

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

Sameer Shende

sameer@paratools.com

Workshop at Solana Training Room, Los Alamos Research Park,
Suite 201, Room 223,

Los Alamos National Laboratory, NM

<http://www.paratools.com/lanl11>

<http://www.hpclinux.com>

ParaTools

Outline - Brief tutorial on the TAU toolset

	Slide #
• Introduction to TAU and a brief demo	7
• Overview of different methods of instrumenting applications	18
• Throttling effect of frequently called small subroutines	37
• Custom profiling	42
• Instrumentation and measurement alternatives	50
• Techniques for manual instrumentation of individual routines	94
• PAPI hardware performance counters	96
• Estimation of tool intrusiveness	112
• Memory and I/O	116
• GPGPU instrumentation	129
• Event traces	139
• Performance analysis with Paraprof and PerfExplorer	163
• Scalasca	209
• Hands-on training with sample codes (provided)	259

ParaTools

Hands-on analysis of own benchmark application

- Review of instrumentation process, address any problems with individual applications
- Determination of routines requiring further investigation, custom profiling if needed
- Running on the HPC system, address any problems with individual application
- Analysis of communication, input/output, scalability, Flop/s using ParaProf and/or PerfExplorer
- Refinement of instrumentation to the users' needs, possible manual instrumentation of individual routines or loops
- Optional: brief presentation of results of individual applications

References:

TAU User Guide

<http://tau.uoregon.edu/tau-usersguide.pdf>

ParaTools

3

Agenda: Day 1

- 8:30am - 10:30am: Introduction to TAU and a short demo. Topics:
 - Introduction to interval, atomic, and context events in TAU.
 - Instrumentation options for C++, C, and UPC programs.
 - Measurement: profiling and tracing.
 - Analysis tools: paraprof, perfexplorer.
- 10:30am - 10:45am: break
- 10:45-noon: Hands-on session #1 for TAU instrumentation on Cielito, Cray XE6.
- noon - 1:30pm: lunch break
- 1:30pm - 3:30pm: Introduction to performance engineering, measurement options for memory and I/O evaluation, and using PAPI.
- 3:30pm - 3:45pm: break
- 3:45pm – 5:00pm: Hands-on session #2:
 - Using PAPI and TAU for performance assessment of compiler optimizations
 - Memory and I/O evaluation.

ParaTools

4

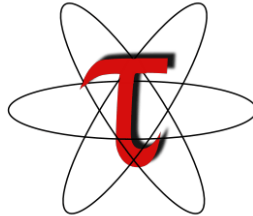
Agenda: Day 2

- 8:30am - 10:30am: Instrumentation of SHMEM, MPI, OpenCL, and CUDA programs, optimization of instrumentation, analysis of performance data collected.
- 10:30am – 10:45am: break
- 10:45am – noon: Hands-on session #3:
 - Comparing performance of sample codes with different compilers on Cray.
 - Instrumentation of OpenCL and CUDA programs on GPGPU platforms.
- noon - 1:30pm: lunch break
- 1:30pm - 3:30pm: Introduction to event tracing, performance databases, and PerfExplorer for cross-experiment performance analysis and a demo.
- 3:30pm - 3:45pm: break
- 3:45pm – 5:00pm:
- Hands-on session #4:
 - Using PerfDMF database, PerfExplorer, and Jumpshot.
 - Scalasca
 - Applying TAU to other sample applications.

Workshop Goals

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

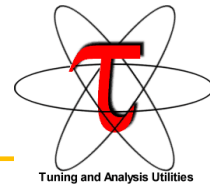
Introduction to TAU and a brief demo



ParaTools

7

TAU Performance System



- <http://tau.uoregon.edu/>
- Multi-level performance instrumentation
 - Multi-language automatic source instrumentation
- Flexible and configurable performance measurement
- Widely-ported parallel performance profiling system
 - Computer system architectures and operating systems
 - Different programming languages and compilers
- Support for multiple parallel programming paradigms
 - Multi-threading, message passing, mixed-mode, hybrid
- Integration in complex software, systems, applications

ParaTools

8

What is TAU?

- TAU is a performance evaluation tool
- It supports parallel profiling and tracing
- Profiling shows you how much (total) time was spent in each routine
- Tracing shows you *when* the events take place in each process along a timeline
- TAU uses a package called PDT for automatic instrumentation of the source code
- Profiling and tracing can measure time as well as hardware performance counters from your CPU
- TAU can automatically instrument your source code (routines, loops, I/O, memory, phases, etc.)
- TAU runs on all HPC platforms and it is free (BSD style license)
- TAU has instrumentation, measurement and analysis tools
 - paraprof is TAU's 3D profile browser
- To use TAU's automatic source instrumentation, you need to set a couple of environment variables and substitute the name of your compiler with a TAU shell script

ParaTools

9

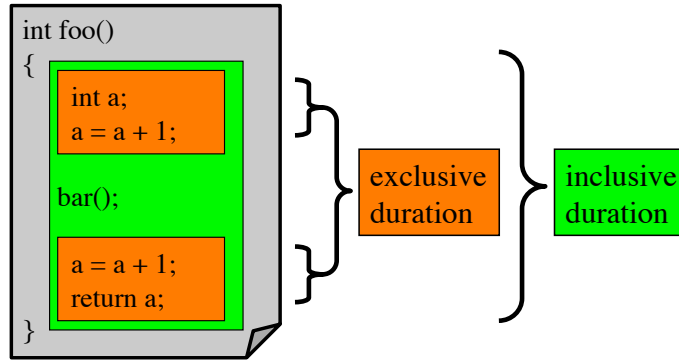
TAU Instrumentation Approach

- Based on direct performance observation
 - Direct instrumentation of program (system) code (probes)
 - Instrumentation invokes performance measurement
 - Event measurement: performance data, meta-data, context
- Support for standard program events
 - Routines, classes and templates
 - Statement-level blocks and loops
 - Begin/End events (Interval events)
- Support for user-defined events
 - Begin/End events specified by user
 - Atomic events (e.g., size of memory allocated/freed)
 - Flexible selection of event statistics
- Provides static events and dynamic events

ParaTools

Inclusive and Exclusive Profiles

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



ParaTools

Interval Events, Atomic Events in TAU

```

xterm
NODE 0;CONTEXT 0;THREAD 0:
-----
%Time   Exclusive   Inclusive   #Call   #Subrs   Inclusive Name
      msec      total msec
-----
100.0    0.187        1.105        1        44      1105659 int main(int, char **) C
93.2     1.030        1.030        1         0      1030654 MPI_Init()
 5.9     0.879         65         40       320     1637 void func(int, int) C
 4.6     51           51         40         0     1277 MPI_Barrier()
 1.2     13           13        120         0     111 MPI_Recv()
 0.8     9            9          1          0     9328 MPI_Finalize()
 0.0     0.137        0.137       120         0         1 MPI_Send()
 0.0     0.086        0.086        40         0         2 MPI_Bcast()
 0.0     0.002        0.002         1          0         2 MPI_Comm_size()
 0.0     0.001        0.001         1          0         1 MPI_Comm_rank()
-----

USER EVENTS Profile :NODE 0, CONTEXT 0, THREAD 0
-----
NumSamples  MaxValue  MinValue  MeanValue  Std. Dev.  Event Name
-----
365 5.138E+04  44.39  3.09E+04  1.234E+04  Heap Memory Used (KB) : Entry
365 5.138E+04  2064  3.115E+04  1.21E+04   Heap Memory Used (KB) : Exit
 40 40         40         40         0         Message size for broadcast
-----
27,1 17%
```

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

Atomic events
(trigger with
value)

```

% setenv TAU_CALLPATH_DEPTH 0
% setenv TAU_TRACK_HEAP 1
```

ParaTools

Atomic Events, Context Events

```

xterm
-----
%Time Exclusive Inclusive #Call #Subrs Inclusive Name
      msec total msec          usec/call
-----
100.0 0.253 1.106 1 44 1106701 int main(int, char **) C
93.2 1.031 1.031 1 0 1031311 MPI_Init()
6.0 1 66 40 320 1650 void func(int, int) C
5.7 63 63 40 0 1588 MPI_Barrier()
0.8 9 9 1 0 9119 MPI_Finalize()
0.1 1 1 120 0 10 MPI_Recv()
0.0 0.141 0.141 120 0 1 MPI_Send()
0.0 0.085 0.085 40 0 2 MPI_Bcast()
0.0 0.001 0.001 1 0 1 MPI_Comm_size()
0.0 0 0 1 0 0 MPI_Comm_rank()

USER EVENTS Profile :NODE 0, CONTEXT 0, THREAD 0
-----
NumSamples MaxValue MinValue MeanValue Std. Dev. Event Name
-----
40 40 40 40 0 Message size for broadcast
365 5.139E+04 44.39 3.091E+04 1.234E+04 Heap Memory Used (KB) : Entry
40 5.139E+04 3097 3.114E+04 1.227E+04 Heap Memory Used (KB) : Entry : MPI_Barrier()
40 5.139E+04 1.13E+04 3.134E+04 1.187E+04 Heap Memory Used (KB) : Entry : MPI_Bcast()
1 2067 2067 2067 0 Heap Memory Used (KB) : Entry : MPI_Comm_rank()
1 2066 2066 2066 0 Heap Memory Used (KB) : Entry : MPI_Comm_size()
1 5.139E+04 5.139E+04 5.139E+04 0.0006905 Heap Memory Used (KB) : Entry : MPI_Finalize()
1 57.56 57.56 57.56 0 Heap Memory Used (KB) : Entry : MPI_Init()
120 5.139E+04 1.13E+04 3.134E+04 1.187E+04 Heap Memory Used (KB) : Entry : MPI_Recv()
120 5.139E+04 1.129E+04 3.134E+04 1.187E+04 Heap Memory Used (KB) : Entry : MPI_Send()
1 44.39 44.39 44.39 0 Heap Memory Used (KB) : Entry : int main(int, char **) C
40 5.036E+04 2068 3.011E+04 1.227E+04 Heap Memory Used (KB) : Entry : void func(int, int) C
4.9 13%

% setenv TAU_CALLPATH_DEPTH 1
% setenv TAU_TRACK_HEAP 1
    
```

Atomic event

Context event
= atomic event
+ executing
context

ParaTools

Context Events (Default)

```

xterm
-----
NODE 0 CONTEXT 0, THREAD 0:
%Time Exclusive Inclusive #Call #Subrs Inclusive Name
      msec total msec          usec/call
-----
100.0 0.357 1.114 1 44 1114040 int main(int, char **) C
92.6 1.031 1.031 1 0 1031066 MPI_Init()
6.7 72 74 40 320 1865 void func(int, int) C
0.7 8 8 1 0 8002 MPI_Finalize()
0.1 1 1 120 0 12 MPI_Recv()
0.1 0.608 0.608 40 0 15 MPI_Barrier()
0.0 0.136 0.136 120 0 1 MPI_Send()
0.0 0.095 0.095 40 0 2 MPI_Bcast()
0.0 0.001 0.001 1 0 1 MPI_Comm_size()
0.0 0 0 1 0 0 MPI_Comm_rank()

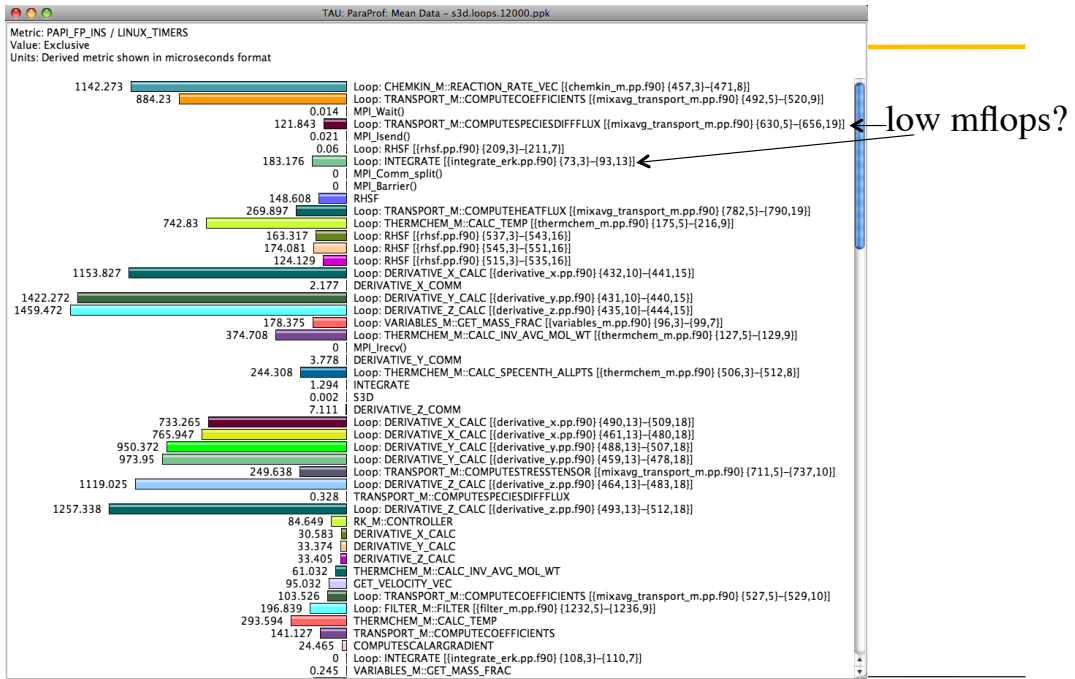
USER EVENTS Profile :NODE 0, CONTEXT 0, THREAD 0
-----
NumSamples MaxValue MinValue MeanValue Std. Dev. Event Name
-----
365 5.139E+04 44.39 3.091E+04 1.234E+04 Heap Memory Used (KB) : Entry
1 44.39 44.39 44.39 0 Heap Memory Used (KB) : Entry : int main(int, char **) C => MPI_Comm_rank()
1 2068 2068 2068 0 Heap Memory Used (KB) : Entry : int main(int, char **) C => MPI_Comm_size()
1 2066 2066 2066 0 Heap Memory Used (KB) : Entry : int main(int, char **) C => MPI_Finalize()
1 5.139E+04 5.139E+04 5.139E+04 0 Heap Memory Used (KB) : Entry : int main(int, char **) C => MPI_Init()
1 57.58 57.58 57.58 0 Heap Memory Used (KB) : Entry : int main(int, char **) C => MPI_Init()
40 5.036E+04 2069 3.011E+04 1.229E+04 Heap Memory Used (KB) : Entry : int main(int, char **) C => void func(int, int) C
40 5.139E+04 3098 3.114E+04 1.227E+04 Heap Memory Used (KB) : Entry : void func(int, int) C => MPI_Barrier()
40 5.139E+04 1.13E+04 3.134E+04 1.187E+04 Heap Memory Used (KB) : Entry : void func(int, int) C => MPI_Bcast()
120 5.139E+04 1.13E+04 3.134E+04 1.187E+04 Heap Memory Used (KB) : Entry : void func(int, int) C => MPI_Recv()
120 5.139E+04 1.13E+04 3.134E+04 1.187E+04 Heap Memory Used (KB) : Entry : void func(int, int) C => MPI_Send()
365 5.139E+04 2065 3.116E+04 1.21E+04 Heap Memory Used (KB) : Exit
3.7 1%

% setenv TAU_CALLPATH_DEPTH 2
% setenv TAU_TRACK_HEAP 1
    
```

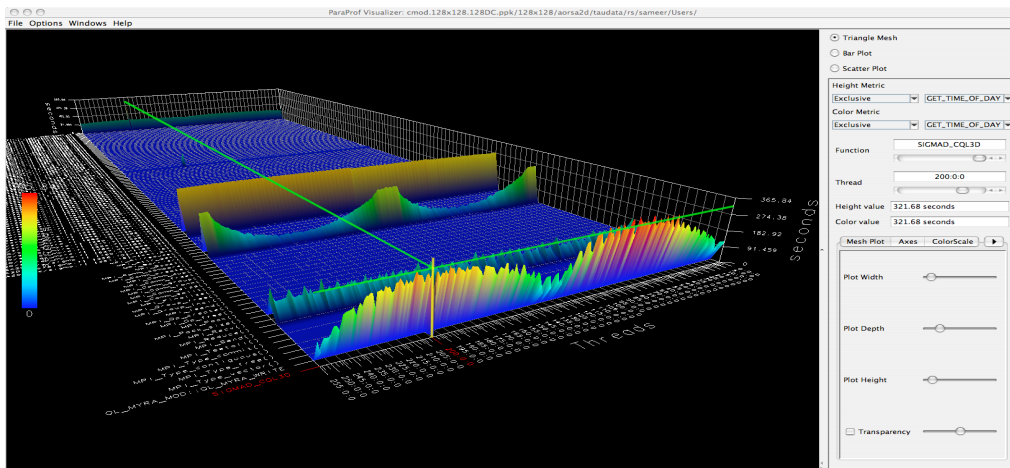
Context event
= atomic event
+ executing
context

ParaTools

ParaProf: Mflops Sorted by Exclusive Time



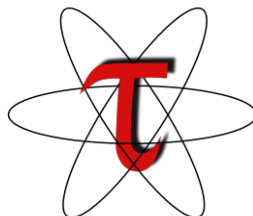
Parallel Profile Visualization: ParaProf



