Parallel Performance Evaluation Tools Workshop

Los Alamos National Laboratory

Encantado Training Room, Suite 201, Room 219

Dec. 5 & 6, 2006, 8:30am - 4:30pm

Sameer Shende

info@paratools.com

http://www.paratools.com/lanl06

Outline

TOPIC	SLIDES
• Introduction to Performance Evaluation	#3 - #11
• PAPI	#12 - #21
• Perflib	#22 - #24
• TAU's Paraprof Profile Browser	#25 - #41
– LAB: Setup, PAPI, Perflib, and paraprof	#42 - #45
• TAU	#46 - #87
– Instrumentation	
– LAB: TAU Selective instrumentation and Memory	#88 - #89
– Meaurement and Instrumentation Optimization	#90 - #126
– LAB: TAU Measurement	#126 - #127
– Instrumentation for threaded programs (Kojak Opari, Java)	#128 - #140
– LAB: Opari	#141 - #142
• Vampir/VNG/OTF: Tracing with TAU	#143 - #166
– LAB: Tracing	# 167 - #168
• KOJAK	#169 - #216
– LAB: Epilog and Expert with TAU	#217 - #218
• TAU's PerfDMF Performance Database and Eclipse PTP IDE	#219 - #228
– LAB: Eclipse PTP	#229 - #230
• TAU's PerfExplorer Multi-experiment data analysis tool	#231 - #253
– LAB: PerfExplorer	#254 - #255

Workshop Goals

- This tutorial is intended as an introduction to portable performance evaluation tools.
- Today you should leave here with a better understanding of…
  - Concepts and steps involved in performance evaluation
  - How to use Perflib to quickly and easily obtain profile data
  - 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, C++,C, MPI for mixed language programming
  - How to generate trace data in different formats
  - How to analyze trace data using KOJAK, Vampir, and Jumpshot
  - How to use Eclipse PTP integrated development environment with TAU

More Information

- PAPI References:
  - PAPI documentation page available from the PAPI website:
    http://icl.cs.utk.edu/papi/
- TAU References:
  - "The TAU Parallel Performance System," Sameer Shende and Allen D. Malony, International Journal of High Performance Computing Applications, ACTS Special Issue, 20(2), pp. 287-331, Summer 2006.
  - TAU Users Guide.
  - Both available from the TAU website:
    http://www.cs.uoregon.edu/research/tau
- KOJAK References
  - KOJAK documentation page
    http://www.fz-juelich.de/zam/kojak/documentation/pr_documentation

Performance Evaluation

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

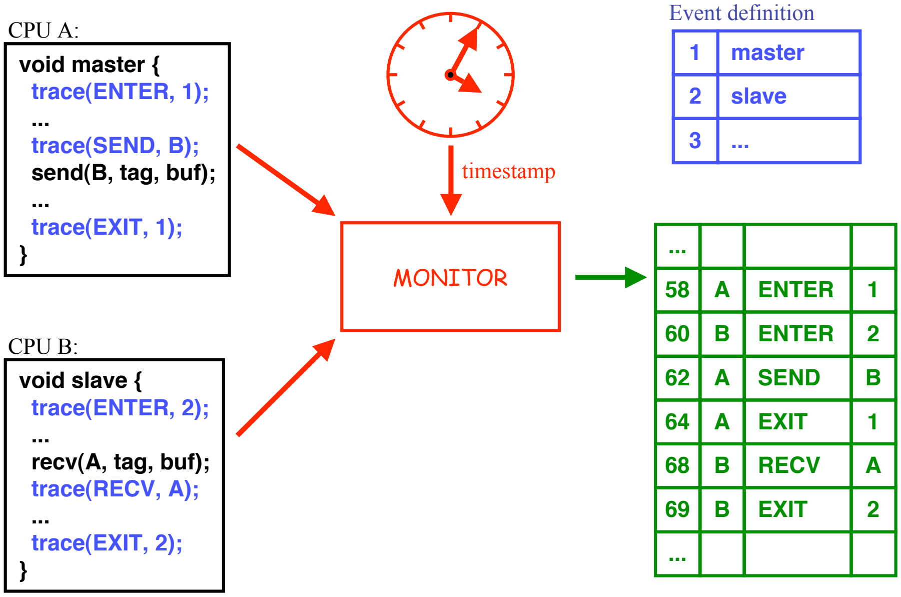
Definitions – Profiling

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

Definitions – Tracing

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

Event Tracing: Instrumentation, Monitor, Trace



CPU A:

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

CPU B:

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

Event definition

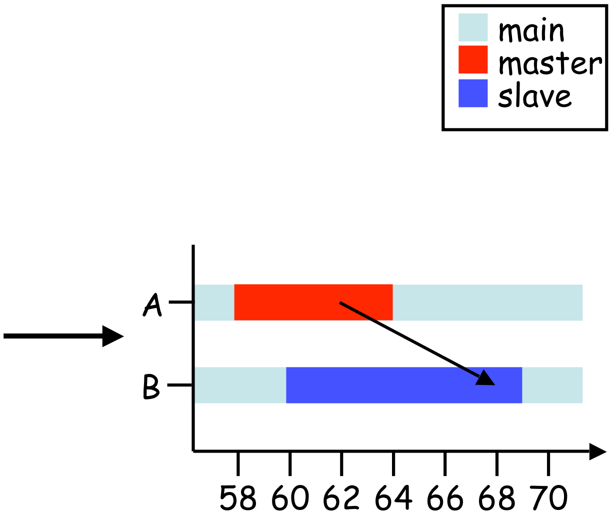
1	master
2	slave
3	...

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

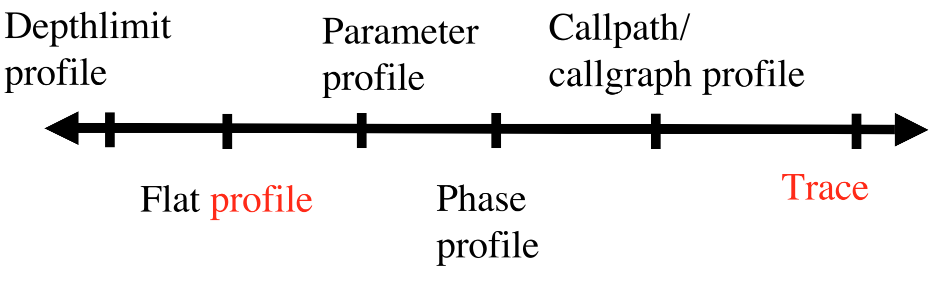
Event Tracing: “Timeline” Visualization

1	master
2	slave
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
...			



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

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

PAPI

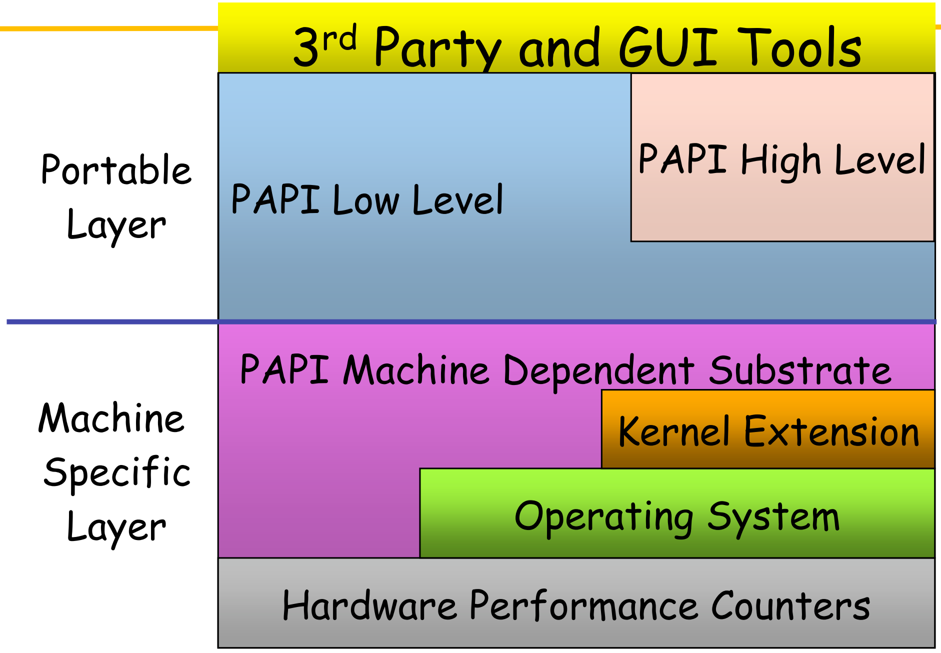
- **P**erformance **A**pplication **P**rogramming **I**nterface
  - The purpose of the PAPI project is to design, standardize and implement a portable and efficient API to access the hardware performance monitor counters found on most modern microprocessors.
- Parallel Tools Consortium project
- Developed by University of Tennessee, Knoxville
- http://icl.cs.utk.edu/papi/

PAPI Counter Interfaces

PAPI provides 3 interfaces to the underlying counter hardware:

1. The low level interface manages hardware events in user defined groups called *EventSets*, and provides access to advanced features.
2. The high level interface 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 Implementation



3rd Party and GUI Tools

Portable Layer

PAPI Low Level

PAPI High Level

Machine Specific Layer

PAPI Machine Dependent Substrate

Kernel Extension

Operating System

Hardware Performance Counters

PAPI Hardware Events

- Preset Events
  - Standarhd 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
- 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
- PAPI Event Chooser to see if events are compatible
  - Use papi_event_chooser <PRESET | NATIVE> [<events> ...]
  - % papi_event_chooser PRESET PAPI_FP_INS PAPI_L2_DCM

papi_avail

```
$ /usr/local/packages/papi-3.5.0/bin/papi_avail
Available events and hardware information.
-------------------------------------------------------------------------
Vendor string and code   : GenuineIntel (1)
Model string and code    : Intel Core (17)
CPU Revision             : 8.000000
CPU Megahertz            : 1000.000000
CPU's in this Node       : 2
Nodes in this System     : 1
Total CPU's              : 2
Number Hardware Counters : 2
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      No      No      Level 3 data cache misses ...
PAPI_FNV_INS    0x80000065      No      No      Floating point inverse instructions
PAPI_FP_OPS     0x80000066      Yes     No      Floating point operations
-------------------------------------------------------------------------
avail.c                                     PASSED
```

PAPI High-level Interface

- Meant for application programmers wanting coarse-grained measurements
- Calls the lower level API
- Allows only PAPI preset events
- Easier to use and less setup (less additional code) than low-level
- Supports 8 calls in C or Fortran:

```
PAPI_start_counters          PAPI_stop_counters

PAPI_read_counters           PAPI_accum_counters

PAPI_num_counters            PAPI_flips

PAPI_ipc                     PAPI_flops
```

PAPI High-level Example

```
#include "papi.h”
#define NUM_EVENTS 2
long_long values[NUM_EVENTS];
unsigned int Events[NUM_EVENTS]={PAPI_TOT_INS,PAPI_TOT_CYC};

 /* Start the counters */
 PAPI_start_counters((int*)Events,NUM_EVENTS);

 /* What we are monitoring… */
 do_work();

 /* Stop counters and store results in values */
 retval = PAPI_stop_counters(values,NUM_EVENTS);
```

Low-level Interface

- Increased efficiency and functionality over the high level PAPI interface
- Obtain information about the executable, the hardware, and the memory environment
- Multiplexing
- Callbacks on counter overflow
- Profiling
- About 60 functions

PAPI Low-level Example

```
#include "papi.h”
#define NUM_EVENTS 2
int Events[NUM_EVENTS]={PAPI_FP_INS,PAPI_TOT_CYC};
int EventSet;
long_long values[NUM_EVENTS];
/* Initialize the Library */
retval = PAPI_library_init(PAPI_VER_CURRENT);
/* Allocate space for the new eventset and do setup */
retval = PAPI_create_eventset(&EventSet);
/* Add Flops and total cycles to the eventset */
retval = PAPI_add_events(EventSet,Events,NUM_EVENTS);
/* Start the counters */
retval = PAPI_start(EventSet);

do_work();  /* What we want to monitor*/

/*Stop counters and store results in values */
retval = PAPI_stop(EventSet,values);
```

Tools that use PAPI
Most users access PAPI from higher-level tools

- TAU (U. Oregon) http://www.cs.uoregon.edu/research/tau/
- PERFLIB (Jeff Brown, LANL) http://www.lanl.gov (jeffb@lanl.gov)
- HPCToolkit (Rice Univ) http://hipersoft.cs.rice.edu/hpctoolkit/
- KOJAK (UTK, FZ Juelich) http://icl.cs.utk.edu/kojak/
- PerfSuite (NCSA) http://perfsuite.ncsa.uiuc.edu/
- Titanium (UC Berkeley) http://www.cs.berkeley.edu/Research/Projects/titanium/
- Open|Speedshop (SGI) http://oss.sgi.com/projects/openspeedshop/
- SvPablo (UNC Renaissance Computing Institute) http://www.renci.unc.edu/Software/Pablo/pablo.htm

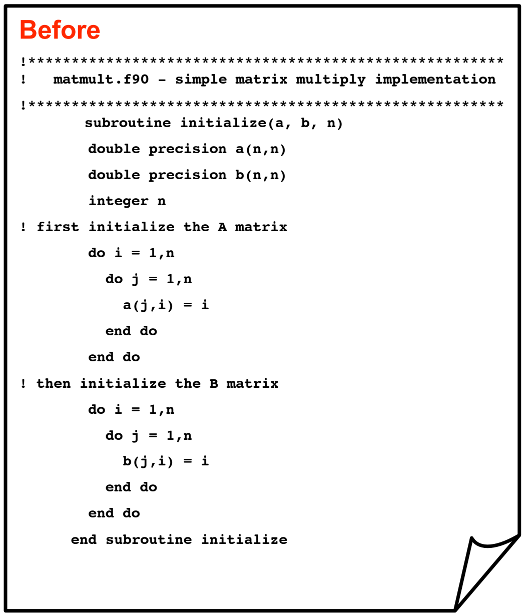
PERFLIB

- Leverage existing timers user application codes
- Provide an easy to use API
- Both manual and automatic (w/ TAU) modes of instrumentation supported
- Memory, cpu, io, flops (PAPI), message traffic, elapsed time available
- Complete callgraph is available
- Incremental program dumps using cycles (at major iteration boundaries)
- Both C and Fortran languages are supported
- Profile data converted to TAU format using perf2tau
- Specify different measurement options at runtime
- Already installed on QSC, QB, lightning, bolt
- Integrated in Crestone projects
- Jeff Brown [jeffb@lanl.gov]

PERFLIB

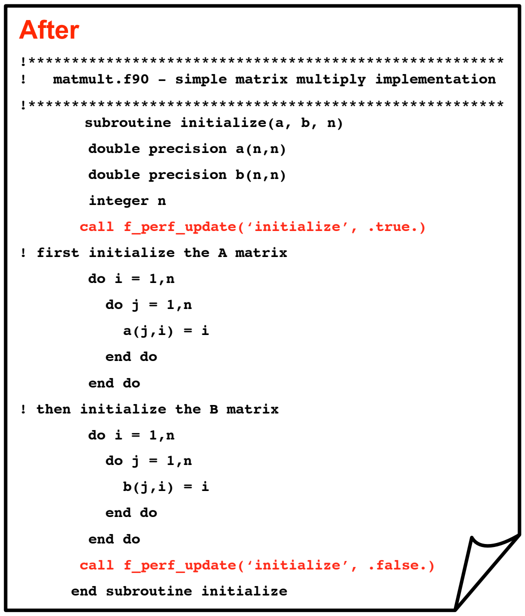
- Easy to use callgraph profiling package. F90 API:
  - call f_perf_update('routine_name', .true.)   ! for entry in routine
  - call f_perf_update('routine_name', .false.) ! for exit from routine
  - call f_perf_dump(<value>)   ! dump perf data at cycles (optional)
- Link with Perflib and PAPI:
  - mpif90 app.o -L/usr/projects/codeopt/PERF/v2.0/lib/linux/i686 -lperfrt -lpapi
- Generate performance data by setting environment variables:
  - % setenv PERF_DATA_DIRECTORY timing
  - % setenv PERF_PROFILE_[TIME, MEMORY, MPI, IO, COUNTERS]
  - % setenv PERF_FILTER_MPI 1000000 (size of messages filter)
  - % mpirun -np 4 ./a.out
  - See /usr/projects/codeopt/PERF/v2.0/doc/ReadMe for further details
- Convert to TAU format and view in TAU's paraprof GUI profile browser:
  - % perf2tau timing; paraprof --pack timing.ppk; paraprof timing.ppk

PERFLIB



Before

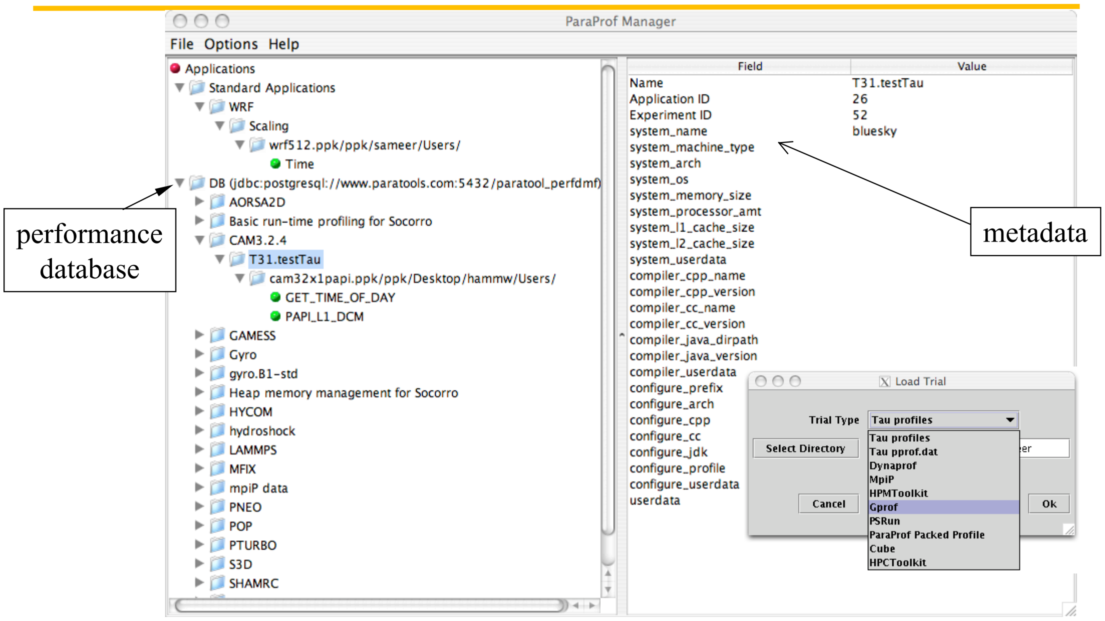
```
!*********************************************************
!   matmult.f90 - simple matrix multiply implementation
!*********************************************************
       subroutine initialize(a, b, n)
        double precision a(n,n)
        double precision b(n,n)
        integer n
! first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



After

