Applies to: critical sections, OpenMP lock API

Other metrics

- LateReceivers/LateSenders
  - counts shown in hierarchies of Synchronizations & Communications below Sends & Receives respectively
- Computational Imbalance
  - load imbalance heuristic calculated as absolute difference from average exclusive execution time
- HWC metrics
  - shown as separate root metrics for each counter
  - only provided in summary reports

Intermediate use of Scalasca

- User-defined region instrumentation
  - EPIK annotation macros API
  - POMP annotation directives
    Selective instrumentation
- Summary collections & analysis experiment
- Trace collection & analysis experiment
- Analysis report effectiveness score
- Customisation of measurement collection
  - Sizing of measurement data structures (e.g., trace buffers)
  - Function filter configuration
  - Optional HWC metrics
- Analysis report algebra

Instrumentation/measurement configuration

- Selective instrumentation
  - Adjust build not to (auto-)instrument particular modules
  - Separate/preprocess sources for functions in same module
  - Entirely avoids instrumentation & overhead
- Selective measurement via function filtering
  - Supported for GCC, IBM & Intel compilers
  - Specify text file listing names of functions (one per line, shell wildcarding) to ignore with EPK_FILTER
  - Use linker/decorated function names [Fortran/C++]

---

cube3_score with (sorted) region breakdown

```
% cube3_score -r epik_smg2000_mano_64/summary.cube | sort ...
flt type   max_tbc     time       %  region
...
    MPI    2061936   293.30   23.89  MPI_Waitall
    COM    2346840    14.79    1.20  hypre_FinalizeCommunication
    COM    2346840    23.49    1.91  hypre_InitializeCommunication
    MPI    7495240    11.38    0.93  MPI_Irecv
    MPI    8149850    41.43    3.37  MPI_Isend
    USR    9426048    10.47    0.85  hypre_StructStencilElementRank
    USR    9426048    21.69    1.77  hypre_StructMatrixExtractPointerByIdx
    USR   11063016    16.80    1.37  hypre_MAlloc$AF10_5
    USR   11454432    25.82    2.10  hypre_MAlloc
    USR   11763336    26.90    2.19  hypre_CAlloc
    USR   23496576    38.16    3.11  hypre_Free

    ANY  162589938  1227.61  100.00  ALL     (254 regions)
    MPI   17649090   456.64   37.20    + MPI ( 13 regions) pure MPI
    COM    9905832   321.80   26.21    + COM ( 32 regions) combined
   MPI&USR
    USR  135034968   311.13   25.34    + USR (207 regions) pure User

max_tbc = est. maximum trace buffer capacity requirement (bytes/process)
   to store all events that would be generated in an equivalent trace
```

cube3_score with trial region filter

```
% cube3_score -r -f smg2000.filt epik_smg2000_mano_64/summary.cube | sort
flt type    max_tbc      time       %  region
...
  - MPI    2061936    293.30   23.89  MPI_Waitall
  - COM    2346840     14.79    1.20  hypre_FinalizeCommunication
  - COM    2346840     23.49    1.91  hypre_InitializeCommunication
  - MPI    7495240     11.38    0.93  MPI_Irecv
  - MPI    8149850     41.43    3.37  MPI_Isend
  + USR    9426048     10.47    0.85  hypre_StructStencilElementRank
  + USR    9426048     21.69    1.77  hypre_StructMatrixExtractPointerByIdx
  + USR   11063016     16.80    1.37  hypre_MAlloc$AF10_5
  + USR   11454432     25.82    2.10  hypre_MAlloc
  + USR   11763336     26.90    2.19  hypre_CAlloc
  + USR   23496576     38.16    3.11  hypre_Free

  - ANY  162589938   1227.61  100.00  ALL     (253 regions)
  - MPI   17649090    456.64   37.20    + MPI ( 13 regions) pure MPI
  - COM    9905832    321.80   26.21    + COM ( 32 regions) combined
   MPI&USR
  - USR  135034968    311.13   25.34    + USR (207 regions) pure User

  + FLT  103570824    182.11   14.83        FLT (  9 regions) filtered
  - FLT   59019114   1045.50   85.17  ALL-FLT (244 regions) remainder
```

Preparation of instrumented executable

- Auto-instrumentation of functions
  - Capability of most (but not all) compilers
  - Currently need separate Scalasca installations for each desired combination of MPI library & compiler suite
  - $(PREP) $(MPIFC) ...
  - $(PREP) $(MPICC) ...
  - $(PREP) $(MPICXX) ...
  - PREP="skin $(SKIN_OPTS)" for instrumented build
  - PREP="" for uninstrumented build for production
- Auto-instrumentation plus API for user-defined regions
  - #include "epik_user.inc" or "epik_user.h"
  - % skin -user $(MPIC) ...