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

Overview of different methods of instrumenting applications



Instrumentation: Events in TAU

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

Instrumentation Techniques

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

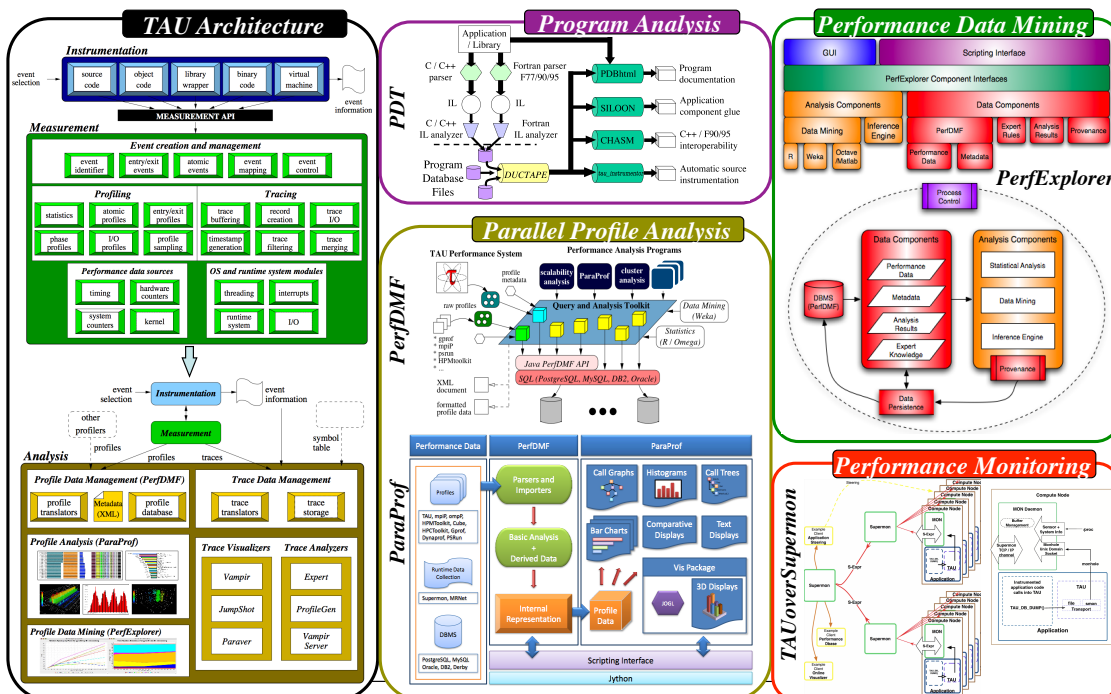
Instrumentation Techniques

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

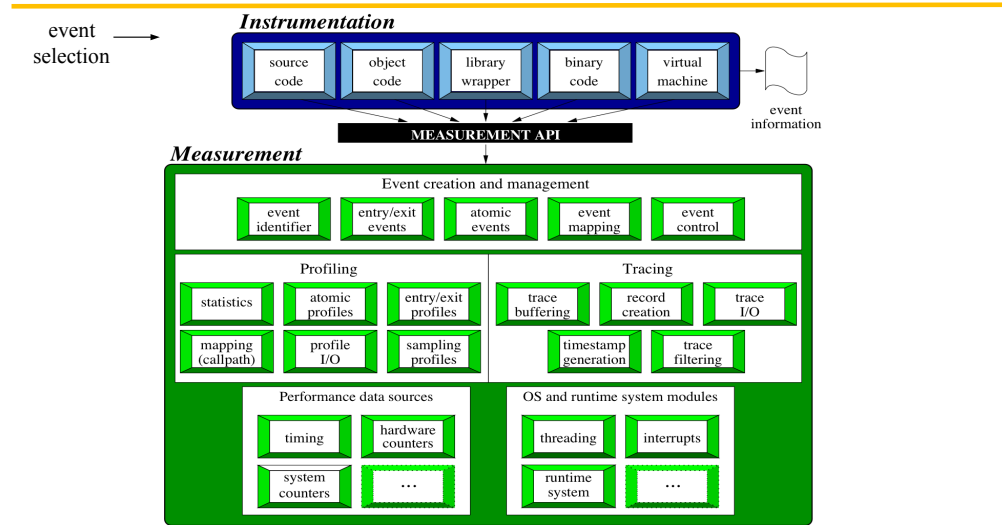
ParaTools

21

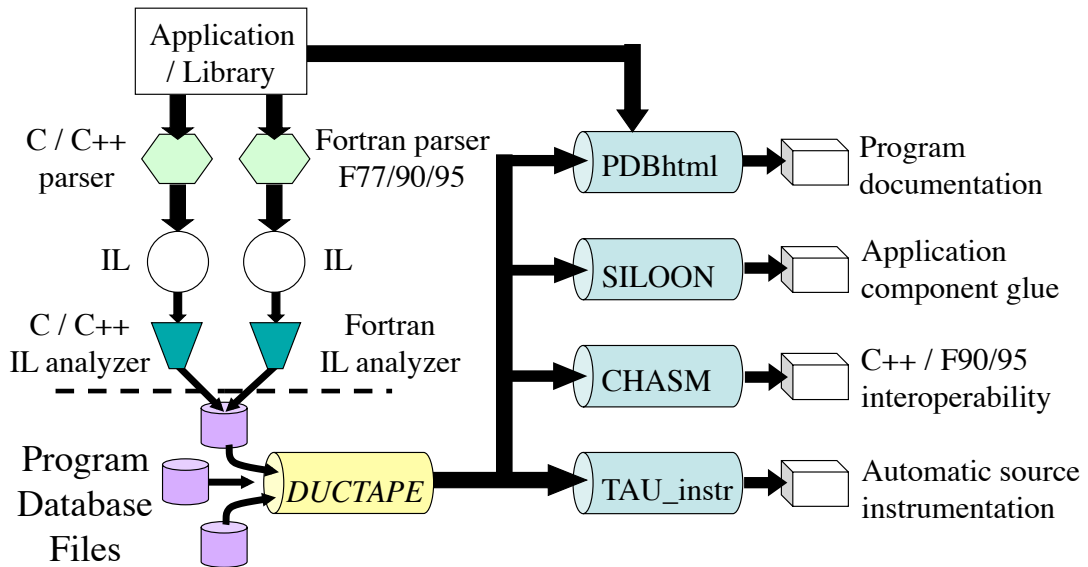
TAU Performance System Components



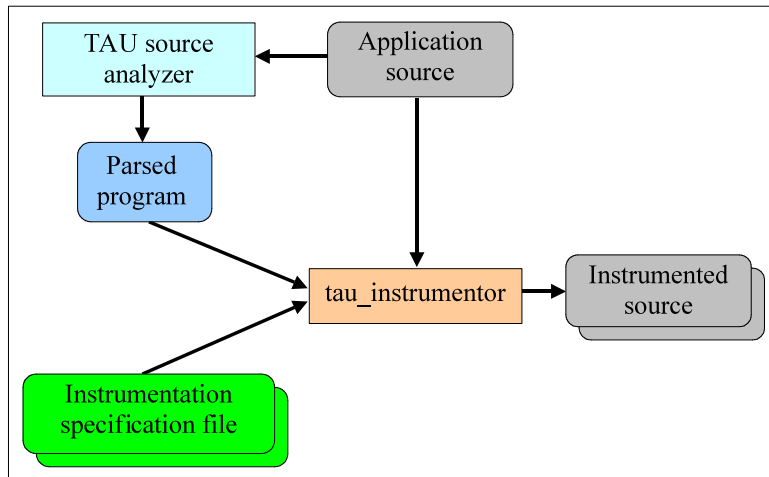
TAU Performance System Architecture



Program Database Toolkit (PDT)



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

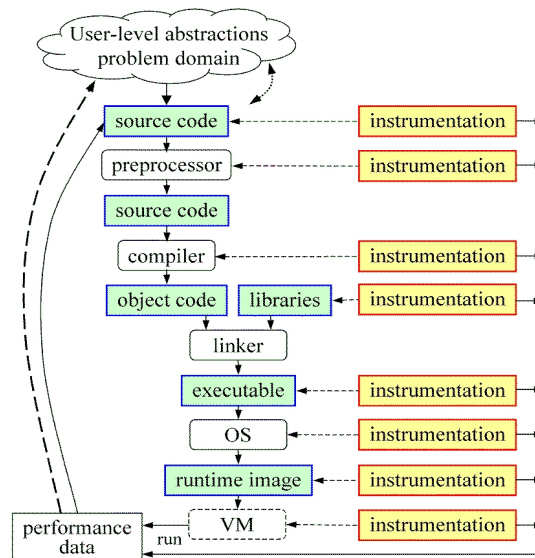


ParaTools

25

Direct Observation: Mapping

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

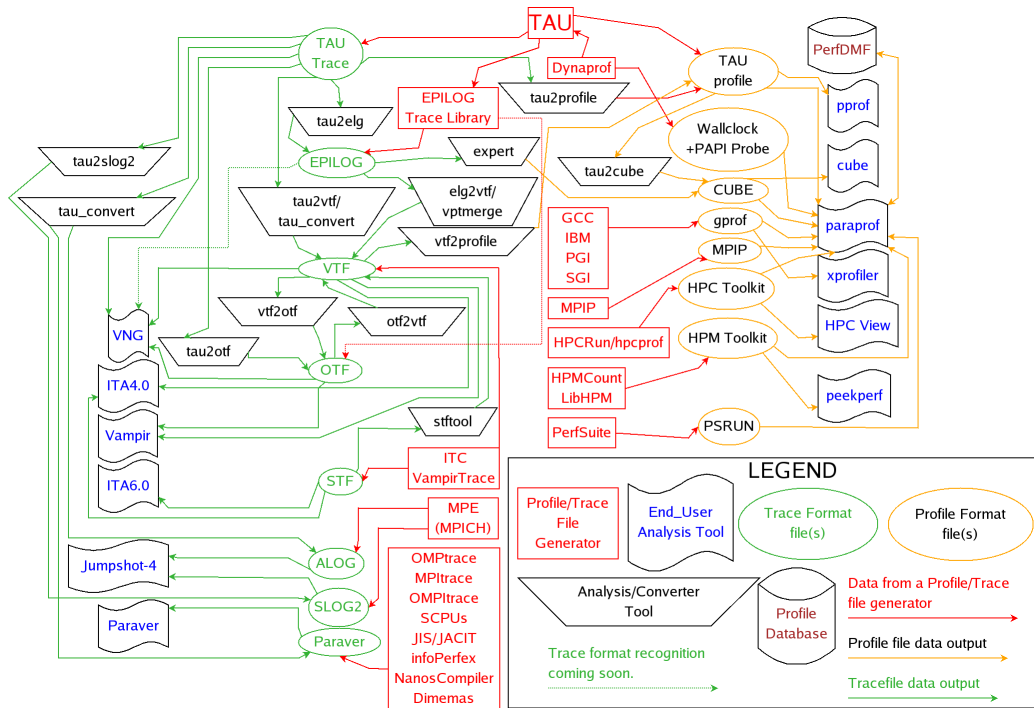


ParaTools

26



Building Bridges to Other Tools



Using TAU: A brief Introduction

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

```
% source /usr/projects/packages/tau/cielito/tau.bashrc (or .cshrc)
% export TAU_MAKEFILE=$TAU/Makefile.tau-papi-mpi-pdt-pgi
% export TAU_OPTIONS='-optVerbose ...' (see tau_compiler.sh -help)
And use tau_f90.sh, tau_cxx.sh or tau_cc.sh as Fortran, C++ or C compilers:
% CC app.cpp
changes to
% tau_cxx.sh app.cpp
```
- Execute application and analyze performance data:
 - % pprof (for text based profile display)
 - % paraprof (for GUI)

TAU Measurement Configuration

```
% cd $TAU; ls Makefile.*
Makefile.tau-pdt-pgi
Makefile.tau-mpi-pdt-pgi
Makefile.tau-pthread-pdt-pgi
Makefile.tau-papi-mpi-pdt-pgi
Makefile.tau-papi-pthread-pdt-pgi
Makefile.tau-mpi-papi-pdt-pgi
```

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

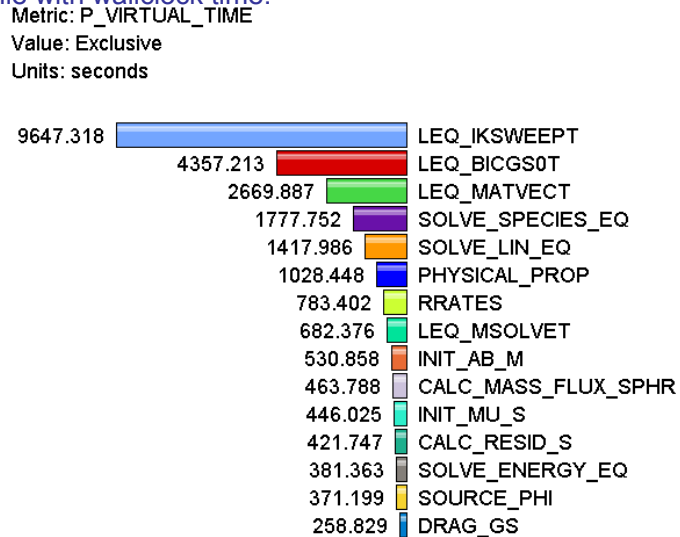
```
Makefile.tau-mpi-pdt-pgi
- Supports MPI instrumentation & PDT for automatic source instrumentation
- % export TAU_MAKEFILE=$TAU/Makefile.tau-mpi-pdt-pgi
- % tau_f90.sh matrix.f90 -o matrix
```

ParaTools

29

Usage Scenarios: Routine Level Profile

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



ParaTools

30

Solution: Generating a flat profile with MPI

```
% export TAU_MAKEFILE=$TAU/Makefile.tau-mpi-pdt-pgi
% export PATH=$TAU_ROOT/bin:$PATH
OR
% source /usr/projects/packages/tau/cielito//tau.bashrc
% tau_cxx.sh app.cpp -o app
(Or make CC=tau_cxx.sh)

% aprun -n 4./app
% pprof
% paraprof &
OR
% paraprof --pack app.ppk
  Move the app.ppk file to your desktop.

% paraprof app.ppk
(Note: you may need module load java if you have an older JVM in your
path)
Click on "node 0" to see figure shown. Right click on node to see
Other windows.
```

ParaTools

31

Automatic Instrumentation

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

Before

```
CXX = CC
F90 = ftn
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 $<
```

ParaTools

32

TAU_COMPILER Commandline Options

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

```
% ftn -c foo.f90
```

Changes to

```
% gfparse foo.f90 $(OPT1)
```

```
% tau_instrumentor foo.pdb foo.f90 -o foo.inst.f90 $(OPT2)
```

```
% ftn -c foo.inst.f90 $(OPT3)
```
- Linking:

```
% ftn foo.o bar.o -o app
```

Changes to

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

```
F90 = tau_f90.sh
```

ParaTools

33

Compile-Time Environment Variables

- Optional parameters for TAU_OPTIONS: [`tau_compiler.sh -help`]
 - optVerbose Turn on verbose debugging messages
 - optComplnst Use compiler based instrumentation
 - optNoComplnst Do not revert to compiler instrumentation if source instrumentation fails.
 - optDetectMemoryLeaks Turn on debugging memory allocations/ de-allocations to track leaks
 - optTrackIO Wrap POSIX I/O call and calculates vol/bw of I/O operations (Requires TAU to be configured with `-iowrapper`)
 - optKeepFiles Does not remove intermediate `.pdb` and `.inst.*` files
 - optPreProcess Preprocess Fortran sources before instrumentation
 - optTauSelectFile="" Specify selective instrumentation file for `tau_instrumentor`
 - optTauWrapFile="" Specify `link_options.tau` generated by `tau_gen_wrapper`
 - 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)`

ParaTools

...