```
!*********************************************************
!   matmult.f90 - simple matrix multiply implementation
!*********************************************************
       subroutine initialize(a, b, n)
        double precision a(n,n)
        double precision b(n,n)
        integer n
       call f_perf_update('initialize', .true.)
! 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
       call f_perf_update('initialize', .false.)
      end subroutine initialize
```

ParaProf – Manager Window



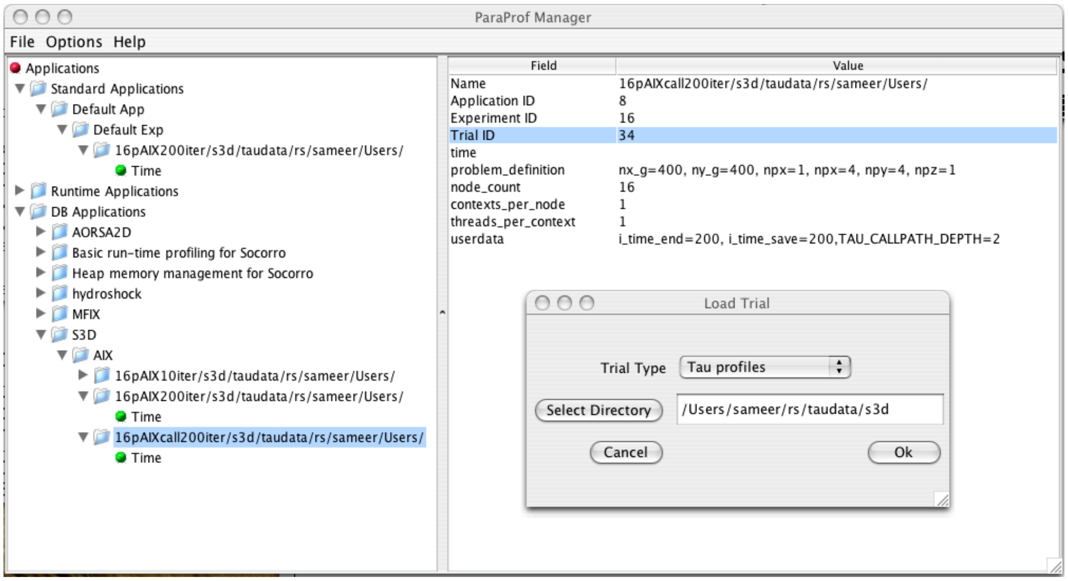
- Applications
  - Standard Applications
    - WRF
      - Scaling
        - wrf512.ppk/ppk/sameer/Users/
          - Time
  - DB (jdbc:postgresql://www.paratools.com:5432/paratool_perfdmf)
    - AORSA2D
    - Basic run-time profiling for Socorro
    - CAM3.2.4
      - T31.testTau
        - cam32x1papi.ppk/ppk/Desktop/hammw/Users/
          - GET_TIME_OF_DAY
          - PAPI_L1_DCM
    - GAMESS
    - Gyro
    - gyro.B1-std
    - Heap memory management for Socorro
    - HYCOM
    - hydroshock
    - LAMMPS
    - MFIX
    - mpiP data
    - PNEO
    - POP
    - PTURBO
    - S3D
    - SHAMRC

Field	Value
Name	T31.testTau
Application ID	26
Experiment ID	52
system_name	bluesky
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	
compiler_cpp_version	
compiler_cc_name	
compiler_cc_version	
compiler_java_dirpath	
compiler_java_version	
compiler_userdata	
configure_prefix	
configure_arch	
configure_cpp	
configure_cc	
configure_jdk	
configure_profile	
configure_userdata	
userdata	

Trial Type options:

- Tau profiles
- Tau pprof.dat
- Dynaprof
- MpiP
- HPMToolkit
- Gprof
- PSRun
- ParaProf Packed Profile
- Cube
- HPCToolkit

Performance Database: Storage of MetaData



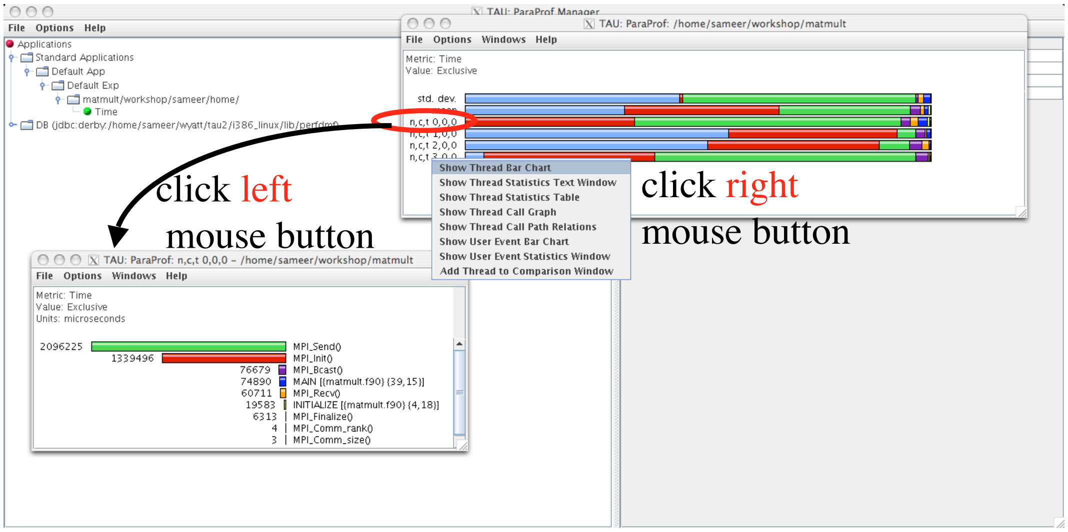
- Applications
  - Standard Applications
    - Default App
      - Default Exp
        - 16pAIX200iter/s3d/taudata/rs/sameer/Users/
          - Time
  - Runtime Applications
  - DB Applications
    - AORSA2D
    - Basic run-time profiling for Socorro
    - Heap memory management for Socorro
    - hydroshock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

Trial Type: Tau profiles

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

ParaProf Main Window


TAU: ParaProf Manager
File Options Help
Applications
Standard Applications
Default App
Default Exp
matmult/workshop/sameer/home/
Time
DB (jdbc:derby:/home/sameer/wyatt/tau2/i386_linux/lib/perfdmf)
TAU: ParaProf: /home/sameer/workshop/matmult
File Options Windows Help
std. dev.
mean
n,c,t 0,0,0
n,c,t 1,0,0
n,c,t 2,0,0
n,c,t 3,0,0
Show Thread Bar Chart
Show Thread Statistics Text Window
Show Thread Statistics Table
Show Thread Call Graph
Show Thread Call Path Relations
Show User Event Bar Chart
Show User Event Statistics Window
Add Thread to Comparison Window
TAU: ParaProf: n,c,t 0,0,0 – /home/sameer/workshop/matmult
File Options Windows Help

Metric: Time
Value: Exclusive

Metric: Time
Value: Exclusive
Units: microseconds

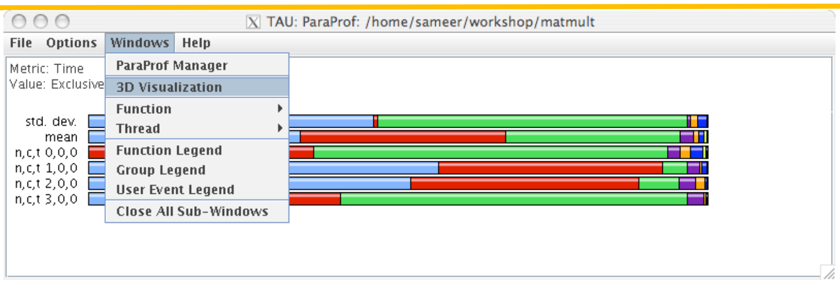
```
2096225  MPI_Send()
1339496  MPI_Init()
  76679  MPI_Bcast()
  74890  MAIN [{matmult.f90} {39,15}]
  60711  MPI_Recv()
  19583  INITIALIZE [{matmult.f90} {4,18}]
   6313  MPI_Finalize()
      4  MPI_Comm_rank()
      3  MPI_Comm_size()
```

click left mouse button

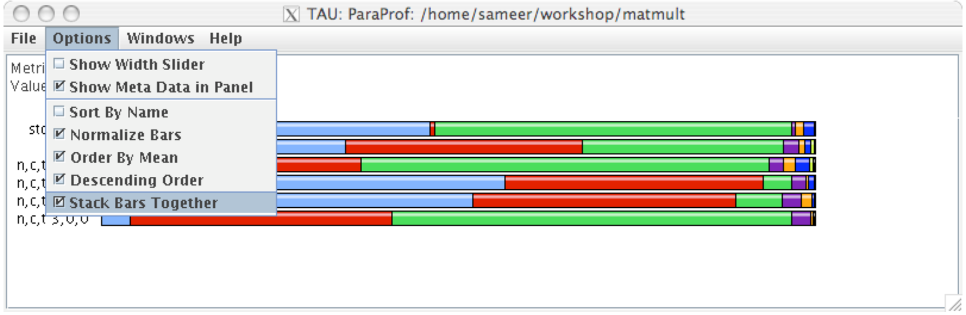
click right mouse button

`% paraprof  matmult.ppk`

Paraprof

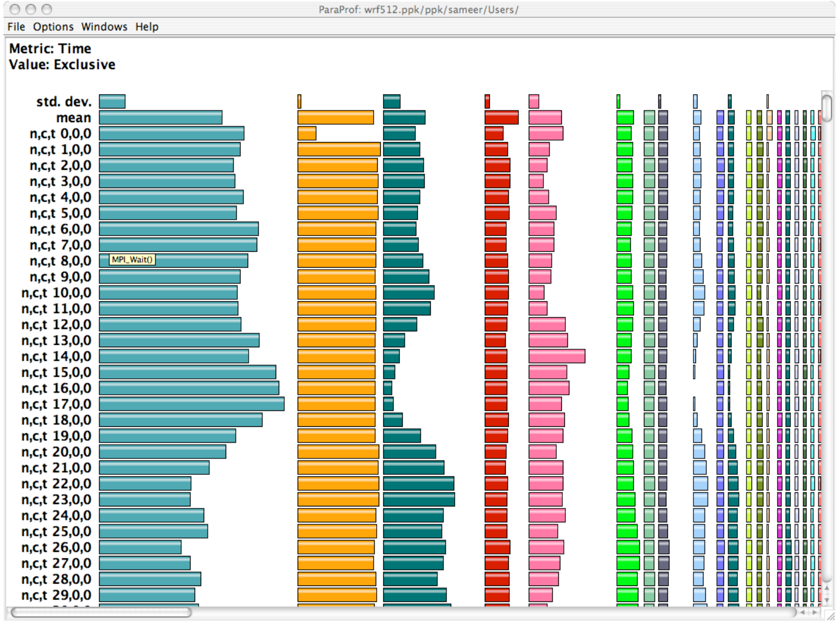


Metric: Time
Value: Exclusive


TAU: ParaProf: /home/sameer/workshop/matmult
File Options Windows Help
Metri
Value
std
n,c,t
n,c,t
n,c,t
n,c,t 3,0,0

☐ Show Width Slider
☑ Show Meta Data in Panel
☐ Sort By Name
☑ Normalize Bars
☑ Order By Mean
☑ Descending Order
☑ Stack Bars Together

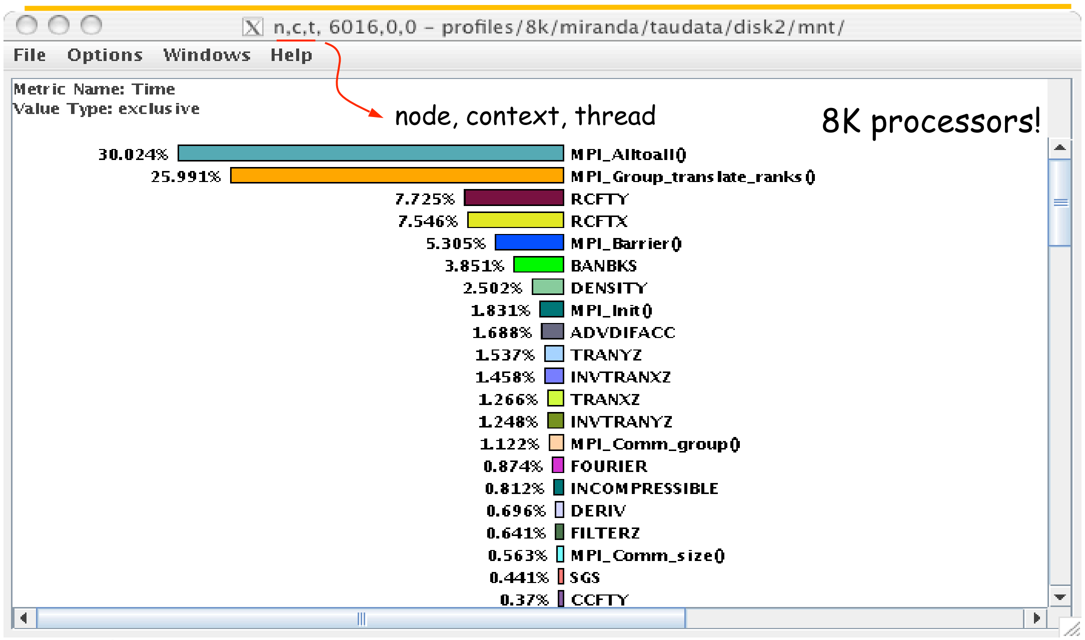
ParaProf Main Window (WRF)



ParaProf: wrf512.ppk/ppk/sameer/Users/

Metric: Time
Value: Exclusive

ParaProf – Flat Profile



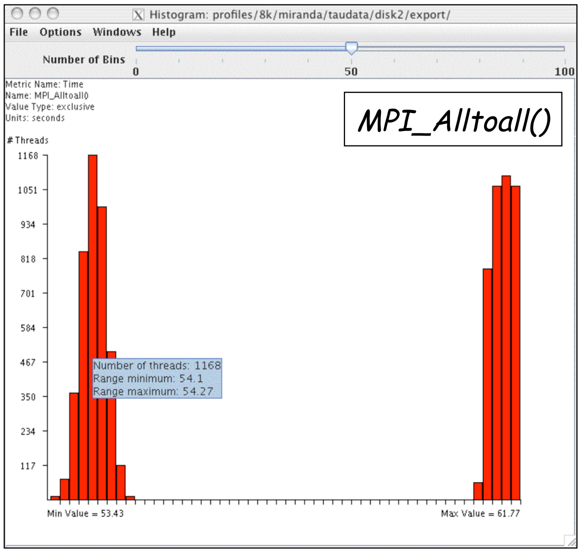
n,c,t, 6016,0,0 – profiles/8k/miranda/taudata/disk2/mnt/

Metric Name: Time
Value Type: exclusive

node, context, thread

8K processors!

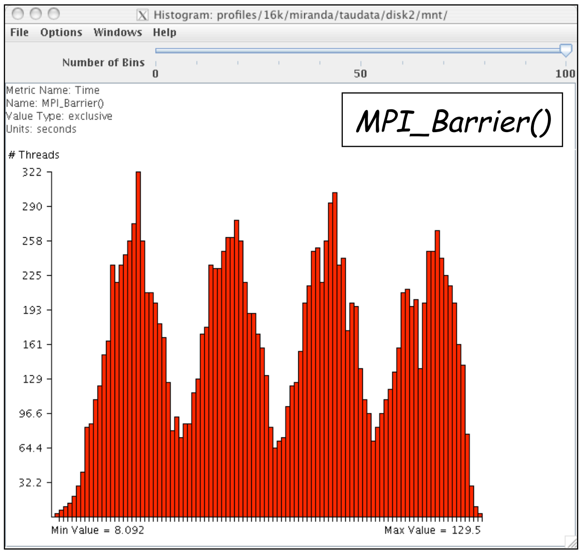
ParaProf – Histogram View



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

MPI_Alltoall()

8k processors

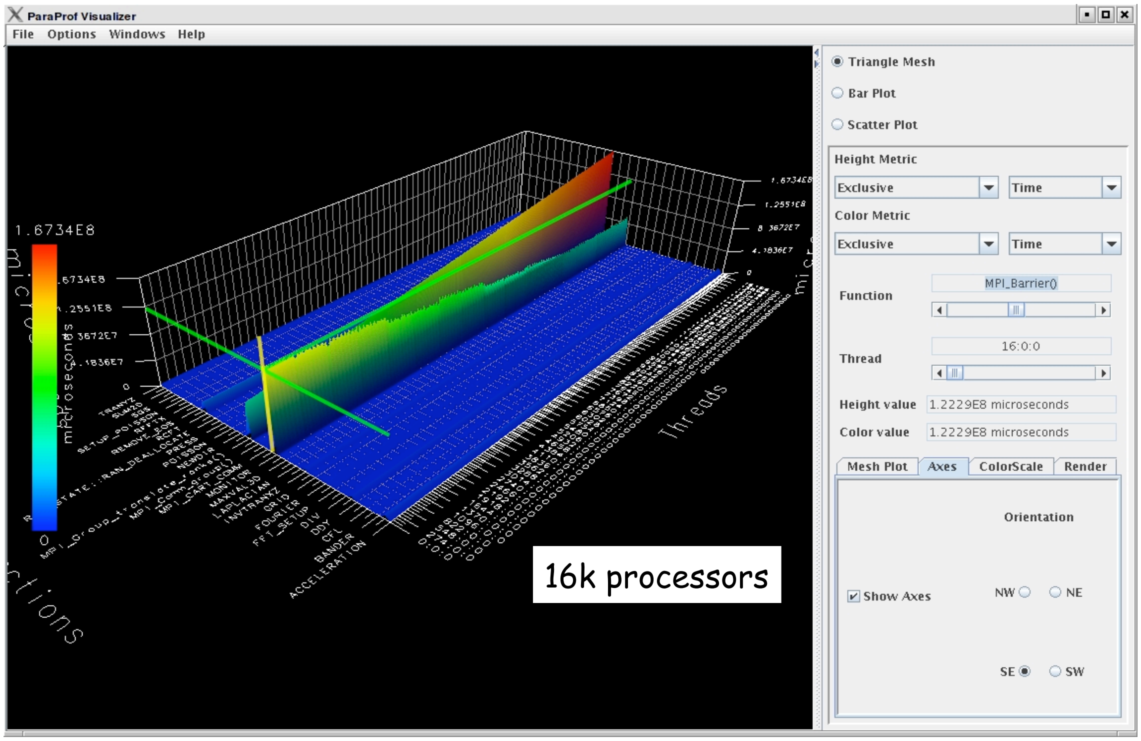


Metric Name: Time
Name: MPI_Barrier()
Value Type: exclusive
Units: seconds

MPI_Barrier()

16k processors

ParaProf – 3D Full Profile



● 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

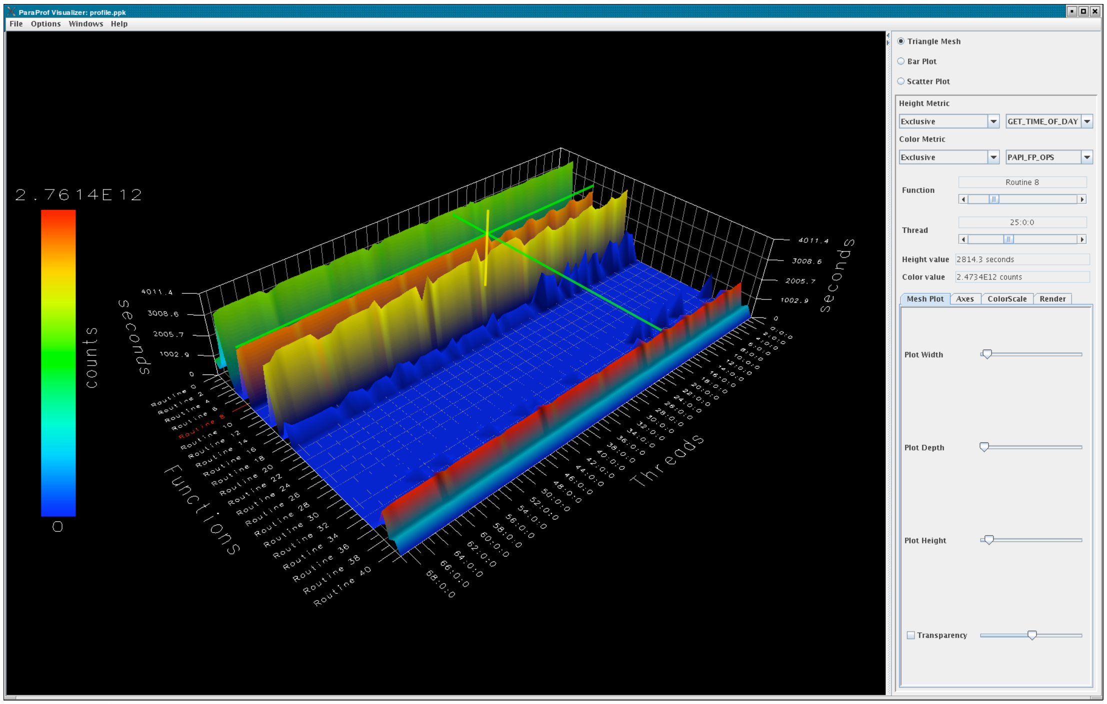
Orientation

☑ Show Axes

NW ○ NE ○ SE ● SW ○

16k processors

ParaProf – 3D Full Profile



● Triangle Mesh

○ Bar Plot

○ Scatter Plot

Height Metric: Exclusive, GET_TIME_OF_DAY

Color Metric: Exclusive, PAPI_FP_OPS

Function: Routine 8

Thread: 25:0:0

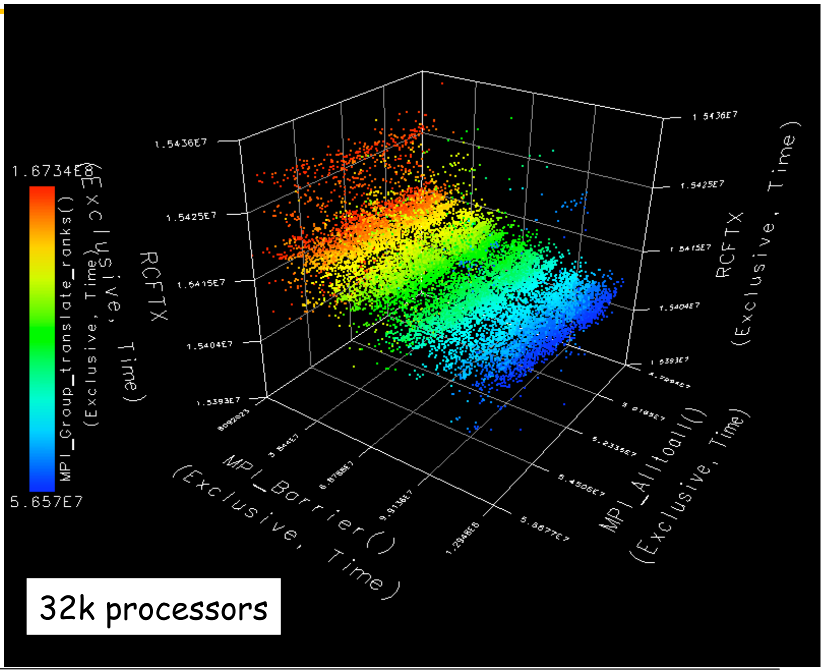
Height value: 2814.3 seconds

Color value: 2.4734E12 counts

□ Transparency

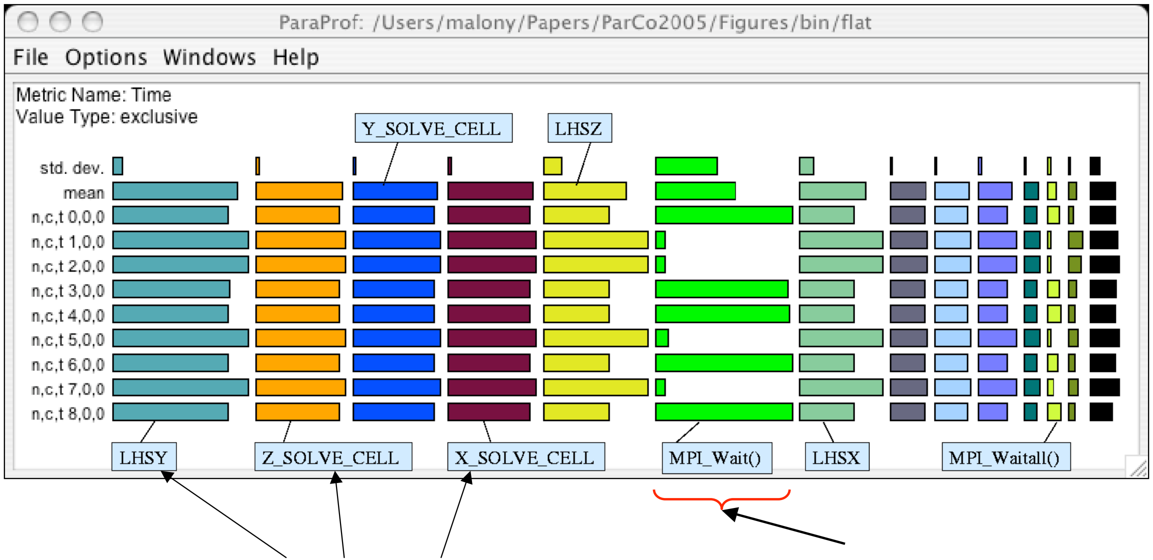
ParaProf – 3D Scatterplot

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



32k processors

ParaProf – Flat Profile (NAS BT)



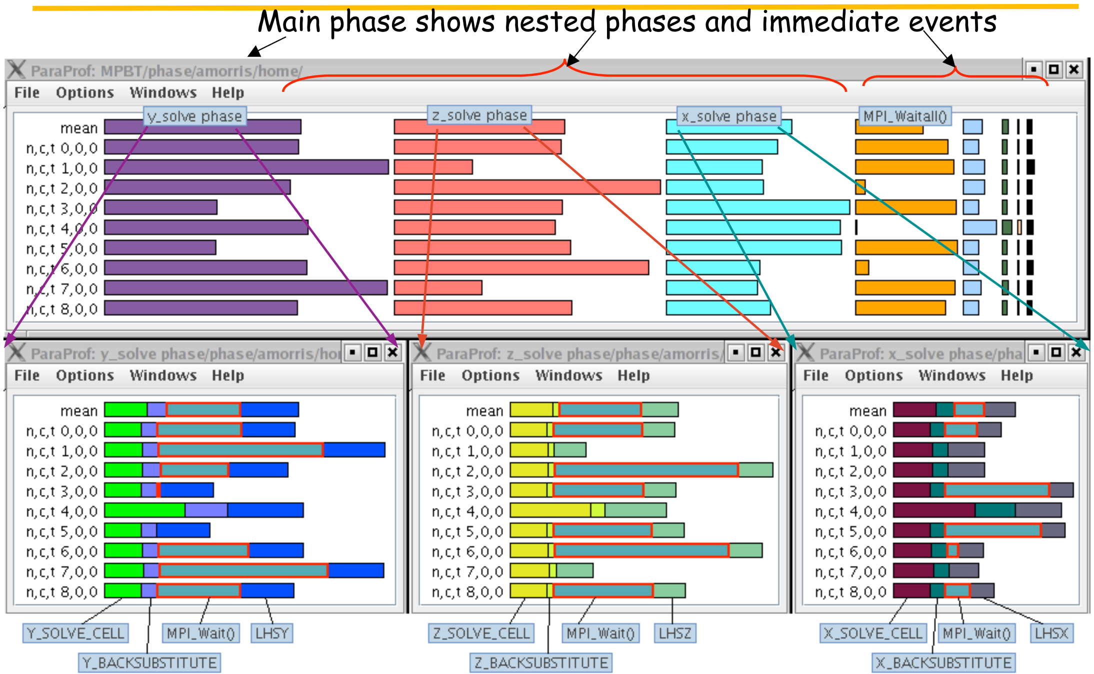
Metric Name: Time
Value Type: exclusive

Application routine names reflect phase semantics

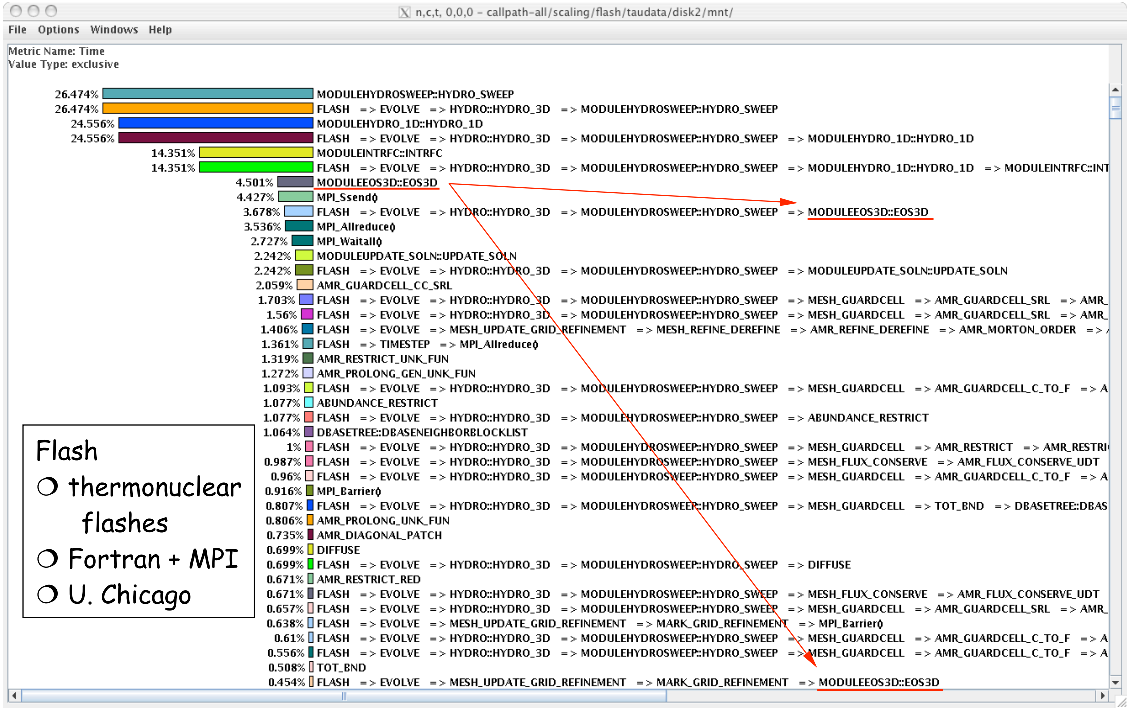
How is MPI_Wait() distributed relative to solver direction?

ParaProf – Phase Profile (NAS BT)

Main phase shows nested phases and immediate events



ParaProf – Callpath Profile (Flash)

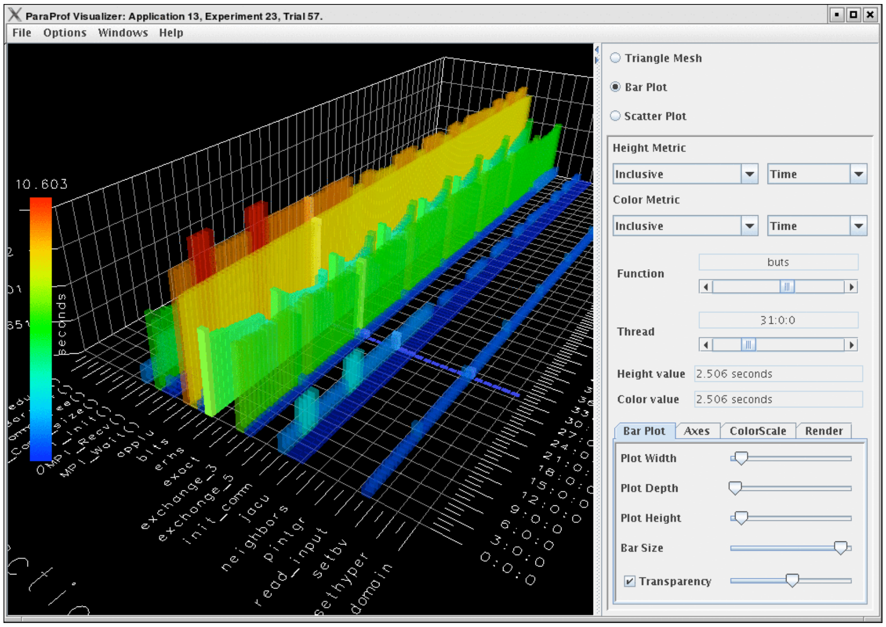


n,c,t, 0,0,0 - callpath-all/scaling/flash/taudata/disk2/mnt/
File Options Windows Help

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



○ 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

Plot Width

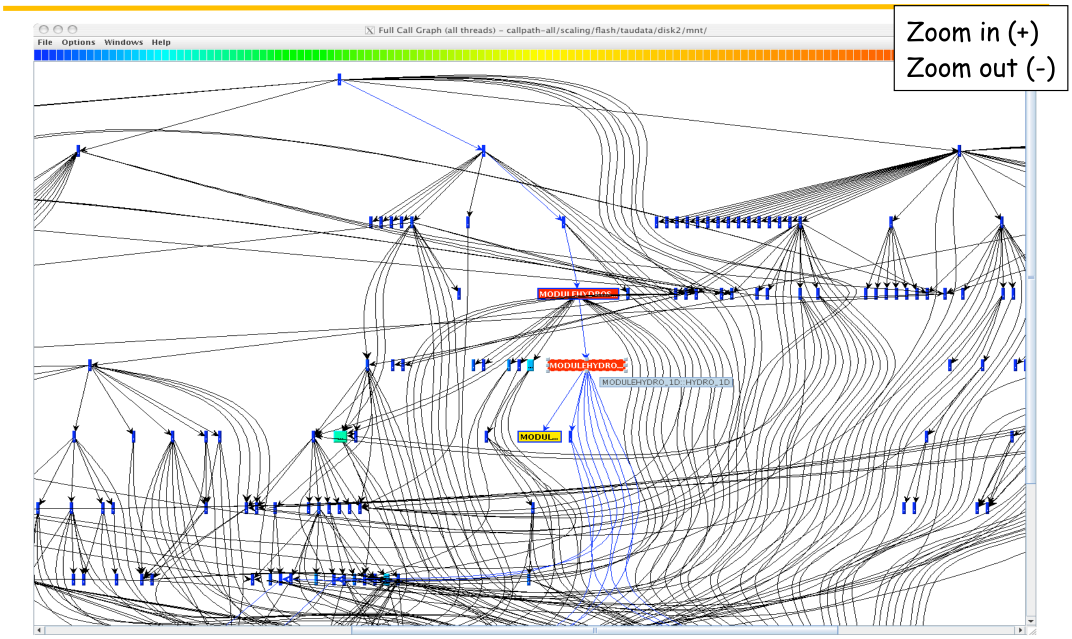
Plot Depth

Plot Height

Bar Size

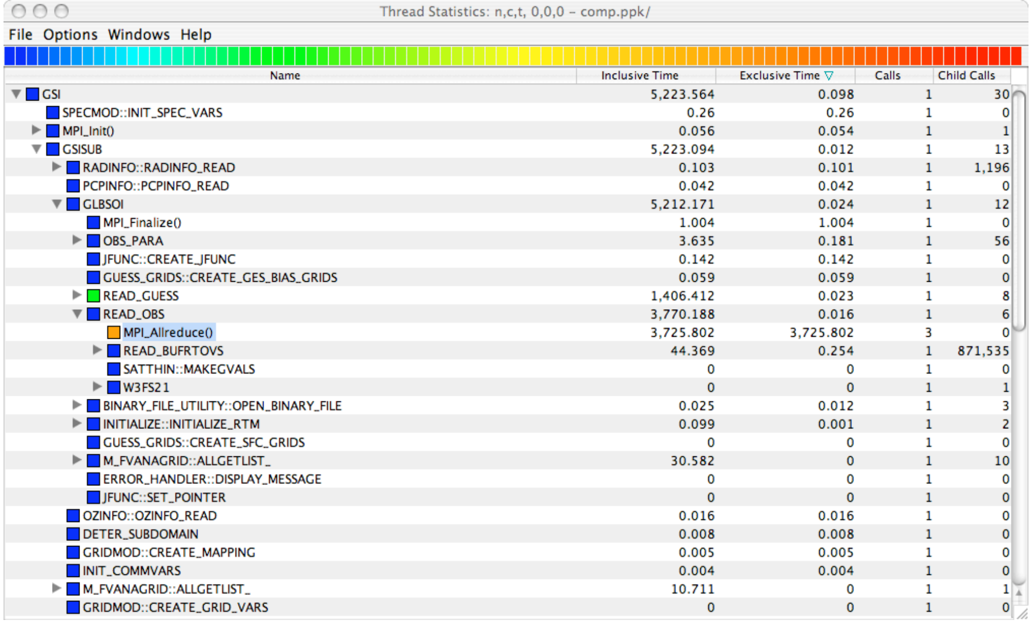
☑ Transparency

ParaProf – Callgraph Zoomed (Flash)


Full Call Graph (all threads) – callpath-all/scaling/flash/taudata/disk2/mnt/
File Options Windows Help
MODULEHYDRO...
MODULEHYDRO_1D::HYDRO_1D
MODUL...

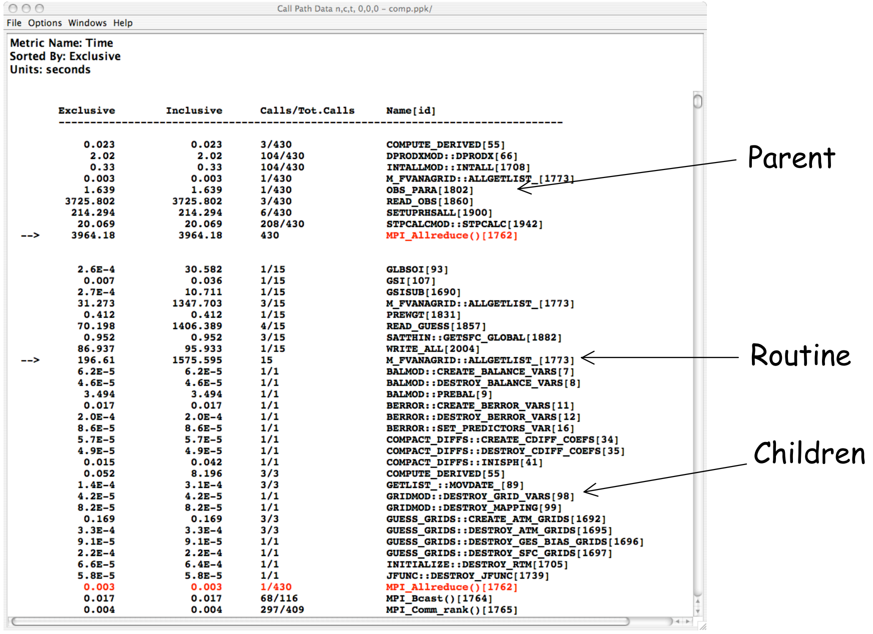
Zoom in (+)
Zoom out (-)

ParaProf - Thread Statistics Table (GSI)


Thread Statistics: n,c,t, 0,0,0 – comp.ppk/
File Options Windows Help

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


Call Path Data n,c,t, 0,0,0 – comp.ppk/
File Options Windows Help
Parent
Routine
Children

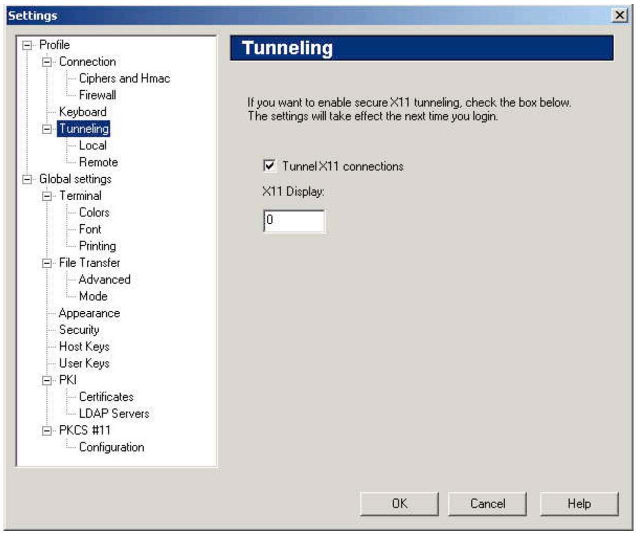
```
Metric Name: Time
Sorted By: Exclusive
Units: seconds


        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]
```

Labs!

Lab 1: PERFLIB and PAPI

Lab Instructions: Logins, Tunneling from Workstations

- Start cygwin and invoke:
  - % startx
- Start F-Secure ssh
  - Edit -> Preferences
  - Select tunneling
- Tools are installed on:
  - ffe1 (flash 32 bit)
  - ffe-64 (flash 64 bit)
  - tlc (32 bit Turquoise)
    - ssh -Y wtrw.lanl.gov
  - qscfe1 (QSC Tru64)
  - laptop
    - ssh -Y ffe1.lanl.gov
- Login to the cluster and try running xclock



If you want to enable secure X11 tunneling, check the box below. The settings will take effect the next time you login.

☑ Tunnel X11 connections

X11 Display: 0

Lab Instructions

- Add one of .cshrc/.bashrc

  source /usr/projects/crestone/sameer/tau.cshrc (on ffe1,ffe64,qscfe1) (or tau.bashrc)
  source /usr/projects/EES_dev/sameer/tau.cshrc (on tlc)
  source /usr/local/packages/tau.cshrc (on laptop)
  to the end of your **.login** file (for tcsh or bash users respectively)
  These files contain LANL specific location information. We will use tau-2.16 for the experiments.

- Get workshop.tar.gz using:

```
% wget http://www.paratools.com/lanl06/workshop.tar.gz
```

Or

```
% cp $TAU/workshop.tar.gz .
% tar zxf workshop.tar.gz
and follow the instructions in the README file.
```

Lab Notes: Running Jobs

- Login front-end nodes (ffe1, ffe-64), bproc master nodes (flashd), bproc compute nodes
- ffe1.lanl.gov (32 bit Linux, no compiles on master node)
  - llogin -n 4
- ffe-64.lanl.gov (64 bit Linux, no compiles on master node)
  - llogin -n 4 -q flash64q
  - flashd% mpirun -np 4 ./a.out
  - Interactive OR Batch:
  - flashd% bsub -q flash64q -W 3 mpirun -np 4 ./a.out
    - to run with 3 minute wallclock time run limit on 4 cpus.
- qscfe1.lanl.gov (Tru64, compile on backend node)
  - llogin -n 4
  - Small job : % prun -n 4 ./a.out
  - % prun -O -n 6 ./sweep3d (-O is to override the node limit. Run 6 cpu job on 4)
  - Big job: % bsub -q largeq -n 512 -o myout.lu -W 15 prun -t -n 512 ./lu.A.512
  - To run sequential job with PAPI, instead of llogin:
    - bsub -a DADD -n 4 -q smallq -ls tcsh -l
    - % `pwd`/a.out

TAU Parallel Performance System

Tuning and Analysis Utilities

- http://www.cs.uoregon.edu/research/tau/
- 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
- Joint project: University of Oregon, ACL LANL and FZJ (NIC)

UNIVERSITY OF OREGON

Los Alamos NATIONAL LABORATORY

NIC

Using TAU: A brief Introduction

- Add one of .cshrc/.bashrc

```
source /usr/projects/crestone/sameer/tau.cshrc  (on ffe1,ffe64,qscfe1)
(or tau.bashrc)
source /usr/projects/EES_dev/sameer/tau.cshrc (on tlc)
source /usr/local/packages/tau.cshrc (on laptop)
```

  to the end of your **.login** file (for bash or csh/tcsh users respectively)

  These files contain LANL specific location information. We will use tau-2.16 for the experiments.

- For using TAU with an uninstrumented MPI application:

```
% mpirun –np 4 a.out
```

  changes to

```
% mpirun –np 4 tau_load.sh a.out
```

Using TAU: A brief Introduction

- To instrument source code:

```
% setenv TAU_MAKEFILE      $TAU/Makefile.tau-mpi-pdt-pgi
```

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

```
% mpif90 foo.f90
```

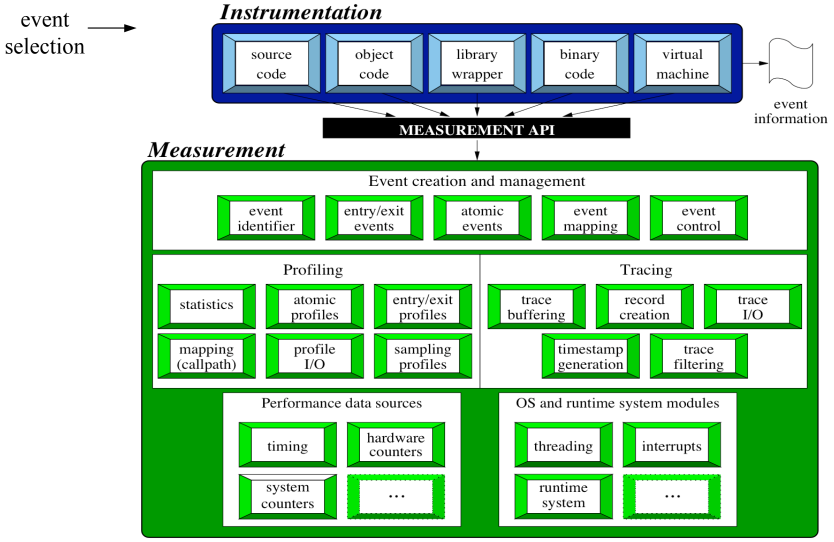
  changes to

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

- Execute application and then run:

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

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

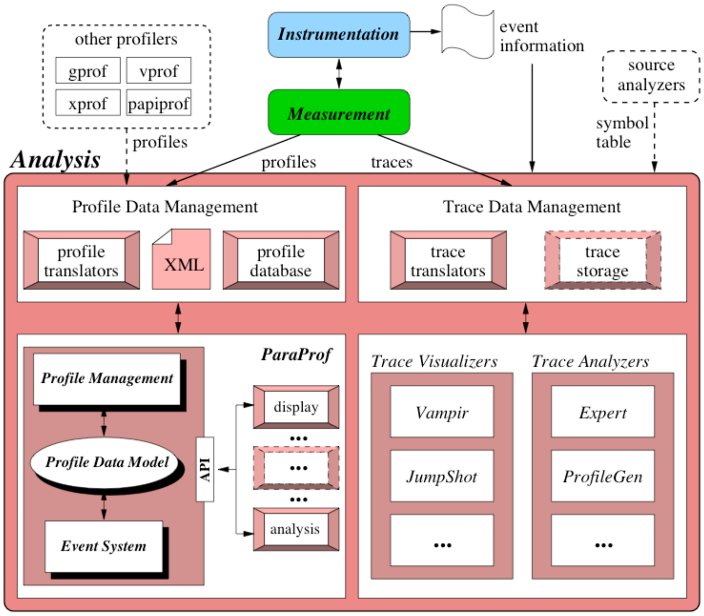
threading

interrupts

runtime system

...

TAU Performance System Architecture



other profilers

gprof

vprof

xprof

papiprof

Instrumentation

event information

source analyzers

Measurement

profiles

symbol table

Analysis

profiles

traces

Profile Data Management

profile translators

XML

profile database

Trace Data Management

trace translators

trace storage

ParaProf

Profile Management

Profile Data Model

Event System

API

display

•••

•••

•••

analysis

Trace Visualizers

Vampir

JumpShot

...

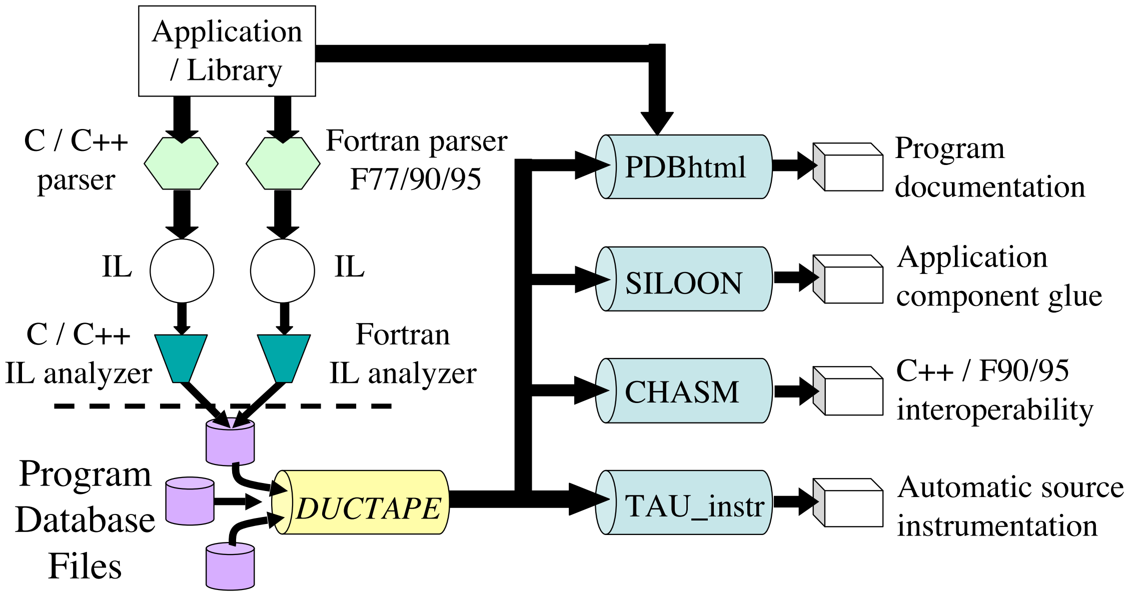
Trace Analyzers

Expert

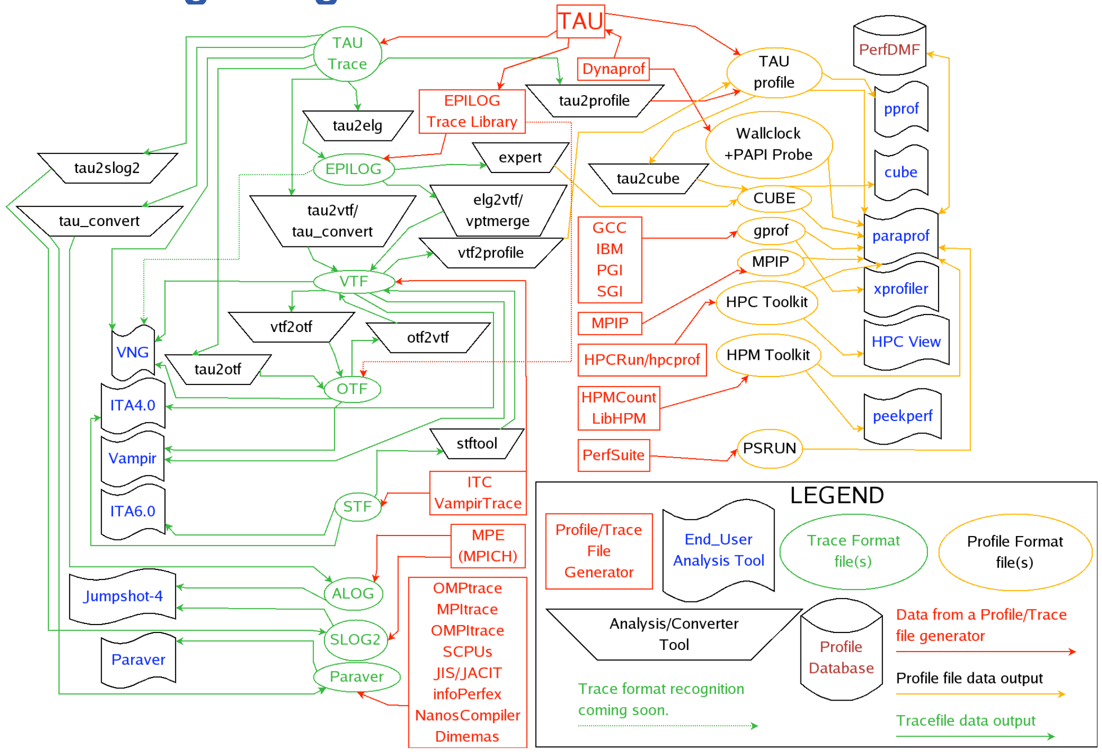
ProfileGen

...

Program Database Toolkit (PDT)



Building Bridges to Other Tools: TAU



TAU Instrumentation Approach

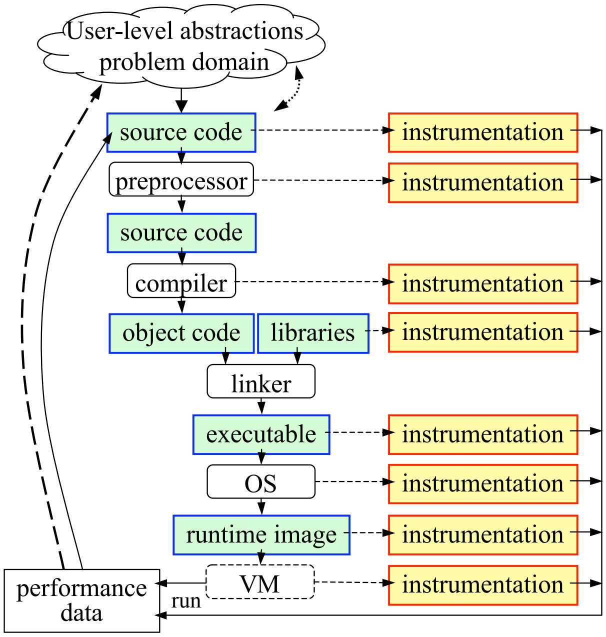
- Support for standard program events
  - Routines
  - Classes and templates
  - Statement-level blocks
- Support for user-defined events
  - Begin/End events ("user-defined timers")
  - Atomic events (e.g., size of memory allocated/freed)
  - Selection of event statistics
- Support definition of "semantic" 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

Multi-Level Instrumentation and Mapping

- Multiple instrumentation interfaces
- Information sharing
  - Between interfaces
- Event selection
  - Within/between levels
- Mapping
  - Associate performance data with high-level semantic abstractions
- Instrumentation targets measurement API with support for mapping



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
  - 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
-perf[inc/lib]=<dir>	Specify location of PERFLIB
-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 –pdt=/opt/pkgs/pdtoolkit-3.9 -mpi
  - Configure using PDT and MPI with GNU compilers
- ./configure -papi=/usr/local/packages/papi -pdt=/usr/local/pdtoolkit-3.9 -mpiinc=/usr/local/include -mpilib=/usr/local/lib -MULTIPLECOUNTERS –c++=icpc –cc=icc –fortran=intel -tag=intel91039; make clean install
  - Use PAPI counters (one or more) with C/C++/F90 automatic instrumentation. Also instrument the MPI library. Use Intel compilers.
- 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.,
  - /opt/tau-2.6/x86_64/lib/Makefile.tau-icpc-mpi-pdt
  - /opt/tau-2.16/x86_64/lib/Makefile.tau-icpc-mpi-pdt-trace

TAU Measurement Configuration – Examples

```
% cd $(TAU)/x86_64/lib; ls Makefile.*pgi
Makefile.tau-pdt-pgi
Makefile.tau-mpi-pdt-pgi
Makefile.tau-callpath-mpi-pdt-pgi
Makefile.tau-mpi-pdt-trace-pgi
Makefile.tau-mpi-compensate-pdt-pgi
Makefile.tau-pthread-pdt-pgi
Makefile.tau-papiwallclock-multiplecounters-papivirtual-mpi-papi-pdt-pgi
Makefile.tau-multiplecounters-mpi-papi-pdt-trace-pgi
Makefile.tau-mpi-pdt-epilog-trace-pgi
Makefile.tau-papiwallclock-multiplecounters-papivirtual-papi-pdt-openmp-opari-pgi
…
```

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

Makefile.tau-mpi-pdt-pgi

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

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
- Instrument application manually/automatically
  – TAU Profiling API
- Typically modify application makefile
  – Select TAU's stub makefile, change name of compiler in Makefile
- Set environment variables
  – TAU_MAKEFILE stub makefile
  – directory where profiles/traces are to be stored
- Execute application
  % mpirun –np <procs> a.out;
- Analyze performance data
  – paraprof, vampir, pprof, paraver …

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 tau_load.sh

```
% mpirun -np 4 tau_load.sh a.out
```

- Requires shared library MPI
- Approach will work with other shared libraries

Instrumenting MPI Applications

- Under Linux you may use tau_load.sh to launch un-instrumented programs under TAU (needs shared MPI objects)
  - Without TAU:

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

  - With TAU:

```
% ls $(TAU)/x86_64/lib/libTAU*pgi*
% mpirun -np 4 tau_load.sh ./a.out
% mpirun -np 4 tau_load.sh -XrunTAUsh-mpi-pdt-trace-pgi.so a.out
loads <taudir>/<arch>/lib/libTAUsh-mpi-pdt-trace-pgi.so shared object
```

- Under AIX, use tau_poe instead of poe
  - Without TAU:

```
% poe a.out -procs 8
```

  - With TAU:

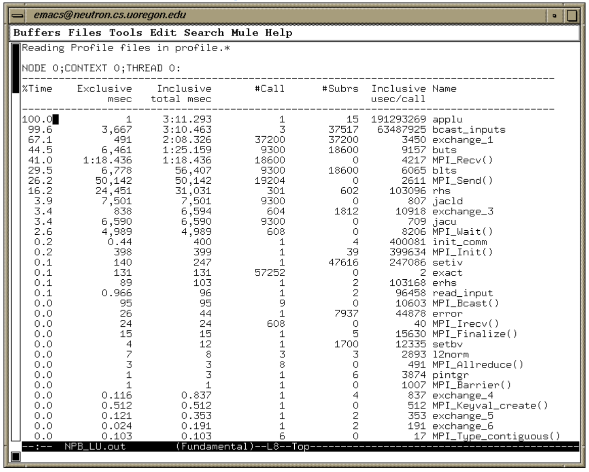
```
% tau_poe a.out -procs 8
% tau_poe -XrunTAUsh-mpi-pdt-trace.so a.out -procs 8
chooses <taudir>/<arch>/lib/libTAUsh-mpi-pdt-trace.so
```

- No change to source code or executables! No need to re-link!
- Only instruments MPI routines. To instrument user routines, you may need to parse the application source code!

-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


emacs@neutron.cs.uoregon.edu
Buffers Files Tools Edit Search Mule Help