Manual instrumentation options

- No instrumentation
  - $(MPIC) [`kconfig -cflags`]
- MPI library instrumentation
  - $(MPIC) [`kconfig -cflags`] `kconfig -libs`
- MPI library & EPIK user instrumentation
  - $(MPIC) `kconfig -cflags` `kconfig -libs` -D**EPIK**
- `kconfig -cflags` is optional for source modules without explicit EPIK API #include

---

EPIK instrumentation API dummy macros

- To use unmodified compile commands (without EPIK API include path) for sources with EPIK API calls, define dummy macros

```
#ifdef EPIK
#include "epik_user.inc" or "epik_user.h"
#else
#define EPIK_FUNC_REG(str)      /* undefined */
#define EPIK_FUNC_START()       /* undefined */
#define EPIK_FUNC_END()         /* undefined */
#define EPIK_USER_REG(id,str)   /* undefined */
#define EPIK_USER_START(id)     /* undefined */
#define EPIK_USER_END(id)       /* undefined */
#endif
```

EPIK instrumentation API

- Manual phase annotation
  - EPIK_FUNC_REG("Fortran function/subroutine")
  - EPIK_FUNC_START()
  - EPIK_USER_REG(tsloop, "<<time step>>")
  - EPIK_USER_START(tsloop)
  - EPIK_USER_END(tsloop)
  - EPIK_FUNC_END()
- Note matching of enter/start annotations
  - all possible exits must be annotated
  - regions must be correctly nested
  - C/C++ function names are automatically registered
  - Fortran function/routine names undefined if not preregistered

---

POMP instrumentation

- Uses pragma/comment directives to annotate regions

C/C++:   Fortran:

```
#pragma pomp inst init              !POMP$ INST INIT
#pragma pomp inst begin(tsloop)     !POMP$ INST BEGIN(tsloop)
  #pragma pomp inst altend(tsloop)    !POMP$ INST
  ALTEND(tsloop)
#pragma pomp inst end(tsloop)       !POMP$ INST END(tsloop)
```

- Directives ignored unless activated with ***skin -pomp***
  - ***all directives in module instrumented***
- Current limitations
  - instrumentation inactive until “inst init”
  - no distinction of functions from other regions
  - last region exit must be marked “end”, all others as “altend”
  - doesn't support C99 `_Pragma` operator

Measurement configuration

- Example configuration
  - EPK_GDIR=/work/$USER # archive location
  - EPK_TITLE=app_$NP # experiment archive title
  - EPK_SUMMARY=1 # runtime summarisation
  - EPK_TRACE=0 # event trace collection
- New archive directory for each experiment
  - $EPK_GDIR/epik_$EPK_TITLE
  - contains intermediate data (e.g., trace files), log/config files and processed analyses
- Configured automatically (overridden) by ***scan*** args