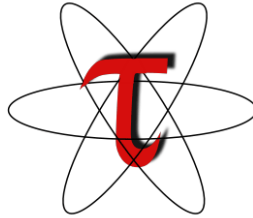
Compiling Fortran Codes with TAU

- If your Fortran code uses free format in .f files (fixed is default for .f), you may use:
% export TAU_OPTIONS='-optPdtF95Opts="-R free" -optVerbose'
- To use the compiler based instrumentation instead of PDT (source-based):
% export TAU_OPTIONS='-optComplnst -optVerbose'
- If your Fortran code uses C preprocessor directives (#include, #ifdef, #endif):
% export TAU_OPTIONS='-optPreProcess -optVerbose -optDetectMemoryLeaks'
- To use an instrumentation specification file:
% export TAU_OPTIONS='-optTauSelectFile=mycmd.tau -optVerbose -optPreProcess'
% cat mycmd.tau
BEGIN_INSTRUMENT_SECTION
memory file="foo.f90" routine="#"
instruments all allocate/deallocate statements in all routines in foo.f90
loops file="*" routine="#"
io file="abc.f90" routine="FOO"
END_INSTRUMENT_SECTION

Environment Variables in TAU

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

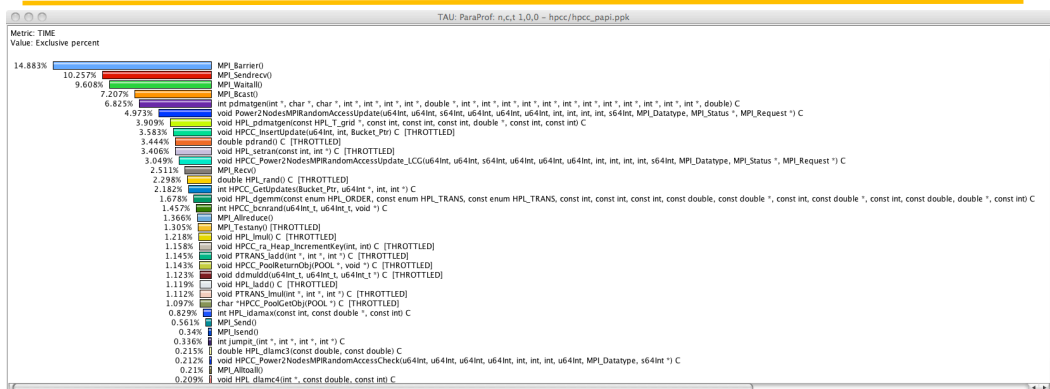
Throttling effect of frequently called small routines



ParaTools

37

Runtime Throttling of Events



ParaTools

38

Optimization of Program Instrumentation

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

```
% export TAU_THROTTLE=1
```

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

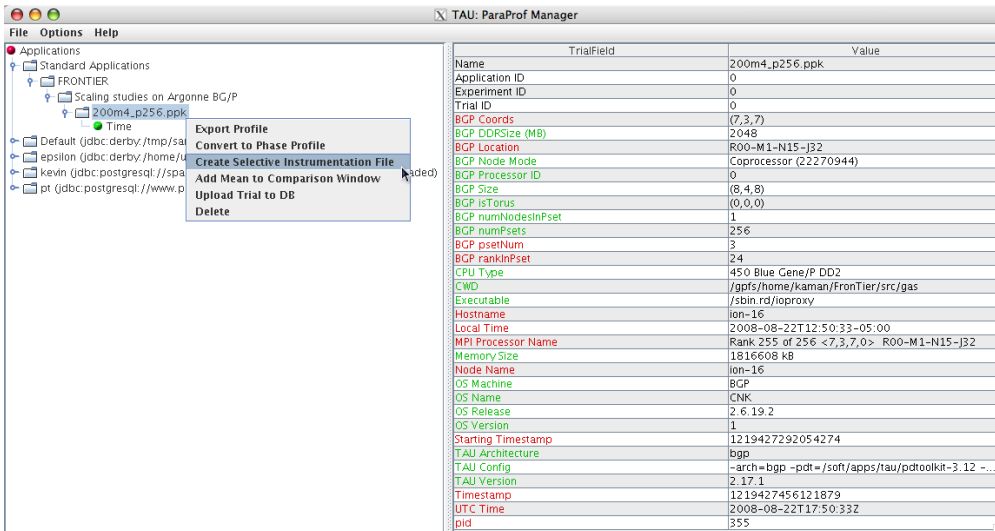
```
% tau_instrumentor [options] -f <file> OR  
% export TAU_OPTIONS='-optTauSelectFile=tau.txt'
```
- Compensation of local instrumentation overhead

```
% export TAU_COMPENSATE=1 (in tau-2.19.2+)
```

ParaTools

39

ParaProf: Creating Selective Instrumentation File

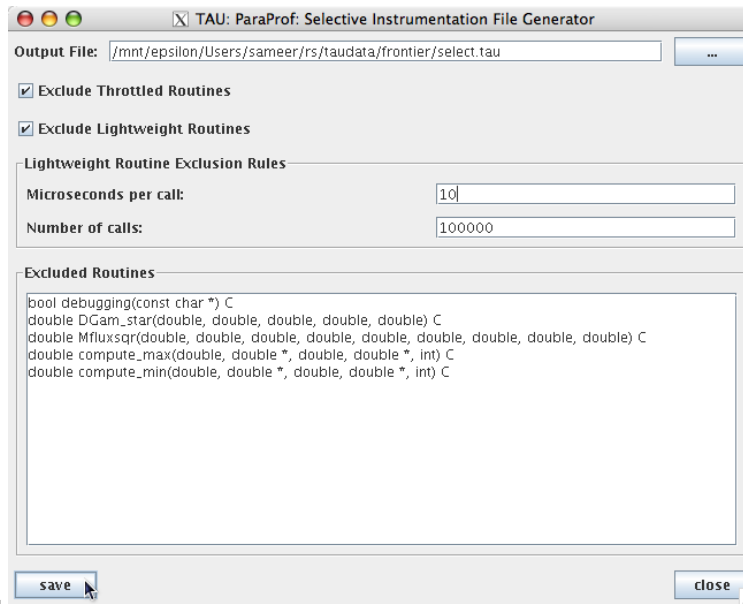


Name	TrialField	Value
Application ID	200m4_p256.ppk	0
Experiment ID		0
Trial ID		0
BGP Coords		(7,3,7)
BGP DDRSize (MB)		2048
BGP Location		R00-M1-N15-J32
BGP Node Mode		Coprocessor (22770944)
BGP Processor ID		0
BGP Size		(8,4,8)
BGP isTorus		(0,0,0)
BGP numNodesInPset		1
BGP numPsets		256
BGP psetNum		3
BGP rankInPset		24
CPU Type		450 Blue Gene/P DD2
CWD		/gpfs/home/kaman/FrontTier/src/gas
Executable		/sbin.rd/loproxy
Hostname		ion-16
Local Time		2008-08-22T12:50:33-05:00
MPI Processor Name		Rank 255 of 256 <7,3,7,0> R00-M1-N15-J32
Memory Size		1816608 kB
Node Name		ion-16
OS Machine		BGP
OS Name		CNK
OS Release		2.6.19.2
OS Version		1
Starting Timestamp		1219427292054274
TAU Architecture		bgp
TAU Config		-arch=bgp -pdt=/soft/apps/tau/pdtoolkit-3.12 ...
TAU Version		2.17.1
Timestamp		1219427456121879
UTC Time		2008-08-22T17:50:33Z
pid		355

ParaTools

40

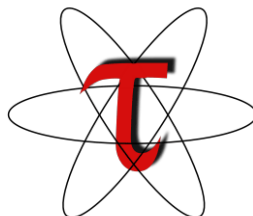
Choosing Rules for Excluding Routines



ParaTools

41

Custom profiling



ParaTools

42

Selective Instrumentation File

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

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

- Specify a list of routines to include for instrumentation

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

- Specify either an include list or an exclude list!

Selective Instrumentation File

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

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

- Specify a list of routines to include for instrumentation

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

Selective Instrumentation File

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

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

ParaTools

45

Instrumentation Specification

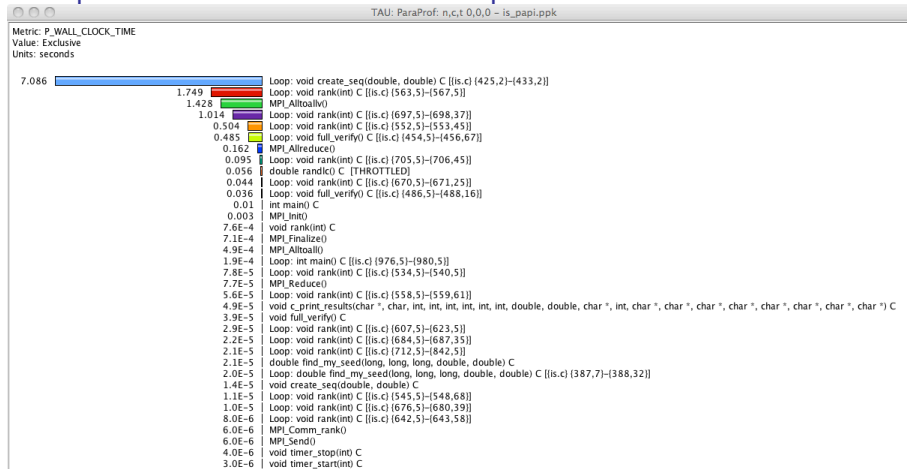
```
% tau_instrumentor
Usage : tau_instrumentor <pdbfile> <sourcefile> [-o <outputfile>] [-noinline]
[-g groupname] [-i headerfile] [-c|-c++|-fortran] [-f <instr_req_file> ]
For selective instrumentation, use -f option
% tau_instrumentor foo.pdb foo.cpp -o foo.inst.cpp -f selective.dat
% cat selective.dat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

46

Usage Scenarios: Loop Level Instrumentation

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



ParaTools

47

Solution: Generating a loop level profile

```
% export TAU_MAKEFILE=$TAU/Makefile.tau-mpi-pdt-pgi
% export TAU_OPTIONS='-optTauSelectFile=select.tau -optVerbose'
% cat select.tau
BEGIN_INSTRUMENT_SECTION
loops routine="#"
END_INSTRUMENT_SECTION

% export PATH=$TAU_ROOT/bin:$PATH
% make CC=tau_cc.sh CXX=tau_cxx.sh F90=tau_f90.sh

% aprun -n 4./a.out
% paraprof --pack app.ppk
Move the app.ppk file to your desktop.

% paraprof app.ppk
```

ParaTools

48

ParaProf's Source Browser: Loop Level Instrumentation

The screenshot displays the ParaProf interface with three function data windows on the left and a source code window on the right.

Function Data Window 1: Loop: TRANSPORT_Me:COMPUTESPECIESDIFFLUX [(mixavg_transport_m.pp.f90) (630,5)-(656,19)]
Metric Name: PAPL_FP_INS / GET_TIME_OF_DAY
Value: Exclusive
Units: Derived metric shown in microseconds format

Mean	std. dev.
114.979	1.088
117.62	n.ct 0,0
115.134	n.ct 1,0
114.709	n.ct 2,0
114.615	n.ct 3,0
113.547	n.ct 4,0
114.381	n.ct 5,0
114.837	n.ct 6,0
114.789	n.ct 7,0

Function Data Window 2: Loop: TRANSPORT_Me:COMPUTESPECIESDIFFLUX [(mixavg_transport_m.pp.f90) (630,5)-(656,19)]
Metric Name: GET_TIME_OF_DAY
Value: Exclusive percent

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

Function Data Window 3: Loop: TRANSPORT_Me:COMPUTESPECIESDIFFLUX [(mixavg_transport_m.pp.f90) (630,5)-(656,19)]
Metric Name: PAPL_LL_DCM
Value: Exclusive
Units: counts

Mean	std. dev.
5.0701E9	836336.1
5.0692E9	n.ct 0,0
5.07E9	n.ct 1,0
5.069E9	n.ct 2,0
5.0701E9	n.ct 3,0
5.0708E9	n.ct 4,0
5.0711E9	n.ct 5,0
5.0712E9	n.ct 6,0
5.0692E9	n.ct 7,0

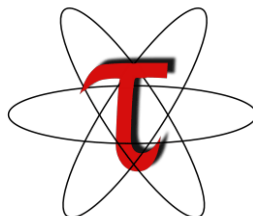
Source Code Window: TAU: ParaProf: Source Browser: /mnt/epsilon/Users/sameer/rs/taudata/s3d/harness/flat/papi8

```

606 grad_mixW(:, :, :, n) = grad_mixW(:, :, :, n)*absolut(:, :, :
607 end do
608
609 ! compute grad_P
610 if (baro_switch) then
611   allocate(grad_P(mv,nz,3))
612   grad_P = 0.0
613   if (vary_in_x == 1) then
614     call derivative_x( mx,ny,nz, Press, grad_P(:, :, :1), scale_x, 1 )
615   endif
616   if (vary_in_y == 1) then
617     call derivative_y( mx,ny,nz, Press, grad_P(:, :, :2), scale_y, 1 )
618   endif
619   if (vary_in_z == 1) then
620     call derivative_z( mx,ny,nz, Press, grad_P(:, :, :3), scale_z, 1 )
621   endif
622 endif
623
624 ! Changed by Ramanan - 03/24/05
625 ! Ds_mixavg is now vhorb
626
627 !grad_P/press and avnolot*grad_T/Temp can be optimized by division before the loop.
628 ! compute diffusive flux for species n in direction n.
629 diffFlux(:, :, :, n_spec,:) = 0.0
630 DIRECTION = do n=1,3
631 SPECIES = do n=1,n_spec-1
632
633 if (baro_switch) then
634   ! driving force includes gradient in mole fraction and baro-diffusion
635   diffFlux(:, :, :, n) = - Ds_mixavg(:, :, :, n) * ( grad_Ys(:, :, :, n,n) &
636     + Ys(:, :, :, n) * ( grad_mixW(:, :, :, n) &
637     + (1 - molwt(n)*avnolot) * grad_P(:, :, :, n)/Press))
638 else
639   ! driving force is just the gradient in mole fraction
640   diffFlux(:, :, :, n) = - Ds_mixavg(:, :, :, n) * ( grad_Ys(:, :, :, n,n) &
641     + Ys(:, :, :, n) * grad_mixW(:, :, :, n) )
642 endif
643
644 ! Add thermal diffusion
645 if (thermoDiff_switch) then
646   diffFlux(:, :, :, n) = diffFlux(:, :, :, n) &
647     - Ds_mixavg(:, :, :, n) * Rs_therm_diff(:, :, :, n) * molwt(n) &
648     * avnolot * grad_T(:, :, :, 0) / Temp
649 endif
650
651 ! compute contribution to nth species diffusive flux
652 ! this will ensure that the sum of the diffusive fluxes is zero.
653 diffFlux(:, :, :, n_spec,n) = diffFlux(:, :, :, n_spec,n) - diffFlux(:, :, :, n,n)
654
655 enddo SPECIES
656 enddo DIRECTION
657
658 if (baro_switch) then
659   deallocate(grad_P)
660 endif
661
662 return
663 end subroutine computeSpeciesDiffFlux
664
665 !!$-----
666
667
668 subroutine computeStressTensor( grad_u)
669

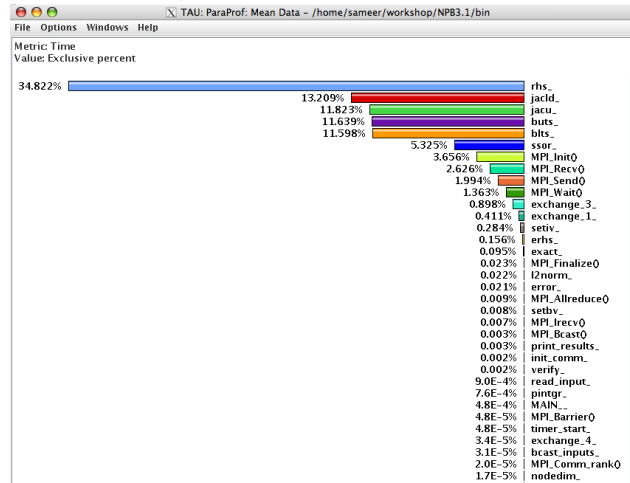
```

Instrumentation and Measurement Alternatives



Usage Scenarios: Compiler-based Instrumentation

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



ParaTools

51

Use Compiler-Based Instrumentation

```
% export TAU_MAKEFILE=$TAU_ROOT  
    /lib/Makefile.tau-mpi-pdt-pgi  
% export TAU_OPTIONS='-optCompInst -optVerbose'  
% export PATH=$TAU_ROOT/bin:$PATH  
% make CXX=tau_cxx.sh CC=tau_cc.sh  
% aprun -n 4 ./a.out  
% paraprof --pack app.ppk  
    Move the app.ppk file to your desktop.  
% paraprof app.ppk
```

ParaTools

52

Profiling a UPC Applications

TAU: ParaProf: Context Events for thread: n,c,t, 0,0,0 - gasp_test.ppk

Name	Total	NumSamples	MaxValue	MinValue	MeanValue	Std. Dev.
BUPC_STATIC_SHARED	892	6	400	4	148.667	179.739
Message size for all-gather	52	5	16	8	10.4	3.2
Message size for all-reduce	4	1	4	4	4	0
Message size for broadcast	9,157	1	9,157	9,157	9,157	0
UPC_MEMCPY	8	2	4	4	4	0
UPC_MEMGET	8	2	4	4	4	0
UPC_MEMPUT	8	2	4	4	4	0

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

Name	Exclusive TIME	Inclusive TIME	Exclusive TIME%	Calls	Child Calls
.TAU application	11.227	13.463	83.4%	1	306
MPI_init_thread()	1.092	1.092	8.1%	1	0
MPI_Test()	0.907	0.907	6.7%	83,681	0
UPC_COLLECTIVE_EXIT	0.196	1.024	1.5%	1	80,623
MPI_Finalize()	0.091	0.091	0.7%	1	0
MPI_Allreduce()	0.038	0.038	0.3%	1	0
MPI_Allgather()	0.008	0.008	0.1%	5	0
UPCR1_ALLOC_gasp_test_	0.005	0.044	0.0%	2	1,842
UPC_ALL_SCATTER	0.003	0.003	0.0%	1	11
UPC_WAIT	0.002	0.015	0.0%	24	953
user_main	0.001	0.023	0.0%	144	214
MPI_Waitany()	0.001	0.001	0.0%	2	0
MPI_Bcast()	0.001	0.001	0.0%	1	0
MPI_Isend()	0.001	0.001	0.0%	193	0
MPI_Irecv()	0.001	0.001	0.0%	195	0
collectives	0	0.01	0.0%	45	73
UPC_NOTIFY	0	0.001	0.0%	24	76
MPI_Testsome()	0	0	0.0%	95	0
UPC_ALL_EXCHANGE	0	0.001	0.0%	1	80
UPC_ALL_GATHER_ALL	0	0.001	0.0%	1	63
UPC_ALL_BROADCAST	0	0.001	0.0%	1	38
MPI_Cancel()	0	0	0.0%	16	0
UPC_ALL_ALLOC	0	0	0.0%	4	24
UPC_ALL_REDUCE	0	0.001	0.0%	1	26
UPC_ALL_PREFIX_REDUCE	0	0	0.0%	1	25
MPI_Wait()	0	0	0.0%	16	0
UPC_ALL_PERMUTE	0	0.001	0.0%	1	18
UPC_ALL_GATHER	0	0	0.0%	1	20
MPI_Comm_create()	0	0	0.0%	3	0
UPC_ALL_LOCK_ALLOC	0	0	0.0%	1	18
C_MALLOC	0	0	0.0%	7	2

ParaTools

53

Profiling a UPC Application

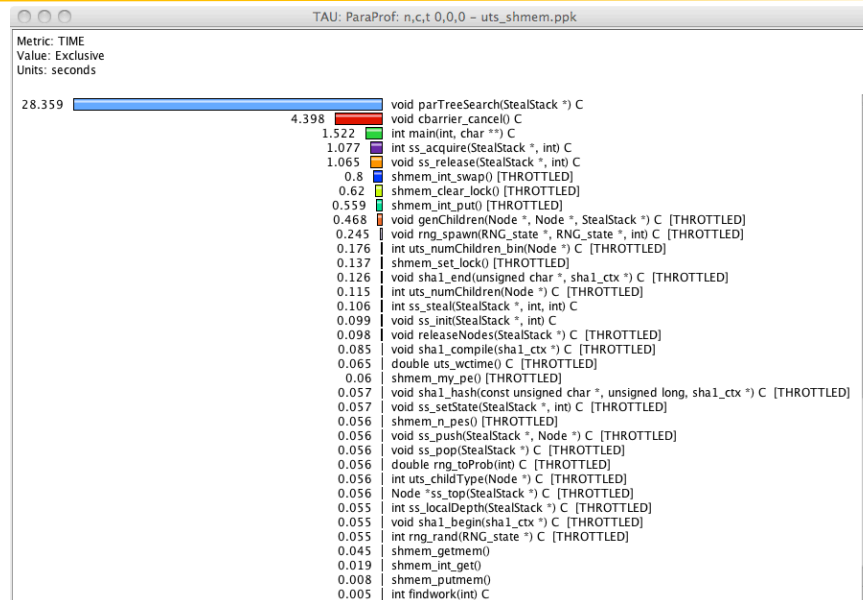
```
% export TAU_MAKEFILE=$TAU/Makefile.tau-upc-mpi
% export PATH=$TAU_ROOT/bin:$PATH
% export TAU_OPTIONS='-optCompInst -optVerbose'
% make CC=tau_cc.sh
(Or edit Makefile and change CXX and F90)
% export TAU_CALLPATH=1
% export TAU_CALLPATH_DEPTH=100