```
Reading Profile files in profile.*

NODE 0;CONTEXT 0;THREAD 0:
---------------------------------------------------------------------------------------
%Time    Exclusive    Inclusive       #Call      #Subrs  Inclusive Name
              msec   total msec                          usec/call
---------------------------------------------------------------------------------------
100.0            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-------------------------------------------
```

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

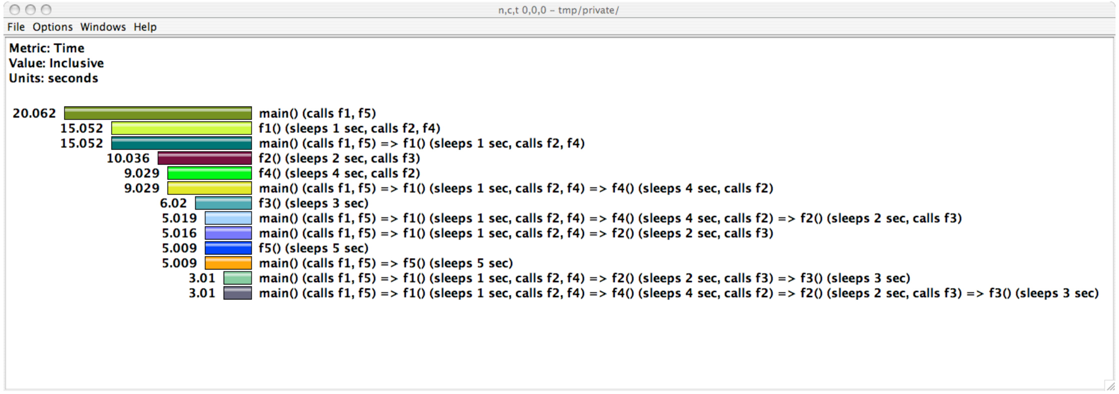
```
int main( )
{ /* takes 100 secs */

  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_OPS. */
```

-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 …
- 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 and papi_avail are two useful tools

-PROFILECALLPATH Configuration Option

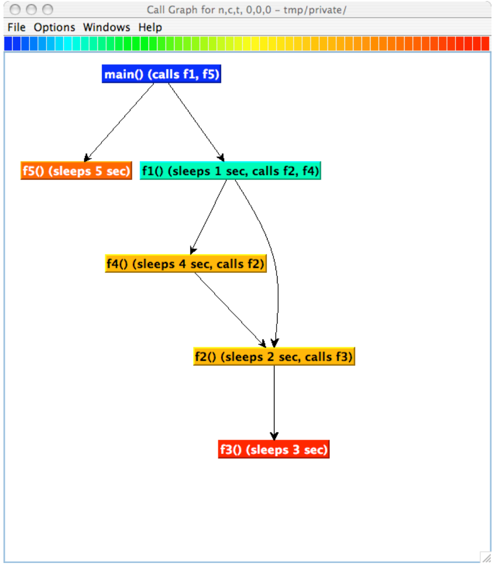
- Generates profiles that show the calling order (edges & nodes in callgraph)
  - A=>B=>C shows the time spent in C when it was called by B and B was called by A
  - Control the depth of callpath using TAU_CALLPATH_DEPTH env. Variable
  - -callpath in the name of the stub Makefile name



```
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


Call Graph for n,c,t, 0,0,0 – tmp/private/
File Options Windows Help
main() (calls f1, f5)
f5() (sleeps 5 sec)
f1() (sleeps 1 sec, calls f2, f4)
f4() (sleeps 4 sec, calls f2)
f2() (sleeps 2 sec, calls f3)
f3() (sleeps 3 sec)

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-icpc-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

-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-icpc-mpi-pdt-trace

-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

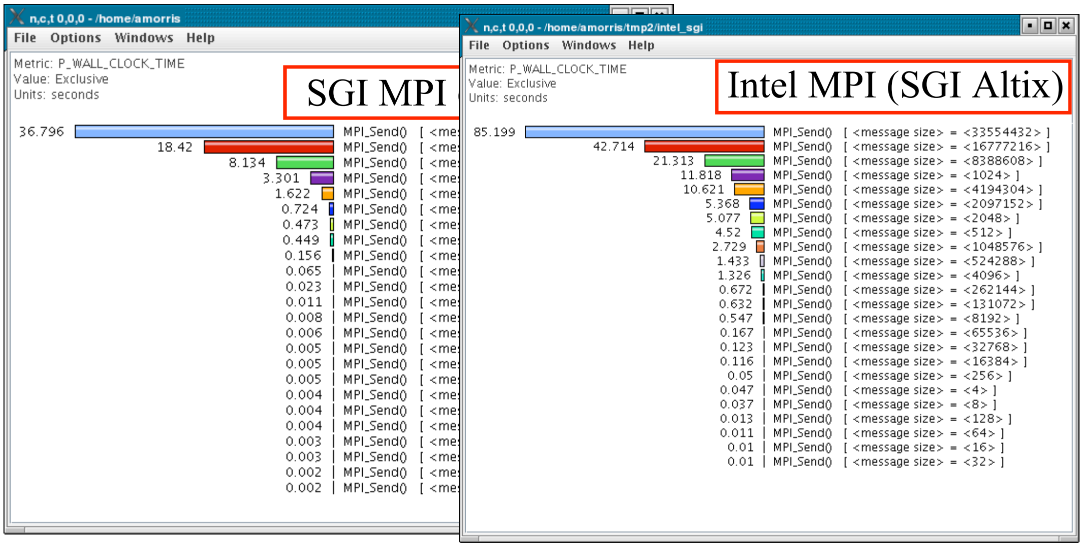
- 5 seconds spent in function "`foo`" becomes
  - 2 seconds for "`foo [ <input> = <25> ]`"
  - 1 seconds for "`foo [ <input> = <5> ]`"
  - …
- Currently used in MPI wrapper library
  - Allows for partitioning of time spent in MPI routines based on parameters (message size, message tag, destination node)
  - 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 tau_load.sh -XrunTAU-icpc-mpi-pdt.so a.out
```



SGI MPI (

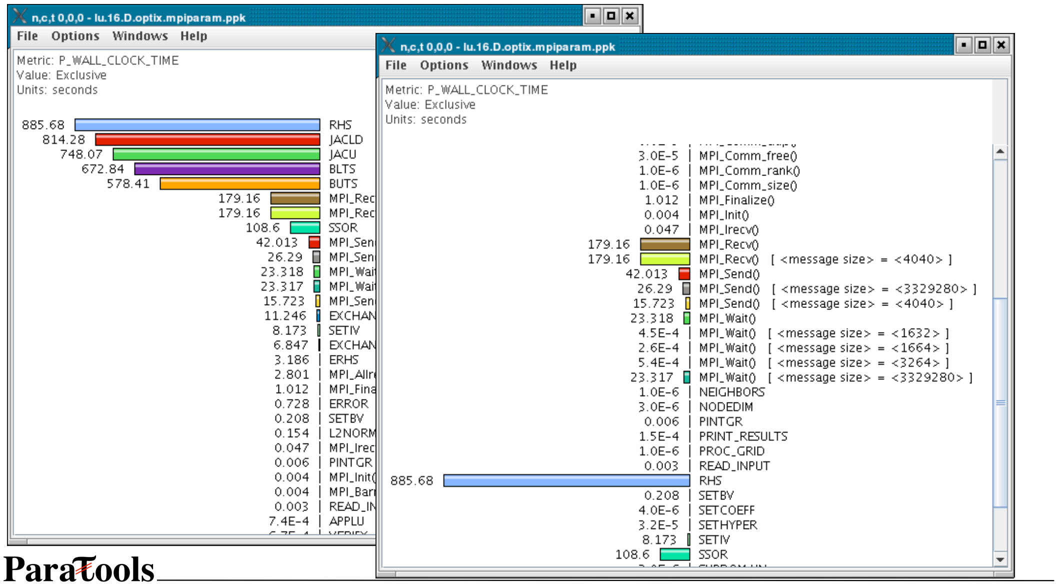
```
n,c,t 0,0,0 - /home/amorris
Metric: P_WALL_CLOCK_TIME
Value: Exclusive
Units: seconds

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
```

Intel MPI (SGI Altix)

```
n,c,t 0,0,0 - /home/amorris/tmp2/intel_sgi
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

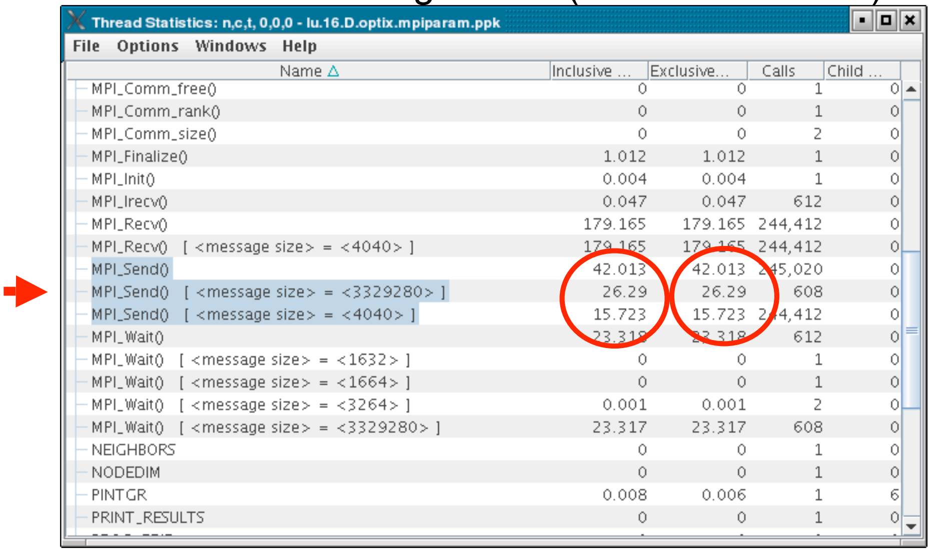


```
n,c,t 0,0,0 - lu.16.D.optix.mpiparam.ppk
Metric: P_WALL_CLOCK_TIME
Value: Exclusive
Units: seconds

885.68  RHS
814.28  JACLD
748.07  JACU
672.84  BLTS
578.41  BUTS
179.16  MPI_Rec
179.16  MPI_Rec
 108.6  SSOR
42.013  MPI_Sen
 26.29  MPI_Sen
23.318  MPI_Wai
23.317  MPI_Wai
15.723  MPI_Sen
11.246  EXCHAN
 8.173  SETIV
 6.847  EXCHAN
 3.186  ERHS
 2.801  MPI_Allr
 1.012  MPI_Fina
 0.728  ERROR
 0.208  SETBV
 0.154  L2NORM
 0.047  MPI_Irec
 0.006  PINTGR
 0.004  MPI_Init(
 0.004  MPI_Barr
 0.003  READ_IN
7.4E-4  APPLU
```

```
n,c,t 0,0,0 - lu.16.D.optix.mpiparam.ppk
Metric: P_WALL_CLOCK_TIME
Value: Exclusive
Units: seconds

 3.0E-5  MPI_Comm_free()
 1.0E-6  MPI_Comm_rank()
 1.0E-6  MPI_Comm_size()
  1.012  MPI_Finalize()
  0.004  MPI_Init()
  0.047  MPI_Irecv()
 179.16  MPI_Recv()
 179.16  MPI_Recv()  [ <message size> = <4040> ]
 42.013  MPI_Send()
  26.29  MPI_Send()  [ <message size> = <3329280> ]
 15.723  MPI_Send()  [ <message size> = <4040> ]
 23.318  MPI_Wait()
 4.5E-4  MPI_Wait()  [ <message size> = <1632> ]
 2.6E-4  MPI_Wait()  [ <message size> = <1664> ]
 5.4E-4  MPI_Wait()  [ <message size> = <3264> ]
 23.317  MPI_Wait()  [ <message size> = <3329280> ]
 1.0E-6  NEIGHBORS
 3.0E-6  NODEDIM
  0.006  PINTGR
 1.5E-4  PRINT_RESULTS
 1.0E-6  PROC_GRID
  0.003  READ_INPUT
 885.68  RHS
  0.208  SETBV
 4.0E-6  SETCOEFF
 3.2E-5  SETHYPER
  8.173  SETIV
  108.6  SSOR
```

Workload Characterization

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

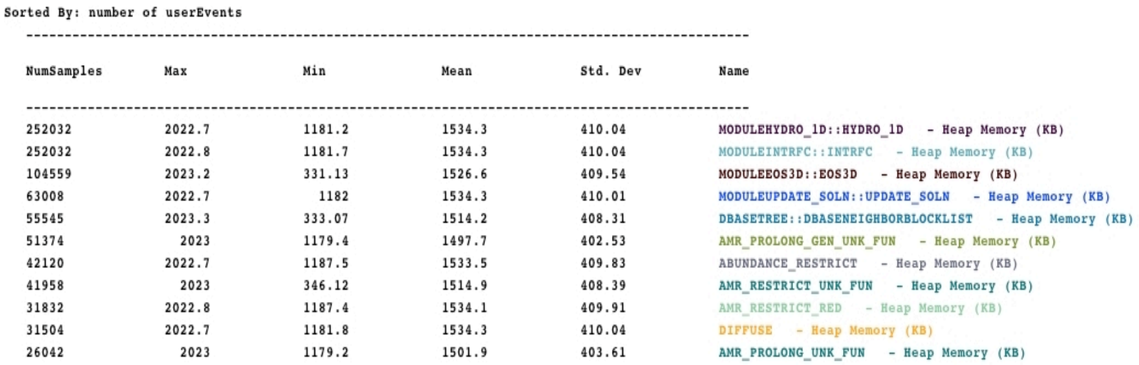

Thread Statistics: n,c,t, 0,0,0 - lu.16.D.optix.mpiparam.ppk
File
Options
Windows
Help

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

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

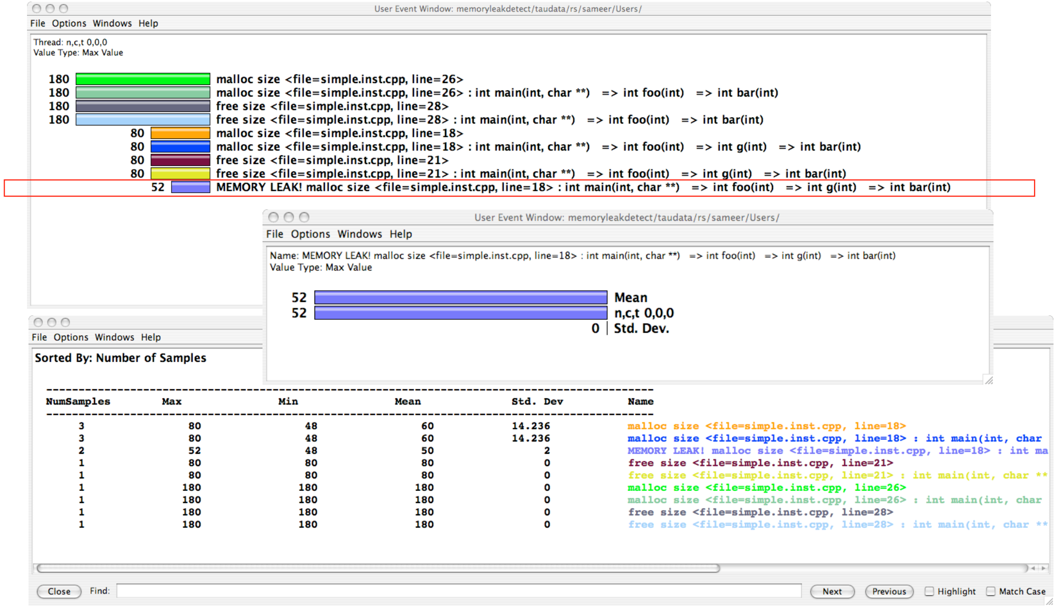
  -optDetectMemoryLeaks option to TAU_COMPILER

  ```
  % setenv TAU_OPTIONS '-optVerbose -optDetectMemoryLeaks'
  % setenv TAU_MAKEFILE <taudir>/<arch>/lib/Makefile.tau-icpc-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

Detecting Memory Leaks in C/C++

```
% setenv TAU_MAKEFILE /opt/tau/x86_64/lib/Makefile.tau-icpc-mpi-pdt
% setenv TAU_OPTIONS '-optVerbose -optDetectMemoryLeaks'
% cat Makefile
CC= tau_cc.sh
LIBS = -lm
OBJS = f1.o f2.o ...
TARGET= a.out
TARGET: $(OBJS)
       $(F90) $(LDFLAGS) $(OBJS) -o $@ $(LIBS)
.c.o:
       $(CC) $(CFLAGS) -c $< -o $@
```

Memory Leak Detection


User Event Window: memoryleakdetect/taudata/rs/sameer/Users/
File Options Windows Help

```
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)
```

User Event Window: memoryleakdetect/taudata/rs/sameer/Users/
File Options Windows Help

```
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 **
```

Labs!

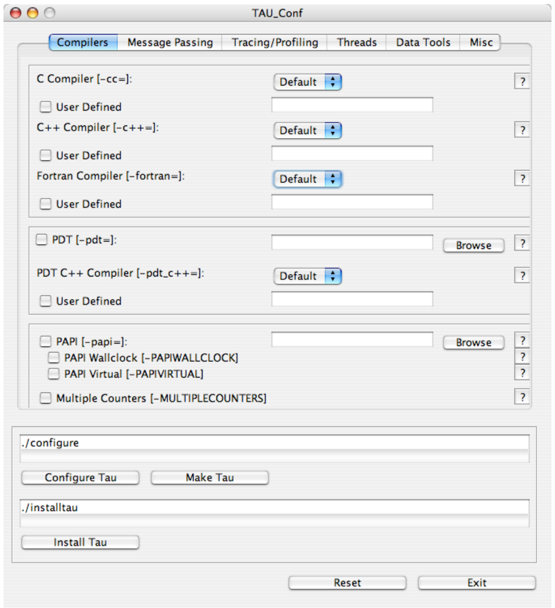
Lab: Intro to TAU, memory evaluation

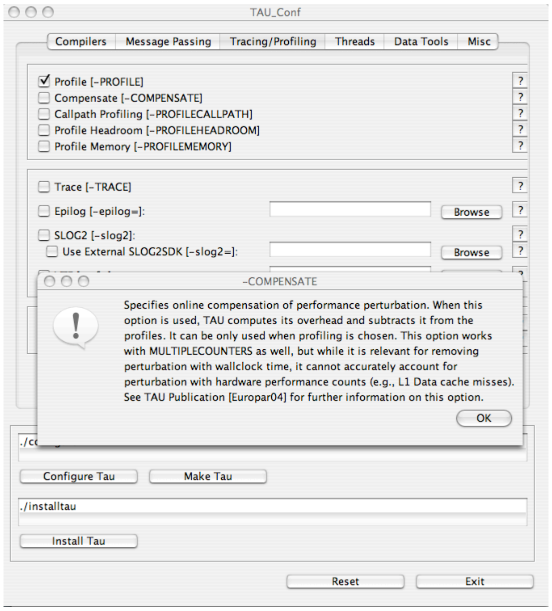
Lab Instructions

- See workshop examples matmult, memoryleakdetect, cthreads
- To profile a code:

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
   $TAU/Makefile.tau-[options]
3. If stub makefile has -multiplecounters 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 …
4. Set TAU_THROTTLE environment variable to throttle
   instrumentation:
   % setenv TAU_THROTTLE 1
5. Build and run workshop examples, then run pprof/paraprof

TAU_SETUP: A GUI for Installing TAU





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.

TAU Manual Instrumentation API for C/C++

- Initialization and runtime configuration
  - TAU_PROFILE_INIT(argc, argv);
    TAU_PROFILE_SET_NODE(myNode);
    TAU_PROFILE_SET_CONTEXT(myContext);
    TAU_PROFILE_EXIT(message);
    TAU_REGISTER_THREAD();
- Function and class methods for C++ only:
  - TAU_PROFILE(name, type, group);
  - TAU_PROFILE (name, type, group);
- Template
  - TAU_TYPE_STRING(variable, type);
    TAU_PROFILE(name, type, group);
    CT (variable);
- User-defined timing
  - TAU_PROFILE_TIMER(timer, name, type, group);
    TAU_PROFILE_START(timer);
    TAU_PROFILE_STOP(timer);

TAU Measurement API (continued)

- Defining application phases
  - TAU_PHASE_CREATE_STATIC(var, name, type, group);
  - TAU_PHASE_CREATE_DYNAMIC(var, name, type, group);
  - TAU_PHASE_START(var)
  - TAU_PHASE_STOP (var)
- User-defined events
  - TAU_REGISTER_EVENT(variable, event_name);
    TAU_EVENT(variable, value);
    TAU_PROFILE_STMT(statement);
- Heap Memory Tracking:
  - TAU_TRACK_MEMORY();
  - TAU_TRACK_MEMORY_HEADROOM();
  - TAU_SET_INTERRUPT_INTERVAL(seconds);
  - TAU_DISABLE_TRACKING_MEMORY[_HEADROOM]();
  - TAU_ENABLE_TRACKING_MEMORY[_HEADROOM]();

Manual Instrumentation – C++ Example

```
#include <TAU.h>
int main(int argc, char **argv)
{
  TAU_PROFILE(“int main(int, char **)”, “ ”, TAU_DEFAULT);
  TAU_PROFILE_INIT(argc, argv);
  TAU_PROFILE_SET_NODE(0); /* for sequential programs */
  foo();
  return 0;
}
int foo(void)
{
  TAU_PROFILE(“int foo(void)”, “ ”, TAU_DEFAULT); // measures entire foo()
   TAU_PROFILE_TIMER(t, “foo(): for loop”, “[23:45 file.cpp]”, TAU_USER);
  TAU_PROFILE_START(t);
  for(int i = 0; i < N ; i++){
    work(i);
  }
  TAU_PROFILE_STOP(t);
  // other statements in foo …
}
```

Manual Instrumentation – F90 Example

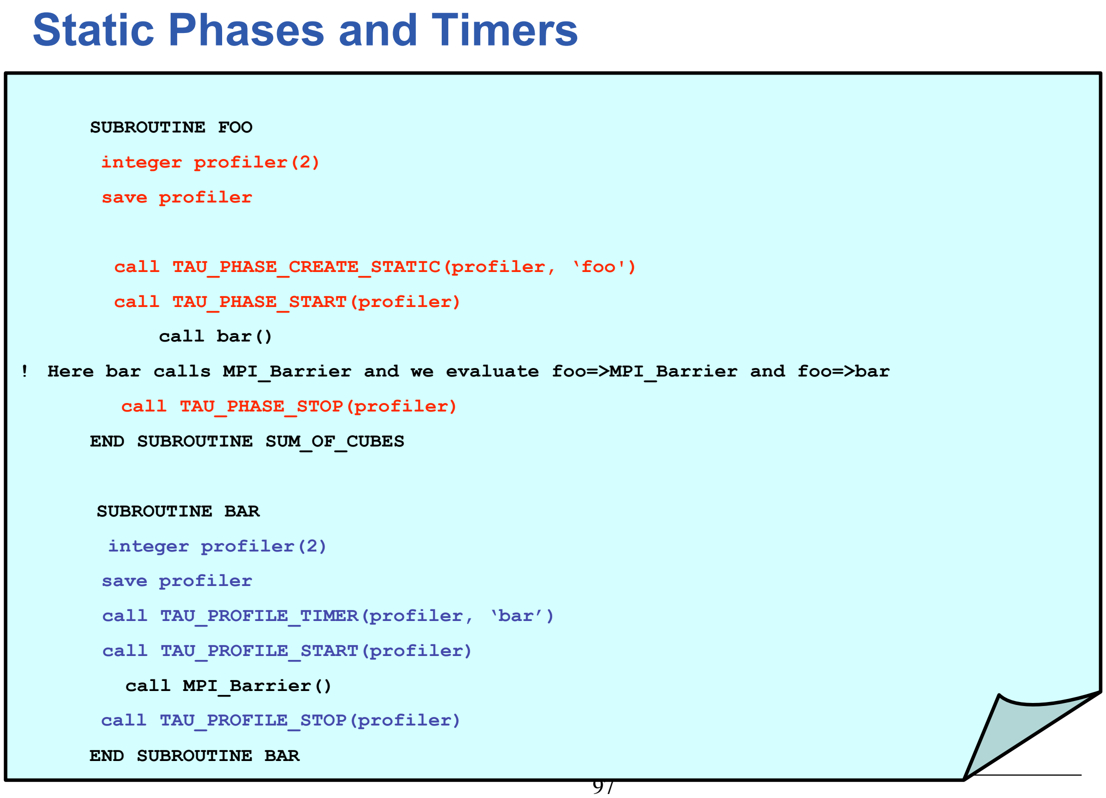
```
cc34567 Cubes program – comment line
      PROGRAM SUM_OF_CUBES
       integer profiler(2)
       save profiler
      INTEGER :: H, T, U
        call TAU_PROFILE_INIT()
        call TAU_PROFILE_TIMER(profiler, 'PROGRAM SUM_OF_CUBES')
        call TAU_PROFILE_START(profiler)
        call TAU_PROFILE_SET_NODE(0)
! This program prints all 3-digit numbers that equal the sum of the cubes of their digits.
      DO H = 1, 9
        DO T = 0, 9
          DO U = 0, 9
          IF (100*H + 10*T + U == H**3 + T**3 + U**3) THEN
             PRINT "(3I1)", H, T, U
          ENDIF
          END DO
        END DO
      END DO
      call TAU_PROFILE_STOP(profiler)
      END PROGRAM SUM_OF_CUBES
```

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.

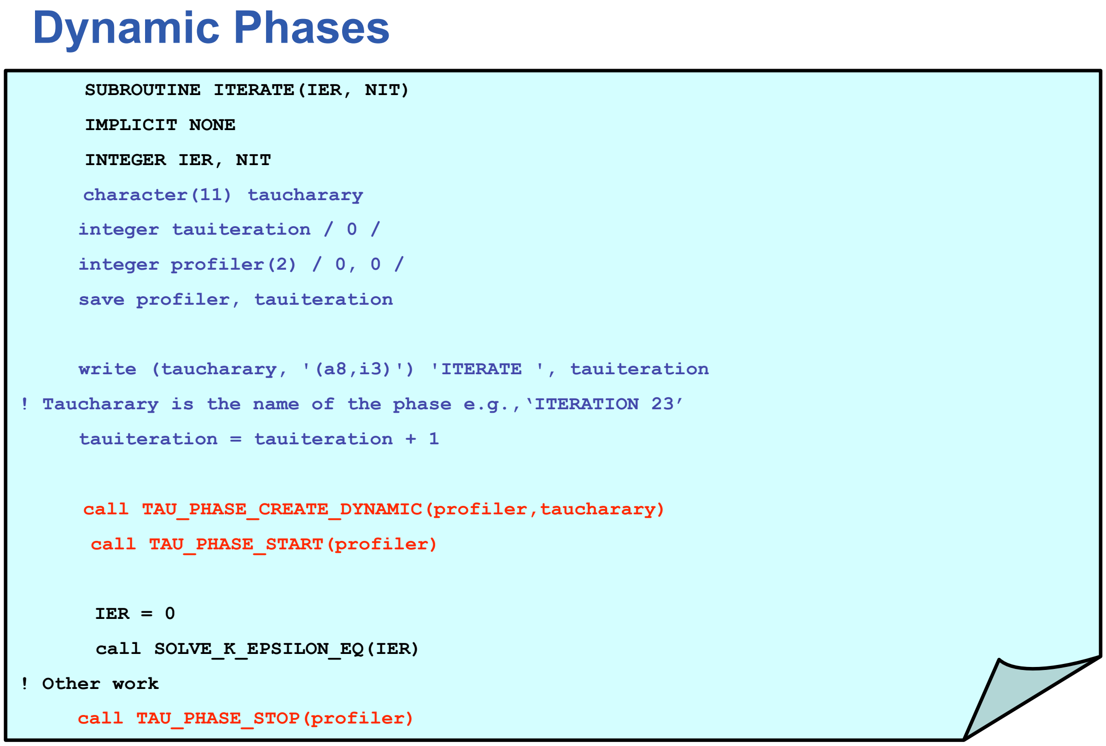
Static Timers in TAU

```
SUBROUTINE SUM_OF_CUBES
 integer profiler(2)
 save profiler
INTEGER :: H, T, U

  call TAU_PROFILE_TIMER(profiler, 'SUM_OF_CUBES')
  call TAU_PROFILE_START(profiler)
! This program prints all 3-digit numbers that
! equal the sum of the cubes of their digits.
DO H = 1, 9
  DO T = 0, 9
    DO U = 0, 9
    IF (100*H + 10*T + U == H**3 + T**3 + U**3) THEN
       PRINT "(3I1)", H, T, U
    ENDIF
    END DO
  END DO
END DO
call TAU_PROFILE_STOP(profiler)
END SUBROUTINE SUM_OF_CUBES
```



Static Phases and Timers

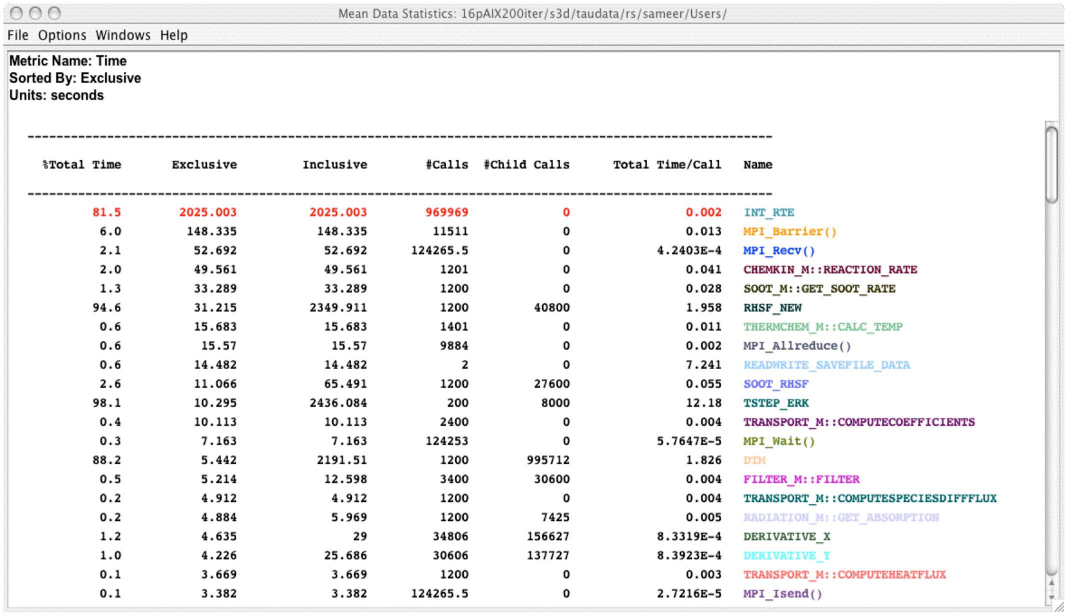
```
      SUBROUTINE FOO
       integer profiler(2)
       save profiler

        call TAU_PHASE_CREATE_STATIC(profiler, ‘foo')
        call TAU_PHASE_START(profiler)
            call bar()
!  Here bar calls MPI_Barrier and we evaluate foo=>MPI_Barrier and foo=>bar
         call TAU_PHASE_STOP(profiler)
      END SUBROUTINE SUM_OF_CUBES

       SUBROUTINE BAR
        integer profiler(2)
       save profiler
       call TAU_PROFILE_TIMER(profiler, ‘bar’)
       call TAU_PROFILE_START(profiler)
         call MPI_Barrier()
       call TAU_PROFILE_STOP(profiler)
      END SUBROUTINE BAR
```



Dynamic Phases

```
      SUBROUTINE ITERATE(IER, NIT)
      IMPLICIT NONE
      INTEGER IER, NIT
      character(11) taucharary
     integer tauiteration / 0 /
     integer profiler(2)  / 0, 0 /
     save profiler, tauiteration

     write (taucharary, '(a8,i3)') 'ITERATE ', tauiteration
! Taucharary is the name of the phase e.g.,‘ITERATION 23’
     tauiteration = tauiteration + 1

      call TAU_PHASE_CREATE_DYNAMIC(profiler,taucharary)
       call TAU_PHASE_START(profiler)

       IER = 0
       call SOLVE_K_EPSILON_EQ(IER)
! Other work
     call TAU_PHASE_STOP(profiler)
```

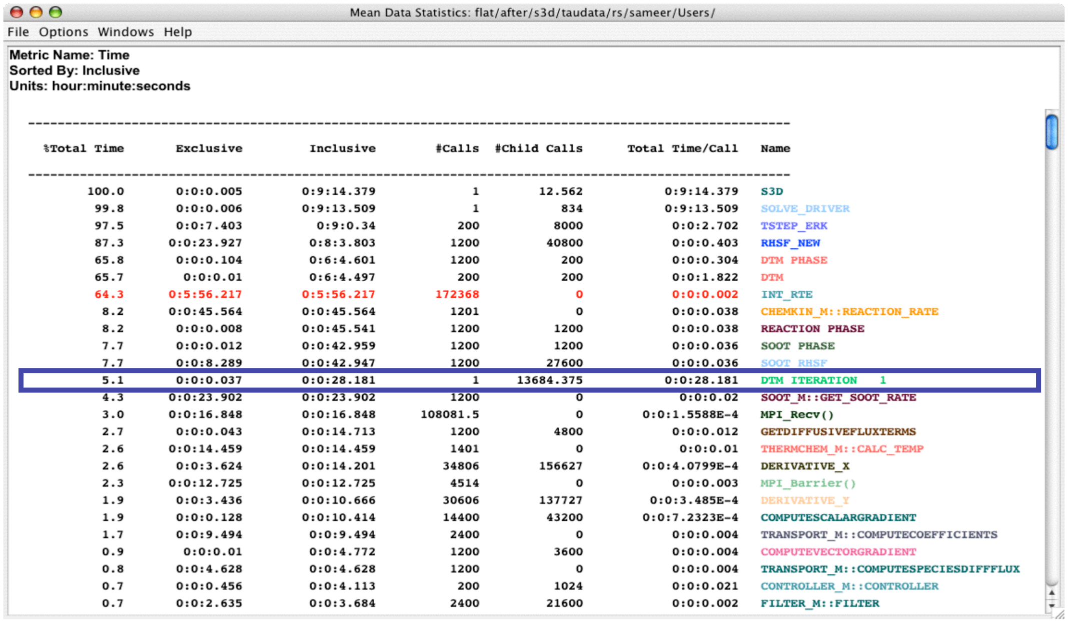
TAU’s ParaProf Profile Browser: Static Timers


Mean Data Statistics: 16pAIX200iter/s3d/taudata/rs/sameer/Users/
File Options Windows Help

Metric Name: Time
Sorted By: Exclusive
Units: seconds

```
---------------------------------------------------------------------------------------------------

 %Total Time       Exclusive         Inclusive       #Calls  #Child Calls     Total Time/Call   Name

---------------------------------------------------------------------------------------------------
       81.5         2025.003          2025.003       969969             0               0.002   INT_RTE
        6.0          148.335           148.335        11511             0               0.013   MPI_Barrier()
        2.1           52.692            52.692     124265.5             0           4.2403E-4   MPI_Recv()
        2.0           49.561            49.561         1201             0               0.041   CHEMKIN_M::REACTION_RATE
        1.3           33.289            33.289         1200             0               0.028   SOOT_M::GET_SOOT_RATE
       94.6           31.215          2349.911         1200         40800               1.958   RHSF_NEW
        0.6           15.683            15.683         1401             0               0.011   THERMCHEM_M::CALC_TEMP
        0.6            15.57             15.57         9884             0               0.002   MPI_Allreduce()
        0.6           14.482            14.482            2             0               7.241   READWRITE_SAVEFILE_DATA
        2.6           11.066            65.491         1200         27600               0.055   SOOT_RHSF
       98.1           10.295          2436.084          200          8000               12.18   TSTEP_ERK
        0.4           10.113            10.113         2400             0               0.004   TRANSPORT_M::COMPUTECOEFFICIENTS
        0.3            7.163             7.163       124253             0           5.7647E-5   MPI_Wait()
       88.2            5.442           2191.51         1200        995712               1.826   DTM
        0.5            5.214            12.598         3400         30600               0.004   FILTER_M::FILTER
        0.2            4.912             4.912         1200             0               0.004   TRANSPORT_M::COMPUTESPECIESDIFFFLUX
        0.2            4.884             5.969         1200          7425               0.005   RADIATION_M::GET_ABSORPTION
        1.2            4.635                29        34806        156627           8.3319E-4   DERIVATIVE_X
        1.0            4.226            25.686        30606        137727           8.3923E-4   DERIVATIVE_Y
        0.1            3.669             3.669         1200             0               0.003   TRANSPORT_M::COMPUTEHEATFLUX
        0.1            3.382             3.382     124265.5             0           2.7216E-5   MPI_Isend()
```

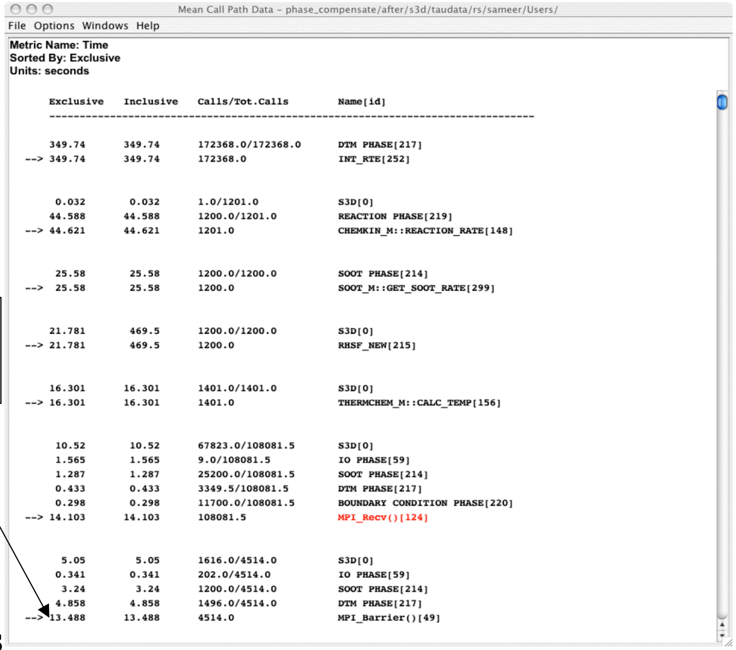
Dynamic Timers


Mean Data Statistics: flat/after/s3d/taudata/rs/sameer/Users/
File Options Windows Help

Metric Name: Time
Sorted By: Inclusive
Units: hour:minute:seconds

```
----------------------------------------------------------------------------------------------------

 %Total Time       Exclusive          Inclusive      #Calls  #Child Calls    Total Time/Call   Name