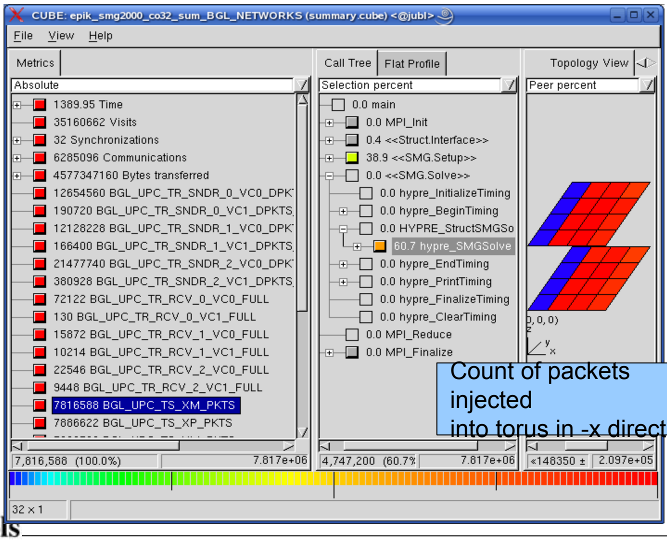
Default EPIK.CONF configuration file extract

```
# E P I K configuration
    –  EPK_TITLE=a         # experiment archive title [scan -e]
    –  EPK_SUMMARY=1 # runtime summarization [scan -s]
    –  EPK_TRACE=0       # event trace collection [scan -t]
    –  EPK_FILTER=        # file listing functions to skip
    –  EPK_METRICS=     # colon-separated list of metrics [-m]
# E P I S O D E configuration
    –  ESD_PATHS=1024 # max. recorded call-paths
    –  ESD_FRAMES=32  # max. call-stack frames
    –  ESD_BUFFER_SIZE=100000 # definitions bytes
# E P I L O G configuration
    –  ELG_BUFFER_SIZE=10000000 # trace bytes
```

Hardware counter metrics

- Available counters (and their interpretation) are platform/processor-specific
  - considered separate root metrics in analyses
- Platform metrics specification
  - defines convenient groups of metrics
  - EPK_METRICS_SPEC=./METRICS.SPEC
- Group/list of counters to measure in experiment
  - EPK_METRICS=POWER4_DC # data-cache
  - EPK_METRICS=BGL_NETWORKS # torus & tree
  - EPK_METRICS=PM_CYC:PM_INST_CMPL
    or PAPI_TOT_CYC:PAPI_TOT_INS

Scalasca summary experiment with HWC metrics



Metrics: Absolute

```
1389.95 Time
35160662 Visits
32 Synchronizations
6285096 Communications
4577347160 Bytes transferred
12654560 BGL_UPC_TR_SNDR_0_VC0_DPK
190720 BGL_UPC_TR_SNDR_0_VC1_DPKTS
12128228 BGL_UPC_TR_SNDR_1_VC0_DPK
166400 BGL_UPC_TR_SNDR_1_VC1_DPKTS
21477740 BGL_UPC_TR_SNDR_2_VC0_DPK
380928 BGL_UPC_TR_SNDR_2_VC1_DPKTS
72122 BGL_UPC_TR_RCV_0_VC0_FULL
130 BGL_UPC_TR_RCV_0_VC1_FULL
15872 BGL_UPC_TR_RCV_1_VC0_FULL
10214 BGL_UPC_TR_RCV_1_VC1_FULL
22546 BGL_UPC_TR_RCV_2_VC0_FULL
9448 BGL_UPC_TR_RCV_2_VC1_FULL
7816588 BGL_UPC_TS_XM_PKTS
7886622 BGL_UPC_TS_XP_PKTS
7,816,588 (100.0%)    7.817e+06
```

Call Tree: Selection percent

```
0.0 main
  0.0 MPI_Init
  0.4 <<Struct.Interface>>
  38.9 <<SMG.Setup>>
  0.0 <<SMG.Solve>>
    0.0 hypre_InitializeTiming
    0.0 hypre_BeginTiming
    0.0 HYPRE_StructSMGSo
      60.7 hypre_SMGSolve
    0.0 hypre_EndTiming
    0.0 hypre_PrintTiming
    0.0 hypre_FinalizeTiming
    0.0 hypre_ClearTiming
  0.0 MPI_Reduce
  0.0 MPI_Finalize
4,747,200 (60.7%    7.817e+06
```

Topology View: Peer percent

```
«148350 ±    2.097e+05
```

Count of packets injected
into torus in -x direction?