% aprun -n 4 ./a.out
% paraprof --pack app.ppk
  Move the app.ppk file to your desktop.
% paraprof app.ppk
(Windows -> Thread -> Call Graph)
```

ParaTools

54

Profiling a SHMEM Application



ParaTools

55

Profiling a SHMEM Application

```
% export TAU_MAKEFILE=$TAU/Makefile.tau-papi-shmem-pdt-pgi
% export PATH=$TAU_ROOT/bin:$PATH
% make CC=tau_cc.sh
(Or edit Makefile and change CXX and F90)

% aprun -n 4 ./a.out
% paraprot --pack app.ppk
  Move the app.ppk file to your desktop.
% paraprot app.ppk
```

ParaTools

56

Instrumentation of OpenMP Constructs

- OpenMP Pragma And Region Instrumentor [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

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

```
% configure -opari -arch=craycnl -mpi -pdt=/apps/pdtoolkit-3.16
% make clean; make install
% export 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
```

ParaTools

61

Re-writing Binaries

- Support for both static and dynamic executables
- Specify the list of routines to instrument/exclude from instrumentation
- Specify the TAU measurement library to be injected
- Simplify the usage of TAU:
 - To instrument:
 - % tau_run a.out -o a.inst
 - To perform measurements, execute the application:
 - % mpirun -np 4 ./a.inst
 - To analyze the data:
 - % paraprof

ParaTools

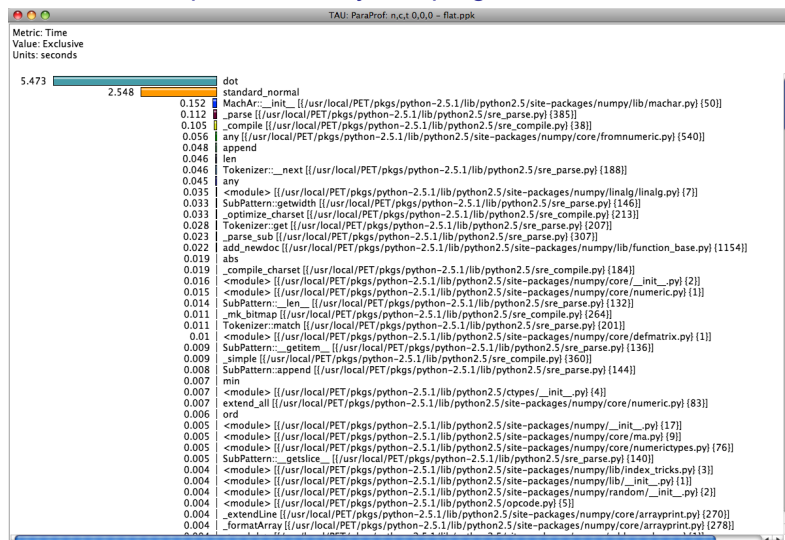
tau_run with NAS PBS

```
livetau@paratools01:~/tutorial
/home/livetau% cd ~/tutorial
/home/livetau/tutorial% # Build an uninstrumented bt NAS Parallel Benchmark
/home/livetau/tutorial% make bt CLASS=W NPROCS=4
/home/livetau/tutorial% cd bin
/home/livetau/tutorial/bin% # Run the instrumented code
/home/livetau/tutorial/bin% mpirun -np 4 ./bt_W.4
/home/livetau/tutorial/bin% # Instrument the executable using TAU with DyninstAPI
/home/livetau/tutorial/bin% tau_run ./bt_W.4 -o ./bt.i
/home/livetau/tutorial/bin% rm -rf profile.* MULT*
/home/livetau/tutorial/bin% mpirun -np 4 ./bt.i
/home/livetau/tutorial/bin% paraprof
/home/livetau/tutorial/bin% # Choose a different TAU configuration
/home/livetau/tutorial/bin% ls $TAU/libTAUsh
libTAUsh-depthlimit-mpi-pdt.so*      libTAUsh-papi-pdt.so*
libTAUsh-mpi-pdt.so*                libTAUsh-papi-pthread-pdt.so*
libTAUsh-mpi-pdt-upc.so*            libTAUsh-param-mpi-pdt.so*
libTAUsh-mpi-python-pdt.so*         libTAUsh-pdt.so*
libTAUsh-papi-mpi-pdt.so*           libTAUsh-pdt-trace.so*
libTAUsh-papi-mpi-pdt-upc.so*       libTAUsh-phase-papi-mpi-pdt.so*
libTAUsh-papi-mpi-pdt-udp.so*       libTAUsh-pthread-pdt.so*
libTAUsh-papi-mpi-pdt-vampirtrace-trace.so* libTAUsh-python-pdt.so*
libTAUsh-papi-mpi-python-pdt.so*
/home/livetau/tutorial/bin% tau_run -XrunTAUsh-papi-mpi-pdt-vampirtrace-trace bt_W.4 -o bt.vpt
/home/livetau/tutorial/bin% setenv VT_METRICS PAPI_FP_INS:PAPI_L1_DCM
/home/livetau/tutorial/bin% mpirun -np 4 ./bt.vpt
/home/livetau/tutorial/bin% vampir bt.vpt.otf &
```

6
3

Usage Scenarios: Instrument a Python program

- Goal: Generate a flat profile for a Python program



Usage Scenarios: Instrument a Python program

Original code:

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

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

Create a wrapper:

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

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

import tau

def OurMain():
    import foo

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

ParaTools

65

Generate a Python Profile

```
% export TAU_MAKEFILE=$TAU_ROOT
    /lib/Makefile.tau-python-pdt
% export PATH=$TAU_ROOT/bin:$PATH
% cat wrapper.py
import tau
def OurMain():
    import foo
    tau.run('OurMain()')
Uninstrumented:
% ./foo.py
Instrumented:
% export PYTHONPATH= <taudir>/x86_64/<lib>/bindings-python-pdt
(same options string as TAU_MAKEFILE)
% ./wrapper.py

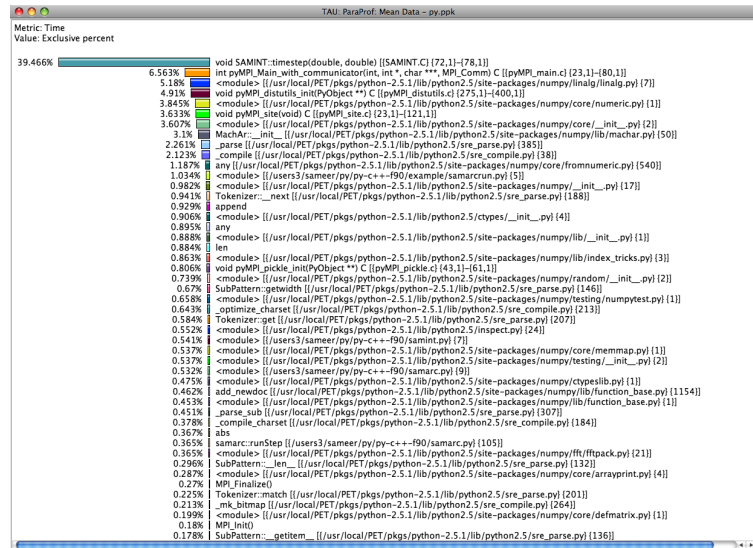
Wrapper invokes foo and generates performance data
% pprof/paraprof
```

ParaTools

66

Usage Scenarios: Mixed Python+F90+C+pyMPI

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



ParaTools

67

Generate a Multi-Language Profile w/ Python

```
% export TAU_MAKEFILE=$TAU_ROOT
/lib/Makefile.tau-python-mpi-pdt

% export PATH=$TAU_ROOT/bin:$PATH

% export TAU_OPTIONS='-optShared -optVerbose...'
(Python needs shared object based TAU library)

% make F90=tau_f90.sh CXX=tau_cxx.sh CC=tau_cc.sh (build libs, pyMPI w/TAU)

% cat wrapper.py
import tau
def OurMain():
    import App
    tau.run('OurMain()')
Uninstrumented:
% mpirun -np 4 pyMPI ./App.py
Instrumented:
% export PYTHONPATH= <taudir>/x86_64/<lib>/bindings-python-mpi-pdt
(same options string as TAU_MAKEFILE)
% export LD_LIBRARY_PATH=<taudir>/x86_64/lib/bindings-python-mpi-pdt:
$LD_LIBRARY_PATH
% mpirun -np 4 tau_exec -T python,mpi,pdt pyMPI ./wrapper.py
(Instruments pyMPI with wrapper.py)
```

68

Using TAU with Java Applications

Step I: Sun JDK 1.6+ [download from www.javasoft.com]

Step II: Configure TAU with JDK (v 1.6 or better)

```
% configure -jdk=/usr/jdk1.6
```

```
% make clean; make install
```

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

For Java (without instrumentation):

```
% java application
```

With instrumentation:

```
% tau_java application
```

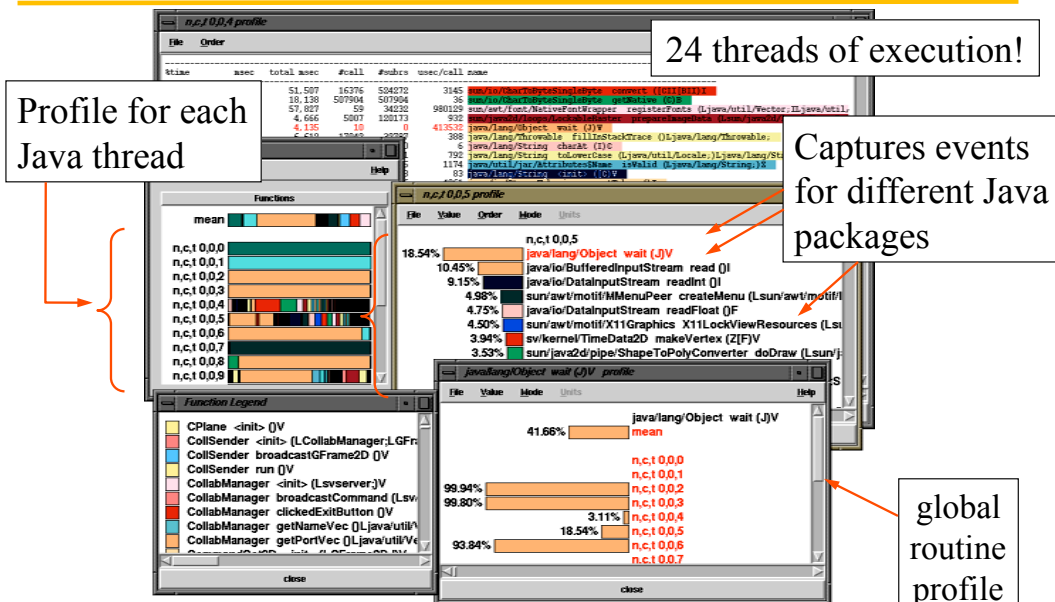
```
% tau_java -tau:agentlib=<different_libTAU.so> -tau:include=<item>
-tau:exclude=<item> application
```

Excludes where item=*.<init>;Foobar.method;sun.*classes

ParaTools

69

TAU Profiling of Java Application (SciVis)

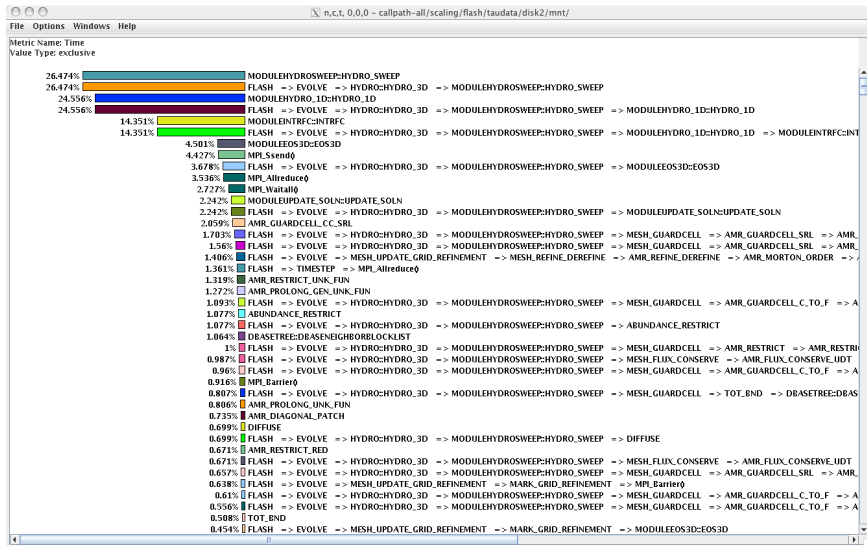


ParaTools

70

Usage Scenarios: Generating Callpath Profile

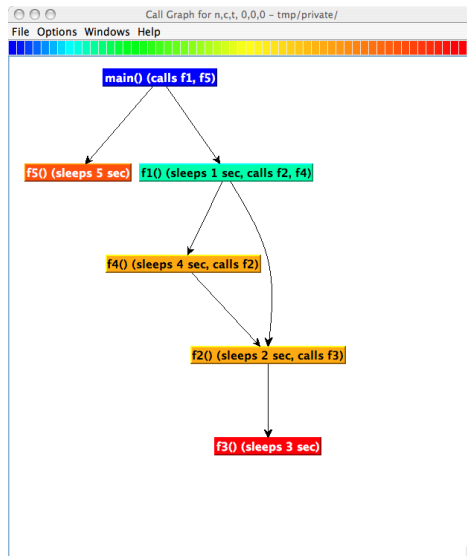
- Callpath profile for a given callpath depth:



ParaTools

Callpath Profile

- Generates program callgraph



ParaTools

Generate a Callpath Profile

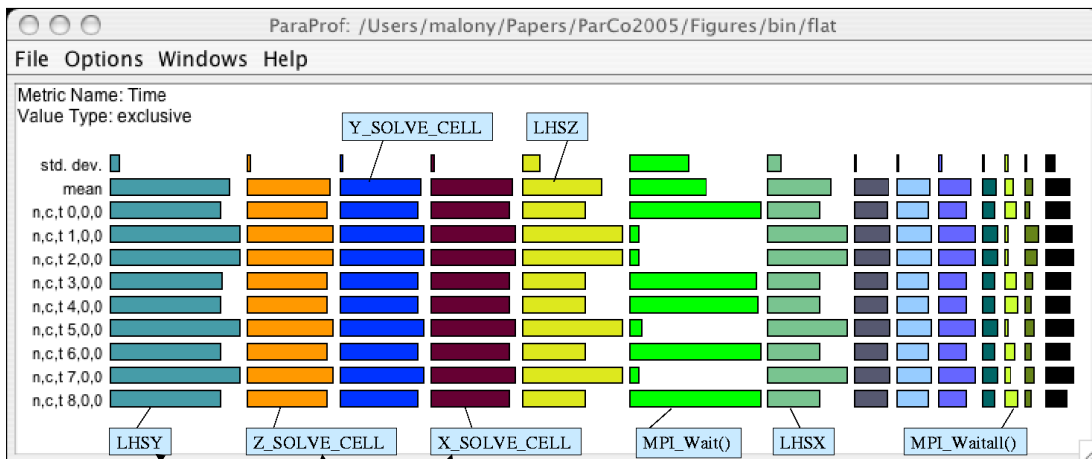
```
% export TAU_MAKEFILE=$TAU/Makefile.tau-mpi-pdt-pgi
% export PATH=$TAU_ROOT/bin:$PATH
% make F90=tau_f90.sh CXX=tau_cxx.sh
(Or edit Makefile and change CXX and F90)
% export TAU_CALLPATH=1
% export TAU_CALLPATH_DEPTH=100