----------------------------------------------------------------------------------------------------
      100.0      0:0:0.005         0:9:14.379           1        12.562        0:9:14.379   S3D
       99.8      0:0:0.006         0:9:13.509           1           834        0:9:13.509   SOLVE_DRIVER
       97.5      0:0:7.403           0:9:0.34         200          8000         0:0:2.702   TSTEP_ERK
       87.3     0:0:23.927          0:8:3.803        1200         40800         0:0:0.403   RHSF_NEW
       65.8      0:0:0.104          0:6:4.601        1200           200         0:0:0.304   DTM PHASE
       65.7       0:0:0.01          0:6:4.497         200           200         0:0:1.822   DTM
       64.3     0:5:56.217         0:5:56.217      172368             0         0:0:0.002   INT_RTE
        8.2     0:0:45.564         0:0:45.564        1201             0         0:0:0.038   CHEMKIN_M::REACTION_RATE
        8.2      0:0:0.008         0:0:45.541        1200          1200         0:0:0.038   REACTION PHASE
        7.7      0:0:0.012         0:0:42.959        1200          1200         0:0:0.036   SOOT PHASE
        7.7      0:0:8.289         0:0:42.947        1200         27600         0:0:0.036   SOOT RHSF
        5.1      0:0:0.037         0:0:28.181           1     13684.375        0:0:28.181   DTM ITERATION    1
        4.3     0:0:23.902         0:0:23.902        1200             0          0:0:0.02   SOOT_M::GET_SOOT_RATE
        3.0     0:0:16.848         0:0:16.848    108081.5             0     0:0:1.5588E-4   MPI_Recv()
        2.7      0:0:0.043         0:0:14.713        1200          4800         0:0:0.012   GETDIFFUSIVEFLUXTERMS
        2.6     0:0:14.459         0:0:14.459        1401             0          0:0:0.01   THERMCHEM_M::CALC_TEMP
        2.6      0:0:3.624         0:0:14.201       34806        156627     0:0:4.0799E-4   DERIVATIVE_X
        2.3     0:0:12.725         0:0:12.725        4514             0         0:0:0.003   MPI_Barrier()
        1.9      0:0:3.436         0:0:10.666       30606        137727      0:0:3.485E-4   DERIVATIVE_Y
        1.9      0:0:0.128         0:0:10.414       14400         43200     0:0:7.2323E-4   COMPUTESCALARGRADIENT
        1.7      0:0:9.494          0:0:9.494        2400             0         0:0:0.004   TRANSPORT_M::COMPUTECOEFFICIENTS
        0.9       0:0:0.01          0:0:4.772        1200          3600         0:0:0.004   COMPUTEVECTORGRADIENT
        0.8      0:0:4.628          0:0:4.628        1200             0         0:0:0.004   TRANSPORT_M::COMPUTESPECIESDIFFFLUX
        0.7      0:0:0.456          0:0:4.113         200          1024         0:0:0.021   CONTROLLER_M::CONTROLLER
        0.7      0:0:2.635          0:0:3.684        2400         21600         0:0:0.002   FILTER_M::FILTER
```

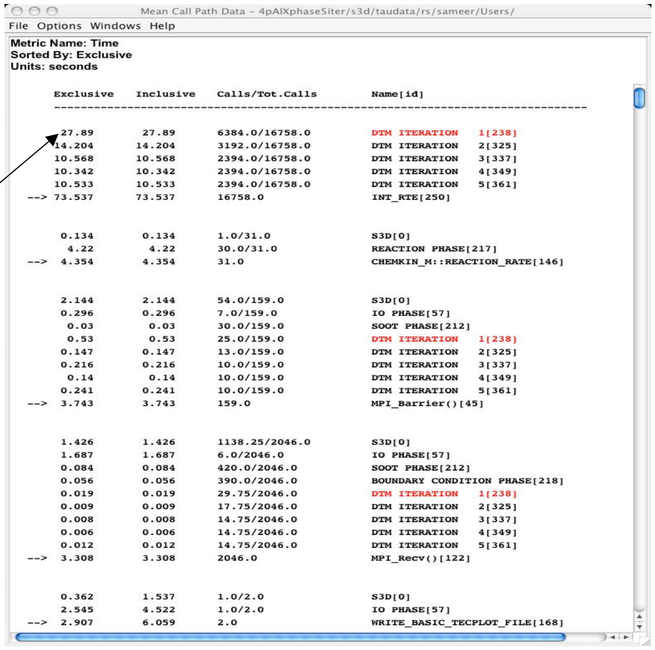
Static Phases


Mean Call Path Data – phase_compensate/after/s3d/taudata/rs/sameer/Users/
File Options Windows Help

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

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

    349.74      349.74      172368.0/172368.0      DTM PHASE[217]
--> 349.74      349.74      172368.0               INT_RTE[252]


     0.032       0.032      1.0/1201.0             S3D[0]
    44.588      44.588      1200.0/1201.0          REACTION PHASE[219]
--> 44.621      44.621      1201.0                 CHEMKIN_M::REACTION_RATE[148]


     25.58       25.58      1200.0/1200.0          SOOT PHASE[214]
-->  25.58       25.58      1200.0                 SOOT_M::GET_SOOT_RATE[299]


    21.781       469.5      1200.0/1200.0          S3D[0]
--> 21.781       469.5      1200.0                 RHSF_NEW[215]


    16.301      16.301      1401.0/1401.0          S3D[0]
--> 16.301      16.301      1401.0                 THERMCHEM_M::CALC_TEMP[156]


     10.52       10.52      67823.0/108081.5       S3D[0]
     1.565       1.565      9.0/108081.5           IO PHASE[59]
     1.287       1.287      25200.0/108081.5       SOOT PHASE[214]
     0.433       0.433      3349.5/108081.5        DTM PHASE[217]
     0.298       0.298      11700.0/108081.5       BOUNDARY CONDITION PHASE[220]
--> 14.103      14.103      108081.5               MPI_Recv()[124]


      5.05        5.05      1616.0/4514.0          S3D[0]
     0.341       0.341      202.0/4514.0           IO PHASE[59]
      3.24        3.24      1200.0/4514.0          SOOT PHASE[214]
     4.858       4.858      1496.0/4514.0          DTM PHASE[217]
--> 13.488      13.488      4514.0                 MPI_Barrier()[49]
```

MPI_Barrier took 4.85 secs out of 13.48 secs in the DTM Phase

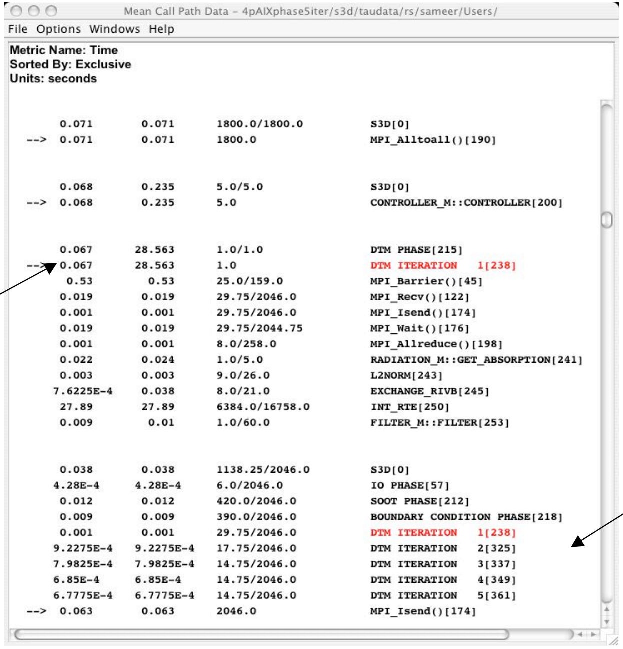
Dynamic Phases


Mean Call Path Data – 4pAIXphase5iter/s3d/taudata/rs/sameer/Users/
File Options Windows Help

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

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

     27.89       27.89      6384.0/16758.0         DTM ITERATION   1[238]
    14.204      14.204      3192.0/16758.0         DTM ITERATION   2[325]
    10.568      10.568      2394.0/16758.0         DTM ITERATION   3[337]
    10.342      10.342      2394.0/16758.0         DTM ITERATION   4[349]
    10.533      10.533      2394.0/16758.0         DTM ITERATION   5[361]
--> 73.537      73.537      16758.0                INT_RTE[250]


     0.134       0.134      1.0/31.0               S3D[0]
      4.22        4.22      30.0/31.0              REACTION PHASE[217]
-->  4.354       4.354      31.0                   CHEMKIN_M::REACTION_RATE[146]


     2.144       2.144      54.0/159.0             S3D[0]
     0.296       0.296      7.0/159.0              IO PHASE[57]
      0.03        0.03      30.0/159.0             SOOT PHASE[212]
      0.53        0.53      25.0/159.0             DTM ITERATION   1[238]
     0.147       0.147      13.0/159.0             DTM ITERATION   2[325]
     0.216       0.216      10.0/159.0             DTM ITERATION   3[337]
      0.14        0.14      10.0/159.0             DTM ITERATION   4[349]
     0.241       0.241      10.0/159.0             DTM ITERATION   5[361]
-->  3.743       3.743      159.0                  MPI_Barrier()[45]


     1.426       1.426      1138.25/2046.0         S3D[0]
     1.687       1.687      6.0/2046.0             IO PHASE[57]
     0.084       0.084      420.0/2046.0           SOOT PHASE[212]
     0.056       0.056      390.0/2046.0           BOUNDARY CONDITION PHASE[218]
     0.019       0.019      29.75/2046.0           DTM ITERATION   1[238]
     0.009       0.009      17.75/2046.0           DTM ITERATION   2[325]
     0.008       0.008      14.75/2046.0           DTM ITERATION   3[337]
     0.006       0.006      14.75/2046.0           DTM ITERATION   4[349]
     0.012       0.012      14.75/2046.0           DTM ITERATION   5[361]
-->  3.308       3.308      2046.0                 MPI_Recv()[122]


     0.362       1.537      1.0/2.0                S3D[0]
     2.545       4.522      1.0/2.0                IO PHASE[57]
-->  2.907       6.059      2.0                    WRITE_BASIC_TECPLOT_FILE[168]
```

The first iteration was expensive for INT_RTE. It took 27.89 secs. Other iterations took less time – 14.2, 10.5, 10.3, 10.5 seconds

Dynamic Phases


Mean Call Path Data – 4pAIXphase5iter/s3d/taudata/rs/sameer/Users/
File Options Windows Help

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


      0.071       0.071      1800.0/1800.0           S3D[0]
 -->  0.071       0.071      1800.0                  MPI_Alltoall()[190]


      0.068       0.235      5.0/5.0                 S3D[0]
 -->  0.068       0.235      5.0                     CONTROLLER_M::CONTROLLER[200]


      0.067      28.563      1.0/1.0                 DTM PHASE[215]
 -->  0.067      28.563      1.0                     DTM ITERATION   1[238]
       0.53        0.53      25.0/159.0              MPI_Barrier()[45]
      0.019       0.019      29.75/2046.0            MPI_Recv()[122]
      0.001       0.001      29.75/2046.0            MPI_Isend()[174]
      0.019       0.019      29.75/2044.75           MPI_Wait()[176]
      0.001       0.001      8.0/258.0               MPI_Allreduce()[198]
      0.022       0.024      1.0/5.0                 RADIATION_M::GET_ABSORPTION[241]
      0.003       0.003      9.0/26.0                L2NORM[243]
    7.6225E-4     0.038      8.0/21.0                EXCHANGE_RIVB[245]
      27.89       27.89      6384.0/16758.0          INT_RTE[250]
      0.009        0.01      1.0/60.0                FILTER_M::FILTER[253]


      0.038       0.038      1138.25/2046.0          S3D[0]
    4.28E-4     4.28E-4      6.0/2046.0              IO PHASE[57]
      0.012       0.012      420.0/2046.0            SOOT PHASE[212]
      0.009       0.009      390.0/2046.0            BOUNDARY CONDITION PHASE[218]
      0.001       0.001      29.75/2046.0            DTM ITERATION   1[238]
    9.2275E-4   9.2275E-4    17.75/2046.0            DTM ITERATION   2[325]
    7.9825E-4   7.9825E-4    14.75/2046.0            DTM ITERATION   3[337]
    6.85E-4     6.85E-4      14.75/2046.0            DTM ITERATION   4[349]
    6.7775E-4   6.7775E-4    14.75/2046.0            DTM ITERATION   5[361]
 -->  0.063       0.063      2046.0                  MPI_Isend()[174]
```

Time spent in MPI_Barrier, MPI_Recv,… in DTM ITERATION 1

Breakdown of time spent in MPI_Isend based on its static and dynamic parent phases

Using TAU – A tutorial

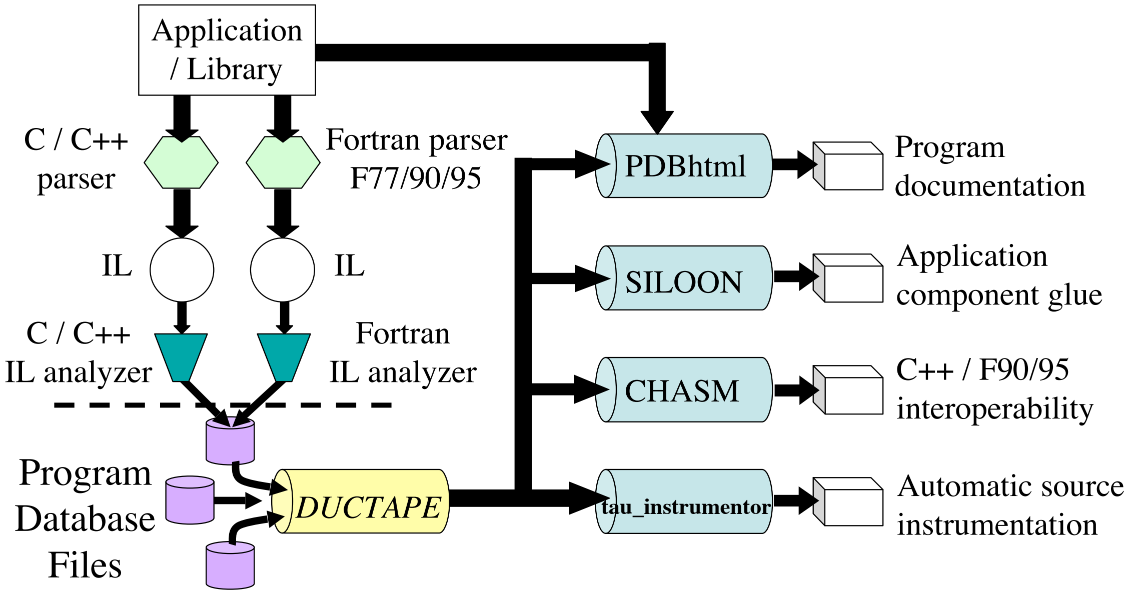
- Configuration
- Instrumentation
  - Manual
  - MPI – Wrapper interposition library
  - PDT- Source rewriting for C,C++, F77/90/95
  - OpenMP – Directive rewriting
  - Component based instrumentation – Proxy components
  - Binary Instrumentation
    - DyninstAPI – Runtime Instrumentation/Rewriting binary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

TAU’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

Program Database Toolkit (PDT)



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

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 $<
```

AutoInstrumentation using TAU_COMPILER

- $(TAU_COMPILER) stub Makefile variable
- Invokes PDT parser, TAU instrumentor, compiler through `tau_compiler.sh` shell script
- Requires minimal changes to application Makefile
  - Compilation rules are not changed
  - User adds $(TAU_COMPILER) before compiler name
    - F90=mpxlf90
Changes to
F90= $(TAU_COMPILER) mpxlf90
- Passes options from TAU stub Makefile to the four compilation stages
- Use tau_cxx.sh, tau_cc.sh, tau_f90.sh scripts OR $(TAU_COMPILER)
- Uses original compilation command if an error occurs

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 – Improving Integration in Makefiles Alternative to using tau_f90.sh/tau_cxx.sh

```
include /usr/tau-2.16/x86_64/Makefile.tau-icpc-mpi-pdt
CXX = $(TAU_COMPILER) mpicxx
F90 = $(TAU_COMPILER) mpif90
CFLAGS =
LIBS = -lm
OBJS = f1.o f2.o f3.o … fn.o

app: $(OBJS)
     $(CXX) $(LDFLAGS) $(OBJS) -o $@ $(LIBS)
.cpp.o:
     $(CXX) $(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)
  % mpxlf90 –c foo.f90 $(OPT3)
  ```
- Linking:

  ```
  % mpxlf90 foo.o bar.o –o app
  ```

  Changes to

  ```
  % mpxlf90 foo.o bar.o –o app $(OPT4)
  ```
- Where options OPT[1-4] default values may be overridden by the user:

  ```
   F90 = $(TAU_COMPILER) $(MYOPTIONS) mpxlf90
  ```

TAU_COMPILER Options

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

-optVerbose	Turn on verbose debugging messages
-optDetectMemoryLeaks	Turn on debugging memory allocations/ de-allocations to track leaks
-optPdtGnuFortranParser	Use gfparse (GNU) instead of f95parse (Cleanscape) for parsing Fortran source code
-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)

...

Overriding Default Options:TAU_COMPILER

```
include $(TAU)/x86_64/lib/
                              Makefile.tau-icpc-mpi-pdt-trace
# Fortran .f files in free format need the -R free option for parsing
# Are there any preprocessor directives in the Fortran source?
MYOPTIONS= -optVerbose -optPreProcess -optPdtF95Opts=’’-R free’’
F90 = $(TAU_COMPILER) $(MYOPTIONS) ifort
OBJS = f1.o f2.o f3.o …
LIBS = -Lappdir -lapplib1 -lapplib2 …

app: $(OBJS)
     $(F90) $(OBJS) -o app $(LIBS)
.f.o:
     $(F90) -c $<
```

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-icpc-mpi-pdt
```

Optimization of Program Instrumentation

- Need to eliminate instrumentation in frequently executing lightweight routines
- Throttling of events at runtime:

  ```
  % setenv TAU_THROTTLE 1
  ```

  Turns off instrumentation in routines that execute over 10000 times (TAU_THROTTLE_NUMCALLS) and take less than 10 microseconds of inclusive time per call (TAU_THROTTLE_PERCALL)
- 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
  ```

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_LIST_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_EXCLUDE_LIST
```

- Specify a list of routines to include for instrumentation

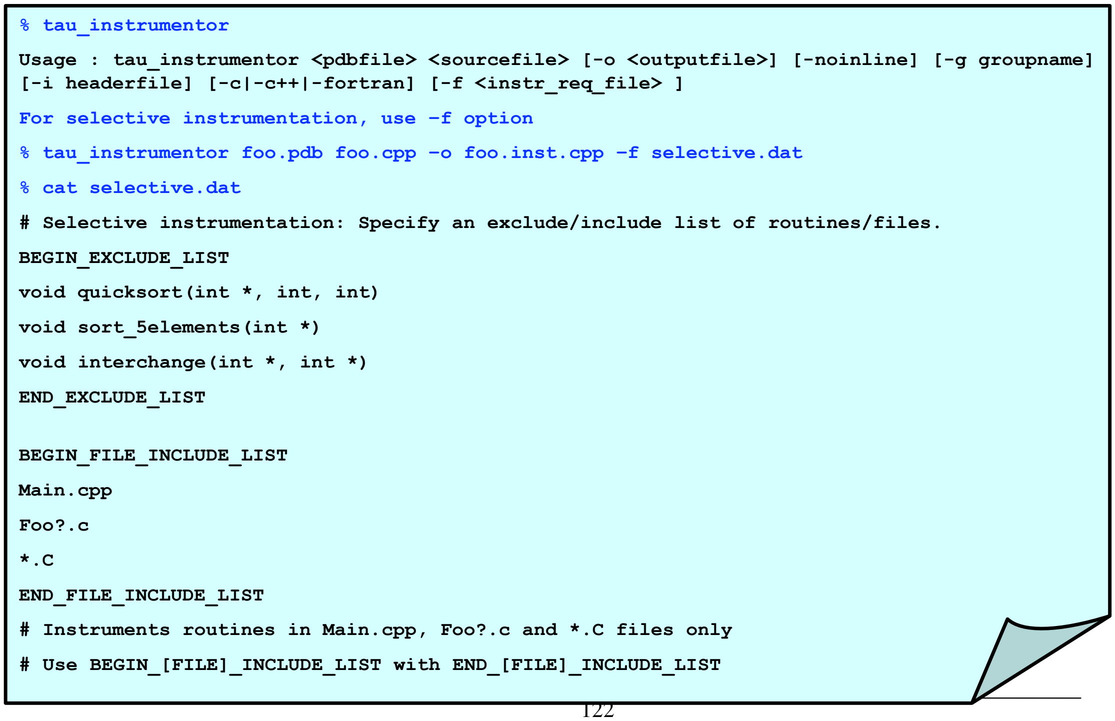
```
BEGIN_FILE_INCLUDE_LIST
main.cpp
foo.f90
END_INCLUDE_LIST_LIST
```

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#”
file=“foo.f90” line = 123 code = "  print *, \" Inside foo\""
exit routine = “int foo()” code = "cout <<\"exiting foo\"<<endl;"
END_INSTRUMENT_SECTION
```

Instrumentation Specification



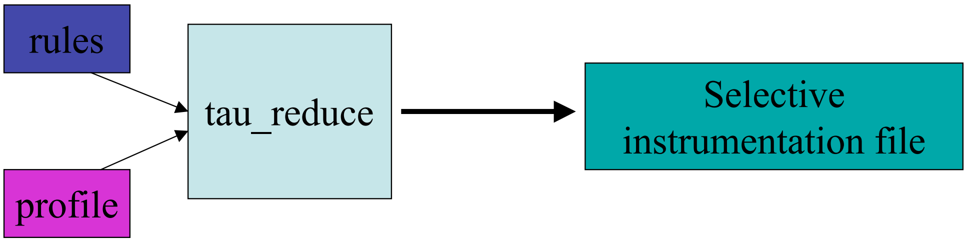
```
% tau_instrumentor
Usage : tau_instrumentor <pdbfile> <sourcefile> [-o <outputfile>] [-noinline] [-g groupname]
[-i headerfile] [-c|-c++|-fortran] [-f <instr_req_file> ]
For selective instrumentation, use –f option
% tau_instrumentor foo.pdb foo.cpp –o foo.inst.cpp –f selective.dat
% cat selective.dat
# 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
```

Automatic Outer Loop Level Instrumentation

```
BEGIN_INSTRUMENT_SECTION
loops file="loop_test.cpp" routine="multiply"
# it also understands # as the wildcard in routine name
# and * and ? wildcards in file name.
# You can also specify the full
# name of the routine as is found in profile files.
#loops file="loop_test.cpp" routine="double multiply#"
END_INSTRUMENT_SECTION

% pprof
NODE 0;CONTEXT 0;THREAD 0:
---------------------------------------------------------------------------------------
%Time    Exclusive    Inclusive       #Call      #Subrs  Inclusive Name
              msec   total msec                          usec/call
---------------------------------------------------------------------------------------
100.0         0.12       25,162           1           1   25162827 int main(int, char **)
100.0        0.175       25,162           1           4   25162707 double multiply()
 90.5       22,778       22,778           1           0   22778959 Loop: double multiply()[ file =
<loop_test.cpp> line,col = <23,3> to <30,3> ]
  9.3        2,345        2,345           1           0    2345823 Loop: double multiply()[ file =
<loop_test.cpp> line,col = <38,3> to <46,7> ]
  0.1           33           33           1           0      33964 Loop: double
multiply()[ file = <loop_test.cpp> line,col = <16,10> to <21,12> ]
```

TAU_REDUCE

- Reads profile files and rules
- Creates selective instrumentation file
  - Specifies which routines should be excluded from instrumentation



Optimizing Instrumentation Overhead: Rules

- #Exclude all events that are members of TAU_USER
  #and use less than 1000 microseconds
  TAU_USER:usec < 1000
- #Exclude all events that have less than 100
  #microseconds and are called only once
  usec < 1000 & numcalls = 1
- #Exclude all events that have less than 1000 usecs per
  #call OR have a (total inclusive) percent less than 5
  usecs/call < 1000
  percent < 5
- Scientific notation can be used
  - **usec>1000 & numcalls>400000 & usecs/call<30 & percent>25**
- Usage:

```
% pprof -d > pprof.dat
% tau_reduce -f pprof.dat -r rules.txt -o select.tau
```

Labs!

Lab: TAU Measurement

Lab Instructions

- See workshop examples:
- papi
- NPB3.1
- Sweep3D

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

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)
```

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

Step I: Configure KOJAK/opari [Download from http://www.fz-juelich.de/zam/kojak/**]**

```
% cd kojak-2.1.1; cp mf/Makefile.defs.ibm Makefile.defs;
  edit Makefile
% make
```

Builds opari

Step II: Configure TAU with Opari (used here with MPI and PDT)

```
% configure -opari=/usr/contrib/TAU/kojak-2.1.1/opari
  -mpiinc=/usr/lpp/ppe.poe/include
  -mpilib=/usr/lpp/ppe.poe/lib
  -pdt=/usr/contrib/TAU/pdtoolkit-3.9
% 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-4.2.1/core; make
```

Set DyninstAPI environment variables (including LD_LIBRARY_PATH)

Step II: Configure TAU with Dyninst

```
% configure -dyninst=/usr/local/dyninstAPI-4.2.1
% 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):

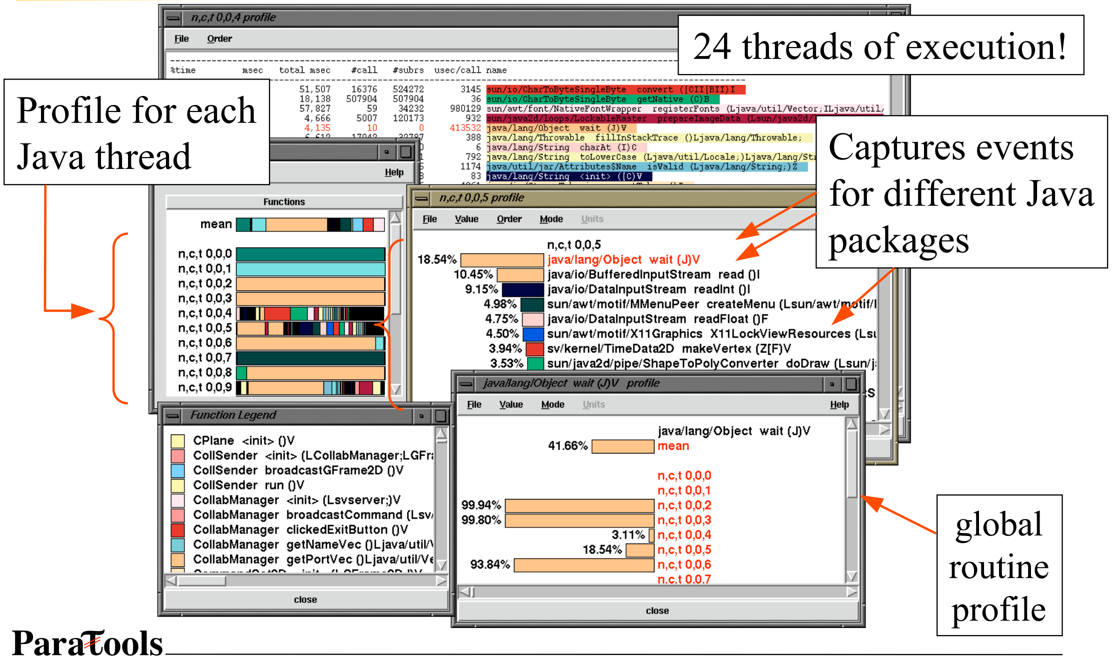
```
% java application
```

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

```
%time        msec   total msec     #call    #subrs  usec/call name
                51,507     16376    524272       3145 sun/io/CharToByteSingleByte  convert ([CII[BII)I
                18,138    507904    507904         36 sun/io/CharToByteSingleByte  getNative (C)B
                57,827        59     34232     980129 sun/awt/font/NativeFontWrapper  registerFonts (Ljava/util/Vector;ILjava/util,
                 4,666      5007    120173        932 sun/java2d/loops/LockableRaster  prepareImageData (Lsun/java2d/
                 4,135        10         0     413532 java/lang/Object  wait (J)V
                                                  388 java/lang/Throwable  fillInStackTrace ()Ljava/lang/Throwable;
                                                    6 java/lang/String  charAt (I)C
                                                  792 java/lang/String  toLowerCase (Ljava/util/Locale;)Ljava/lang/Str
                                                 1174 java/util/jar/Attributes$Name  isValid (Ljava/lang/String;)Z
                                                   83 java/lang/String  <init> ([C)V
```

- mean
- n,c,t 0,0,0
- n,c,t 0,0,1
- n,c,t 0,0,2
- n,c,t 0,0,3
- n,c,t 0,0,4
- n,c,t 0,0,5
- n,c,t 0,0,6
- n,c,t 0,0,7
- n,c,t 0,0,8
- n,c,t 0,0,9

n,c,t 0,0,5

18.54%	java/lang/Object wait (J)V
10.45%	java/io/BufferedInputStream read ()I
9.15%	java/io/DataInputStream readInt ()I
4.98%	sun/awt/motif/MMenuPeer createMenu (Lsun/awt/motif/I
4.75%	java/io/DataInputStream readFloat ()F
4.50%	sun/awt/motif/X11Graphics X11LockViewResources (Lsu
3.94%	sv/kernel/TimeData2D makeVertex (Z[F)V
3.53%	sun/java2d/pipe/ShapeToPolyConverter doDraw (Lsun/j

- CPlane <init> ()V
- CollSender <init> (LCollabManager;LGFra
- CollSender broadcastGFrame2D ()V
- CollSender run ()V
- CollabManager <init> (Lsvserver;)V
- CollabManager broadcastCommand (Lsv
- CollabManager clickedExitButton ()V
- CollabManager getNameVec ()Ljava/util/V
- CollabManager getPortVec ()Ljava/util/Ve

java/lang/Object wait (J)V

41.66%	mean
	n,c,t 0,0,0
	n,c,t 0,0,1
99.94%	n,c,t 0,0,2
99.80%	n,c,t 0,0,3
3.11%	n,c,t 0,0,4
18.54%	n,c,t 0,0,5
93.84%	n,c,t 0,0,6
	n,c,t 0,0,7

Using TAU with Python Applications

Step I: Configure TAU with Python

```
% configure -pythoninc=/usr/include/python2.4/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>]
```

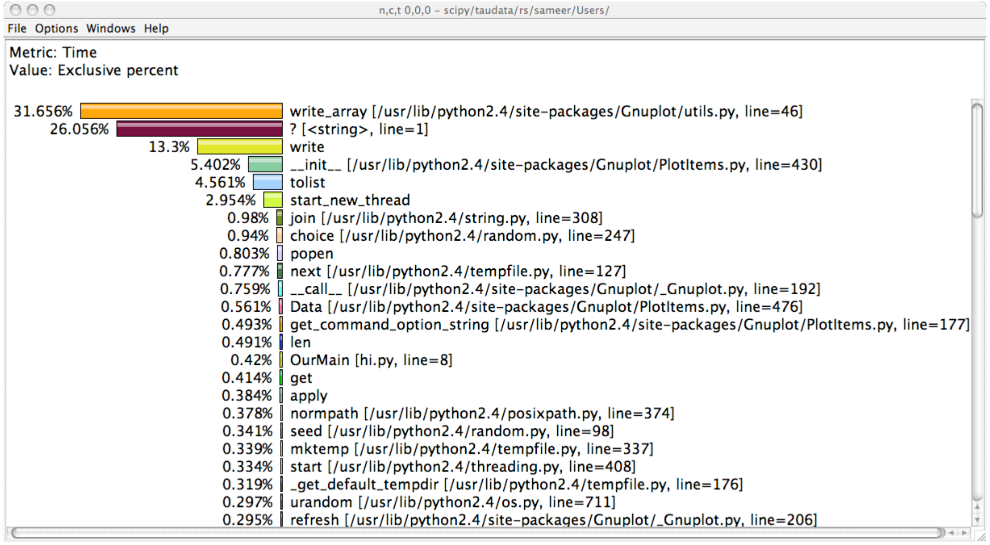
Python Automatic Instrumentation Example

```
#!/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


n,c,t 0,0,0 – scipy/taudata/rs/sameer/Users/
File Options Windows Help

```
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]
```

Labs!

Lab: Kojak’s Opari and Multi-threaded Executions

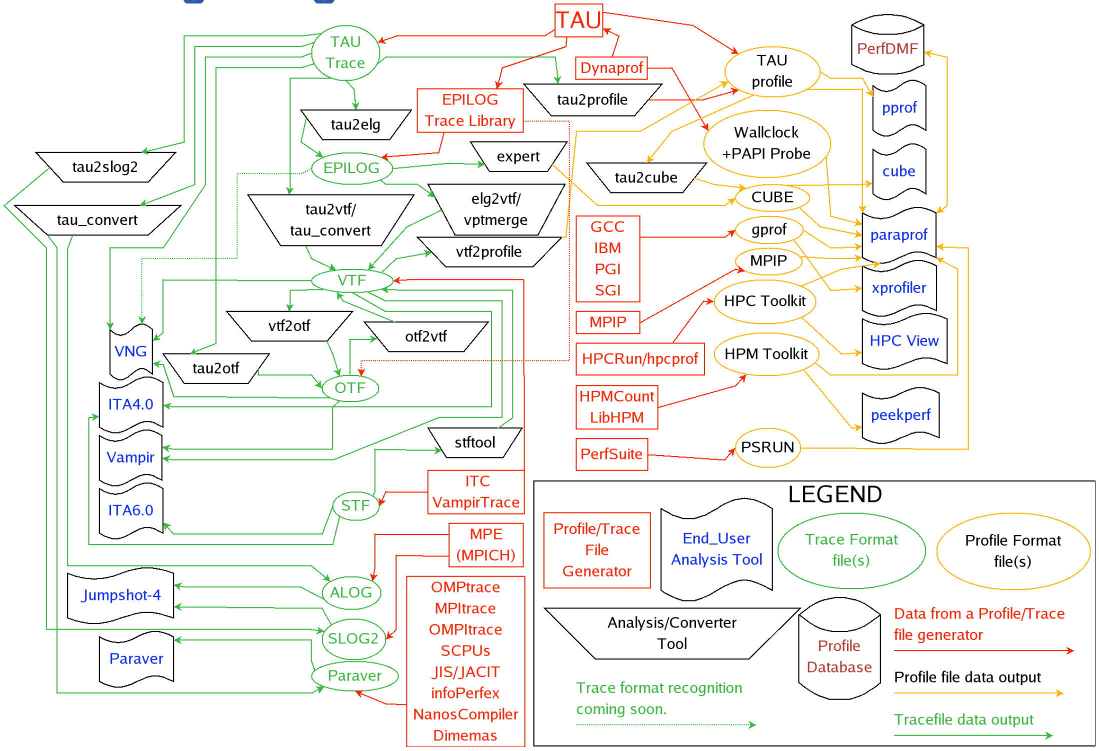
Lab Instructions

- See workshop examples:
- papi
- NPB3.1
- Sweep3D

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



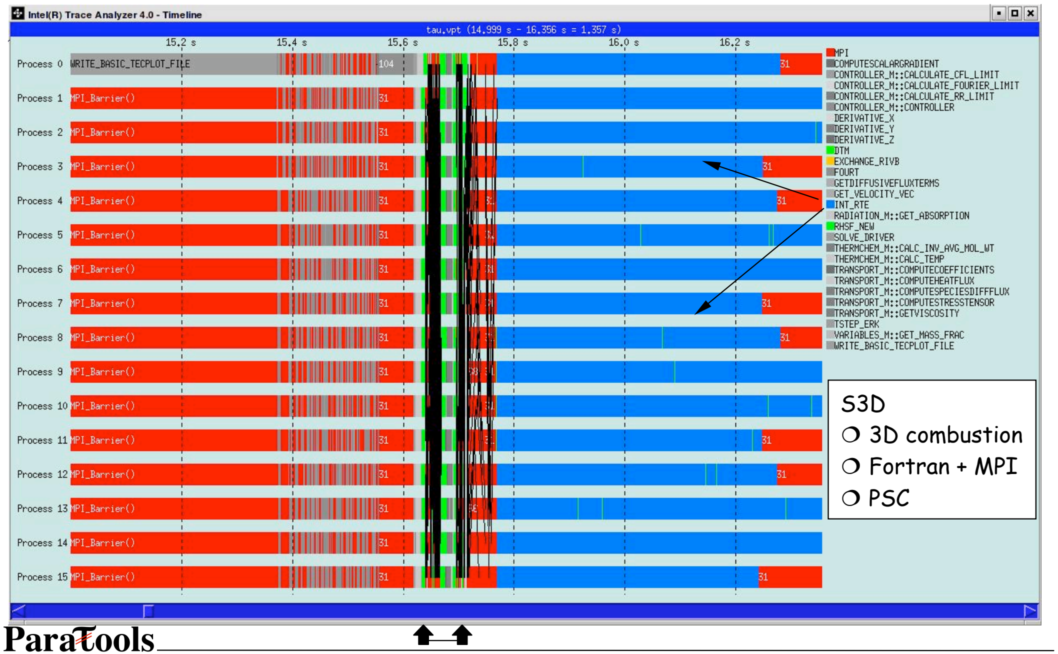
TAU
TAU Trace
PerfDMF
Dynaprof
TAU profile
EPILOG Trace Library
tau2profile
pprof
tau2elg
Wallclock +PAPI Probe
tau2slog2
expert
EPILOG
tau2cube
cube
tau2vtf/ tau_convert
elg2vtf/ vptmerge
CUBE
tau_convert
GCC IBM PGI SGI
gprof
paraprof
vtf2profile
MPIP
VTF
xprofiler
MPIP
HPC Toolkit
vtf2otf
otf2vtf
VNG
HPC View
tau2otf
HPCRun/hpcprof
HPM Toolkit
OTF
ITA4.0
HPMCount LibHPM
peekperf
stftool
PerfSuite
PSRUN
Vampir
ITC VampirTrace
STF
ITA6.0
MPE (MPICH)
Jumpshot-4
ALOG
OMPtrace MPItrace OMPItrace SCPUs JIS/JACIT infoPerfex NanosCompiler Dimemas
SLOG2
Paraver
Paraver

LEGEND
Profile/Trace File Generator
End_User Analysis Tool
Trace Format file(s)
Profile Format file(s)
Analysis/Converter Tool
Profile Database
Data from a Profile/Trace file generator
Profile file data output
Tracefile data output
Trace format recognition coming soon.