CUBE algebra tools

- CUBE files can be compared/combined with some useful command line tools
- Note that these work directly on CUBE files and not on archive directories
  - Reads CUBE2 & CUBE3 files, but only writes CUBE3 files
- General usage:

  -
    ```
    cube3_tool   [-o <output file>]   <input file>
    ```
- If no output file is specified, *tool*.cube is generated

CUBE algebra tools (2)

- cube3_merge
  - combines multiple analysis reports into integrated report
  - merges metric, call-path & system trees
  - takes metric severities from first available report
  - e.g., combine measurements of sets of HWC metrics in summary report(s) with a (non-HWC) trace analysis report into a “holistic” analysis report

  ```
  % cube3_merge trace.cube summary_HWC[1234].cube
  ```

    - Metrics listed in order of appearance in input reports
    - User-defined hierarchies of measured & derived HWC metrics not yet supported by CUBE3!

CUBE algebra tools (3)

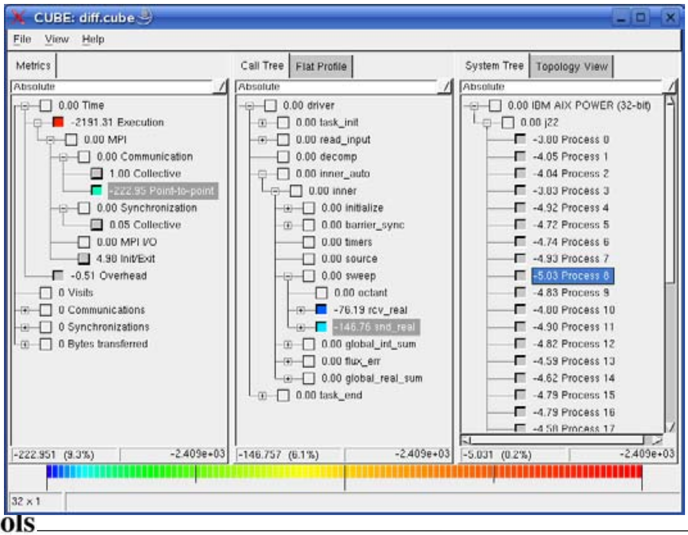
- `cube3_mean`
  - Can eliminate “measurement noise” by averaging the results of several experiments

- `cube3_cut [-p prune] [-r root]`
  - Creates a new CUBE file without pruned subtrees and/or containing only the specified call tree node as new root(s)

- `cube3_diff`
  - Calculates the difference of two experiments
  - Useful to measure improvement/degradation due to a modification

---

Difference experiment:
JUMP – JUBL (different architectures)



Labs!

Lab: PAPI, TAU, Vampir, and Scalasca/KOJAK

Lab Instructions (for LLNL systems)

Get workshop.tar.gz using:

```
% wget
  http://www.paratools.com/llnl09/workshop.tar.gz
```

Or

```
% cp
  /usr/global/tools/tau/training/src/workshop.tar.gz
  .
```

```
% tar zxf workshop.tar.gz
```

```
source /usr/global/tools/tau/training/src/tau.cshrc
```

OR

```
source /usr/global/tools/tau/training/src/tau.bashrc
```

in your .login file and then follow the instructions in the README file.

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
   ```
   % setenv TAU_MAKEFILE
   /usr/global/tools/tau/training/tau-
   2.18.2/bgp/lib/Makefile.tau-[options]
   ```
3. If stub makefile has -papi in its name, set COUNTER[1-<n>] environment variables:
   ```
   % setenv COUNTER1 GET_TIME_OF_DAY
   % setenv COUNTER2 PAPI_L2_DCM
   % setenv COUNTER3 PAPI_TOT_CYC … OR
   % setenv TAU_METRICS TIME:PAPI_L2_DCM:PAPI_TOT_CYC
   ```
4. Build and run workshop examples, then run pprof/paraprof

Support Acknowledgements

- Department of Energy (DOE)
  - Office of Science contracts
  - LLNL-LANL-SNL ASC/NNSA Level 3 contract
  - LLNL ParaTools contracts
- Department of Defense (DoD)
  - PET
- National Science Foundation (NSF)
  - POINT
- University of Oregon
  - A. Malony, A. Morris, K. Huck, W. Spear, S. Biersdorff, A. Nataraj
- University of Tennessee, Knoxville
  - Dr. David Cronk and Dr. Shirley Moore
- T.U. Dresden, GWT
  - Dr. Wolfgang Nagel and Dr. Holger Brunst
- Research Centre Juelich
  - Dr. Bernd Mohr, Dr. Felix Wolf