% aprun -n 4 ./a.out
% paraprof --pack app.ppk
  Move the app.ppk file to your desktop.
% paraprof app.ppk
(Windows -> Thread -> Call Graph)
```

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

Phase Profiling (NAS BT, Flat Profile)



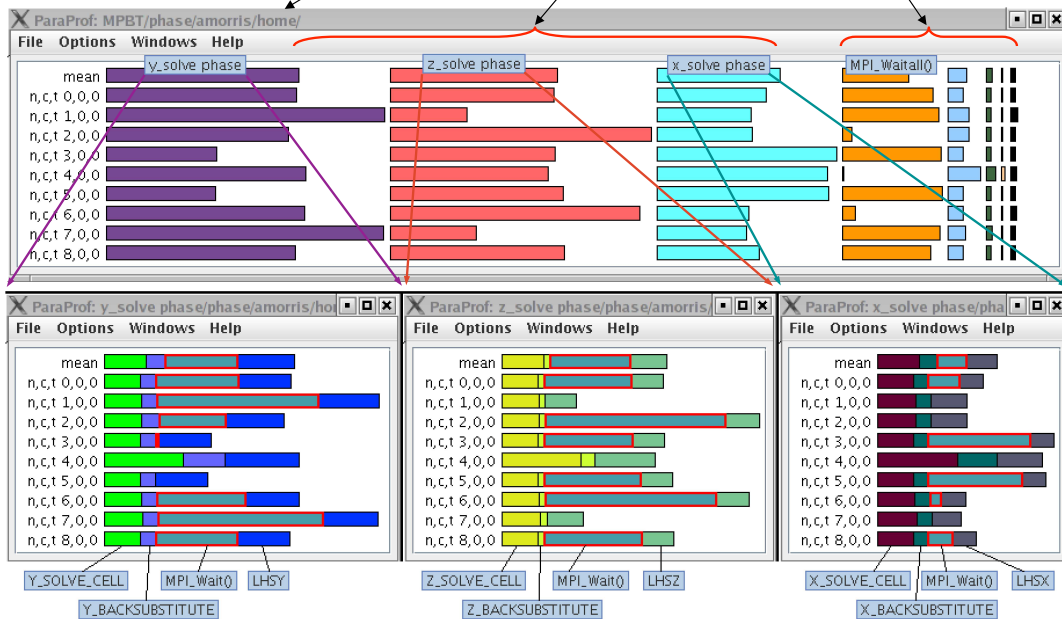
Application routine names reflect phase semantics

How is MPI_Wait() distributed relative to solver direction?

ParaTools

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

Main phase shows nested phases and immediate events

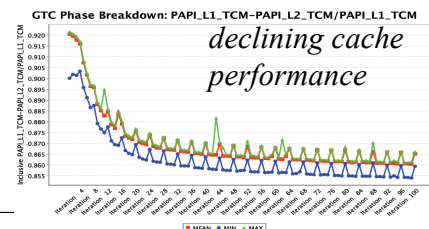
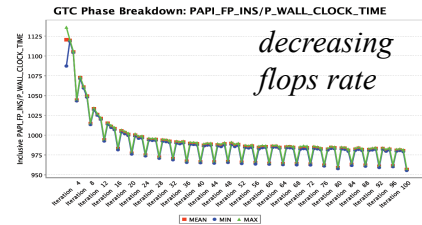
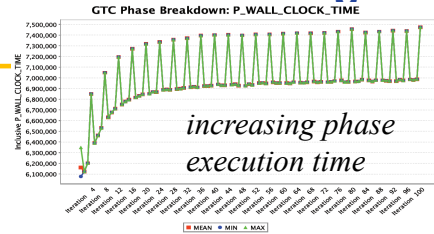


TAU Timers and Phases

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

Performance Dynamics: Phase-Based Profiling

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



Library interposition/wrapping: tau_exec, tau_wrap

- TAU provides a wealth of options to measure the performance of an application
- Need to simplify TAU usage to easily evaluate performance properties, including I/O, memory, and communication
- Designed a new tool (*tau_exec*) that leverages runtime instrumentation by pre-loading measurement libraries
- Works on dynamic executables (default under Linux, not on Cray)
- Substitutes I/O, MPI, and memory allocation/deallocation routines with instrumented calls
 - Interval events (e.g., time spent in write())
 - Atomic events (e.g., how much memory was allocated)
- Measure I/O and memory usage

ParaTools

TAU Execution Command (tau_exec)

- Uninstrumented execution
 - % mpirun -np 256 ./a.out
- Track MPI performance (-T <options>)
 - % mpirun -np 256 **tau_exec** ./a.out
- Track I/O and MPI performance (MPI by default, use -T serial for serial)
 - % mpirun -np 256 **tau_exec -io** ./a.out
- Track memory operations
 - % setenv TAU_TRACK_MEMORY_LEAKS 1
 - % mpirun -np 256 **tau_exec -T papi,mpi,pdt -memory** ./a.out
- Track I/O performance and memory operations
 - % mpirun -np 256 **tau_exec -io -memory** ./a.out
- Track GPGPU operations
 - % mpirun -np 256 **tau_exec -cuda** ./a.out
 - % **tau_exec -T serial -cuda** ./a.out
 - % **tau_exec -T serial -opencl** ./a.out

ParaTools

Library wrapping: tau_gen_wrapper

- How to instrument an external library without source?
 - Source may not be available
 - Library may be too cumbersome to build (with instrumentation)
- Build a library wrapper tools
 - Used PDT to parse header files
 - Generate new header files with instrumentation files
 - Three methods to instrument: runtime preloading, linking, redirecting headers to re-define functions
- Application is instrumented
- Add the `-optTauWrapFile=<wrapperdir>/link_options.tau` file to `TAU_OPTIONS` env var while compiling with `tau_cc.sh`, etc.
- Wrapped library
 - Redirects references at routine callsite to a wrapper call
 - Wrapper internally calls the original
 - Wrapper has TAU measurement code

Paratools

81

HDF5 Library Wrapping

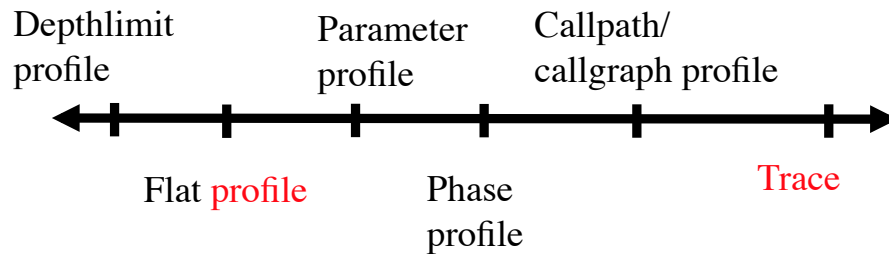
```
$ tau_gen_wrapper hdf5.h /usr/lib/libhdf5.a -f select.tau

Usage : tau_gen_wrapper <header> <library> [-r|-d|-w (default)] [-g groupname] [-i
headerfile] [-c|-c++|-fortran] [-f <instr_req_file> ]
• instruments using runtime preloading (-r), or -Wl,-wrap linker (-w), redirection
of header file to redefine the wrapped routine (-d)
• instrumentation specification file (select.tau)
• -g group may be specified (hdf5)
• tau_exec loads libhdf5_wrap.so shared library using -loadlib=<libwrap_pkg.so>
• creates the wrapper/ directory with linkoptions.tau passed to the TAU_OPTIONS
environment variable using -optTauWrapFile=<file>

NODE 0;CONTEXT 0;THREAD 0:
-----
%Time      Exclusive    Inclusive    #Call    #Subrs    Inclusive Name
          msec      total msec
-----
100.0      0.057        1            1         13        1236 .TAU Application
 70.8      0.875        0.875        1          0          875 hid_t H5Fcreate()
  9.7      0.12         0.12         1          0          120 herr_t H5Fclose()
  6.0      0.074        0.074        1          0           74 hid_t H5Dcreate()
  3.1      0.038        0.038        1          0           38 herr_t H5Dwrite()
  2.6      0.032        0.032        1          0           32 herr_t H5Dclose()
  2.1      0.026        0.026        1          0           26 herr_t H5check_version()
  0.6      0.008        0.008        1          0            8 hid_t H5Screate_simple()
  0.2      0.002        0.002        1          0            2 herr_t H5Tset_order()
  0.2      0.002        0.002        1          0            2 hid_t H5Tcopy()
  0.1      0.001        0.001        1          0            1 herr_t H5Sclose()
```

82

Performance Evaluation Alternatives



Each alternative has:

- one metric/counter
- multiple counters

Volume of performance data

ParaTools

83

-PROFILEPARAM Configuration Option

- Idea: partition performance data for individual functions based on runtime parameters
- Enable by configuring with **-PROFILEPARAM**
- Choose TAU stub makefile with **-param** in its name
- TAU call: `TAU_PROFILE_PARAM1L (value, "name")`

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

ParaTools

84

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

ParaTools

85

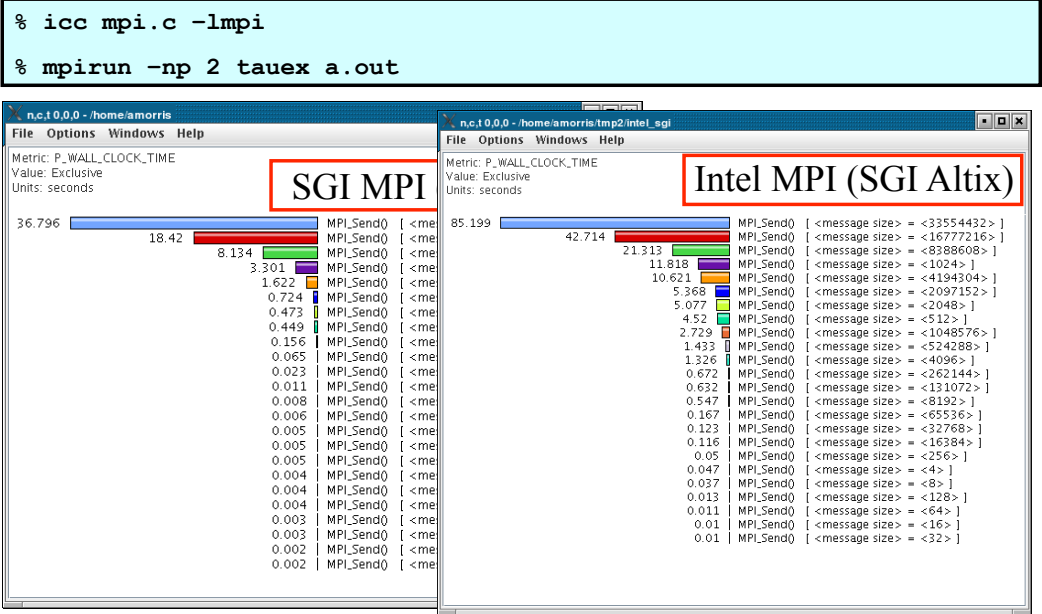
Workload Characterization

```
#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();
}
```

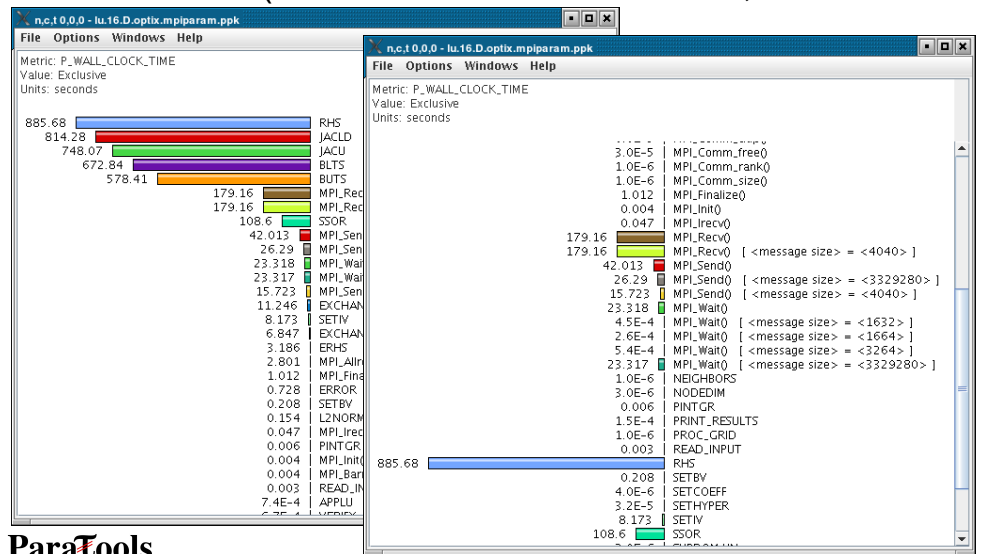
86

Workload Characterization



Workload Characterization

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



ParaTools

Workload Characterization

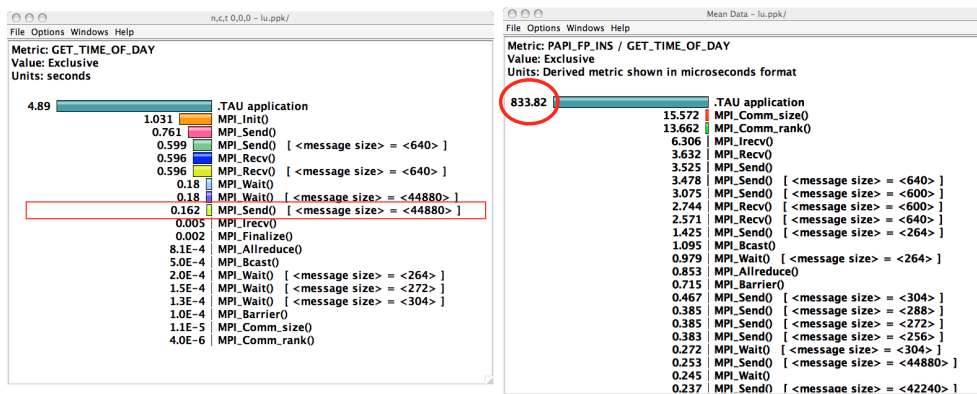
- Two different message sizes (~3.3MB and ~4K)

Name	Inclusive ...	Exclusive...	Calls	Child ...
MPI_Comm_tree()	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()	23.317	23.317	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

ParaTools

89

Job Tracking: ParaProf profile browser



LU spent 0.162 seconds sending messages of size 44880

It got 833.82 Mflops!

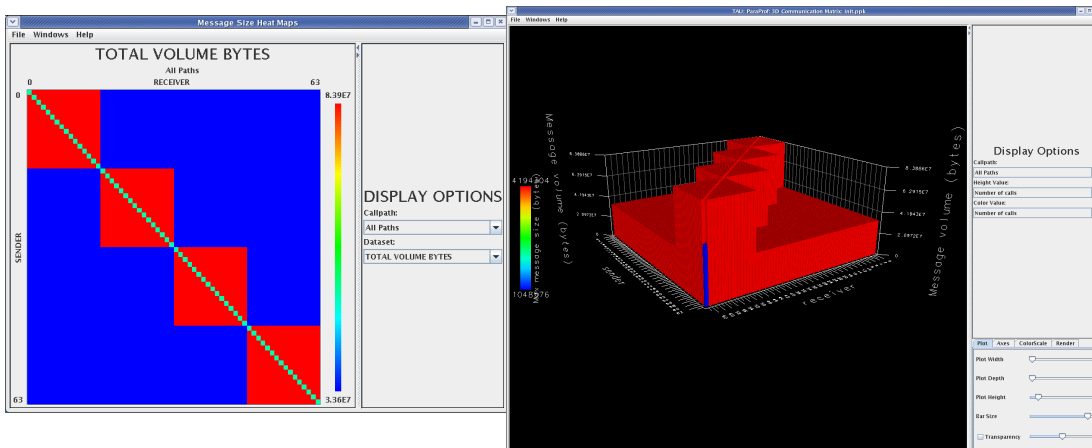
ParaTools

90

Parameter Based Profiling

```
% export TAU_MAKEFILE=$TAU/Makefile.tau-mpi-param-pdt-pgi
% export PATH=$TAU_ROOT/bin:$PATH
% make CC=tau_cc.sh
% aprun -n 4 ./a.out
% paraprof
```

ParaProf: Communication Matrix Display



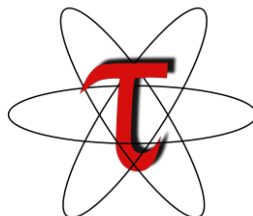
Communication Matrix

```
% export TAU_MAKEFILE=$TAU/Makefile.tau-mpi-pdt
% export PATH=$TAU_ROOT/bin:$PATH
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)

%
% export TAU_COMM_MATRIX=1
% aprun -n 4 ./a.out (setting the environment variables)

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

Techniques for manual instrumentation of individual routines



Instrumenting a C code

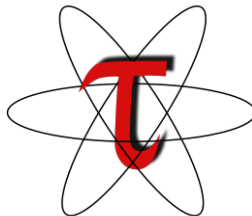
```
#include <TAU.h>
int foo(int x) {
    TAU_START("foo");
    for (i = 0; i < x; i++) { // do work
    }
    TAU_STOP("foo");
}

int main(int argc, char **argv) {
    TAU_INIT(&argc, &argv);
    TAU_START("main");
    TAU_PROFILE_SET_NODE(rank);
    ...
    TAU_STOP("main");
}
% gcc -I<taudir>/include foo.c -o foo -L<taudir>/<arch>/lib -lTAU
% ./a.out
% pprof; paraprof
NOTE: Replace TAU_START("foo") with call TAU_START('foo')
      in Fortran. See <taudir>/include/TAU.h for full API.
```

ParaTools

95

Using PAPI and TAU



ParaTools

96

Hardware Counters

Hardware performance counters available on most modern microprocessors can provide insight into:

1. Whole program timing
2. Cache behaviors
3. Branch behaviors
4. Memory and resource access patterns
5. Pipeline stalls
6. Floating point efficiency
7. Instructions per cycle

Hardware counter information can be obtained with:

1. Subroutine or basic block resolution
2. Process or thread attribution

What's PAPI?