TAU Performance System Interfaces

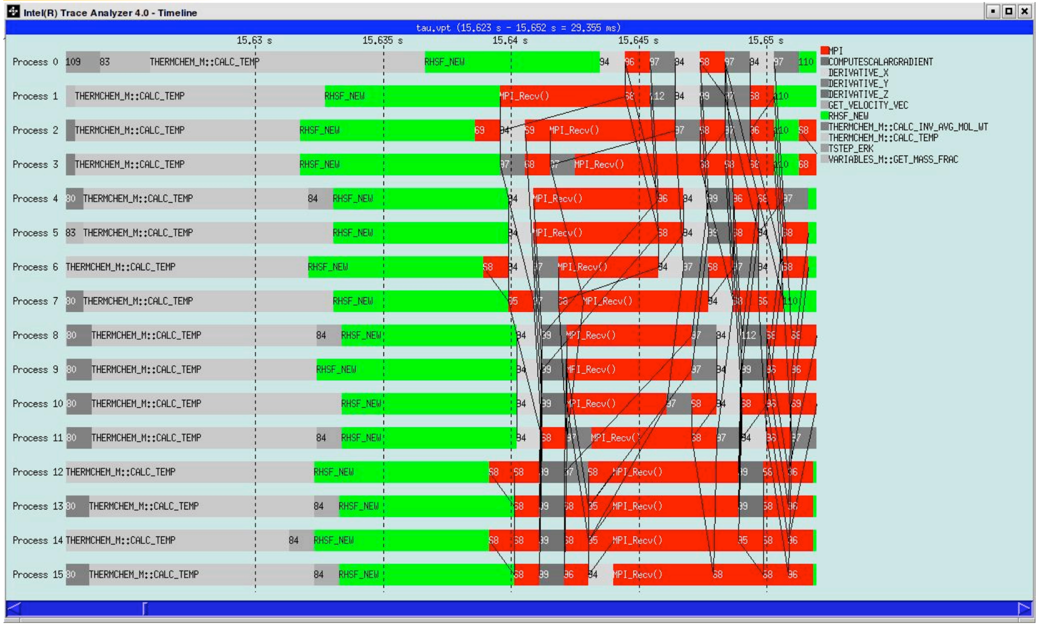
- PDT [U. Oregon, LANL, FZJ] for instrumentation of C++, C99, F95 source code
- PAPI [UTK] for accessing hardware performance counters data
- DyninstAPI [U. Maryland, U. Wisconsin] for runtime instrumentation
- KOJAK [FZJ, UTK]
  - Epilog trace generation library
  - CUBE callgraph visualizer
  - Opari 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]

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



S3D
- 3D combustion
- Fortran + MPI
- PSC

Vampir – Trace Zoomed (S3D)

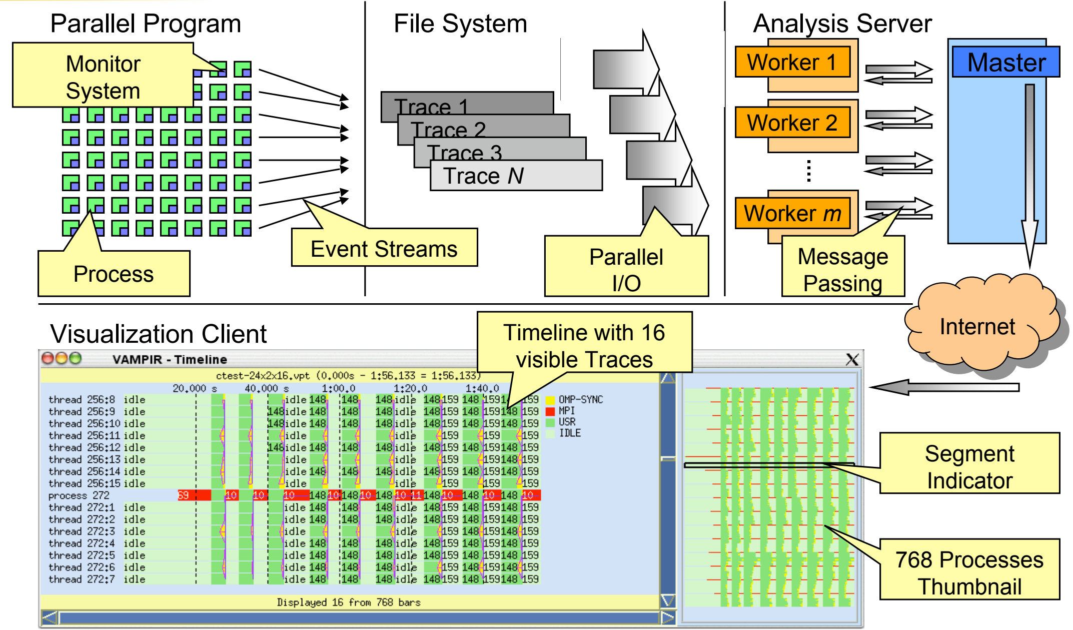


Vampir, VNG, and OTF

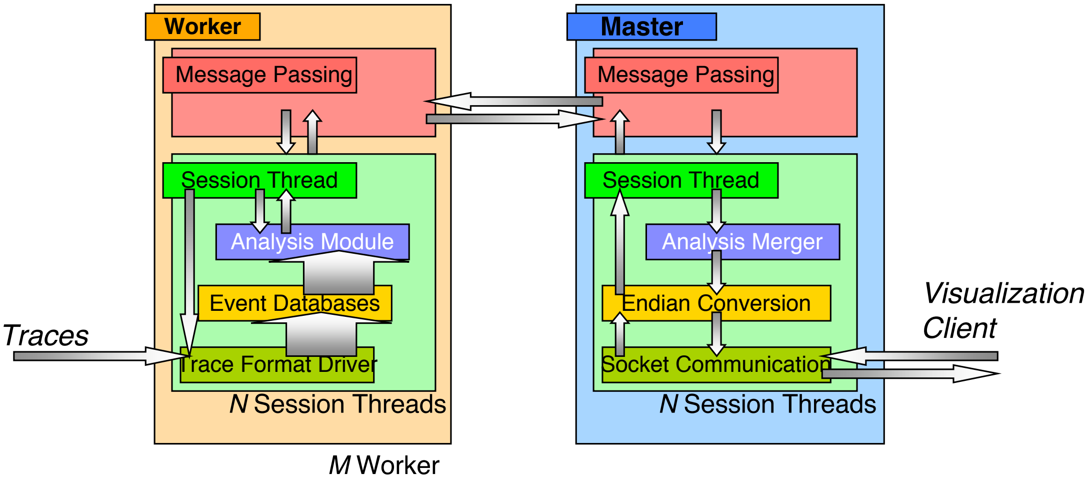
- Commercial trace based tools developed at ZiH, T.U. Dresden
  - Wolfgang Nagel, Holger Brunst and others…
- Vampir Trace Visualizer (aka Intel ® Trace Analyzer v4.0)
  - Sequential program
- Vampir Next Generation (VNG)
  - Client (vng) runs on a desktop, server (vngd) on a cluster
  - Parallel trace analysis
  - Orders of magnitude bigger traces (more memory)
  - State of the art in parallel trace visualization
- Open Trace Format (OTF)
  - Hierarchical trace format, efficient streams based parallel access with VNGD
  - Replacement for proprietary formats such as STF
  - Tracing library available with a evaluation license now. Open source package at SC'06.

http://www.vampir-ng.de

Vampir Next Generation (VNG) Architecture

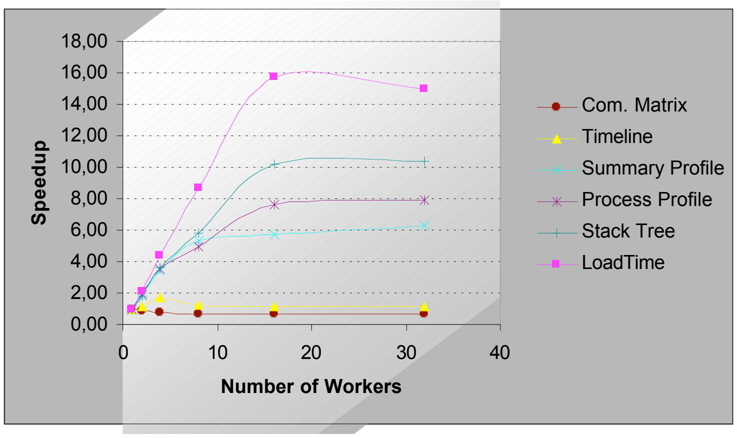


VNG Parallel Analysis Server



Scalability of VNG [Holger Brunst, WAPA 2005]

- sPPM
- 16 CPUs
- 200 MB



Number of Workers	1	2	4	8	16	32
Load Time	47,33	22,48	10,80	5,43	3,01	3,16
Timeline	0,10	0,09	0,06	0,08	0,09	0,09
Summary Profile	1,59	0,87	0,47	0,30	0,28	0,25
Process Profile	1,32	0,70	0,38	0,26	0,17	0,17
Com. Matrix	0,06	0,07	0,08	0,09	0,09	0,09
Stack Tree	2,57	1,39	0,70	0,44	0,25	0,25

VNG Analysis Server Architecture

- Implementation using MPI and Pthreads
- Client/server approach
- MPI and pthreads are available on most platforms
- Workload and data distribution among “physical” MPI processes
- Support of multiple visualization clients by using virtual sessions handled by individual threads
- Sessions are scheduled as threads

TAU Tracing Enhancements

- Configure TAU with -TRACE –otf=<dir> option

```
% configure -TRACE -otf=<dir> …
```

Generates tau_merge, tau2vtf, tau2otf tools in <tau>/<arch>/bin directory

```
% tau_f90.sh app.f90 -o app
```

- Instrument and execute application

```
% mpirun -np 4 app
```

- Merge and convert trace files to OTF format

```
% tau2otf tau.trc tau.edf app.otf [-z][-n <nstreams>]
% vampir app.otf
```

OR use VNG to analyze OTF/VTF trace files

Environment Variables

- Configure TAU with -TRACE –otf=<dir> option

```
% configure -TRACE -otf=<dir>
   -MULTIPLECOUNTERS -papi=<dir> -mpi
   -pdt=dir …
```

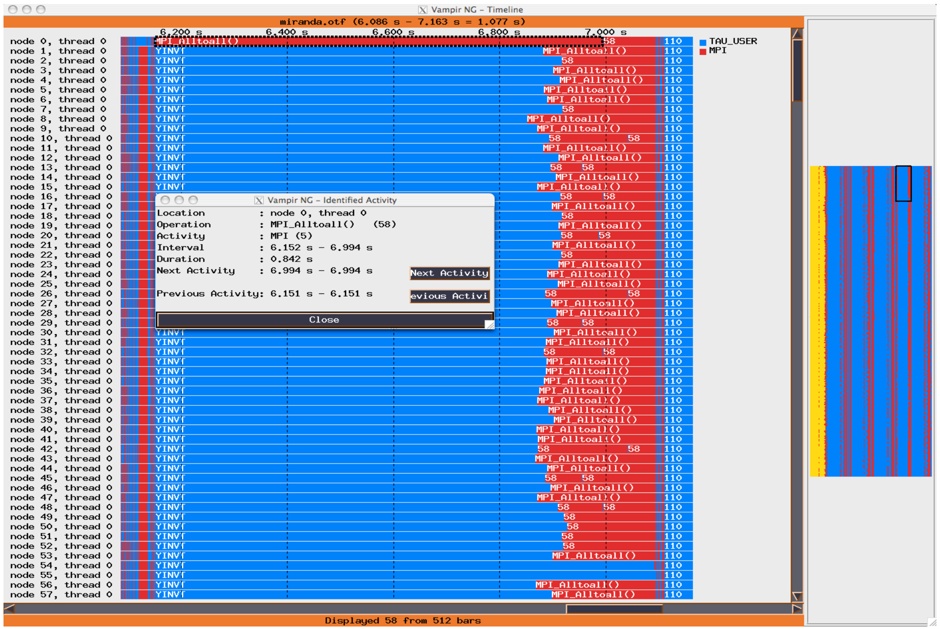
- Set environment variables

```
% setenv TRACEDIR /p/gm1/<login>/traces
% setenv COUNTER1 GET_TIME_OF_DAY (reqd)
% setenv COUNTER2 PAPI_FP_INS
% setenv COUNTER3 PAPI_TOT_CYC …
```

- Execute application

```
% mpirun -np 32 ./a.out [args]

% tau_treemerge.pl; tau2otf/tau2vtf ...
```

VNG Timeline Display



Vampir NG – Timeline

miranda.otf (6.086 s – 7.163 s = 1.077 s)

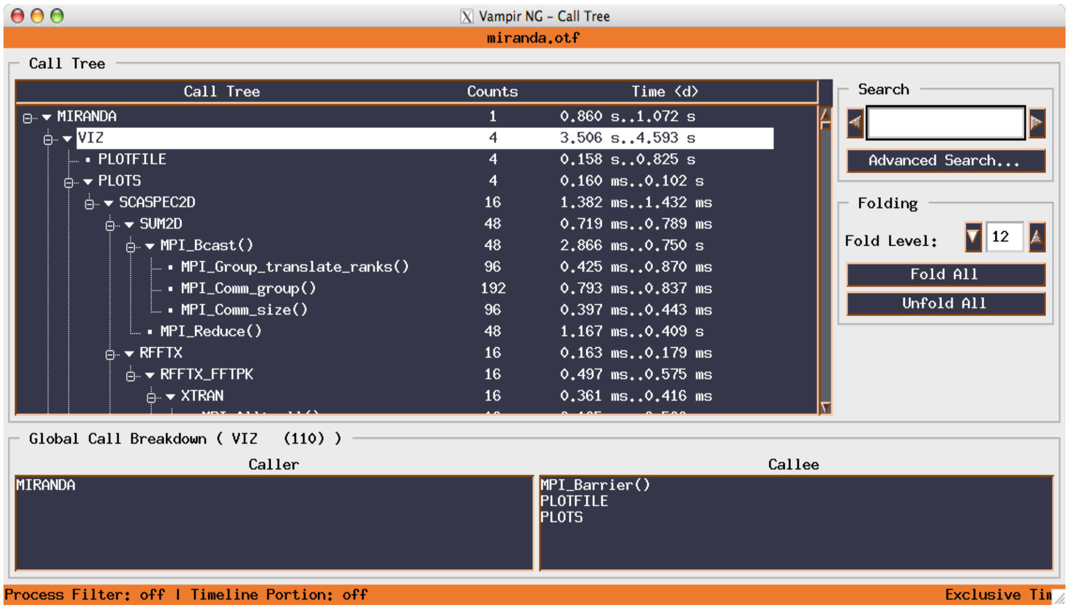
Vampir NG – Identified Activity

```
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
```

Displayed 58 from 512 bars

ParaTools

VNG Calltree Display



Vampir NG – Call Tree

miranda.otf

Call Tree

```
Call Tree                                   Counts   Time <d>
MIRANDA                                     1        0.860 s..1.072 s
  VIZ                                       4        3.506 s..4.593 s
    PLOTFILE                                4        0.158 s..0.825 s
    PLOTS                                   4        0.160 ms..0.102 s
      SCASPEC2D                             16       1.382 ms..1.432 ms
        SUM2D                               48       0.719 ms..0.789 ms
          MPI_Bcast()                       48       2.866 ms..0.750 s
            MPI_Group_translate_ranks()     96       0.425 ms..0.870 ms
            MPI_Comm_group()                192      0.793 ms..0.837 ms
            MPI_Comm_size()                 96       0.397 ms..0.443 ms
          MPI_Reduce()                      48       1.167 ms..0.409 s
        RFFTX                               16       0.163 ms..0.179 ms
          RFFTX_FFTPK                       16       0.497 ms..0.575 ms
            XTRAN                           16       0.361 ms..0.416 ms
```

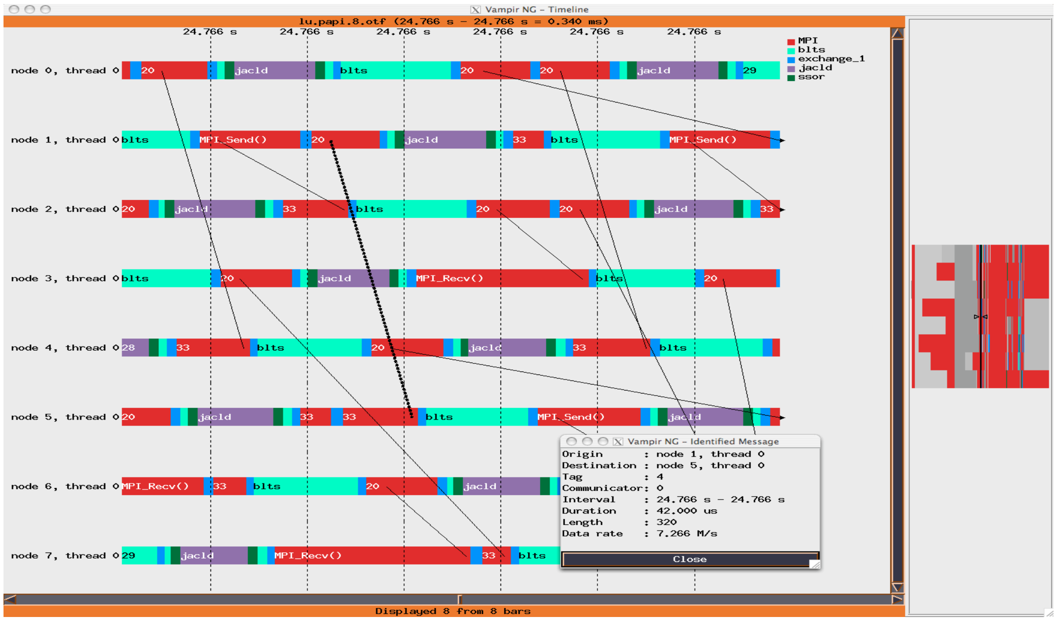
Global Call Breakdown (VIZ (110))

Caller	Callee
MIRANDA	MPI_Barrier() PLOTFILE PLOTS

Process Filter: off | Timeline Portion: off — Exclusive Tin

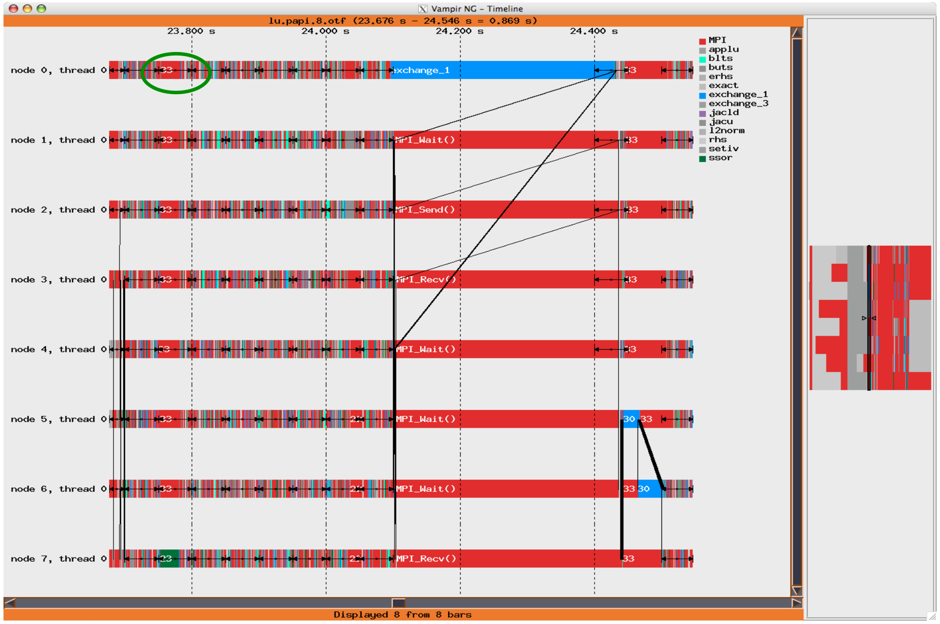
ParaTools

VNG Timeline Zoomed In

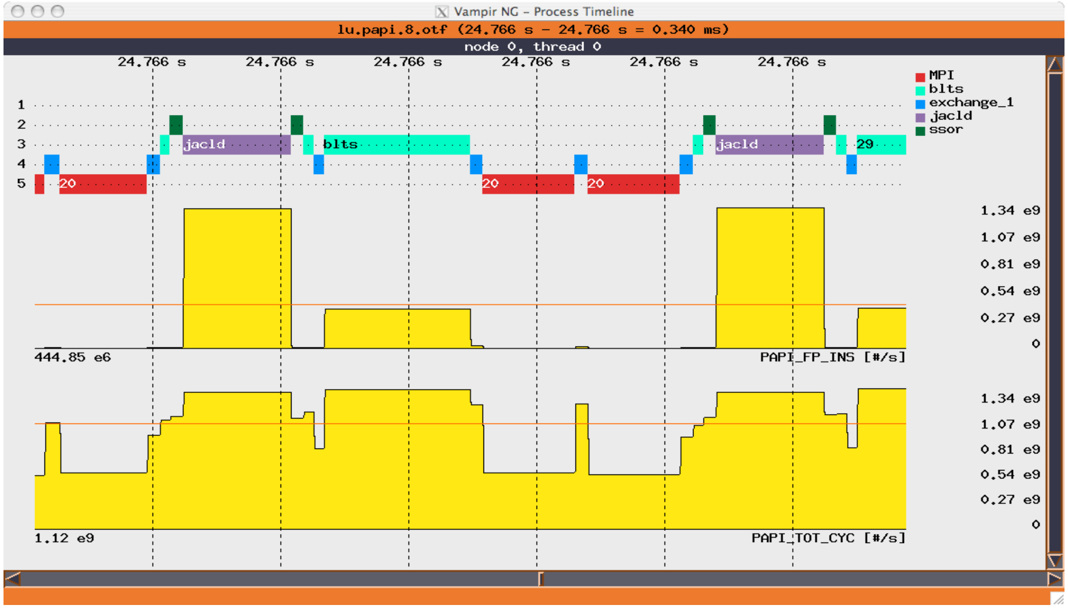


```
Origin       : node 1, thread 0
Destination  : node 5, thread 0
Tag          : 4
Communicator: 0
Interval     : 24.766 s – 24.766 s
Duration     : 42.000 us
Length       : 320
Data rate    : 7.266 M/s
```

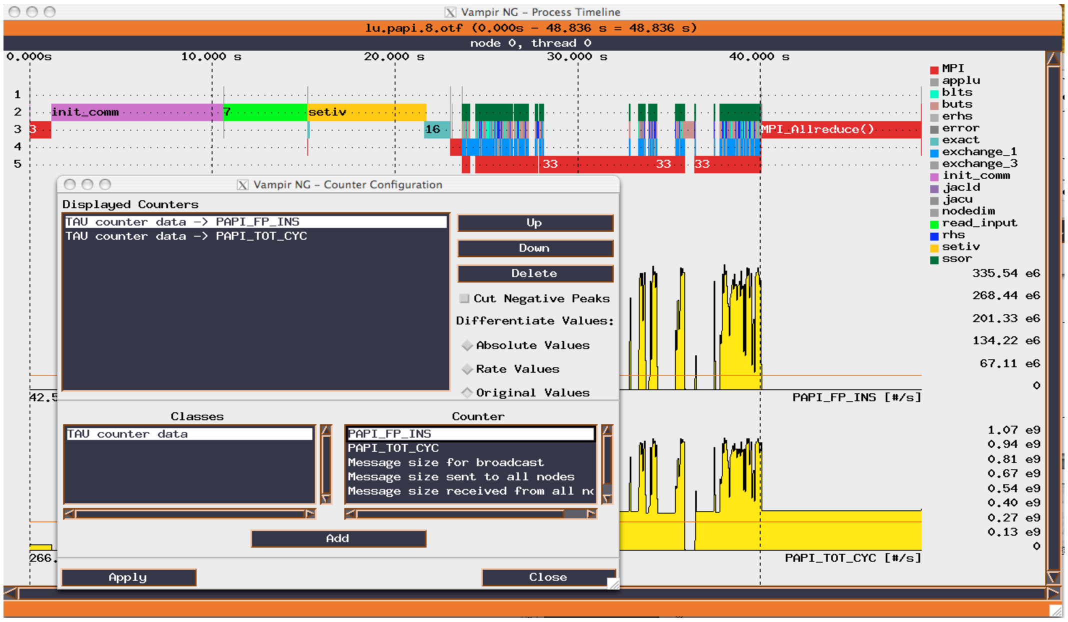
VNG Grouping of Interprocess Communications



VNG Process Timeline with PAPI Counters



OTF/VNG Support for Counters



Displayed Counters

TAU counter data –> PAPI_FP_INS
TAU counter data –> PAPI_TOT_CYC

☐ Cut Negative Peaks

Differentiate Values:

◇ Absolute Values
◇ Rate Values
◇ Original Values

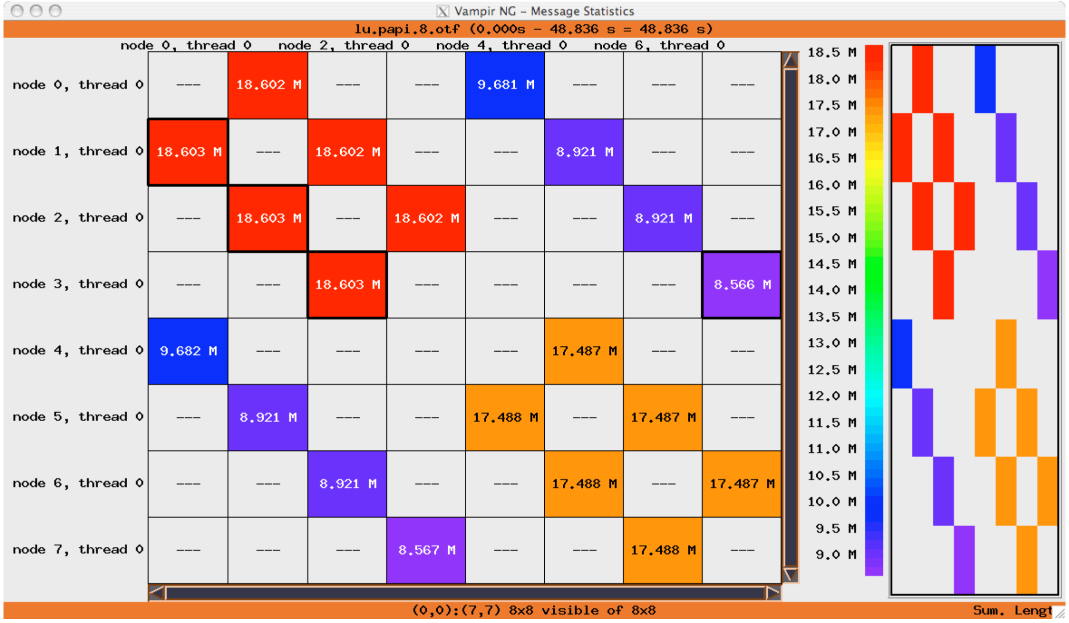
Classes

TAU counter data

Counter

PAPI_FP_INS
PAPI_TOT_CYC
Message size for broadcast
Message size sent to all nodes
Message size received from all nc

VNG Communication Matrix Display

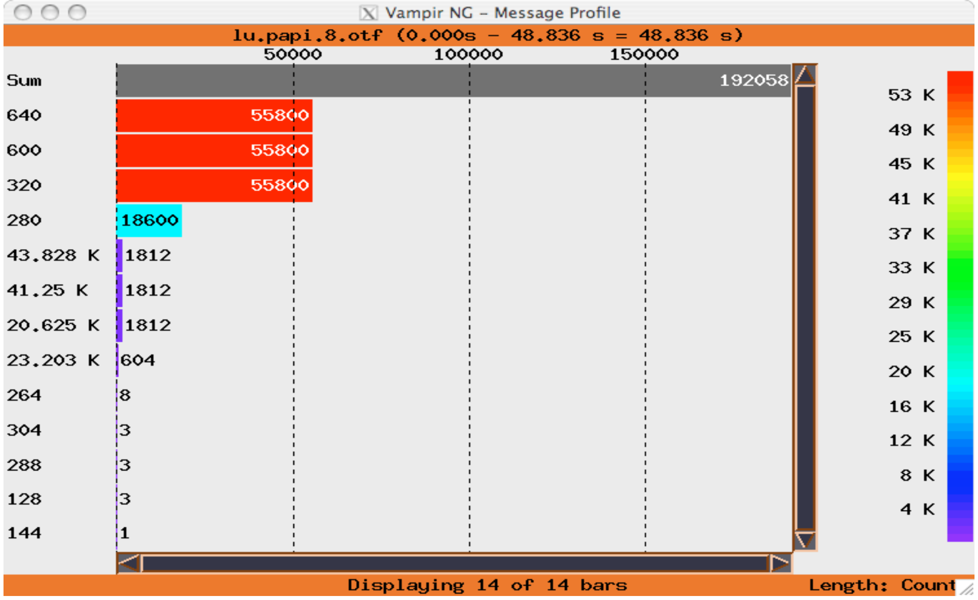


lu.papi.8.otf (0.000s – 48.836 s = 48.836 s)

	node 0, thread 0		node 2, thread 0		node 4, thread 0		node 6, thread 0	
node 0, thread 0	---	18.602 M	---	---	9.681 M	---	---	---
node 1, thread 0	18.603 M	---	18.602 M	---	---	8.921 M	---	---
node 2, thread 0	---	18.603 M	---	18.602 M	---	---	8.921 M	---
node 3, thread 0	---	---	18.603 M	---	---	---	---	8.566 M
node 4, thread 0	9.682 M	---	---	---	---	17.487 M	---	---
node 5, thread 0	---	8.921 M	---	---	17.488 M	---	17.487 M	---
node 6, thread 0	---	---	8.921 M	---	---	17.488 M	---	17.487 M
node 7, thread 0	---	---	---	8.567 M	---	---	17.488 M	---

(0,0):(7,7) 8x8 visible of 8x8

VNG Message Profile

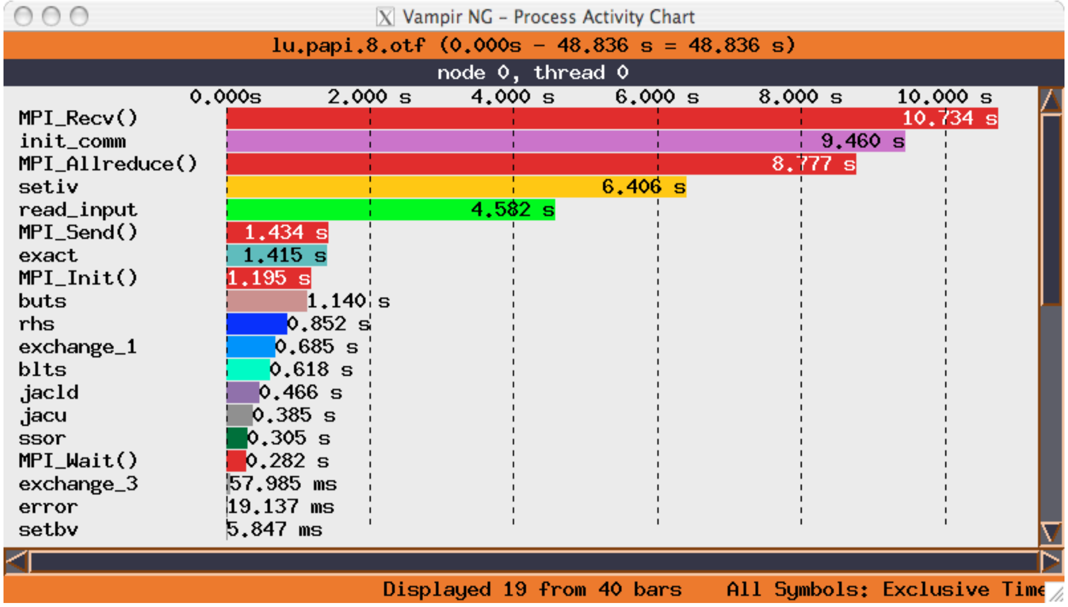


lu.papi.8.otf (0.000s – 48.836 s = 48.836 s)

Sum	192058
640	55800
600	55800
320	55800
280	18600
43.828 K	1812
41.25 K	1812
20.625 K	1812
23.203 K	604
264	8
304	3
288	3
128	3
144	1

Displaying 14 of 14 bars

VNG Process Activity Chart



Vampir NG – Process Activity Chart

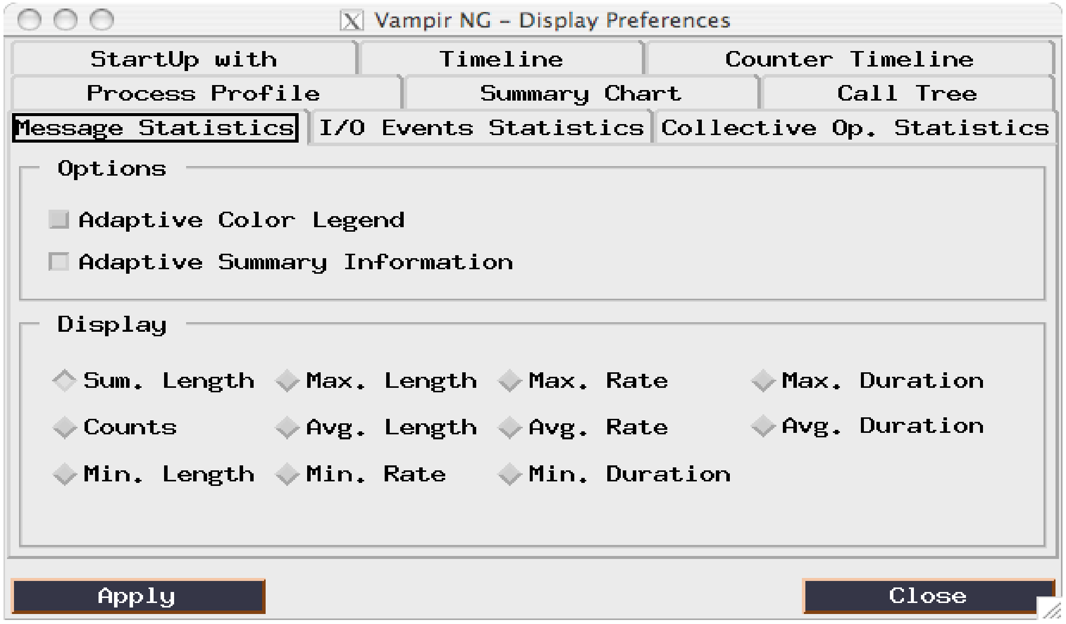
lu.papi.8.otf (0.000s – 48.836 s = 48.836 s)

node 0, thread 0

```
MPI_Recv()       10.734 s
init_comm        9.460 s
MPI_Allreduce()  8.777 s
setiv            6.406 s
read_input       4.582 s
MPI_Send()       1.434 s
exact            1.415 s
MPI_Init()       1.195 s
buts             1.140 s
rhs              0.852 s
exchange_1       0.685 s
blts             0.618 s
jacld            0.466 s
jacu             0.385 s
ssor             0.305 s
MPI_Wait()       0.282 s
exchange_3       57.985 ms
error            19.137 ms
setbv            5.847 ms
```

Displayed 19 from 40 bars   All Symbols: Exclusive Time

VNG Preferences



Vampir NG – Display Preferences

Options

□ Adaptive Color Legend

□ Adaptive Summary Information

Display

◇ Sum. Length ◇ Max. Length ◇ Max. Rate ◇ Max. Duration

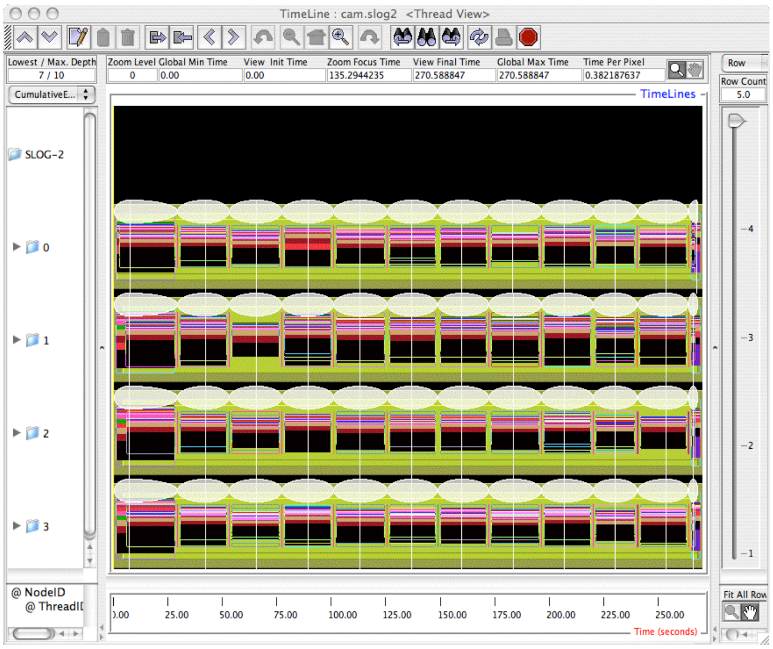
◇ Counts ◇ Avg. Length ◇ Avg. Rate ◇ Avg. Duration

◇ Min. Length ◇ Min. Rate ◇ Min. Duration

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 in TAU’s <arch>/bin directory, bundled with TAU
- Java-based tracefile visualization tool for postmortem performance analysis of MPI programs
- Latest version is Jumpshot-4 for SLOG-2 format
  - Scalable level of detail support
  - Timeline and histogram views
  - Scrolling and zooming
  - Search/scan facility

Jumpshot



TimeLine : cam.slog2 <Thread View>

Lowest / Max. Depth	Zoom Level	Global Min Time	View Init Time	Zoom Focus Time	View Final Time	Global Max Time	Time Per Pixel	Row Count
7 / 10	0	0.00	0.00	135.2944235	270.588847	270.588847	0.382187637	5.0

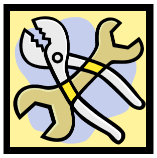
Labs!

Lab: Tracing and Vampir/VNG

KOJAK Project

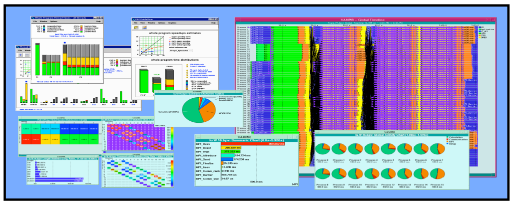
- Collaborative research project between
  - Forschungszentrum Jülich
  - University of Tennessee
- Automatic performance analysis
  - MPI and/or OpenMP applications
  - Parallel communication analysis
  - CPU and memory analysis
- WWW
  - http://www.fz-juelich.de/zam/kojak/
  - htttp://icl.cs.utk.edu/kojak/
- Contact
  - kojak@cs.utk.edu

Software-engineering problem



Efficient development of efficient code

- Tools are needed that help optimize applications by
  - Collecting relevant performance data
  - Automatically identifying the causes of performance problems
- Requirements
  - Expressiveness and accuracy of results
  - Scalability
  - Convenience of use



KOJAK Tools

- KOJAK Trace Analysis Environment
  - Automatic event trace analysis for MPI and / or OpenMP applications
  - Includes tools for trace generation and analysis
    - EPILOG tracing library
    - EXPERT trace analyzer
  - Display of results using CUBE

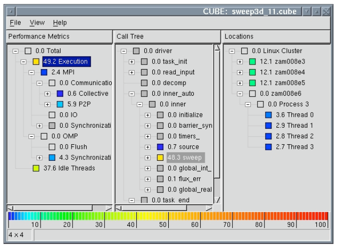
(1) Wolf, Felix: **“Automatic Performance Analysis on Parallel Computers with SMP Nodes”** , Dissertation, RWTH Aachen, NIC Series, Volume 17, Februar 2003. http://www.fz-juelich.de/nic-series/volume17/volume17.html

(2) Wolf F., Mohr, B. **"Automatic performance analysis of hybrid MPI/OpenMP applications,"** *Journal of Systems Architecture, Special Issue 'Evolutions in parallel distributed and network-based processing'*, Clematis, A., D'Agostino, D. eds. Elsevier, 49(10-11), pp. 421-439, November, 2003.

KOJAK Tools (2)

- CUBE
  - Generic display for call-tree profiles
  - Automatic comparison of different experiments
  - Download http://icl.cs.utk.edu/kojak/cube/

Song, F., Wolf, F., Bhatia, N., Dongarra, J., Moore, S. **"An Algebra for Cross-Experiment Performance Analysis,"** *2004 International Conference on Parallel Processing (ICPP-04)*, Montreal, Quebec, Canada, August 2004.



KOJAK: Supported Platforms

- Instrumentation and measurement only (analysis on front-end or workstation)
  - Cray T3E, Cray XD1, Cray X1, and Cray XT3
  - IBM BlueGene/L
  - Hitachi SR-8000
  - NEC SX
- Full support (instrumentation, measurement, and automatic analysis)
  - Linux IA32, IA64, and EMT64/x86_64 based clusters
  - IBM AIX Power3 and Power4 based clusters (SP2, Regatta)
  - SGI Irix MIPS based clusters (Origin 2K, Origin 3K)
  - SGI Linux IA64 based clusters (Altix)
  - SUN Solaris Sparc and x86/x86_64 based clusters (SunFire, …)
  - DEC/HP Tru64 Alpha based clusters (Alphaserver, …)

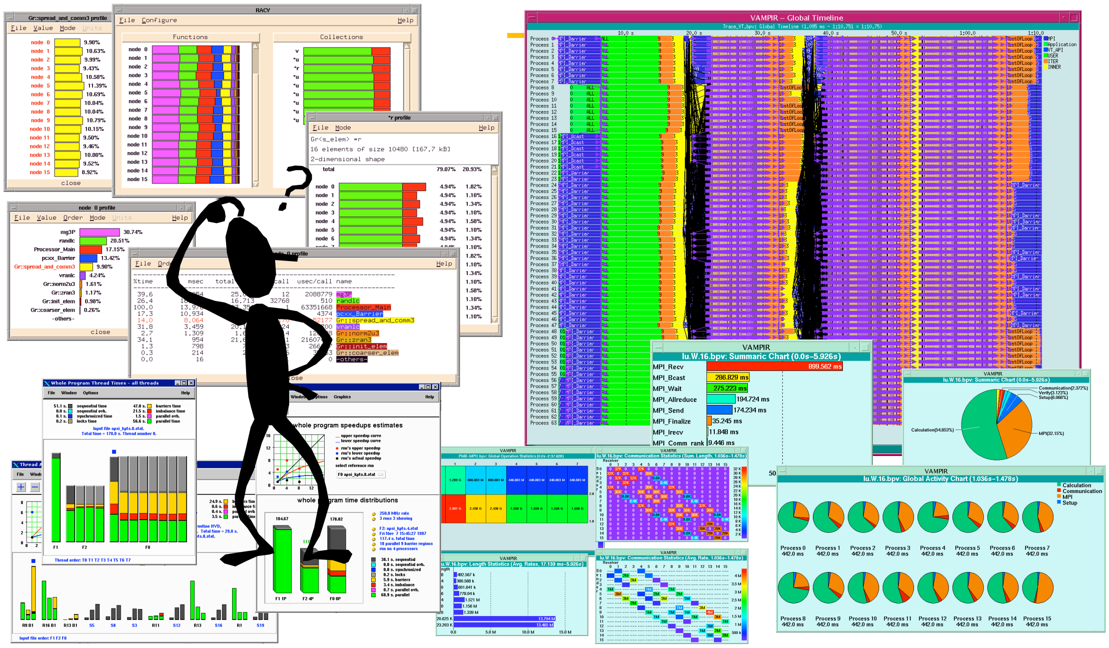
Installation

- Install wxWidgets http://www.wxwidgets.org
- Install libxml2 http://www.xmlsoft.org
- The following commands should be in your search path
  - xml2-config
  - wx-config
- Install CUBE http://icl.cs.utk.edu/kojak/cube/
  - Two options
    - CUBE library only (analysis without presentation)
    - CUBE library + GUI
  - Follow the installation instructions in the manual
- Install KOJAK http://www.fz-juelich.de/zam/kojak/
  - Follow the installation instructions in ./INSTALL

KOJAK Documentation

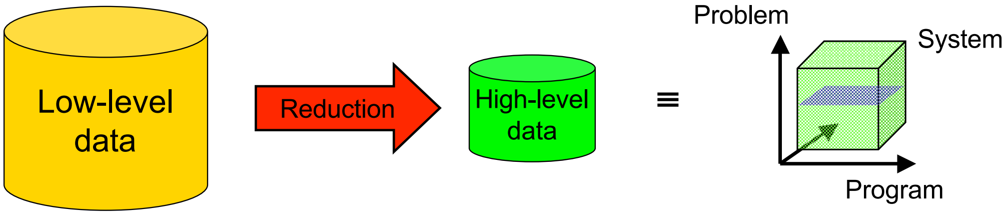
- Installation
  - File INSTALL in top-level build directory
- Usage instructions
  - File USAGE in $(PREFIX)/doc install directory
- Complementary documentation
  - CUBE documentation
    - http://icl.cs.utk.edu/kojak/cube/
  - Specification of EPILOG trace format
    - File epilog.ps in $(PREFIX)/epilog/doc/epilog.ps or
    - http://www.fz-juelich.de/zam/docs/autoren2004/wolf/

Low-level View of Performance Behavior

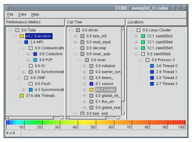

VAMPIR – Global Timeline
VAMPIR
RACY
Functions
Collections
node 0 profile
Whole Program Thread Times - all threads
whole program speedups estimates
whole program time distributions
lu.W.16.bpv: Summaric Chart (0.0s–5.926s)
MPI_Recv
MPI_Bcast
MPI_Wait
MPI_Allreduce
MPI_Send
MPI_Finalize
MPI_Irecv
MPI_Comm_rank
lu.W.16.bpv: Global Activity Chart (1.036s–1.478s)
Calculation
Communication
MPI
Setup

Automatic Performance Analysis

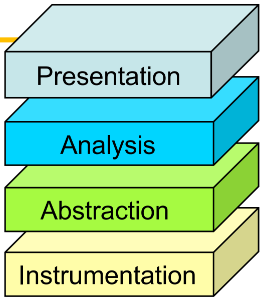
- Transformation of low-level performance data



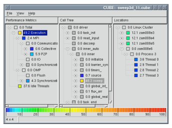
- Take event traces of MPI/OpenMP applications
- Search for execution patterns
- Calculate mapping
  - Problem, call path, system resource ⇒ time
- Display in performance browser

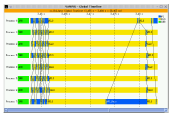


KOJAK Layers

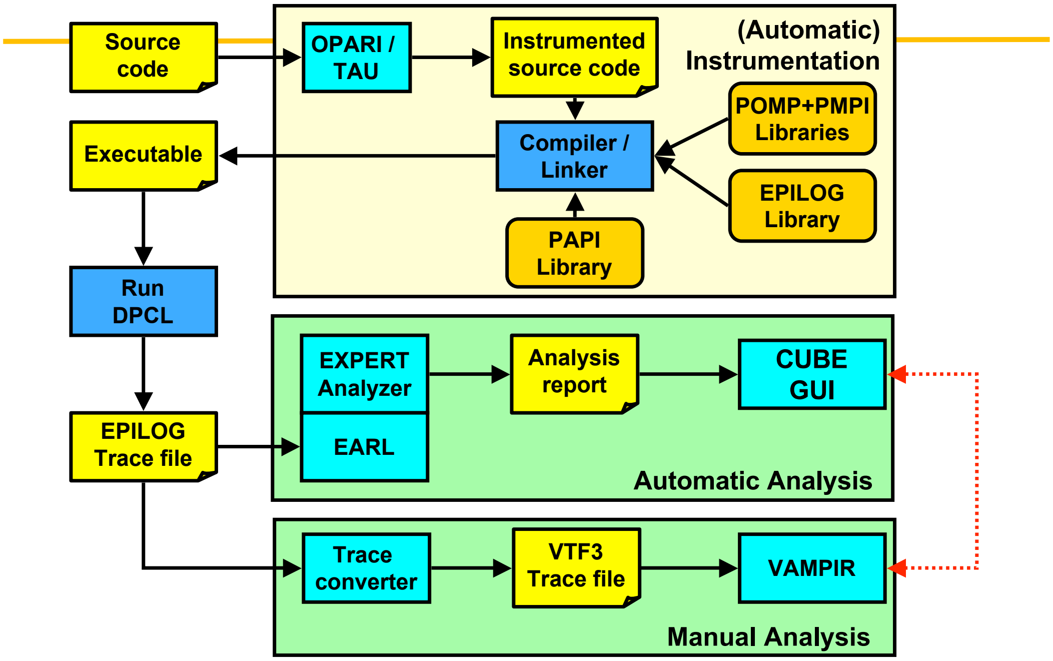


- Instrumentation
  - Insertion of extra code to generate event trace
- Abstract representation of event trace
  - Simplified specification of performance problems
  - Simplified extension of predefined problems
- Automatic analysis
  - Classification and quantification of performance behavior
  - Automatic comparison of multiple experiments
    - Not yet released
- Presentation
  - Navigating / browsing through performance space
  - Can be combined with VAMPIR time-line display

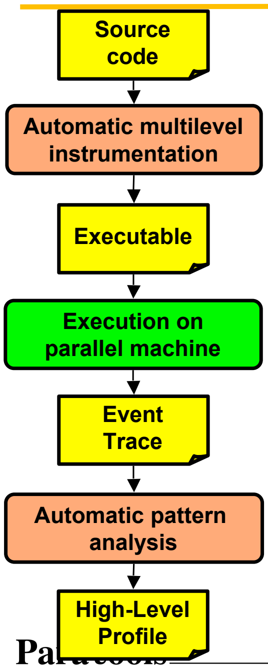




KOJAK Architecture



Analysis process



Source code

Automatic multilevel instrumentation

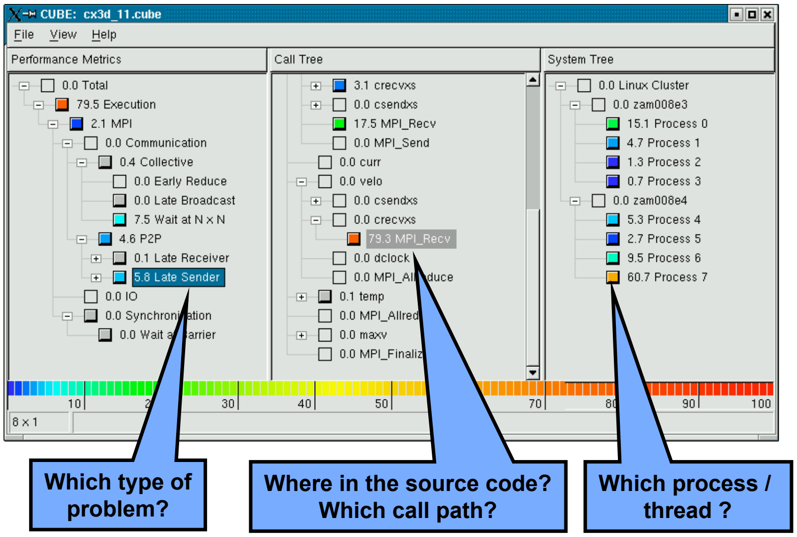
Executable

Execution on parallel machine

Event Trace

Automatic pattern analysis

High-Level Profile



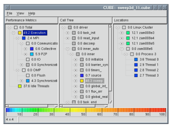
Which type of problem?

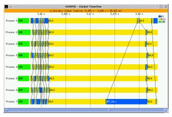
Where in the source code? Which call path?

Which process / thread ?

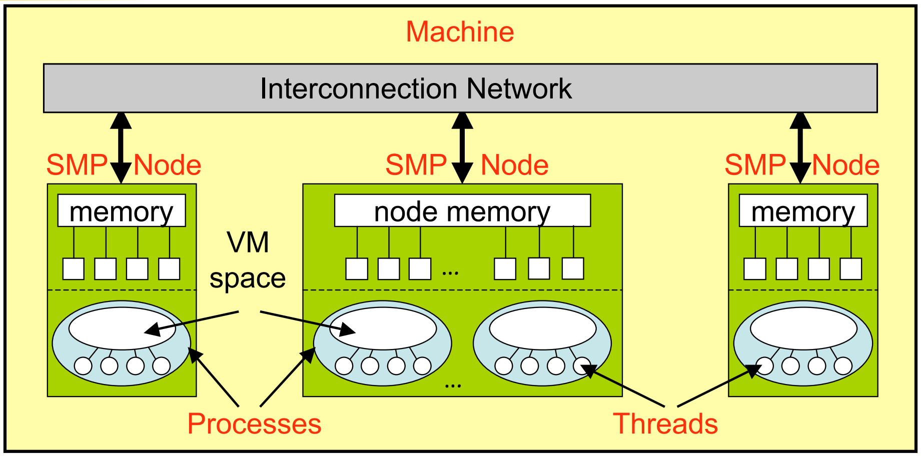
EPILOG Trace File Format

- Event Processing, Investigating, and LOGging
- MPI and OpenMP support (i.e., thread-safe)
  - Region enter and exit
  - Collective region enter and exit (MPI & OpenMP)
  - Message send and receive
  - Parallel region fork and join
  - Lock acquire and release
- Stores source code + HW counter information
- Input of the EXPERT analyzer
- Visualization using VAMPIR
  - EPILOG ⇨VTF3 converter





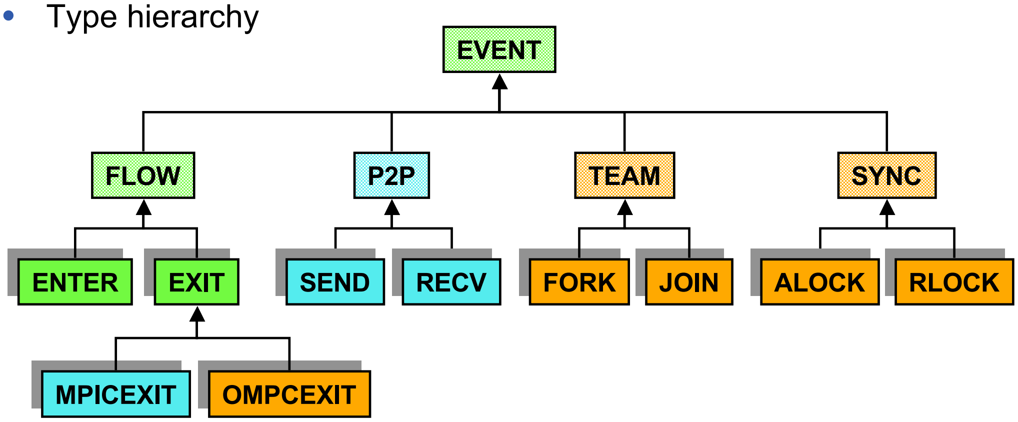
EPILOG Trace File Format (2)



- Hierarchical location ID
  - (machine, node, process, thread)
- Specification
  - http://www.fz-juelich.de/zam/docs/autoren2004/wolf

KOJAK Event Model

- Type hierarchy



- Event type
  - Set of attributes (time, location, position, …)
- Event trace
  - Sequence of events in chronological order

Instrumentation

- Generating event traces requires extra code to be inserted into the application
- Supported programming languages
  - C, C++, Fortran
- Automatic instrumentation of MPI
  - PMPI wrapper library
- Automatic instrumentation of OpenMP
  - POMP wrapper library in combination with OPARI
- Automatic instrumentation of user code / functions
  - Using compiler-supplied profiling interface and **kinst** tool
  - Using TAU
- Manual instrumentation of user code / functions
  - Using POMP directives and **kinst-pomp** tool

TAU Source Code Instrumentor

- Based on PDTOOLKIT
- Part of the TAU performance framework
- Supports
  - f77, f90, C, and C++
  - OpenMP, MPI
  - HW performance counters
  - Selective instrumentation
- http://www.cs.uoregon.edu/research/tau/
- Configure with -epilog=<dir> to specify location of EPILOG library

Tuning and Analysis Utilities

KOJAK Runtime Environment

- ELG_PFORM_GDIR
  - Name of global, cluster-wide directory to store final trace file
  - Default platform specific, typically "."
- ELG_PFORM_LDIR
  - Name of node-local directory to store temporary trace files
  - Default platform specific, typically "/tmp"
- ELG_FILE_PREFIX
  - Prefix used for names of EPILOG trace files
  - Default "a"
- ELG_BUFFER_SIZE
  - Size of per-process event trace buffer in bytes
  - Default 10000000
- ELG_VERBOSE
  - Print EPILOG related control information during measurement
  - Default no

Hardware Counters

- Small set of CPU registers that count events
  - Events: signal related to a processor's function
- Original purpose
  - Verification and evaluation of CPU design
- Can help answer question
  - How efficiently is my application mapped onto the underlying architecture?
- KOJAK supports hardware counter analysis
  - Can be recorded as part of ENTER/EXIT event records
- Uses PAPI for portable access to counters

Hardware Counters (2)

- Request counters using environment variable
  - ELG_METRICS
- EPILOG defines names for most common counters
- See file HWCOUNTERS in $(PREFIX)/doc installation directory
- Request counters as colon-separated list

```
export ELG_METRICS=L1_D_MISS:FLOATING_POINT
```

Running the Application

- Run your instrumented application
-

```
> mpirun -np 4 a.out
> ls
a.elg  a.out  …
```

- KOJAK includes several tools to check trace file
  - ASCII representation of trace file
    - elg_print <file>
  - Event counts and simple statisitics
    - elg_stat <file>
  - Correct order of SEND and RECV events
    - elg_msgord <file>
    - Needed to check correct clock synchronization

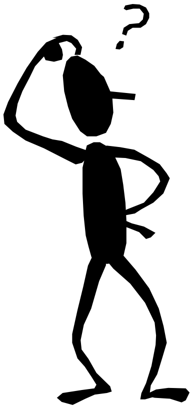
Simple Statistics with elg_stat

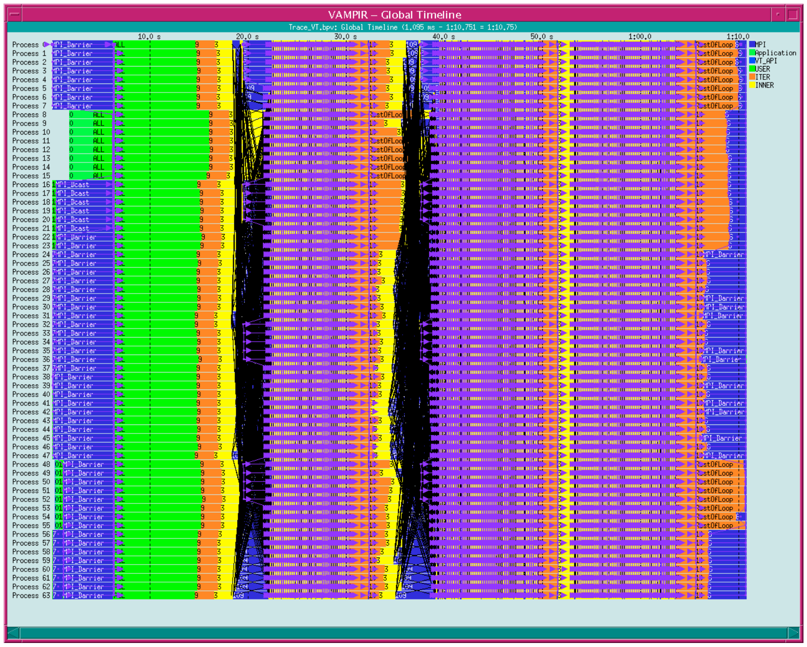
```
          ENTER :    119     90    119     90
           EXIT :     71     54     71     54
       MPI_SEND :      0      0      0      0
       MPI_RECV :      0      0      0      0
   MPI_COLLEXIT :     12      0     12      0
       OMP_FORK :      9      0      9      0
       OMP_JOIN :      9      0      9      0
      OMP_ALOCK :      0      0      0      0
      OMP_RLOCK :      0      0      0      0
   OMP_COLLEXIT :     36     36     36     36
       ENTER_CS :      0      0      0      0

                  MPI_Barrier :    18 :      9      0      9      0
                    MPI_Bcast :     6 :      3      0      3      0
                MPI_Comm_free :     2 :      1      0      1      0
               MPI_Comm_split :     2 :      1      0      1      0
                 MPI_Finalize :     2 :      1      0      1      0
                     MPI_Init :     2 :      1      0      1      0
                         step :   216 :     54     54     54     54
                   sequential :    18 :      9      0      9      0
               !$omp parallel :    36 :      9      9      9      9
               !$omp ibarrier :    36 :      9      9      9      9
                    !$omp for :    36 :      9      9      9      9
               !$omp ibarrier :    36 :      9      9      9      9
                     parallel :     6 :      3      0      3      0
                         main :     2 :      1      0      1      0
```

Analyzing Performance Data

- Analyzing event traces with EXPERT
  - Overview of pattern analysis
  - Running EXPERT
- Viewing results with CUBE
  - Data model
  - Basic principles
  - Usage
  - Viewing EXPERT / CONE output
  - Performance algebra
- Appendix
  - KOJAK application performance properties

Searching Event Traces Manually



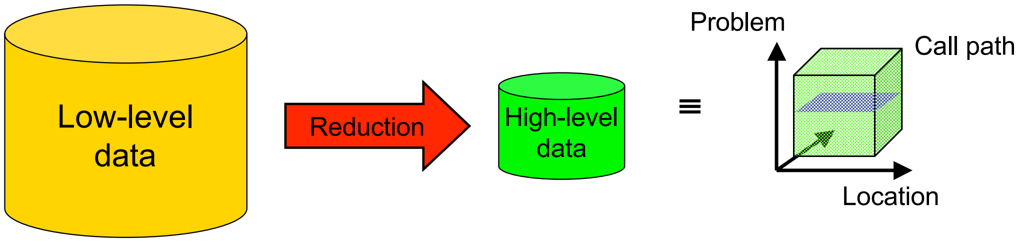


VAMPIR – Global Timeline

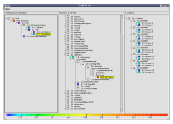
Trace_VT.bpv: Global Timeline (1.095 ms - 1:10.751 = 1:10.75)

Automatic Trace Analysis

- Automatic performance analysis



- Take event traces of MPI/OpenMP applications
- Search for execution patterns
- Calculate mapping
  - Problem, program resource, system resource ⇒ time
- Display in performance browser



EXPERT

- Offline trace analyzer
  - Input format: EPILOG
- Transforms traces into compact representation of performance behavior
  - Mapping of call paths, process or threads into metric space
- Implemented in C++
  - KOJAK 1.0 version was in Python
  - We still maintain a development version in Python to validate design changes
- Uses EARL library to access event trace

EARL Library

- Provides random access to individual events
- Computes links between corresponding events
  - E.g., From RECV to SEND event
- Identifies groups of events that represent an aspect of the program's execution state
  - E.g., all SEND events of messages in transit at a given moment
- Implemented in C++
  - Makes extensive use of STL
- Language bindings
  - C++
  - Python

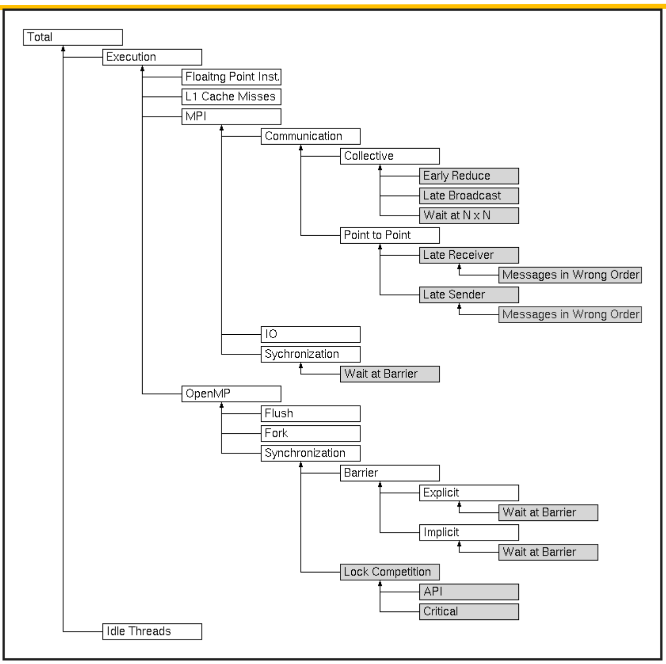
Pattern Specification

- Pattern
  - Compound event
  - Set of primitive events (= constitutents)
  - Relationships between constituents
  - Constraints
- Patterns specified as C++ class
  - Provides callback method to be called upon occurrence of a specific event type in event stream (root event)
  - Uses links or state information to find remaining constituents
  - Calculates (call path, location) matrix containing the time spent on a specific behavior in a particular (call path, location) pair
  - Location can be a process or a thread

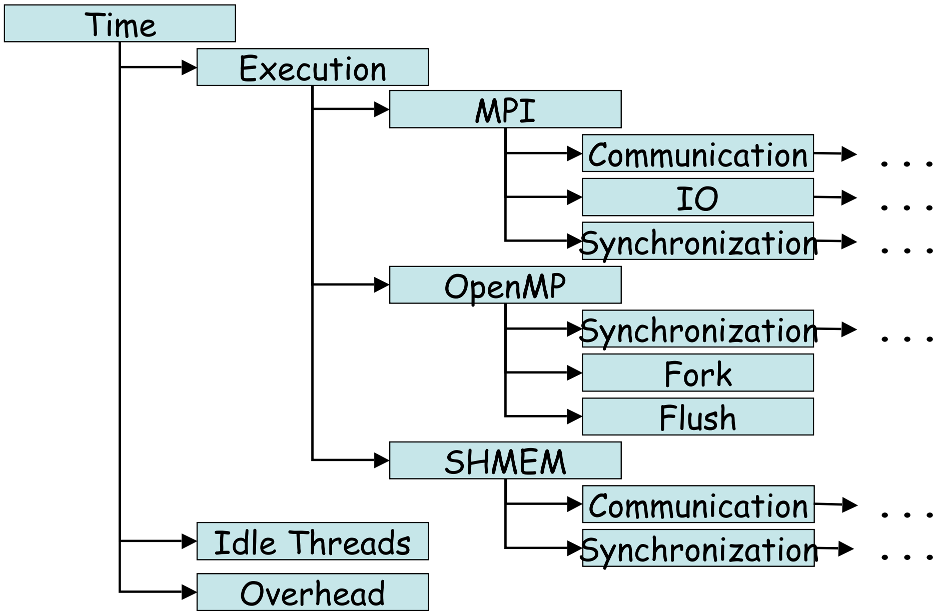
Pattern Specification (2)

- Two types of patterns
- Profiling patterns
  - Simple profiling information
    - E.g.,How much time was spent in MPI calls?
  - Described by pairs of events
    - ENTER and EXIT of certain routine (e.g., MPI)
- Patterns describing complex inefficiency situations
  - Usually described by more than two events
  - E.g., late sender or synchronization before all-to-all operations
- All patterns are arranged in an inclusion hierarchy
  - Inclusion of execution-time interval sets exhibiting the performance behavior
  - E.g., execution time includes communication time

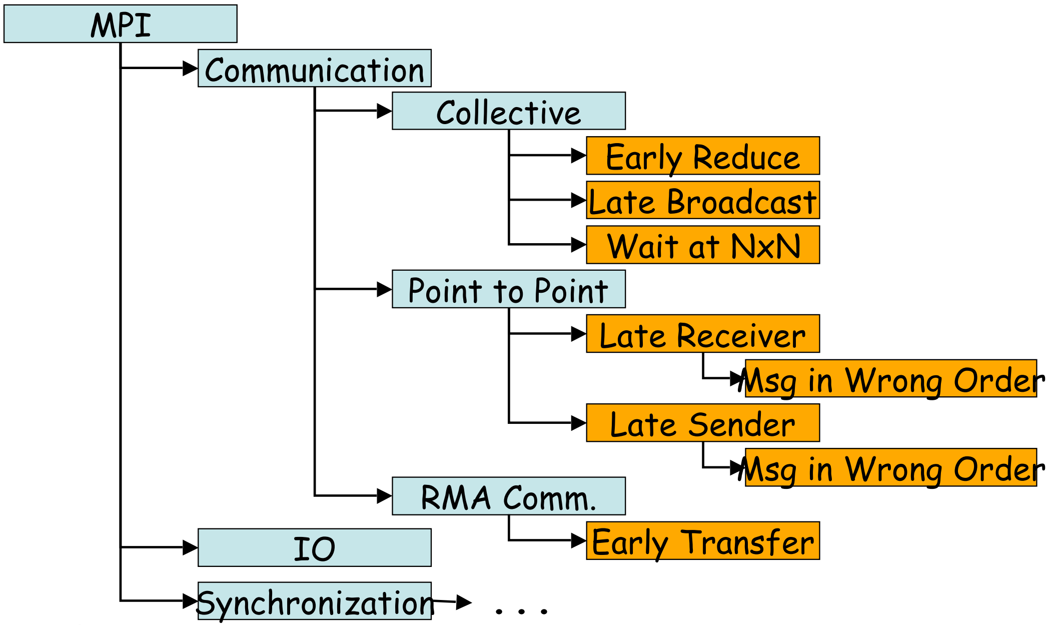
Pattern Hierarchy



KOJAK: Basic Pattern Hierarchy

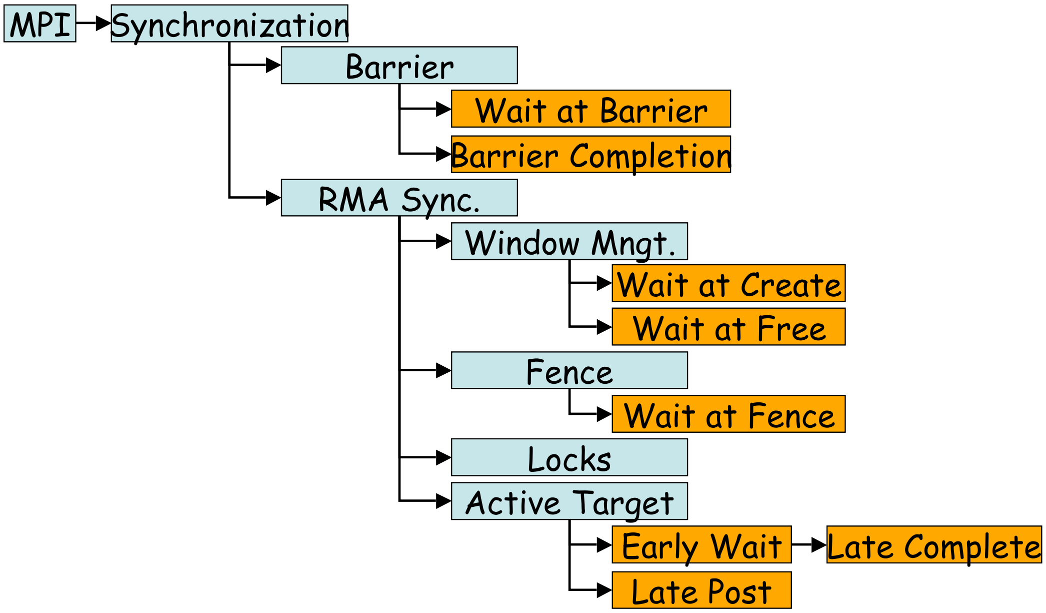


KOJAK: MPI Pattern Hierarchy I



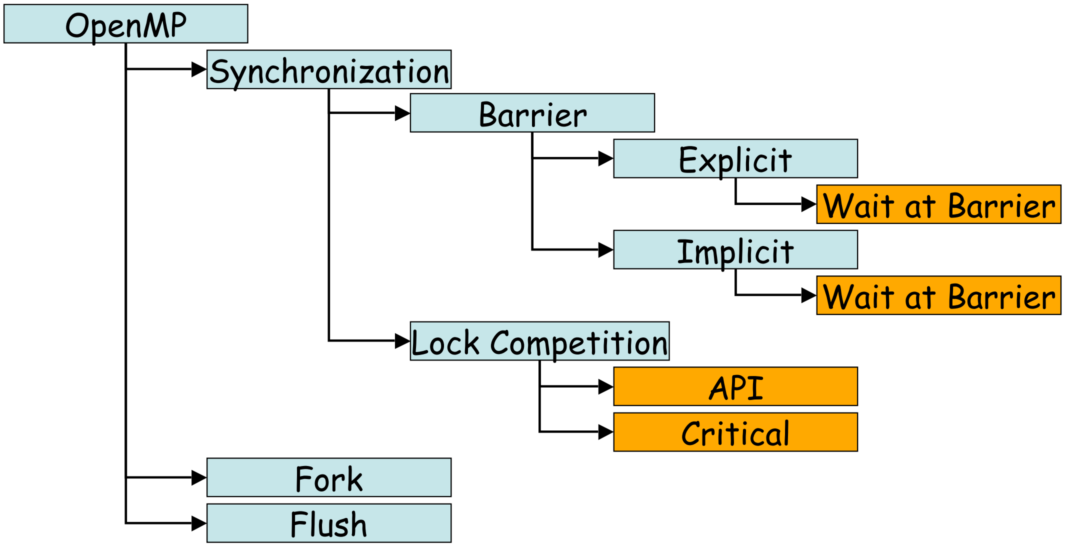
- MPI
  - Communication
    - Collective
      - Early Reduce
      - Late Broadcast
      - Wait at NxN
    - Point to Point
      - Late Receiver
        - Msg in Wrong Order
      - Late Sender
        - Msg in Wrong Order
    - RMA Comm.
      - Early Transfer
  - IO
  - Synchronization → . . .