-
- Open Source software from U. Tennessee, Knoxville
 - <http://icl.cs.utk.edu/papi>
 - Middleware to provide a consistent programming interface for the performance counter hardware found in most major microprocessors.
 - Countable events are defined in two ways:
 - Platform-neutral *preset* events
 - Platform-dependent *native* events
 - Presets can be **derived** from multiple *native events*
 - All events are referenced by name and collected in EventSets

PAPI Utilities: *papi_avail*

```
$ utils/papi_avail -h
Usage: utils/papi_avail [options]
Options:

General command options:
-a, --avail    Display only available preset events
-d, --detail   Display detailed information about all preset events
-e EVENTNAME   Display detail information about specified preset or native event
-h, --help     Print this help message

Event filtering options:
--br          Display branch related PAPI preset events
--cache       Display cache related PAPI preset events
--cnd         Display conditional PAPI preset events
--fp          Display Floating Point related PAPI preset events
--ins         Display instruction related PAPI preset events
--idl         Display Stalled or Idle PAPI preset events
--l1          Display level 1 cache related PAPI preset events
--l2          Display level 2 cache related PAPI preset events
--l3          Display level 3 cache related PAPI preset events
--mem         Display memory related PAPI preset events
--msc         Display miscellaneous PAPI preset events
--tlb         Display Translation Lookaside Buffer PAPI preset events

This program provides information about PAPI preset and native events.
PAPI preset event filters can be combined in a logical OR.
```

PAPI Utilities: *papi_avail*

```
$ utils/papi_avail
Available events and hardware information.
-----
PAPI Version      : 4.0.0.0
Vendor string and code : GenuineIntel (1)
Model string and code  : Intel Core i7 (21)
CPU Revision      : 5.000000
CPUID Info        : Family: 6 Model: 26 Stepping: 5
CPU Megahertz     : 2926.000000
CPU Clock Megahertz : 2926
Hdw Threads per core : 1
Cores per Socket  : 4
NUMA Nodes        : 2
CPU's per Node    : 4
Total CPU's       : 8
Number Hardware Counters : 7
Max Multiplex Counters : 32
-----
The following correspond to fields in the PAPI_event_info_t structure.

[MORE...]
```

PAPI Utilities: *papi_avail*

```
[CONTINUED...]  
  
-----  
The following correspond to fields in the PAPI_event_info_t structure.  
  
Name          Code      Avail Deriv Description (Note)  
PAPI_L1_DCM   0x80000000 No     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_VEC_SP   0x80000069 Yes    No    Single precision vector/SIMD instructions  
PAPI_VEC_DP   0x8000006a Yes    No    Double precision vector/SIMD instructions  
  
-----  
Of 107 possible events, 34 are available, of which 9 are derived.  
  
avail.c          PASSED
```

PAPI Utilities: *papi_avail*

```
$ utils/papi_avail -e PAPI_FP_OPS  
[...]  
  
-----  
The following correspond to fields in the PAPI_event_info_t structure.  
  
Event name:          PAPI_FP_OPS  
Event Code:          0x80000066  
Number of Native Events: 2  
Short Description:   |FP operations|  
Long Description:    |Floating point operations|  
Developer's Notes:   ||  
Derived Type:        |DERIVED_ADD|  
Postfix Processing String: ||  
Native Code[0]: 0x4000801b |FP_COMP_OPS_EXE:SSE_SINGLE_PRECISION|  
Number of Register Values: 2  
Register[ 0]: 0x0000000f |Event Selector|  
Register[ 1]: 0x00004010 |Event Code|  
Native Event Description: |Floating point computational micro-ops, masks:SSE* FP single precision Uops|  
  
Native Code[1]: 0x4000801b |FP_COMP_OPS_EXE:SSE_DOUBLE_PRECISION|  
Number of Register Values: 2  
Register[ 0]: 0x0000000f |Event Selector|  
Register[ 1]: 0x00008010 |Event Code|  
Native Event Description: |Floating point computational micro-ops, masks:SSE* FP double precision Uops|  
  
-----
```

PAPI Utilities: *papi_native_avail*

```
UNIX> utils/papi_native_avail
Available native events and hardware information.
-----
[...]
Event Code   Symbol   | Long Description |
-----
0x40000010   BR_INST_EXEC | Branch instructions executed |
40000410     :ANY      | Branch instructions executed |
40000810     :COND     | Conditional branch instructions executed |
40001010     :DIRECT   | Unconditional branches executed |
40002010     :DIRECT_NEAR_CALL | Unconditional call branches executed |
40004010     :INDIRECT_NEAR_CALL | Indirect call branches executed |
40008010     :INDIRECT_NON_CALL | Indirect non call branches executed |
40010010     :NEAR_CALLS | Call branches executed |
40020010     :NON_CALLS | All non call branches executed |
40040010     :RETURN_NEAR | Indirect return branches executed |
40080010     :TAKEN    | Taken branches executed |
-----
0x40000011   BR_INST_RETIRED | Retired branch instructions |
40000411     :ALL_BRANCHES | Retired branch instructions (Precise Event) |
40000811     :CONDITIONAL | Retired conditional branch instructions (Precise |
| Event) |
40001011     :NEAR_CALL | Retired near call instructions (Precise Event) |
-----
[...]
```

PAPI Utilities: *papi_native_avail*

```
UNIX> utils/papi_native_avail -e DATA_CACHE_REFILLS
Available native events and hardware information.
-----
[...]
-----
The following correspond to fields in the PAPI_event_info_t structure.

Event name:          DATA_CACHE_REFILLS
Event Code:          0x4000000b
Number of Register Values: 2
Description:         |Data Cache Refills from L2 or System|
Register[ 0]:        0x0000000f |Event Selector|
Register[ 1]:        0x00000042 |Event Code|

Unit Masks:
Mask Info:           |:SYSTEM|Refill from System|
Register[ 0]:        0x0000000f |Event Selector|
Register[ 1]:        0x00000142 |Event Code|
Mask Info:           |:L2_SHARED|Shared-state line from L2|
Register[ 0]:        0x0000000f |Event Selector|
Register[ 1]:        0x00000242 |Event Code|
Mask Info:           |:L2_EXCLUSIVE|Exclusive-state line from L2|
Register[ 0]:        0x0000000f |Event Selector|
Register[ 1]:        0x00000442 |Event Code|
```

PAPI Utilities: *papi_event_chooser*

```
$ utils/papi_event_chooser PRESET PAPI_FP_OPS
Event Chooser: Available events which can be added with given events.
-----
[...]
-----
      Name          Code      Deriv Description (Note)
PAPI_L1_DCM 0x80000000 No   Level 1 data cache misses
PAPI_L1_ICM 0x80000001 No   Level 1 instruction cache misses
PAPI_L2_ICM 0x80000003 No   Level 2 instruction cache misses
[...]
PAPI_L1_DCA 0x80000040 No   Level 1 data cache accesses
PAPI_L2_DCR 0x80000044 No   Level 2 data cache reads
PAPI_L2_DCW 0x80000047 No   Level 2 data cache writes
PAPI_L1_ICA 0x8000004c No   Level 1 instruction cache accesses
PAPI_L2_ICA 0x8000004d No   Level 2 instruction cache accesses
PAPI_L2_TCA 0x80000059 No   Level 2 total cache accesses
PAPI_L2_TCW 0x8000005f No   Level 2 total cache writes
PAPI_FML_INS 0x80000061 No   Floating point multiply instructions
PAPI_FDV_INS 0x80000063 No   Floating point divide instructions
-----
Total events reported: 34
event_chooser.c                                     PASSED
```

PAPI Utilities: *papi_event_chooser*

```
$ utils/papi_event_chooser PRESET PAPI_FP_OPS PAPI_L1_DCM
Event Chooser: Available events which can be added with given events.
-----
[...]
-----
      Name          Code      Deriv Description (Note)
PAPI_TOT_INS 0x80000032 No   Instructions completed
PAPI_TOT_CYC 0x8000003b No   Total cycles
-----
Total events reported: 2
event_chooser.c                                     PASSED
```

PAPI Utilities: *papi_event_chooser*

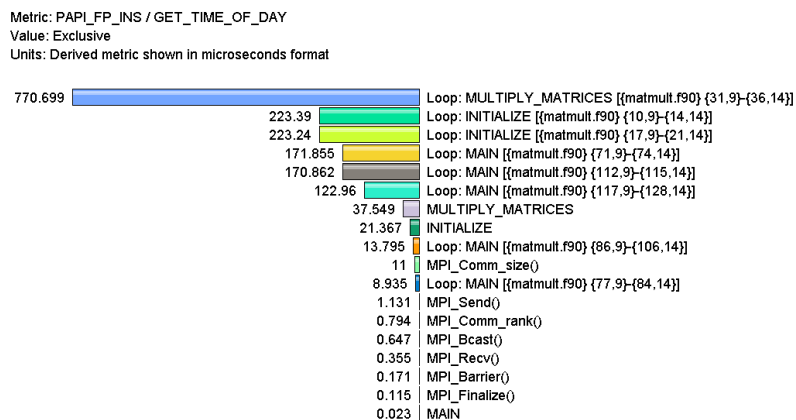
```

$ utils/papi_event_chooser NATIVE RESOURCE_STALLS:LD_ST X87_OPS_RETIRED
INSTRUCTIONS_RETIRED
[...]
-----
UNHALTED_CORE_CYCLES      0x40000000
|count core clock cycles whenever the clock signal on the specific core is running (not
|halted). Alias to event CPU_CLK_UNHALTED:CORE_P|
|Register Value[0]: 0x20003      Event Selector|
|Register Value[1]: 0x3c         Event Code|
-----
UNHALTED_REFERENCE_CYCLES 0x40000002
|Unhalted reference cycles. Alias to event CPU_CLK_UNHALTED:REF|
|Register Value[0]: 0x40000      Event Selector|
|Register Value[1]: 0x13c        Event Code|
-----
CPU_CLK_UNHALTED          0x40000028
|Core cycles when core is not halted|
|Register Value[0]: 0x60000      Event Selector|
|Register Value[1]: 0x3c         Event Code|
    0x40001028 :CORE_P |Core cycles when core is not halted|
    0x40008028 :NO_OTHER |Bus cycles when core is active and the other is halted|
-----
Total events reported: 3
event_chooser.c                PASSED

```

Usage Scenarios: Calculate mflops in Loops

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



Generate a PAPI profile with 2 or more counters

```

% export TAU_MAKEFILE=$TAU/Makefile.tau-papi-mpi-pdt-pgi
% export TAU_OPTIONS='-optTauSelectFile=select.tau -optVerbose'
% cat select.tau
  BEGIN_INSTRUMENT_SECTION
    loops routine="#"
  END_INSTRUMENT_SECTION

% export PATH=$TAU_ROOT/bin:$PATH
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)
%
% export TAU_METRICS=TIME:PAPI_FP_INS:PAPI_L1_DCM
% aprun -n 4 ./a.out
% paraprof --pack app.ppk
  Move the app.ppk file to your desktop.
% paraprof app.ppk
  Choose Options -> Show Derived Metrics Panel -> "PAPI_FP_INS", click "/", "TIME", click
  "Apply" and choose the derived metric.

```

ParaTools

Derived Metrics in ParaProf

The screenshot shows the TAU ParaProf Manager window. On the left is a tree view of applications, with 'PAPI_FP_INS' selected under 'Standard Applications'. The main area displays a table of metrics and their values for the selected application.

Truefield	Value
Name	is_papi.ppk
Application ID	0
Experiment ID	0
Trail ID	0
CPU Cores	6
CPU Mhz	2500.000
CPU Type	6-Core AMD Opteron(tm) Processor 23 (D0)
CPU Vendor	AuthenticAMD
CMD	/jstree/widow/scratch/sameer/workshop/NP83.1/bin
Cache Size	512 KB
Command Line	./is.C.S
Executable	/usr/spool/alpha/4660266/is.C.S
File Type Index	0
File Type Name	ParaProf packed Profile
Hostname	nd00772
Local Time	2011-06-26T23:52:45-04:00
MP Processor Name	nd00772
Memory Size	16385772 KB
Node Name	nd00772
OS Machine	x86_64
OS Name	Linux
OS Release	2.6.18-60-0.69.1.1.0102.S589.2.2.73-ent
OS Version	#1 SMP Tue Nov 16 18:03:32 CST 2010
Starting Timestamp	1309146753453169
TAU Architecture	craype
TAU Config	-papi=/opt/xt-tools/papi/3.7.2/r23/ -arch=craycnl -pdt=/ccs/proj/perc/TOOLS/pdt/pdtoolkit-3.16-pf -pdt_c++g++ -mpi...
TAU Makefile	/ccs/proj/perc/TOOLS/tau/tau-2.20.2/craycnl/lib/Makefile.tau-papi-mpi-pdt-pgi
TAU Metadata Merge Time	3.5E-05 seconds
TAU Version	2.20.2
TAU_CALLPATH	off
TAU_CALLPATH_DEPTH	100
TAU_COMM_MATRIX	off
TAU_COMPENSATE	off
TAU_CUPTI_API	runtime
TAU_PROFILE	on
TAU_PROFILE_FORMAT	profile
TAU_SAMPLING	off
TAU_THROTTLE	on
TAU_THROTTLE_NUMCALLS	100000
TAU_THROTTLE_PERCALL	10
TAU_TRACE	off
TAU_TRACK_HEADROOM	off
TAU_TRACK_HEAP	off
TAU_TRACK_IO_PARAMS	off
TAU_TRACK_MEMORY_LEAKS	off
TAU_TRACK_MESSAGE	off
Timestamp	1309146765912145
UTC Time	2011-06-27T03:52:45Z
pid	30336

At the bottom, there is an 'Expression' field containing 'PAPI_FP_INS"/>

ParaTools

ParaProf's Source Browser: Loop Level Instrumentation

TAU: ParaProf: Function Data Window: s3d_callpath_papi.ppk

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

	1.088	std. dev.
114.979	mean	
117.62	n.ct 0,0	
115.134	n.ct 1,0	
114.709	n.ct 2,0	
114.615	n.ct 3,0	
113.547	n.ct 4,0	
114.381	n.ct 5,0	
114.837	n.ct 6,0	
114.789	n.ct 7,0	

TAU: ParaProf: Function Data Window: s3d_callpath_papi.ppk

Name: Loop: TRANSPORT_Mc:COMPUTESPECIESDIFFFLUX [(mixavg_transport_m.pp.f90) (630,5)-(656,19)]
Metric Name: GET_TIME_OF_DAY
Value: Exclusive percent

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

TAU: ParaProf: Function Data Window: s3d_callpath_papi.ppk

Name: Loop: TRANSPORT_Mc:COMPUTESPECIESDIFFFLUX [(mixavg_transport_m.pp.f90) (630,5)-(656,19)]
Metric Name: PAPI_LL_DCM
Value: Exclusive
Units: counts

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

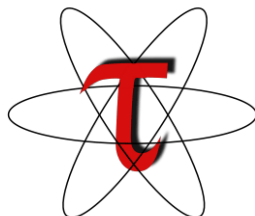
TAU: ParaProf: Source Browser: /mnt/epsilon/Users/sameer/rs/taudata/s3d/harness/flat/papi8

```

606 grad_mlxmw(:,1:n) = grad_mlxmw(:,1:n)*absolut(:,1)
607 end do
608
609 ! compute grad_P
610 if (baro_switch) then
611   allocate(grad_P(mv,nz,3))
612   grad_P = 0.0
613   if (vary_in_x == 1) then
614     call derivative_x( mx,ny,nz, Press, grad_P(:,1:1), scale_x, 1 )
615   endif
616   if (vary_in_y == 1) then
617     call derivative_y( mx,ny,nz, Press, grad_P(:,1:2), scale_y, 1 )
618   endif
619   if (vary_in_z == 1) then
620     call derivative_z( mx,ny,nz, Press, grad_P(:,1:3), scale_z, 1 )
621   endif
622 endif
623
624 ! Changed by Ramanan - 03/24/05
625 ! Ds_mlxavg is now vrbnd
626
627 !grad_P/press and avnolot*grad_T/Temp can be optimized by division before the loop.
628 ! compute diffusive flux for species n in direction n.
629 diffFlux(:,1:n,spec,:) = 0.0
630 DIRECTION = do n=1,3
631 SPECIES = do n=1,n_spec-1
632
633 if (baro_switch) then
634   ! driving force includes gradient in mole fraction and baro-diffusion
635   diffFlux(:,1:n,n) = - Ds_mlxavg(:,1:n) * ( grad_Ys(:,1:n,n) &
636     + Ys(:,1:n) * ( grad_mlxmw(:,1:n) &
637     + (1 - molwt(n)*avnolot) * grad_P(:,1:n,Press) )
638   else
639     ! driving force is just the gradient in mole fraction
640     diffFlux(:,1:n,n) = - Ds_mlxavg(:,1:n) * ( grad_Ys(:,1:n,n) &
641     + Ys(:,1:n) * grad_mlxmw(:,1:n) )
642   endif
643
644 ! Add thermal diffusion
645 if (thermDIFF_switch) then
646   diffFlux(:,1:n,n) = diffFlux(:,1:n,n) &
647     - Ds_mlxavg(:,1:n) * Rs_therm_diff(:,1:n,n) * molwt(n) &
648     * avnolot * grad_T(:,1:n,0) / Temp
649 endif
650
651 ! compute contribution to nth species diffusive flux
652 ! this will ensure that the sum of the diffusive fluxes is zero.
653 diffFlux(:,1:n,spec,n) = diffFlux(:,1:n,spec,n) - diffFlux(:,1:n,n)
654
655 enddo SPECIES
656 enddo DIRECTION
657
658 if (baro_switch) then
659   deallocate(grad_P)
660 endif
661
662 return
663 end subroutine computeSpeciesDiffFlux
664
665 !!-----
666
667
668 subroutine computeStressTensor( grad_u)
669