KOJAK: MPI Pattern Hierarchy II



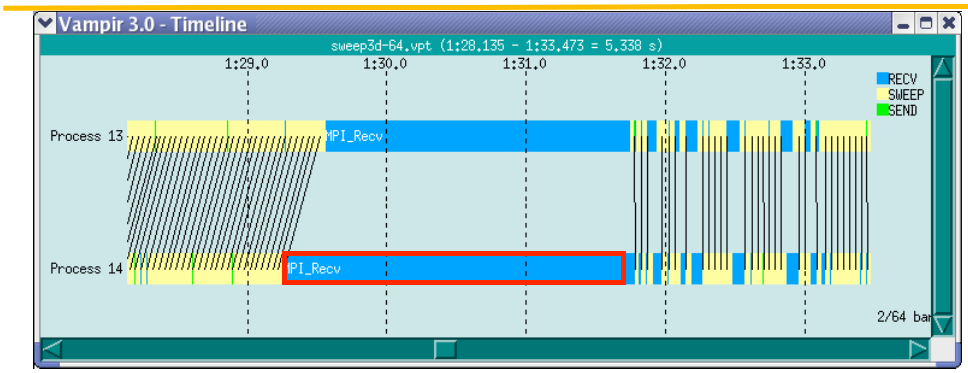
- MPI → Synchronization
  - Barrier
    - Wait at Barrier
    - Barrier Completion
  - RMA Sync.
    - Window Mngt.
      - Wait at Create
      - Wait at Free
    - Fence
      - Wait at Fence
    - Locks
    - Active Target
      - Early Wait → Late Complete
      - Late Post

KOJAK: OpenMP Pattern Hierarchy



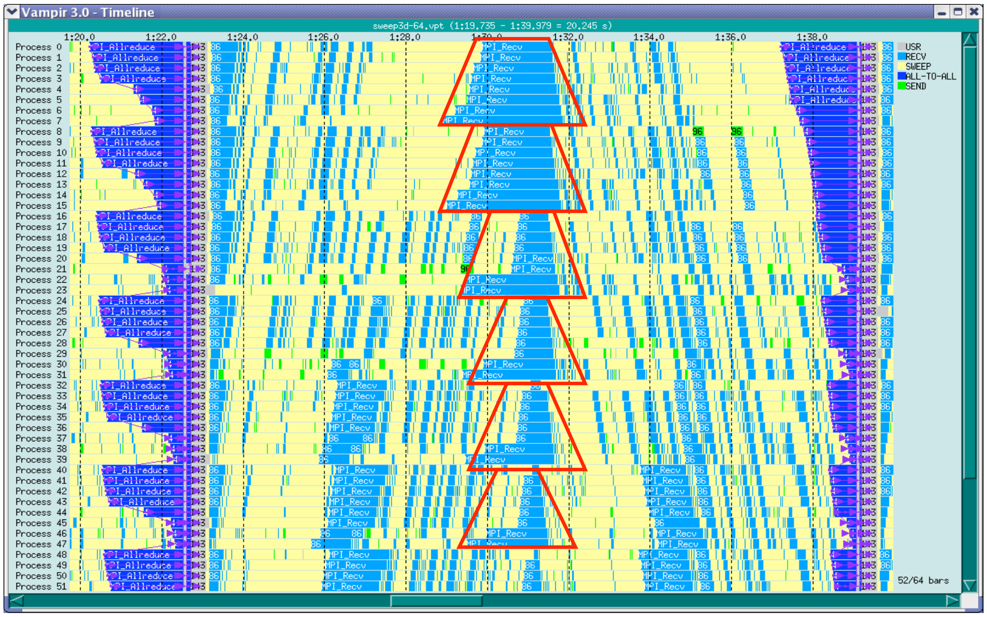
- OpenMP
  - Synchronization
    - Barrier
      - Explicit
        - Wait at Barrier
      - Implicit
        - Wait at Barrier
    - Lock Competition
      - API
      - Critical
  - Fork
  - Flush

Micro-patterns



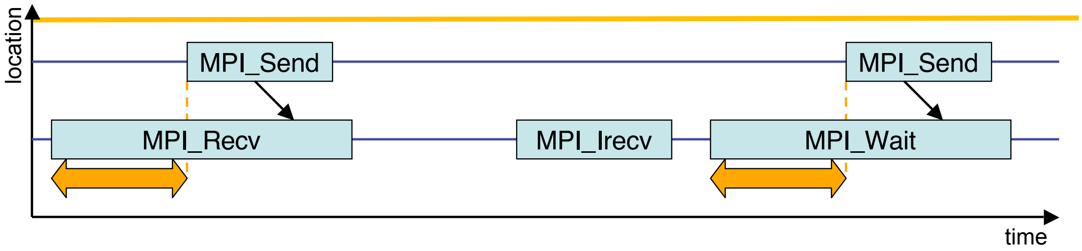
- Undesired wait states as a result of untimely arrival of processes or threads at synchronization points
  - Late sender
  - Wait at n-to-n

Macro-patterns

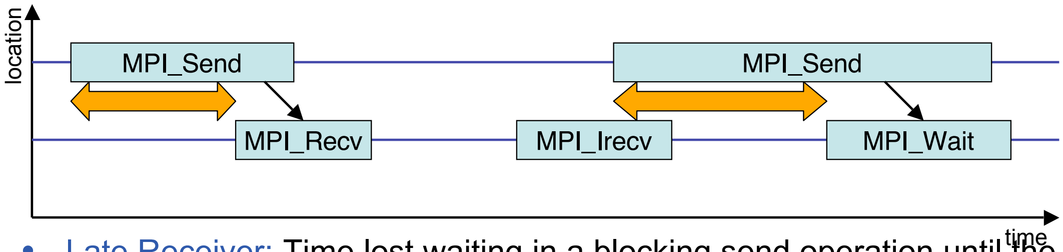


ParaTools

KOJAK MPI-1 Pattern: Late Sender / Receiver



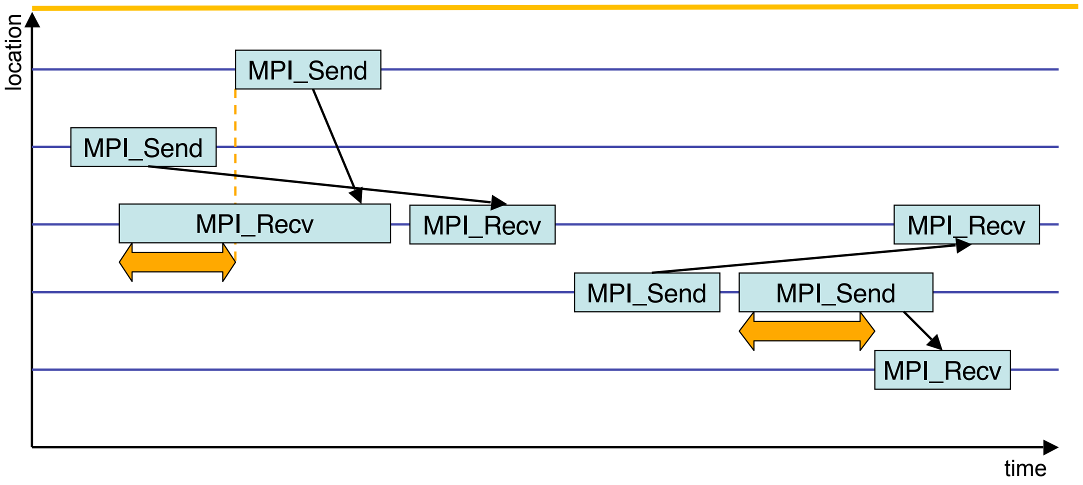
- Late Sender: Time lost waiting caused by a blocking receive operation posted earlier than the corresponding send operation



- Late Receiver: Time lost waiting in a blocking send operation until the corresponding receive operation is called

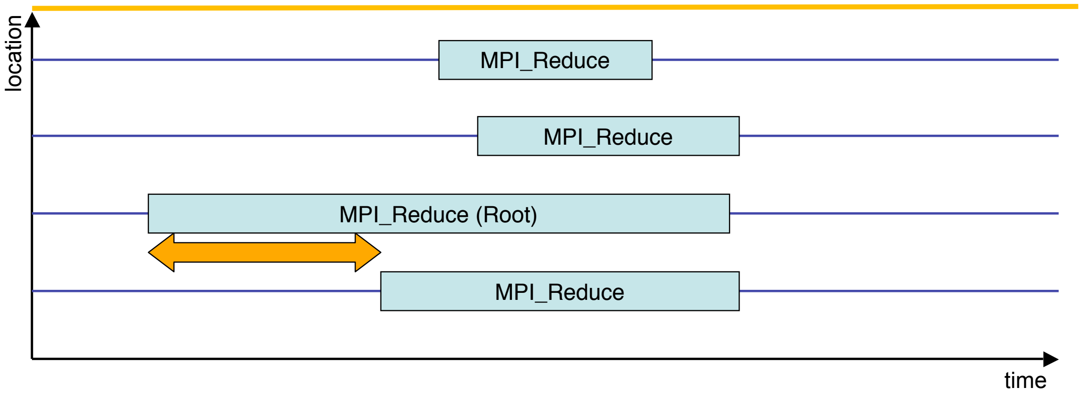
ParaTools

KOJAK MPI-1 Pattern: Wrong Order



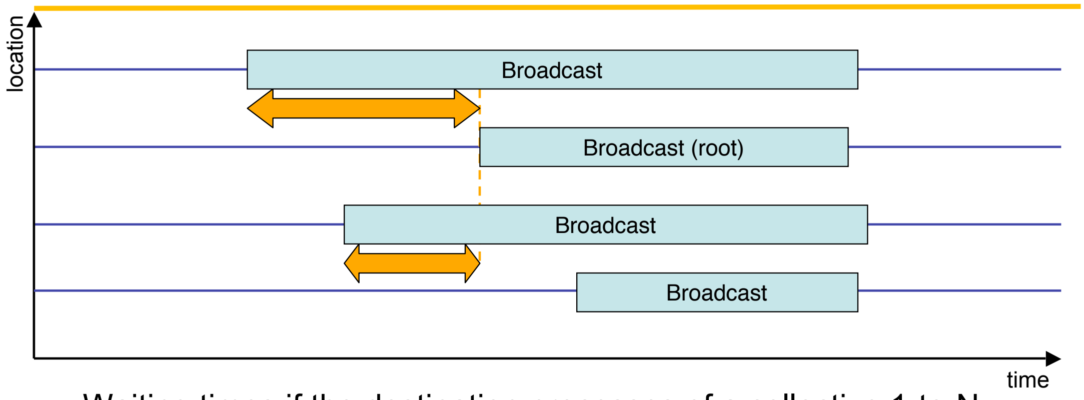
- Late Sender / Receiver patterns caused by messages received/sent in wrong order
- Sub patterns of Late Sender / Receiver

KOJAK MPI-1 Collective Pattern: Early Reduce



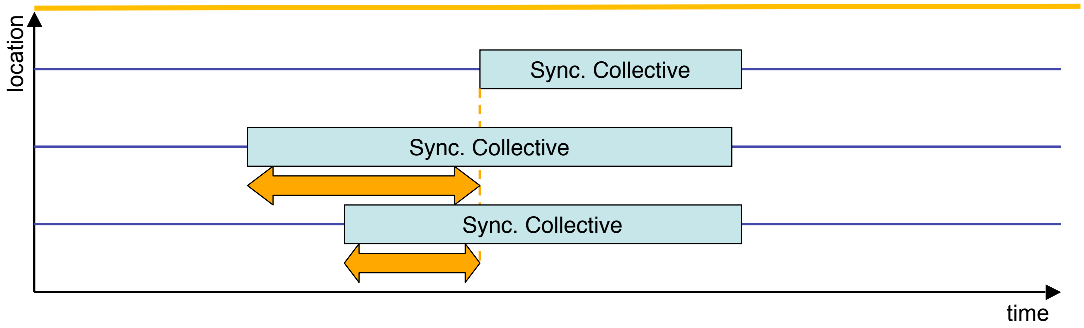
- Waiting time if the destination process (root) of a collective N-to-1 communication operation enters the operation earlier than its sending counterparts
- Applies to MPI calls MPI_Reduce(), MPI_Gather(), MPI_Gatherv()

KOJAK Collective Pattern: Late Broadcast



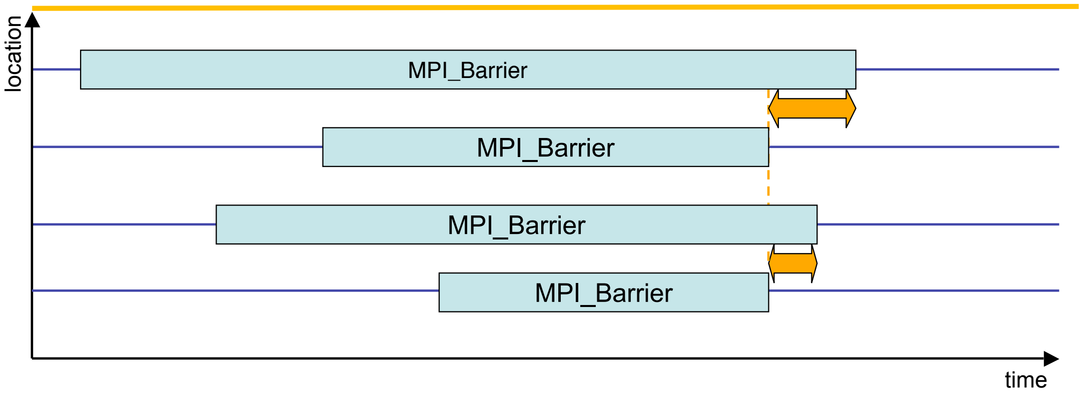
- Waiting times if the destination processes of a collective 1-to-N communication operation enter the operation earlier than the source process (root)
- MPI-1: Applies to MPI_Bcast(), MPI_Scatter(), MPI_Scatterv()
- SHMEM: Applies to shmem_broadcast()

KOJAK Generic Pattern: Wait at



- Time spent waiting in front of a collective synchronizing operation call until the last process reaches the operation
- Pattern instances:
  - Wait at NxN (MPI)
  - Wait at Barrier (MPI)
  - Wait at NxN (SHMEM)
  - Wait at Barrier (SHMEM)
  - Wait at Barrier (OpenMP)
  - Wait at Create (MPI-2)
  - Wait at Free (MPI-2)
  - Wait at Fence (MPI-2)

KOJAK MPI-1 Collective Pattern: Barrier Completion



- Time spent in MPI barriers after the first process has left the operation

Basic Search Strategy

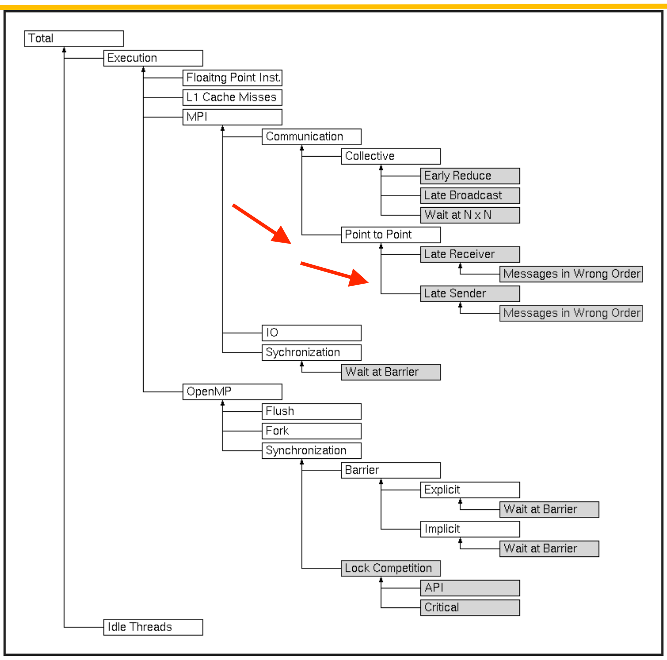
- Register each pattern for specific event type
  - Type of root event
- Read the trace file one from the beginning to the end
  - Depending on the type of the current event
    - Invoke callback method of pattern classes registered for it
  - Callback method
    - Accesses additional events to identify remaining constituents
    - To do this it may follow links or obtain state information
- Pattern from an implementation viewpoint
  - Set of events hold together by links and state-set boundaries

Observation
Many patterns describe related phenomena

Improved Search Strategy in KOJAK 2.0

- Exploit specialization relationships among different patterns
- Pass on compound-event instances from more general pattern (class) to more specific pattern (class)
  - Along a path in the pattern hierarchy
- Previous implementation
  - Patterns could register only for primitive events (e.g., RECV)
- New implementation
  - Patterns can publish compound events
  - Patterns can register for primitive events and compound events

Pathway of Example Pattern



- Total
  - Execution
    - Floaitng Point Inst.
    - L1 Cache Misses
    - MPI
      - Communication
        - Collective
          - Early Reduce
          - Late Broadcast
          - Wait at N x N
        - Point to Point
          - Late Receiver
            - Messages in Wrong Order
          - Late Sender
            - Messages in Wrong Order
      - IO
      - Sychronization
        - Wait at Barrier
    - OpenMP
      - Flush
      - Fork
      - Synchronization
        - Barrier
          - Explicit
            - Wait at Barrier
          - Implicit
            - Wait at Barrier
        - Lock Competition
          - API
          - Critical
  - Idle Threads

To use KOJAK with TAU

- Choose a TAU stub makefile with a -epilog in its name

  ```
  % setenv TAU_MAKEFILE $(TAU)/x86_64/lib/Makefile.tau-mpi-pdt-epilog-trace-pgi
  ```

- Change CC to tau_cc.sh, F90 to tau_f90.sh in your Makefile
- Build and execute the program

  ```
  % make; mpirun -np 6 sweep3d.mpi
  ```

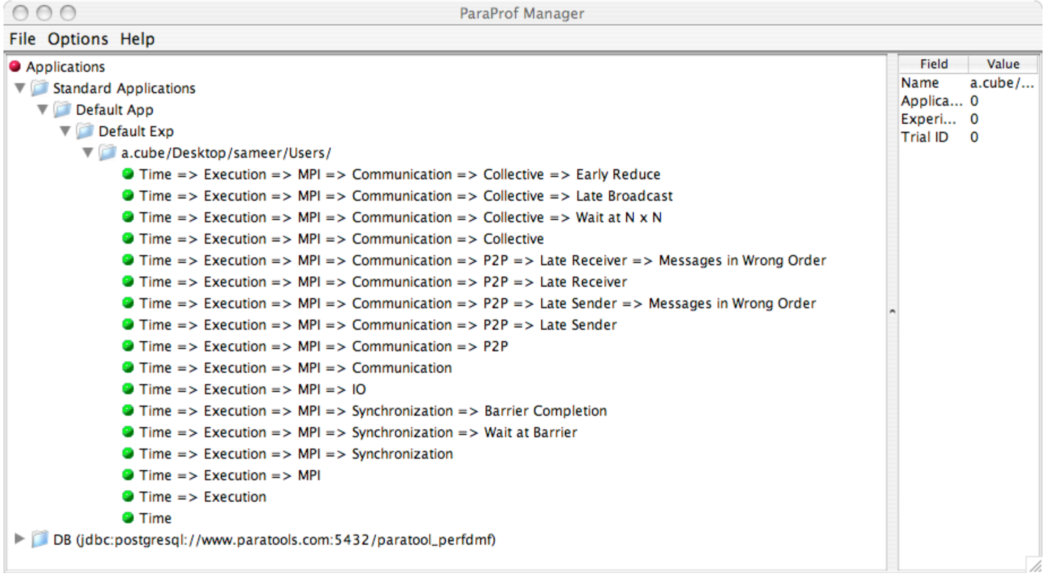
- Run the Expert tool on the generated a.elg merged trace file

  ```
  % expert a.elg
  ```

- Load the generated a.cube file in Paraprof and click on metrics of interest

  ```
  % paraprof a.cube
  ```

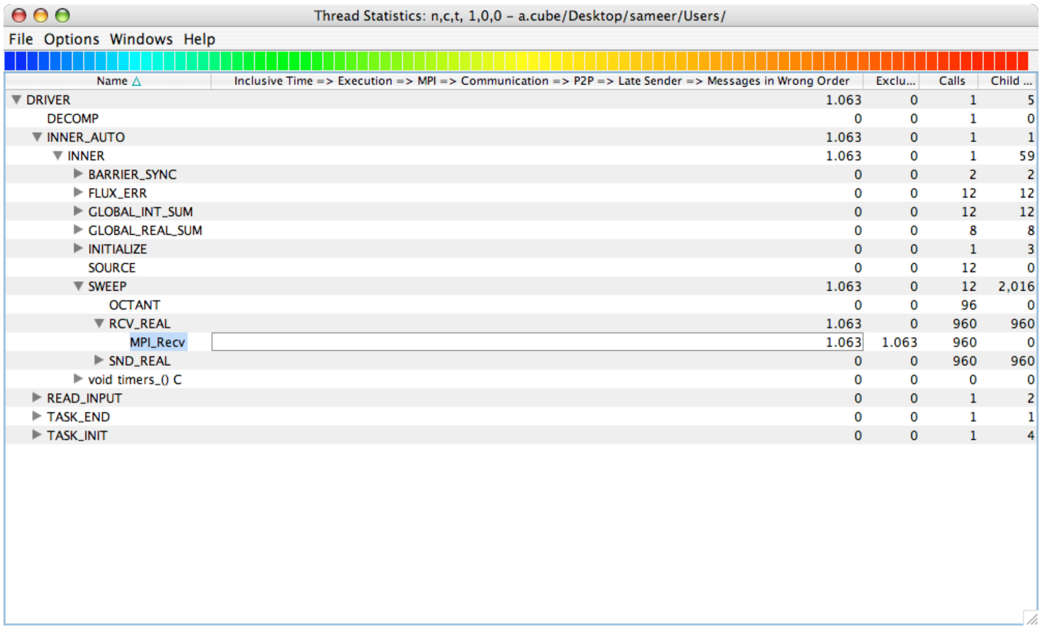
ParaProf: Performance Bottlenecks



```
● Applications
▼ Standard Applications
  ▼ Default App
    ▼ Default Exp
      ▼ a.cube/Desktop/sameer/Users/
          ● Time => Execution => MPI => Communication => Collective => Early Reduce
          ● Time => Execution => MPI => Communication => Collective => Late Broadcast
          ● Time => Execution => MPI => Communication => Collective => Wait at N x N
          ● Time => Execution => MPI => Communication => Collective
          ● Time => Execution => MPI => Communication => P2P => Late Receiver => Messages in Wrong Order
          ● Time => Execution => MPI => Communication => P2P => Late Receiver
          ● Time => Execution => MPI => Communication => P2P => Late Sender => Messages in Wrong Order
          ● Time => Execution => MPI => Communication => P2P => Late Sender
          ● Time => Execution => MPI => Communication => P2P
          ● Time => Execution => MPI => Communication
          ● Time => Execution => MPI => IO
          ● Time => Execution => MPI => Synchronization => Barrier Completion
          ● Time => Execution => MPI => Synchronization => Wait at Barrier
          ● Time => Execution => MPI => Synchronization
          ● Time => Execution => MPI
          ● Time => Execution
          ● Time
▶ DB (jdbc:postgresql://www.paratools.com:5432/paratool_perfdmf)
```

Field	Value
Name	a.cube/...
Applica...	0
Experi...	0
Trial ID	0

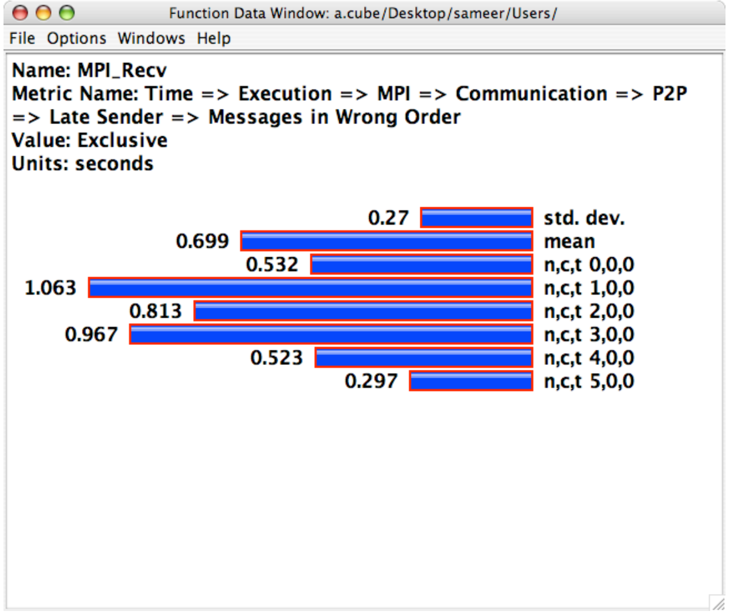
ParaProf: Time spent in late sender bottleneck



Thread Statistics: n,c,t, 1,0,0 – a.cube/Desktop/sameer/Users/

Name △	Inclusive Time => Execution => MPI => Communication => P2P => Late Sender => Messages in Wrong Order	Exclu...	Calls	Child ...
▼ DRIVER	1.063	0	1	5
DECOMP	0	0	1	0
▼ INNER_AUTO	1.063	0	1	1
▼ INNER	1.063	0	1	59
▶ BARRIER_SYNC	0	0	2	2
▶ FLUX_ERR	0	0	12	12
▶ GLOBAL_INT_SUM	0	0	12	12
▶ GLOBAL_REAL_SUM	0	0	8	8
▶ INITIALIZE	0	0	1	3
SOURCE	0	0	12	0
▼ SWEEP	1.063	0	12	2,016
OCTANT	0	0	96	0
▼ RCV_REAL	1.063	0	960	960
MPI_Recv	1.063	1.063	960	0
▶ SND_REAL	0	0	960	960
▶ void timers_() C	0	0	0	0
▶ READ_INPUT	0	0	1	2
▶ TASK_END	0	0	1	1
▶ TASK_INIT	0	0	1	4

ParaProf: Time spent in late sender bottleneck



Function Data Window: a.cube/Desktop/sameer/Users/

Name: MPI_Recv
Metric Name: Time => Execution => MPI => Communication => P2P => Late Sender => Messages in Wrong Order
Value: Exclusive
Units: seconds

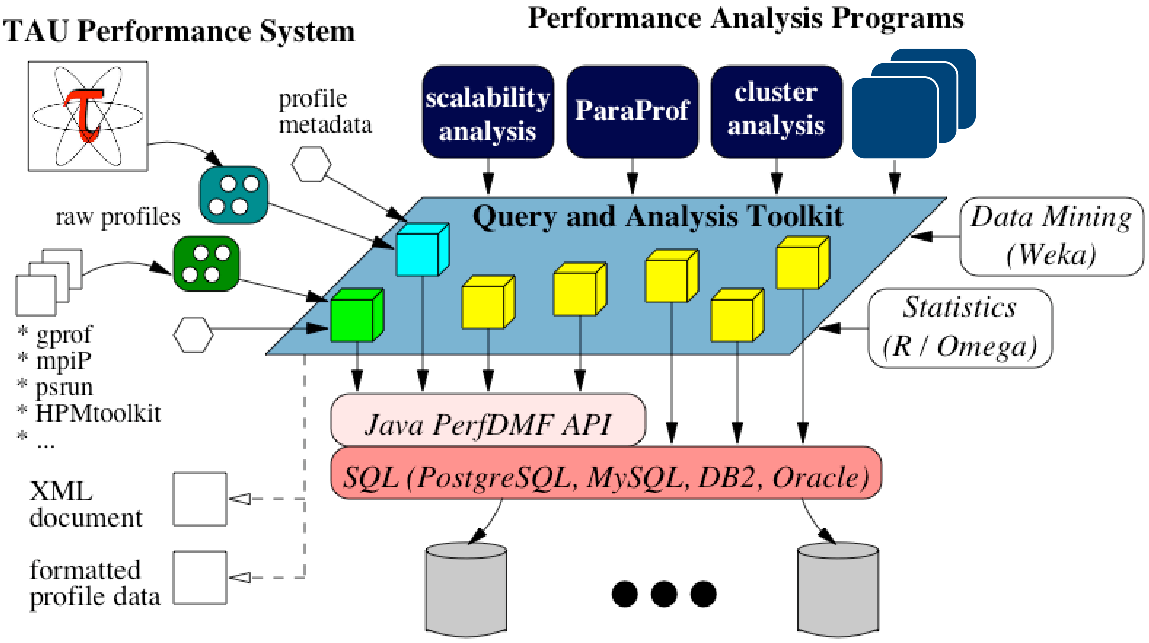
Labs!

Lab: TAU and KOJAK

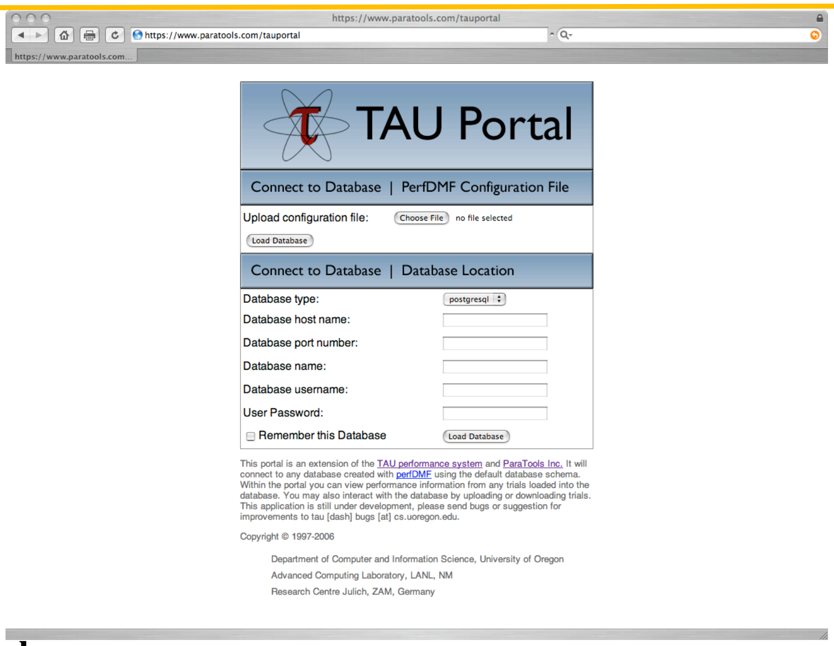
Lab Instructions

- See workshop example:
- Sweep3D

PerfDMF: Performance Data Mgmt. Framework



TAU Portal - www.paratools.com/tauportal


https://www.paratools.com/tauportal
https://www.paratools.com/tauportal
https://www.paratools.com...

TAU Portal

Connect to Database | PerfDMF Configuration File

Upload configuration file: Choose File no file selected

Load Database

Connect to Database | Database Location

Database type: postgresql

Database host name:

Database port number:

Database name:

Database username:

User Password:

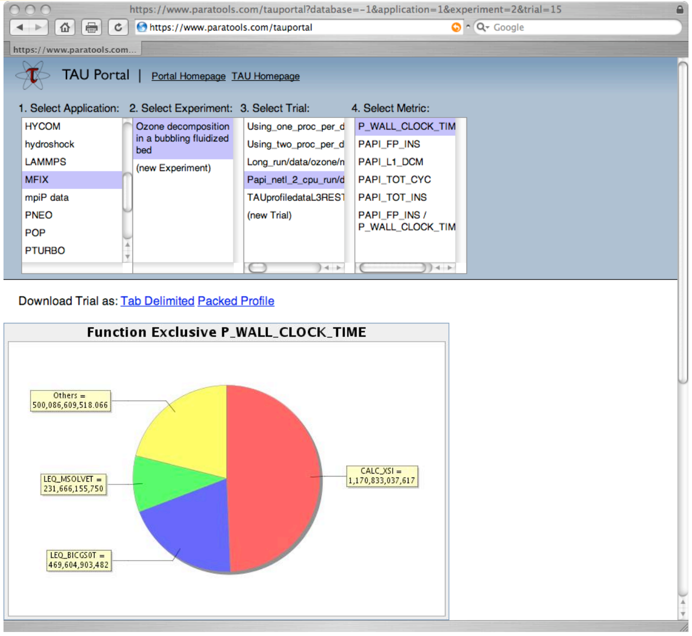
□ Remember this Database Load Database

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



https://www.paratools.com/tauportal?database=-1&application=1&experiment=2&trial=15

TAU Portal | Portal Homepage TAU Homepage

1. Select Application: MFIX
2. Select Experiment: Ozone decomposition in a bubbling fluidized bed
3. Select Trial: Papi_netl_2_cpu_run/d
4. Select Metric: P_WALL_CLOCK_TIM

Download Trial as: Tab Delimited Packed Profile

Function Exclusive P_WALL_CLOCK_TIME

Using Performance Database (PerfDMF)

- Configure PerfDMF (Done by each user)

  % perfdmf_configure

  - 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

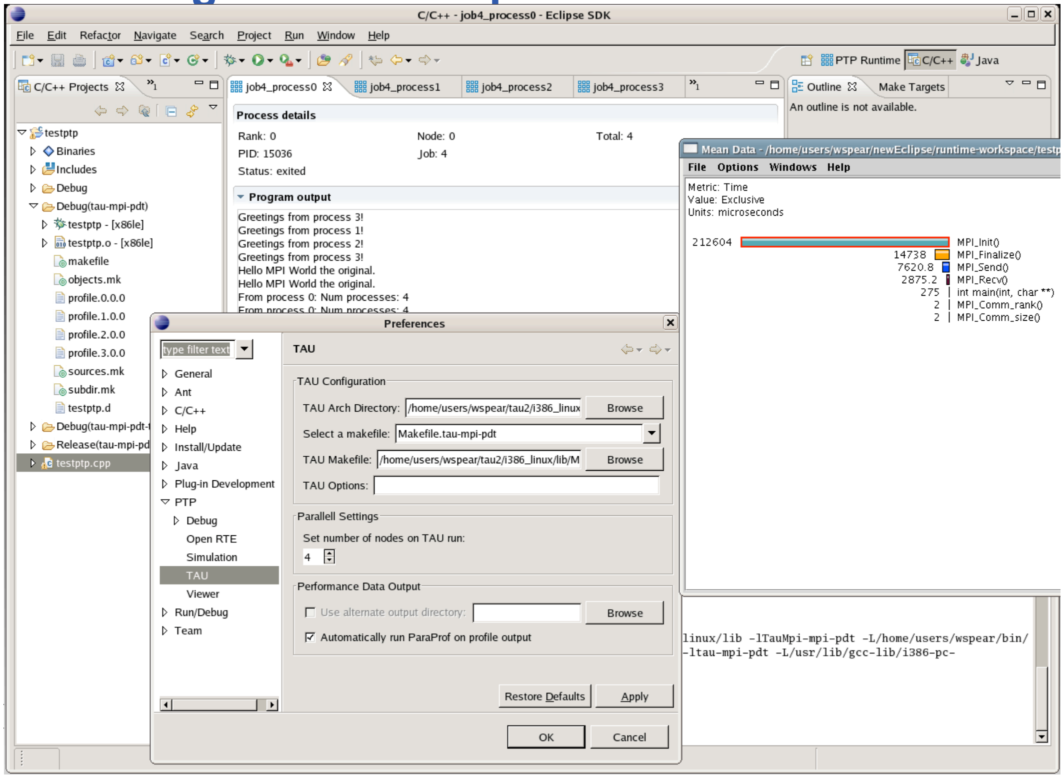
- Configure PerfExplorer (Done by each user)

  % perfexplorer_configure

- Execute PerfExplorer

  % perfexplorer

TAU integration in Eclipse PTP IDE



An outline is not available.

Process details

Rank: 0	Node: 0	Total: 4
PID: 15036	Job: 4
Status: exited

Program output

```
Greetings from process 3!
Greetings from process 1!
Greetings from process 2!
Greetings from process 3!
Hello MPI World the original.
Hello MPI World the original.
From process 0: Num processes: 4
From process 0: Num processes: 4
```

Mean Data - /home/users/wspear/newEclipse/runtime-workspace/testp

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

212604          MPI_Init()
 14738          MPI_Finalize()
 7620.8         MPI_Send()
 2875.2         MPI_Recv()
    275         int main(int, char **)
      2         MPI_Comm_rank()
      2         MPI_Comm_size()
```

```
linux/lib -lTauMpi-mpi-pdt -L/home/users/wspear/bin/
-ltau-mpi-pdt -L/usr/lib/gcc-lib/i386-pc-
```

Preferences

TAU

TAU Configuration

TAU Arch Directory: /home/users/wspear/tau2/i386_linux Browse

Select a makefile: Makefile.tau-mpi-pdt

TAU Makefile: /home/users/wspear/tau2/i386_linux/lib/M Browse

TAU Options:

Parallell Settings

Set number of nodes on TAU run:

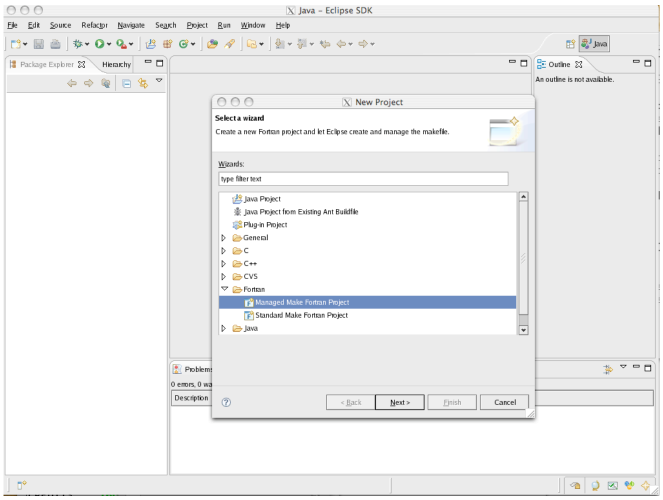
4

Performance Data Output

☐ Use alternate output directory: Browse

☑ Automatically run ParaProf on profile output

Getting Started with Eclipse PTP!



An outline is not available.

0 errors, 0 wa

New Project

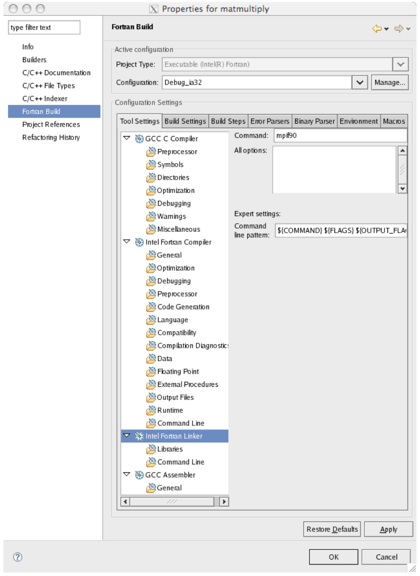
Select a wizard

Create a new Fortran project and let Eclipse create and manage the makefile.

Wizards:

type filter text

- Java Project
- Java Project from Existing Ant Buildfile
- Plug-in Project
- General
- C
- C++
- CVS
- Fortran
  - Managed Make Fortran Project
  - Standard Make Fortran Project
- Java



Fortran Build

Active configuration

Project Type: Executable (Intel(R) Fortran)

Configuration: Debug_ia32 Manage...

Configuration Settings

- GCC C Compiler
  - Preprocessor
  - Symbols
  - Directories
  - Optimization
  - Debugging
  - Warnings
  - Miscellaneous
- Intel Fortran Compiler
  - General
  - Optimization
  - Debugging
  - Preprocessor
  - Code Generation
  - Language
  - Compatibility
  - Compilation Diagnostic
  - Data
  - Floating Point
  - External Procedures
  - Output Files
  - Runtime
  - Command Line
- Intel Fortran Linker
  - Libraries
  - Command Line
- GCC Assembler
  - General

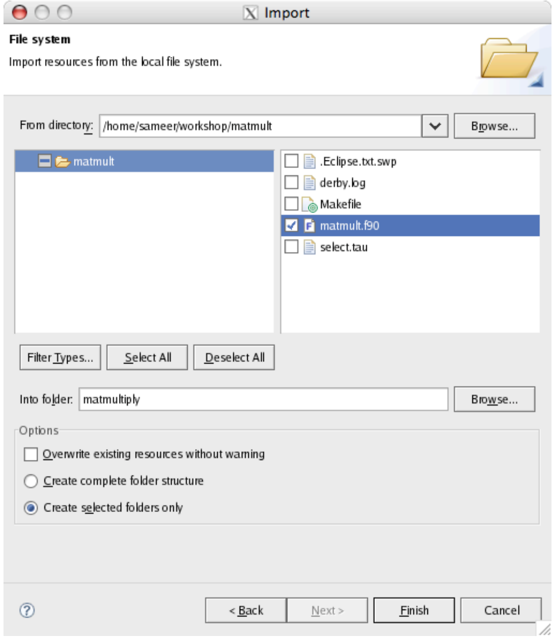
Command: mpif90

All options:

Expert settings:

Command line pattern: ${COMMAND} ${FLAGS} ${OUTPUT_FLA

Eclipse PTP - Importing files



Import

File system

Import resources from the local file system.

From directory: /home/sameer/workshop/matmult

matmult

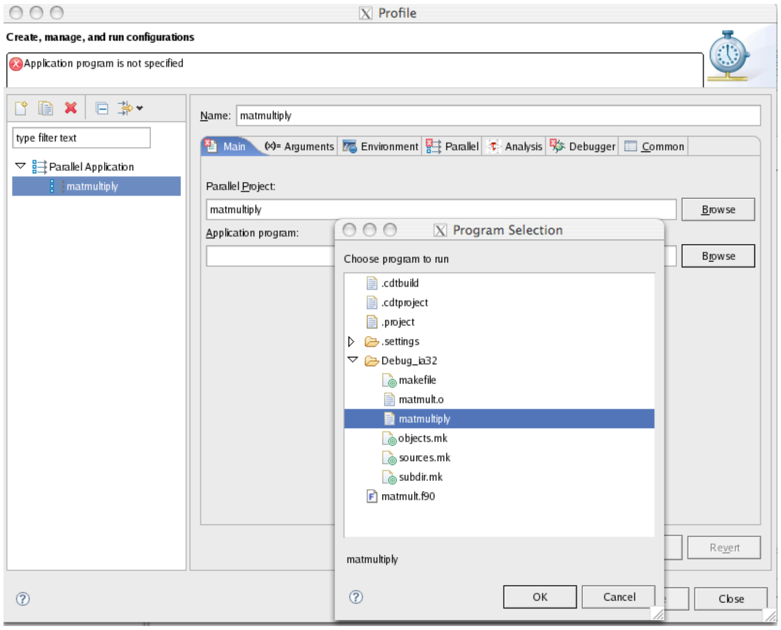
- □ .Eclipse.txt.swp
- □ derby.log
- □ Makefile
- ☑ matmult.f90
- □ select.tau

Into folder: matmultiply

Options

- □ Overwrite existing resources without warning
- ○ Create complete folder structure
- ● Create selected folders only

Program Selection in PTP



Profile

Create, manage, and run configurations

Application program is not specified

type filter text

- Parallel Application
  - matmultiply

Name: matmultiply

Main | Arguments | Environment | Parallel | Analysis | Debugger | Common

Parallel Project: matmultiply

Application program:

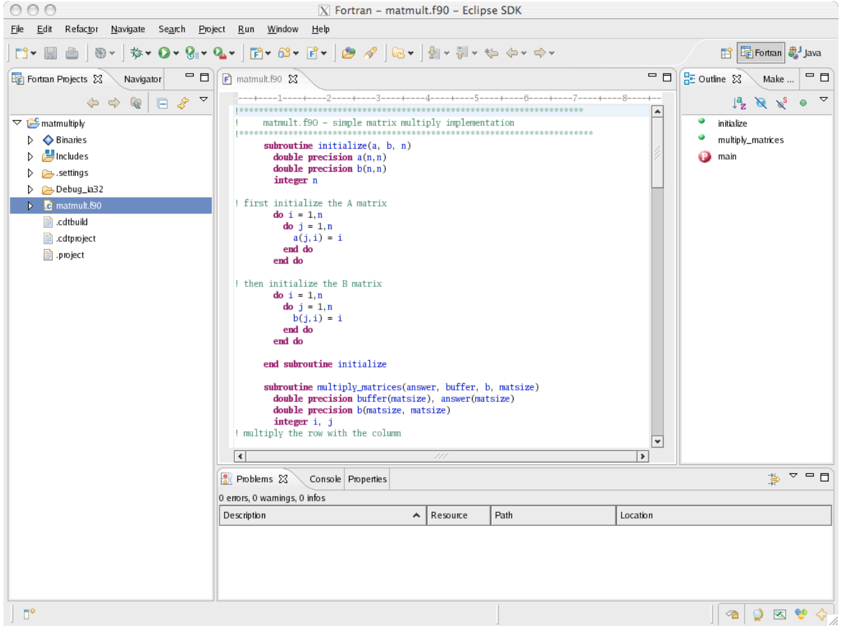
Program Selection

Choose program to run

- .cdtbuild
- .cdtproject
- .project
- .settings
- Debug_ia32
  - makefile
  - matmult.o
  - matmultiply
  - objects.mk
  - sources.mk
  - subdir.mk
- matmult.f90

matmultiply

Getting Started with Eclipse PTP!



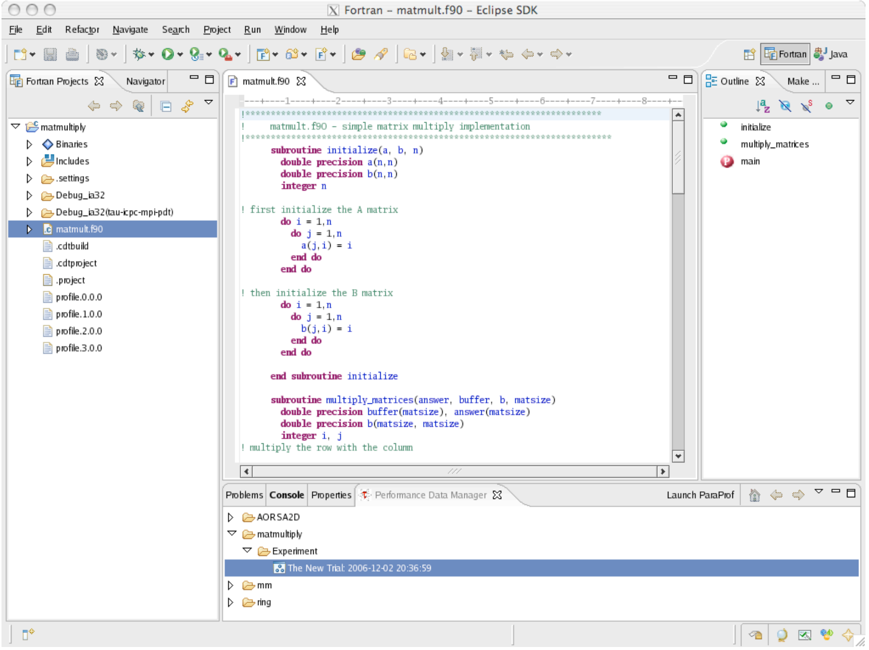
```
!******************************************************************
!     matmult.f90 - simple matrix multiply implementation
!**********************************************************************
      subroutine initialize(a, b, n)
        double precision a(n,n)
        double precision b(n,n)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

0 errors, 0 warnings, 0 infos

Description	Resource	Path	Location

ParaTools

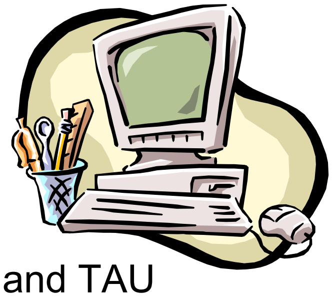
Performance Evaluation within PTP



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

ParaTools

Labs!



Lab: Eclipse PTP and TAU

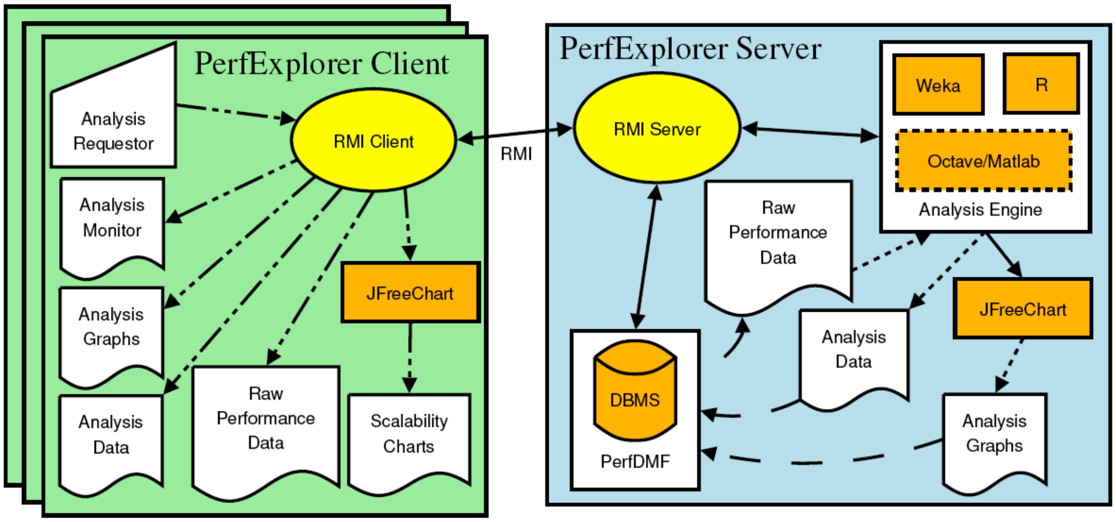
Lab Instructions

- See workshop examples:
- eclipse
- matmult

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

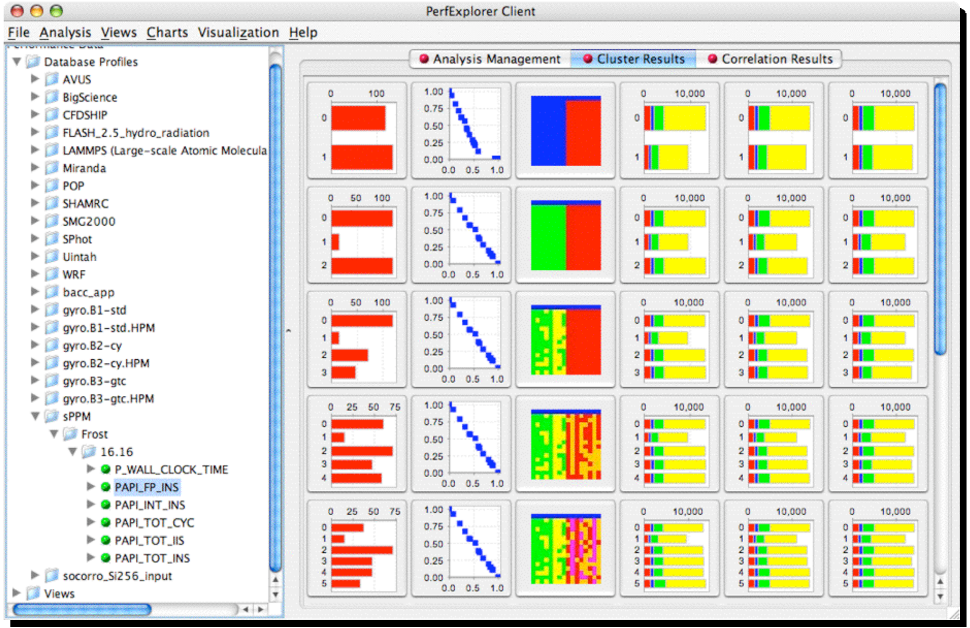
PerfExplorer - Cluster Analysis

- 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 - Comparative Analysis

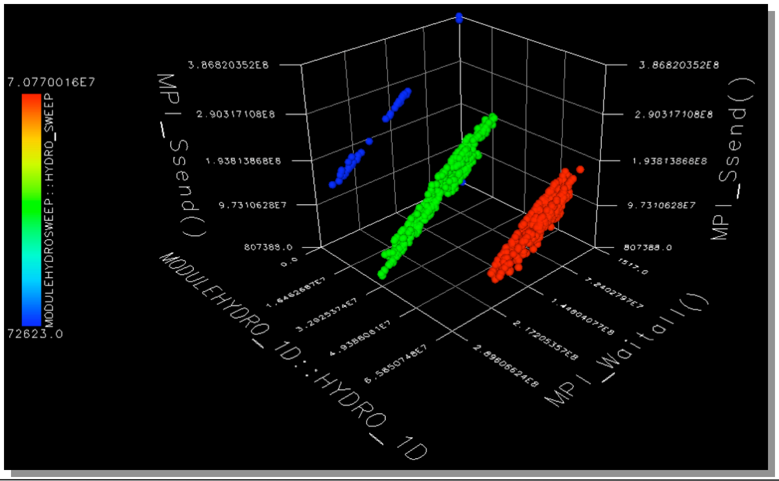
- Relative speedup, efficiency
  - total runtime, by event, one event, by phase
- Breakdown of total runtime
- Group fraction of total runtime
- Correlating events to total runtime
- Timesteps per second
- Performance Evaluation Research Center (PERC)
  - PERC tools study (led by ORNL, Pat Worley)
 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

PerfExplorer - Cluster Analysis (sPPM)



PerfExplorer - Cluster Analysis

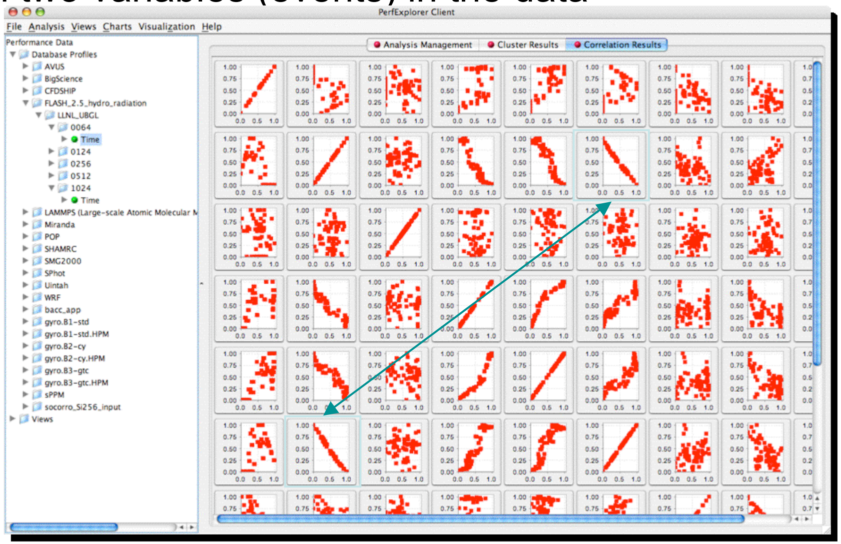
- Four significant events automatically selected (from 16K processors)
- Clusters and correlations are visible



ParaTools

PerfExplorer - Correlation Analysis (Flash)

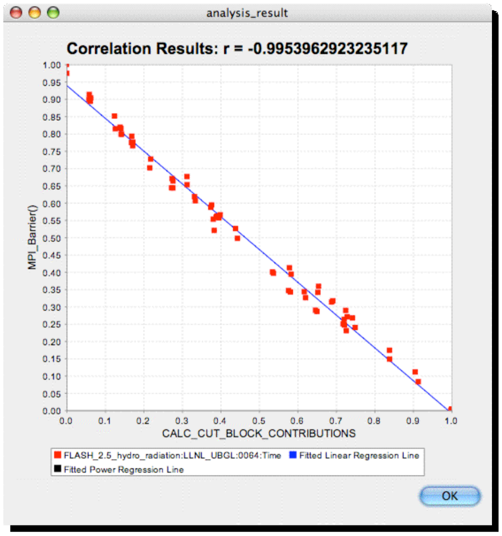
- Describes strength and direction of a linear relationship between two variables (events) in the data



ParaTools

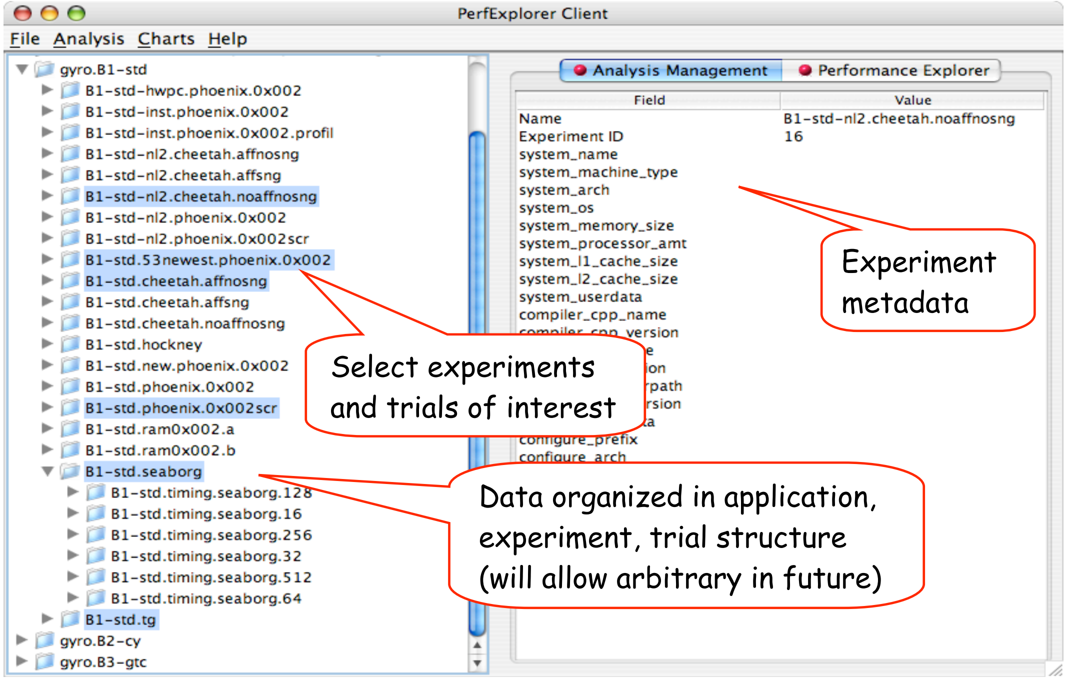
PerfExplorer - Correlation Analysis (Flash)

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



Correlation Results: r = -0.9953962923235117

PerfExplorer - Interface



- gyro.B1-std
  - B1-std-hwpc.phoenix.0x002
  - B1-std-inst.phoenix.0x002
  - B1-std-inst.phoenix.0x002.profil
  - B1-std-nl2.cheetah.affnosng
  - B1-std-nl2.cheetah.affsng
  - B1-std-nl2.cheetah.noaffnosng
  - B1-std-nl2.phoenix.0x002
  - B1-std-nl2.phoenix.0x002scr
  - B1-std.53newest.phoenix.0x002
  - B1-std.cheetah.affnosng
  - B1-std.cheetah.affsng
  - B1-std.cheetah.noaffnosng
  - B1-std.hockney
  - B1-std.new.phoenix.0x002
  - B1-std.phoenix.0x002
  - B1-std.phoenix.0x002scr
  - B1-std.ram0x002.a
  - B1-std.ram0x002.b
  - B1-std.seaborg
    - B1-std.timing.seaborg.128
    - B1-std.timing.seaborg.16
    - B1-std.timing.seaborg.256
    - B1-std.timing.seaborg.32
    - B1-std.timing.seaborg.512
    - B1-std.timing.seaborg.64
  - B1-std.tg
- gyro.B2-cy
- gyro.B3-gtc

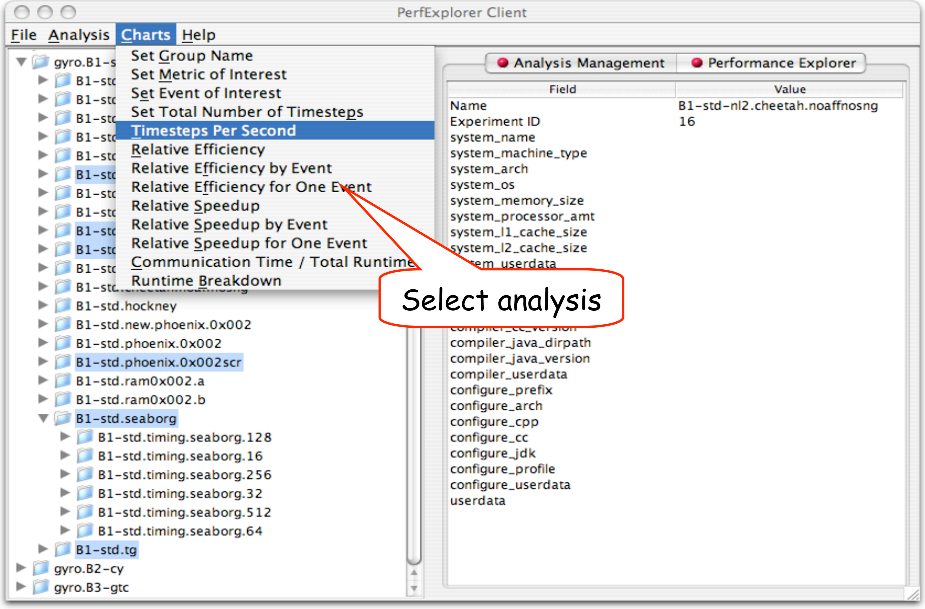
Field	Value
Name	B1-std-nl2.cheetah.noaffnosng
Experiment ID	16
system_name	
system_machine_type	
system_arch	
system_os	
system_memory_size	
system_processor_amt	
system_l1_cache_size	
system_l2_cache_size	
system_userdata	
compiler_cpp_name	
compiler_cpp_version	
e	
ion	
rpath	
rsion	
a	
configure_prefix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

Experiment metadata

Select experiments and trials of interest

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

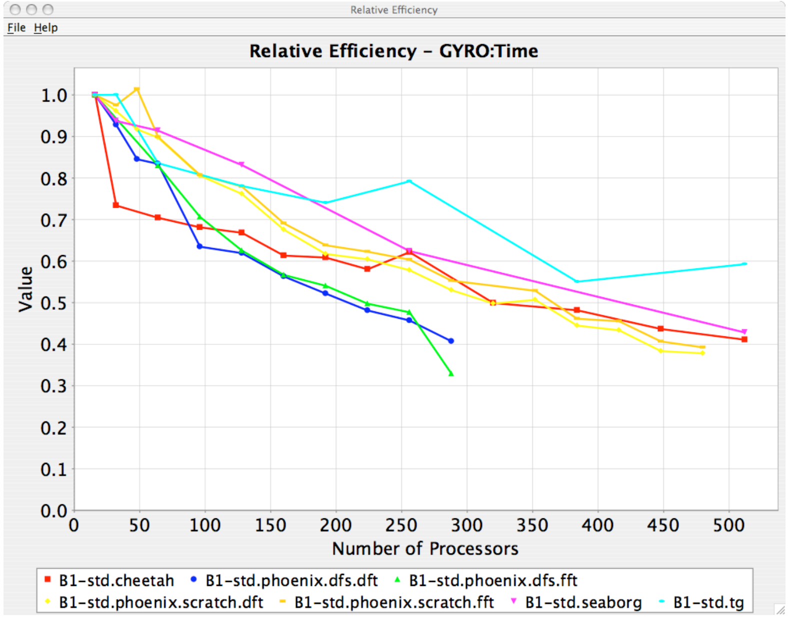
PerfExplorer - Interface



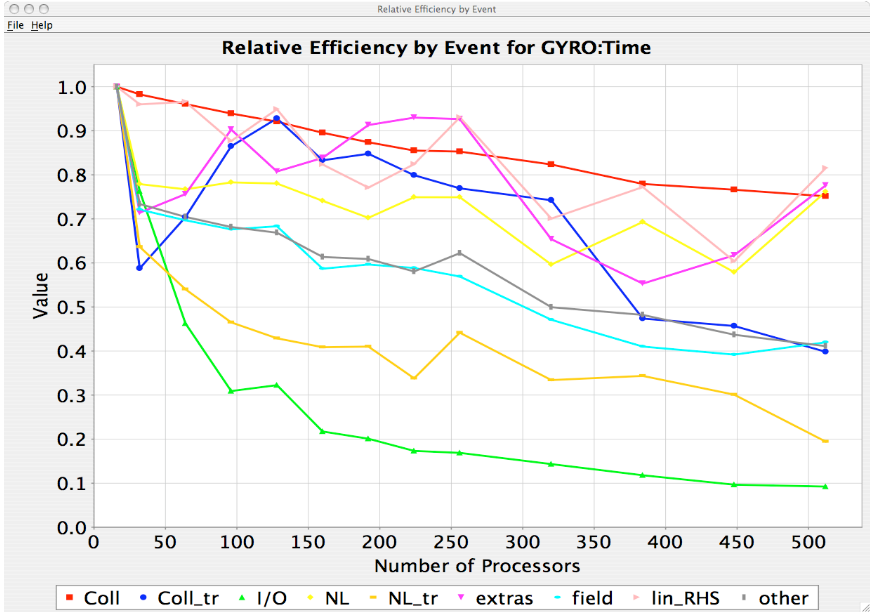
Field	Value
Name	B1-std-nl2.cheetah.noaffnosng
Experiment ID	16
system_name	
system_machine_type	
system_arch	
system_os	
system_memory_size	
system_processor_amt	
system_l1_cache_size	
system_l2_cache_size	
system_userdata	
compiler_cc_version	
compiler_java_dirpath	
compiler_java_version	
compiler_userdata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

Select analysis

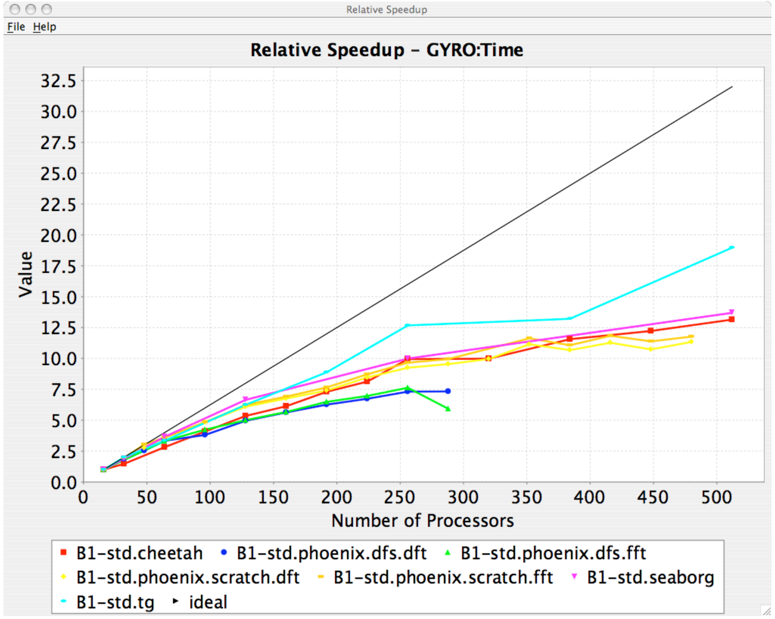
PerfExplorer - Relative Efficiency Plots



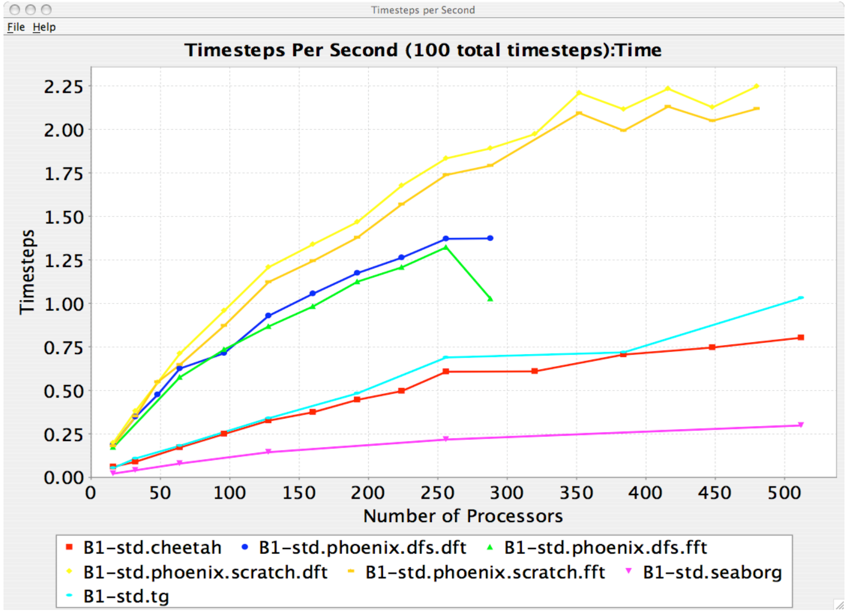
PerfExplorer - Relative Efficiency by Routine



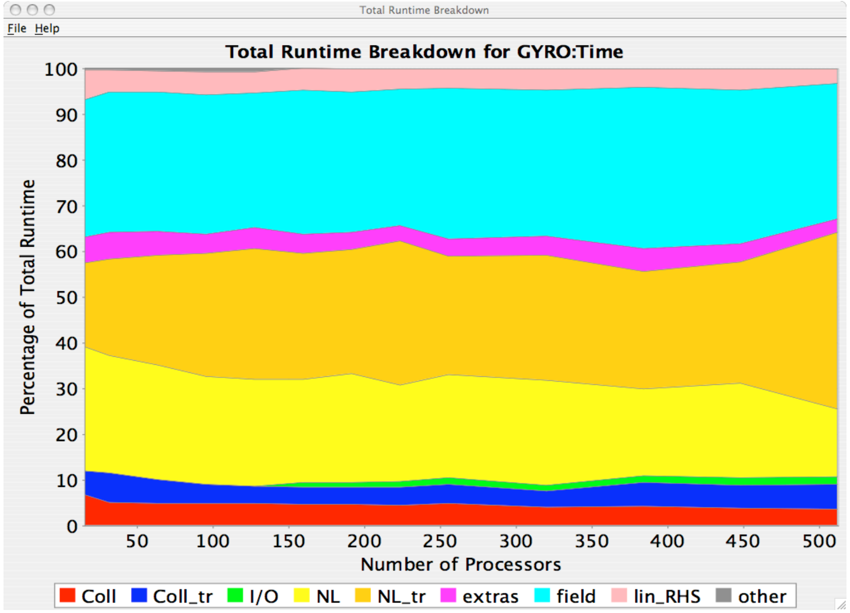
PerfExplorer - Relative Speedup



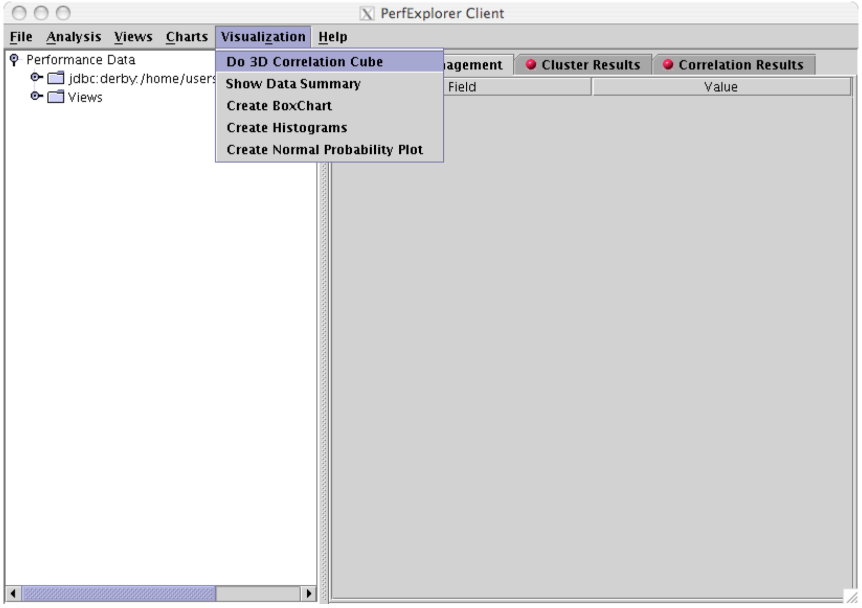
PerfExplorer - Timesteps Per Second



PerfExplorer - Runtime Breakdown

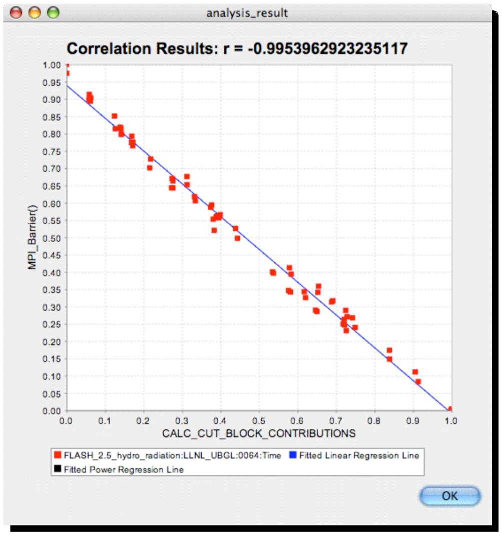


PerfExplorer - Correlation Studies



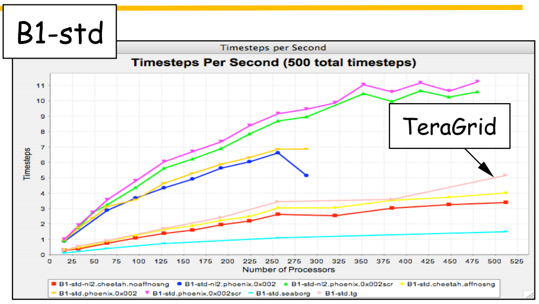
PerfExplorer - Correlation Analysis

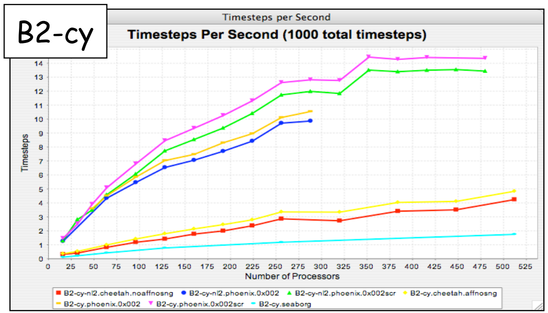
- -0.995 indicates strong, negative relationship
- As CALC_CUT_ BLOCK_CONTRIBUTIONS() increases in execution time, MPI_Barrier() decreases

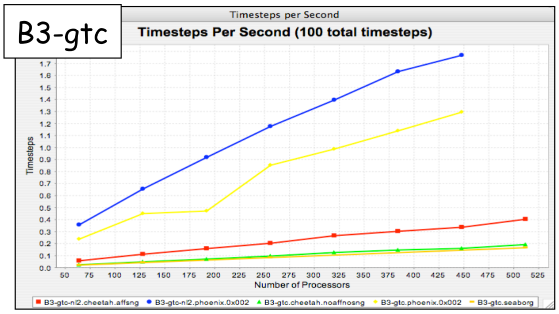


PerfExplorer - Timesteps per Second for GYRO

- 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

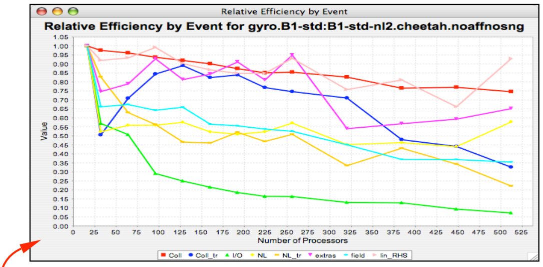


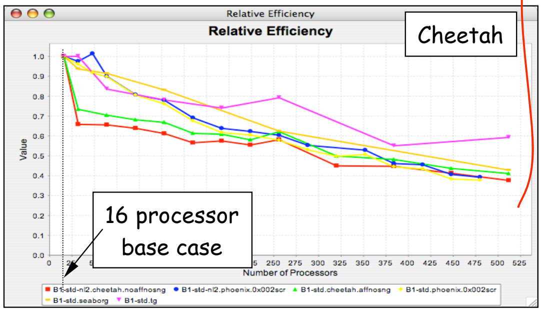


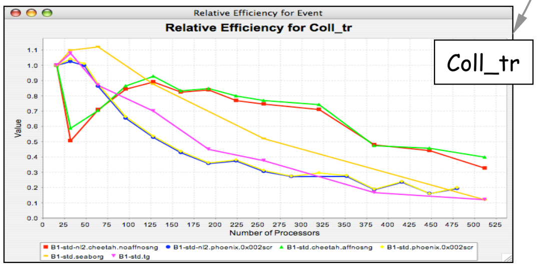


PerfExplorer - Relative Efficiency (B1-std)

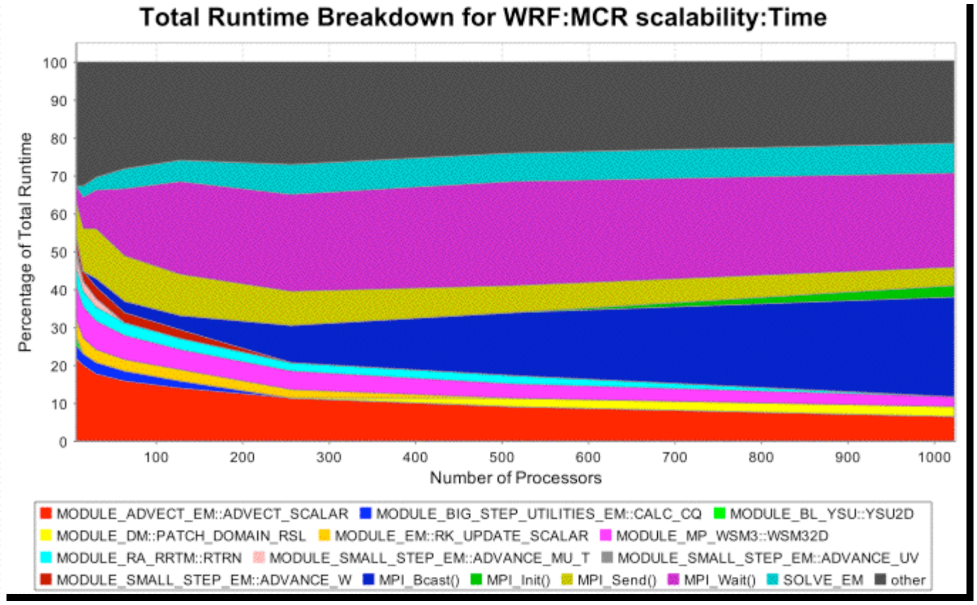
- By experiment (B1-std)
  - Total runtime (Cheetah (red))
- By event for one experiment
  - Coll_tr (blue) is significant
- By experiment for one event
  - Shows how Coll_tr behaves for all experiments



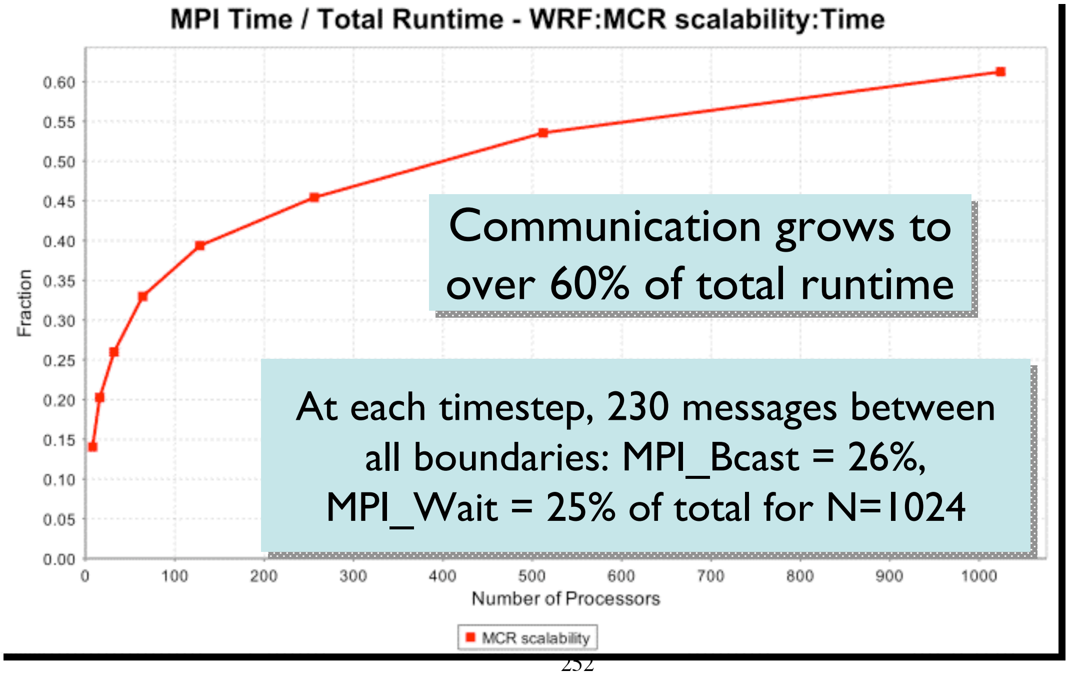




PerfExplorer - Runtime Breakdown



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

Concluding Discussion

- Performance tools must be used effectively
- More intelligent performance systems for productive use
  - Evolve to application-specific performance technology
  - Deal with scale by “full range” performance exploration
  - Autonomic and integrated tools
  - Knowledge-based and knowledge-driven process
- Performance observation methods do not necessarily need to change in a fundamental sense
  - More automatically controlled and efficiently use
- Develop next-generation tools and deliver to community
- Open source with support by ParaTools, Inc.
- http://www.cs.uoregon.edu/research/tau

Labs!

Lab: PerfExplorer

Lab Instructions

- See workshop example:
- perfexplorer

Support Acknowledgements

- Los Alamos National Laboratory (LANL)
- Department of Energy (DOE)
  - Office of Science MICS office contracts
  - University of Utah ASC Level 1 sub-contract
  - LLNL, LANL ASC contracts
  - Argonne National Laboratory FastOS contracts
- NSF
- T.U. Dresden, GWT
  - Dr. Wolfgang Nagel and Holger Brunst
- Research Centre Juelich
  - Dr. Bernd Mohr
- University of Oregon
  - Dr. Allen D. Malony, Alan Morris, Wyatt Spear