```

Estimation of tool intrusiveness



PAPI Utilities: *papi_cost*

```
$ utils/papi_cost -h
This is the PAPI cost program.
It computes min / max / mean / std. deviation for PAPI start/stop pairs;
for PAPI reads, and for PAPI_accums.
Usage:

    cost [options] [parameters]
    cost TESTS_QUIET

Options:

-b BINS          set the number of bins for the graphical
                 distribution of costs. Default: 100
-d              show a graphical distribution of costs
-h              print this help message
-s              show number of iterations above the first
                 10 std deviations
-t THRESHOLD    set the threshold for the number of
                 iterations. Default: 100,000
```

PAPI Utilities: *papi_cost*

```
$ utils/papi_cost
Cost of execution for PAPI start/stop and PAPI read.
This test takes a while. Please be patient...
Performing start/stop test...

Total cost for PAPI_start/stop(2 counters) over 1000000 iterations
min cycles   : 63
max cycles   : 17991
mean cycles  : 69.000000
std deviation: 34.035263
Performing start/stop test...

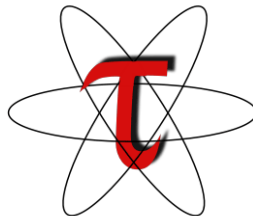
Performing read test...

Total cost for PAPI_read(2 counters) over 1000000 iterations
min cycles   : 288
max cycles   : 102429
mean cycles  : 301.000000
std deviation: 144.694053
cost.c                               PASSED
```

PAPI Utilities: *papi_cost*

```
Cost distribution profile
63:***** 99969 counts *****
153:
243:
[...]
1863:
1953:*****
2043:
2133:*****
2223:
2313:
2403:*****
2493:*****
2583:*****
2673:*****
2763:*****
2853:*****
2943:
3033:*****
3123:*****
3213:*****
3303:
3393:
3483:
3573:
3663:*****
```

Memory and I/O evaluation



Library interposition/wrapping: tau_exec, tau_wrap

- TAU provides a wealth of options to measure the performance of an application
- Need to simplify TAU usage to easily evaluate performance properties, including I/O, memory, and communication
- Designed a new tool (*tau_exec*) that leverages runtime instrumentation by pre-loading measurement libraries
- Works on dynamic executables (default under Linux)
- Substitutes I/O, MPI, and memory allocation/deallocation routines with instrumented calls
 - Interval events (e.g., time spent in write())
 - Atomic events (e.g., how much memory was allocated)
- Measure I/O and memory usage

ParaTools

TAU Execution Command (tau_exec)

- Configure TAU with `-iowrapper` configuration option
- Uninstrumented execution
 - `% mpirun -np 256 ./a.out`
- Track MPI performance
 - `% mpirun -np 256 tau_exec ./a.out`
- Track I/O and MPI performance (MPI enabled by default)
 - `% mpirun -np 256 tau_exec -io ./a.out`
- Track memory operations
 - `% setenv TAU_TRACK_MEMORY_LEAKS 1`
 - `% mpirun -np 256 tau_exec -memory ./a.out`
- Track I/O performance and memory operations
 - `% mpirun -np 256 tau_exec -io -memory ./a.out`
- Track GPGPU operations
 - `% mpirun -np 256 tau_exec -cuda ./a.out`

ParaTools

A New Approach: tau_exec

- Runtime instrumentation by pre-loading the measurement library
- Works on dynamic executables (default under Linux)
- Substitutes I/O, MPI and memory allocation/deallocation routines with instrumented calls
- Track interval events (e.g., time spent in write()) as well as atomic events (e.g., how much memory was allocated) in wrappers
- Accurately measure I/O and memory usage

Tracking I/O in static binaries (Cray)

- The linker can substitute TAU's I/O wrapper and intercept POSIX I/O Calls
- We can track parameters that flow through the I/O calls
- Configure TAU with `-iowrappers`
- Use `-optTrackIO` in `TAU_OPTIONS`

Tracking I/O in static binaries

```
% export TAU_MAKEFILE=$TAU/Makefile.tau-mpi-pdt-pgi
% export PATH=$TAU_ROOT/bin:$PATH
% export TAU_OPTIONS='-optTrackIO -optVerbose'
% make CC=tau_cc.sh CXX=tau_cxx.sh F90=tau_f90.sh
% aprun -n 4 ./a.out
% paraprof -pack ioprofile.ppk
% export TAU_TRACK_IO_PARAMS 1
% aprun -n 4 ./a.out (to track parameters used in POSIX I/O
  calls as context events)
```

Issues

- Heap memory usage reported by the mallinfo() call is not 64-bit clean.
 - 32 bit counters in Linux roll over when > 4GB memory is used
 - We keep track of heap memory usage in 64 bit counters inside TAU
- Compensation of perturbation introduced by tool
 - Only show what application uses
 - Create guards for TAU calls to not track I/O and memory allocations/de-allocations performed inside TAU
- Provide broad POSIX I/O and memory coverage

I/O Calls Supported

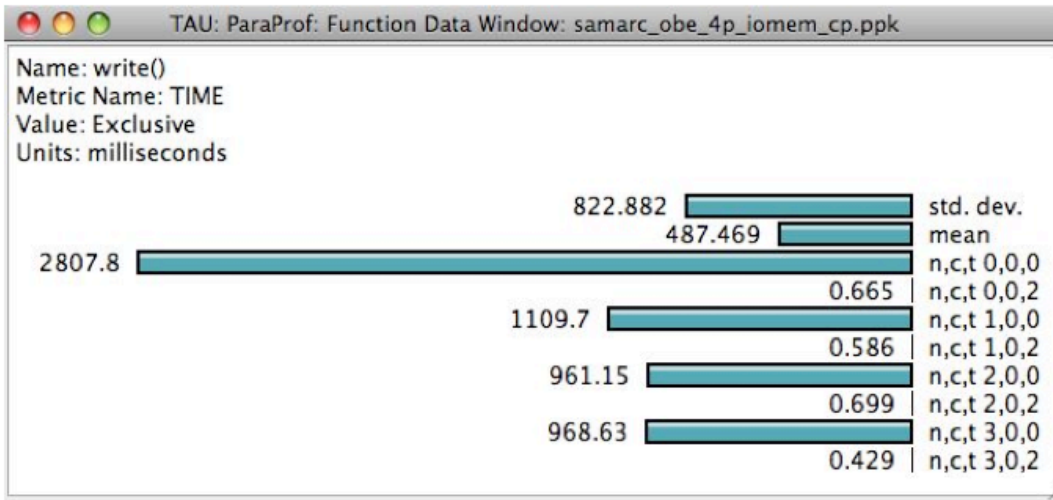
Unbuffered I/O	Buffered I/O	Communication	Control	Asynchronous I/O
open	fopen	socket	fcntl	aio_read
open64	fopen64	pipe	rewind	aio_write
close	fdopen	socketpair	lseek	aio_suspend
read	freopen	bind	lseek64	aio_cancel
write	fclose	accept	fseek	aio_return
readv	fprintf	connect	dup	lio_listio
writew	fscanf	recv	dup2	
creat	fwrite	send	mkstep	
creat64	fread	sendto	tmpfile	
		recvfrom		
		pclose		

ParaTools

Tracking I/O in Each File

Name	Total	NumSamples	MaxValue	MinValue	MeanValue	Std.Dev
Bytes Read	20,024	32	8,192	4	625.75	2,014.699
Bytes Read <file="/opt/openmpi/tm/intel/1.4/etc/openmpi-mca-params.conf">	2,812	1	2,812	2,812	2,812	0
Bytes Read <file="/opt/openmpi/tm/intel/1.4/share/openmpi/help-mpi-btl-opensib.txt">	8,192	1	8,192	8,192	8,192	0
Bytes Read <file="/opt/openmpi/tm/intel/1.4/share/openmpi/mca-btl-opensib-device-params.ini">	8,727	2	8,192	595	4,363.5	3,828.5
Bytes Read <file="/sys/class/infiniband/mthca0/node_type">	8	1	8	8	8	0
Bytes Read <file="/sys/class/infiniband/mthca0/ports/1/qids/0">	41	1	41	41	41	0
Bytes Read <file="/sys/class/infiniband_verbs/abi_version">	8	1	8	8	8	0
Bytes Read <file="/sys/class/infiniband_verbs/uverbs0/abi_version">	8	1	8	8	8	0
Bytes Read <file="/sys/class/infiniband_verbs/uverbs0/device/device">	24	3	8	8	8	0
Bytes Read <file="/sys/class/infiniband_verbs/uverbs0/device/vendor">	24	3	8	8	8	0
Bytes Read <file="/sys/class/infiniband_verbs/uverbs0/ibdev">	64	1	64	64	64	0
Bytes Read <file="/sys/devices/system/cpu/cpu0/topology/core_id">	7	1	7	7	7	0
Bytes Read <file="/sys/devices/system/cpu/cpu0/topology/physical_package_id">	7	1	7	7	7	0
Bytes Read <file="/sys/devices/system/cpu/cpu1/topology/core_id">	7	1	7	7	7	0
Bytes Read <file="/sys/devices/system/cpu/cpu1/topology/physical_package_id">	7	1	7	7	7	0
Bytes Read <file="/sys/devices/system/cpu/cpu2/topology/core_id">	7	1	7	7	7	0
Bytes Read <file="/sys/devices/system/cpu/cpu2/topology/physical_package_id">	7	1	7	7	7	0
Bytes Read <file="/sys/devices/system/cpu/cpu3/topology/core_id">	7	1	7	7	7	0
Bytes Read <file="/sys/devices/system/cpu/cpu3/topology/physical_package_id">	7	1	7	7	7	0
Bytes Read <file="/sys/devices/system/cpu/cpu4/topology/core_id">	7	1	7	7	7	0
Bytes Read <file="/sys/devices/system/cpu/cpu4/topology/physical_package_id">	7	1	7	7	7	0
Bytes Read <file="/sys/devices/system/cpu/cpu5/topology/core_id">	7	1	7	7	7	0
Bytes Read <file="/sys/devices/system/cpu/cpu5/topology/physical_package_id">	7	1	7	7	7	0
Bytes Read <file="/sys/devices/system/cpu/cpu6/topology/core_id">	7	1	7	7	7	0
Bytes Read <file="/sys/devices/system/cpu/cpu6/topology/physical_package_id">	7	1	7	7	7	0
Bytes Read <file="/sys/devices/system/cpu/cpu7/topology/core_id">	7	1	7	7	7	0
Bytes Read <file="/sys/devices/system/cpu/cpu7/topology/physical_package_id">	7	1	7	7	7	0
Bytes Read <file="/pipe">	4	1	4	4	4	0
READ Bandwidth (MB/s)	2,932.118	32	1,170.286	0.001	91.629	269.282
READ Bandwidth (MB/s) <file="/opt/openmpi/tm/intel/1.4/etc/openmpi-mca-params.conf">	312.444	1	312.444	312.444	312.444	0
READ Bandwidth (MB/s) <file="/opt/openmpi/tm/intel/1.4/share/openmpi/help-mpi-btl-opensib.txt">	1,170.286	1	1,170.286	1,170.286	1,170.286	0
READ Bandwidth (MB/s) <file="/opt/openmpi/tm/intel/1.4/share/openmpi/mca-btl-opensib-device-params.i">	1,291.5	2	1,024	267.5	645.75	378.25
READ Bandwidth (MB/s) <file="/sys/class/infiniband/mthca0/node_type">	4	1	4	4	4	0
READ Bandwidth (MB/s) <file="/sys/class/infiniband/mthca0/ports/1/qids/0">	0.304	1	0.304	0.304	0.304	0
READ Bandwidth (MB/s) <file="/sys/class/infiniband_verbs/abi_version">	4	1	4	4	4	0
READ Bandwidth (MB/s) <file="/sys/class/infiniband_verbs/uverbs0/abi_version">	4	1	4	4	4	0
READ Bandwidth (MB/s) <file="/sys/class/infiniband_verbs/uverbs0/device/device">	16	3	8	4	5.333	1.886
READ Bandwidth (MB/s) <file="/sys/class/infiniband_verbs/uverbs0/device/vendor">	20	3	8	4	6.667	1.886
READ Bandwidth (MB/s) <file="/sys/class/infiniband_verbs/uverbs0/ibdev">	32	1	32	32	32	0

Time Spent in POSIX I/O write()



ParaTools

125

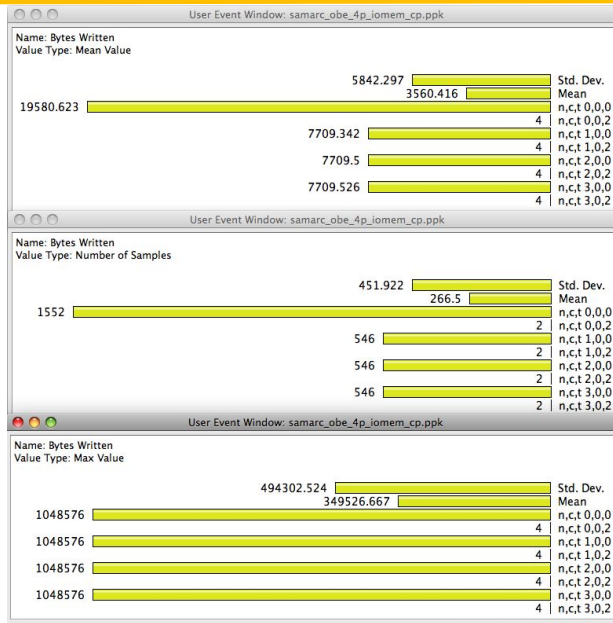
Volume of I/O by File, Memory

Name	Total	MeanValue	NumSamples	MinValue	MaxValue	Std. Dev.
writeRestartData	968.63	0.429	3	0	0.665	0.429
writeRestartData	961.15	0.699	2	0	0.699	0.699
writeRestartData	1109.7	0.586	1	0	0.586	0.586
writeRestartData	822.882		0	0	822.882	822.882
writeRestartData	487.469		0	0	487.469	487.469
writeRestartData	0.665		0	0	0.665	0.665
writeRestartData	2807.8		0	0	2807.8	2807.8
WRITE Bandwidth (MB/s) <file="samarc/restore.00002/nodes.00004/proc.00001">	74.565		117	0	2,156.889	246.386
WRITE Bandwidth (MB/s) <file="samarc/restore.00001/nodes.00004/proc.00001">	77.594		117	0	1,941.2	228.366
WRITE Bandwidth (MB/s)	76.08		234	0	2,156.889	237.551
Bytes Written <file="samarc/restore.00002/nodes.00004/proc.00001">	2,097,552	17,927.795	117	1	1,048,576	133,362.946
Bytes Written <file="samarc/restore.00001/nodes.00004/proc.00001">	2,097,552	17,927.795	117	1	1,048,576	133,362.946
Bytes Written	4,195,104	17,927.795	234	1	1,048,576	133,362.946

ParaTools

126

Bytes Written

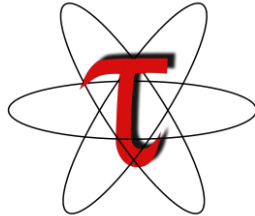


ParaTools

Memory Leaks in MPI

TAU: ParaProf: Context Events for thread: n,c,t,0,0,0 - samarc_obe_4p_iomem_cp.ppk						
Name	Total	MeanValue	NumSamples	MaxValue	MinValue	Std. Dev.
TAU application						
MPL_Finalize()						
free size	23,901,253	22,719.822	1,052	2,099,200	2	186,920.948
malloc size	5,013,902	65,972.395	76	5,000,000	2	569,732.815
MEMORY LEAK!	5,000,264	500,026.4	10	5,000,000	3	1,499,991.2
read()						
Bytes Read	4	4	1	4	4	0
READ Bandwidth (MB/s) <file="pipe">		0.308	1	0.308	0.308	0
Bytes Read <file="pipe">	4	4	1	4	4	0
READ Bandwidth (MB/s)		0.308	1	0.308	0.308	0
write()						
WRITE Bandwidth (MB/s)		0.635	102	12	0	1.472
Bytes Written <file="/dev/infiniband/rdma_cm">	24	24	1	24	24	0
Bytes Written	1,456	14.275	102	28	4	5.149
WRITE Bandwidth (MB/s) <file="/dev/infiniband/uverbs0">		0.528	97	12	0.089	1.32
Bytes Written <file="pipe">	64	16	4	28	4	12
WRITE Bandwidth (MB/s) <file="/dev/infiniband/rdma_cm">		1.714	1	1.714	1.714	0
Bytes Written <file="/dev/infiniband/uverbs0">	1,368	14.103	97	24	12	4.562
WRITE Bandwidth (MB/s) <file="pipe">		2.967	4	5.6	0	2.644
writev()						
WRITE Bandwidth (MB/s)		4.108	2	7.667	0.549	3.559
Bytes Written	297	148.5	2	230	67	81.5
WRITE Bandwidth (MB/s) <file="socket">		4.108	2	7.667	0.549	3.559
Bytes Written <file="socket">	297	148.5	2	230	67	81.5
readv()						
Bytes Read	112	28	4	36	20	8
READ Bandwidth (MB/s) <file="socket">		25.5	4	36	10	11.079
Bytes Read <file="socket">	112	28	4	36	20	8
READ Bandwidth (MB/s)		25.5	4	36	10	11.079
MPL_Comm_free()						
free size	10,952	195.571	56	1,024	48	255.353
read()						
MPL_Type_free()						
MPL_Init()						
fopen64()						
free size	231,314	263.456	878	568	35	221.272
MEMORY LEAK!	1,105,956	1,868.169	592	7,200	32	3,078.574
malloc size	1,358,286	901.318	1,507	7,200	32	2,087.737
OurMain()						
fclose()						

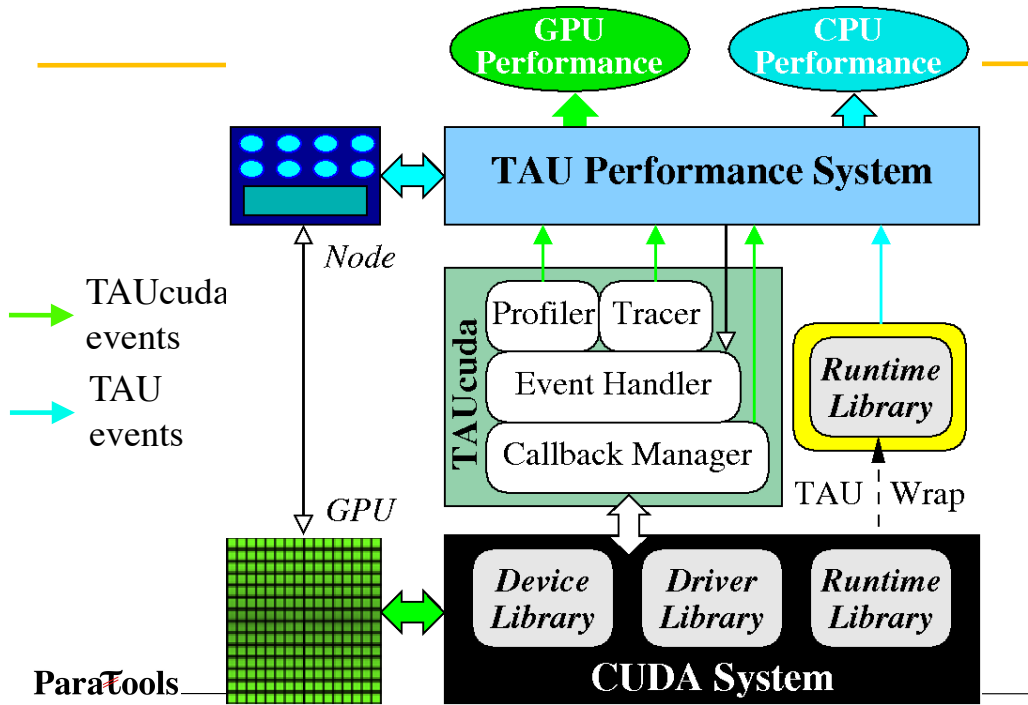
Profiling GPGPU Executions



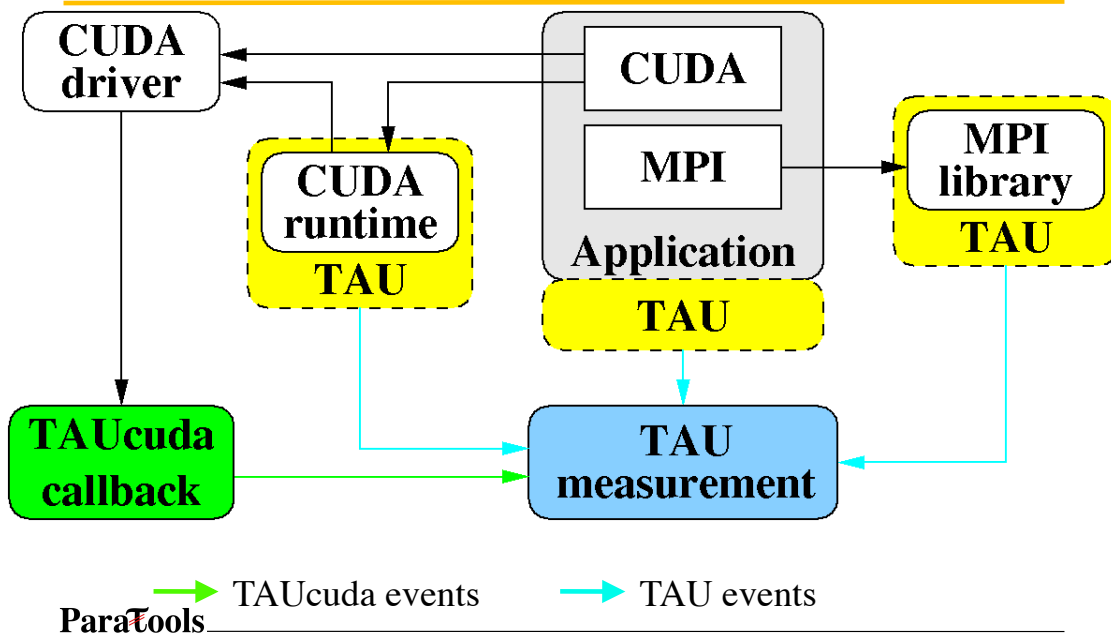
Profiling GPGPU Executions

- GPGPU compilers (e.g., CAPS hmpp and PGI) can now automatically generate GPGPU code using manual annotation of loop-level constructs and routines (hmpp)
- The loops (and routines for HMPP) are transferred automatically to the GPGPU
- TAU intercepts the runtime library routines and examines the arguments
- Shows events as seen from the host
- Profiles and traces GPGPU execution

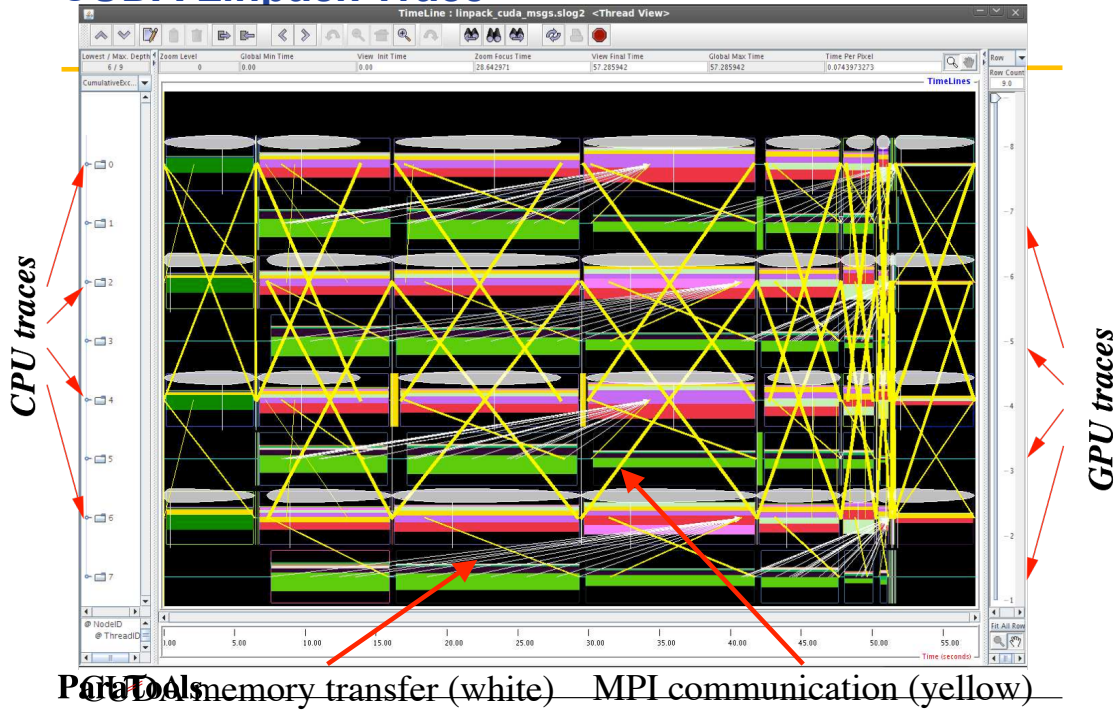
TAUcuda Architecture



TAUcuda Instrumentation

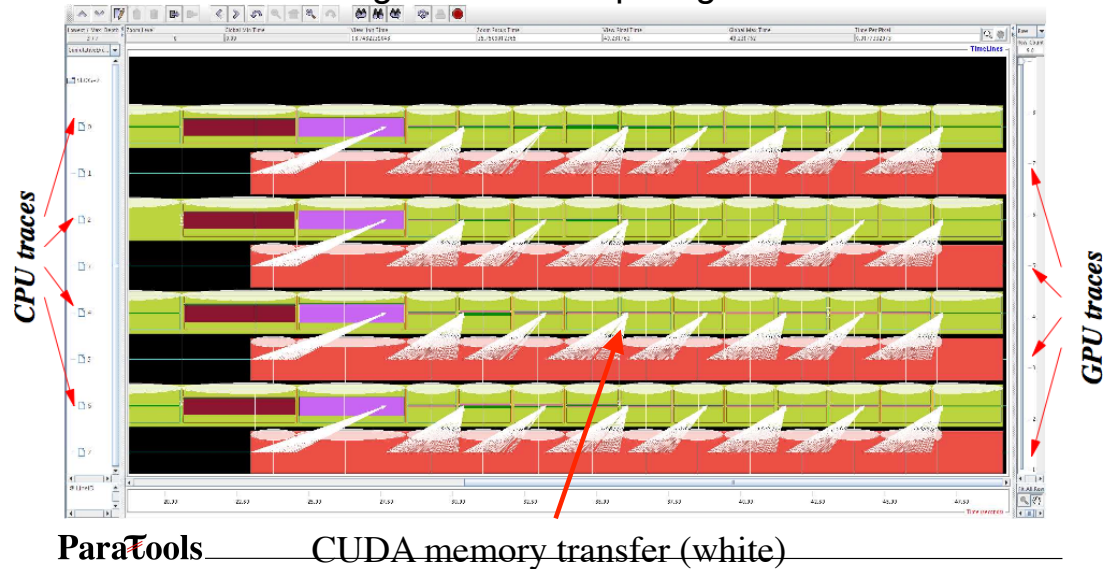


CUDA Linpack Trace

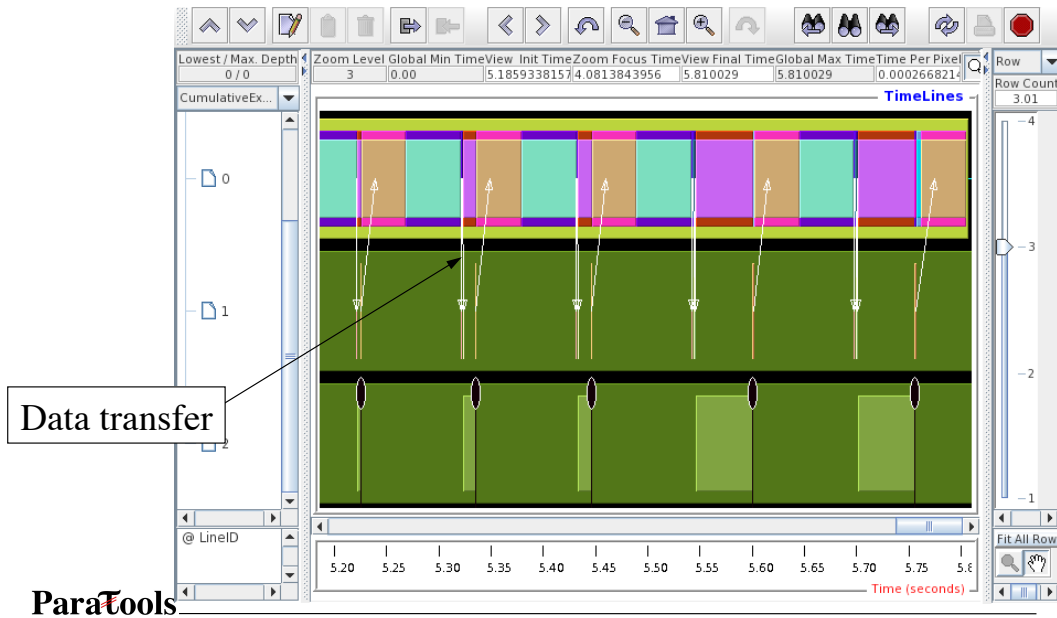


SHOC Stencil2D (512 iterations, 4 CPUxGPU)

- Scalable Heterogeneous Computing benchmark suite

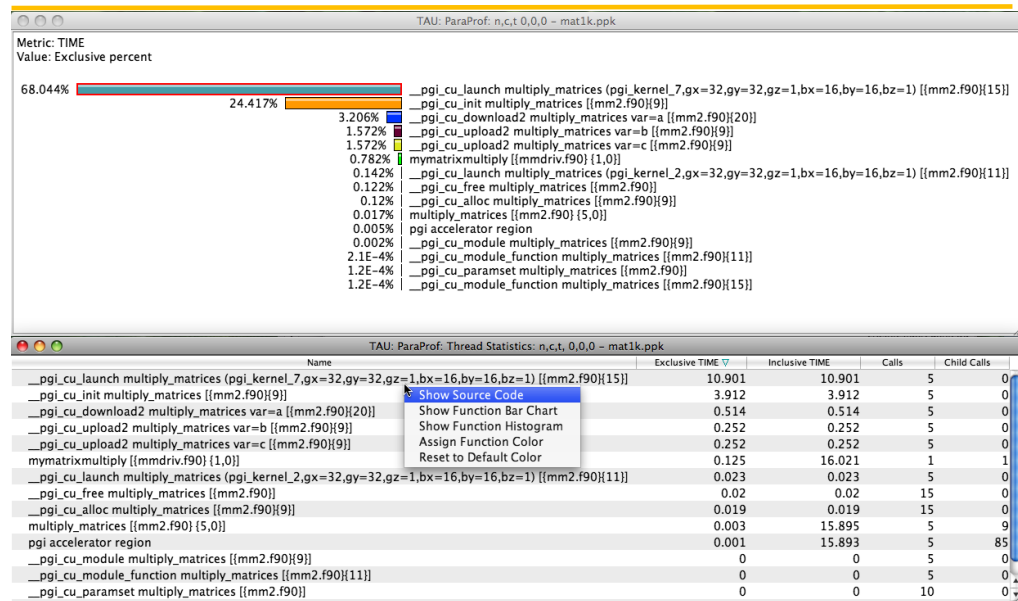


Scaling NAMD with CUDA (Jumpshot with TAU)



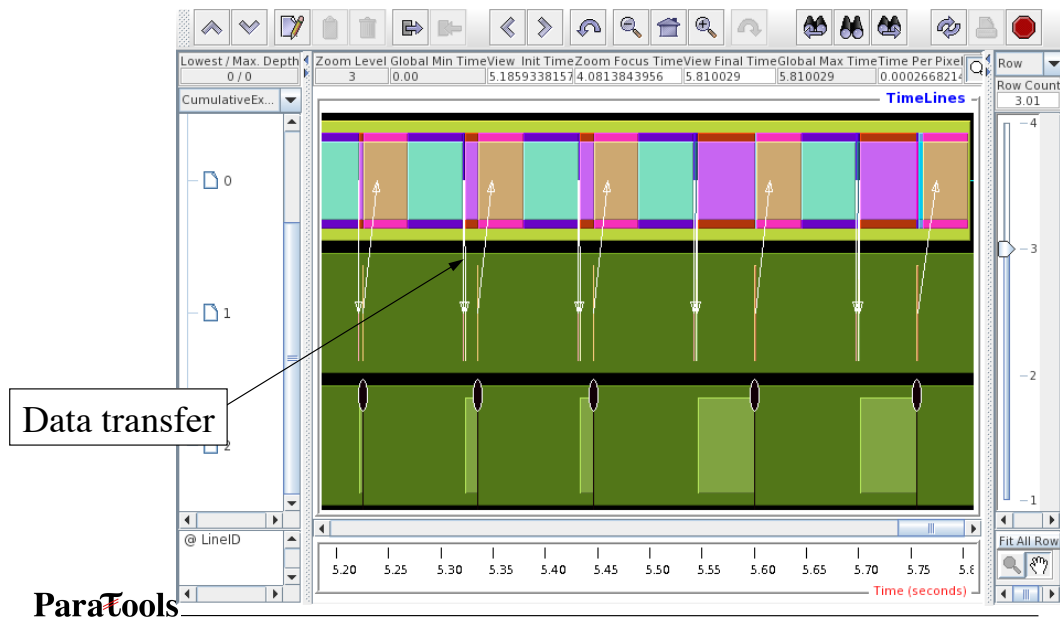
ParaTools

Measuring Performance of PGI GPGPU Accelerated Code



ParaTools

Scaling NAMD with CUDA (Jumpshot with TAU)



Using TAU for Tracing GPGPU Applications

Step I: Configure TAU with `-cuda=<dir>` for the SDK directory

Step II: Use the `tau_exec` script to launch the application

```
% export TAU_TRACE=1 (to enable tracing)
% tau_exec -T serial -cuda ./oceanFFT (for non-MPI code)
```

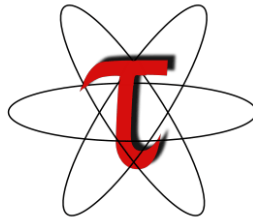
Step III: Merge trace files to create `tau.trc` and `tau.edf`:

```
% tau_multimerge
```

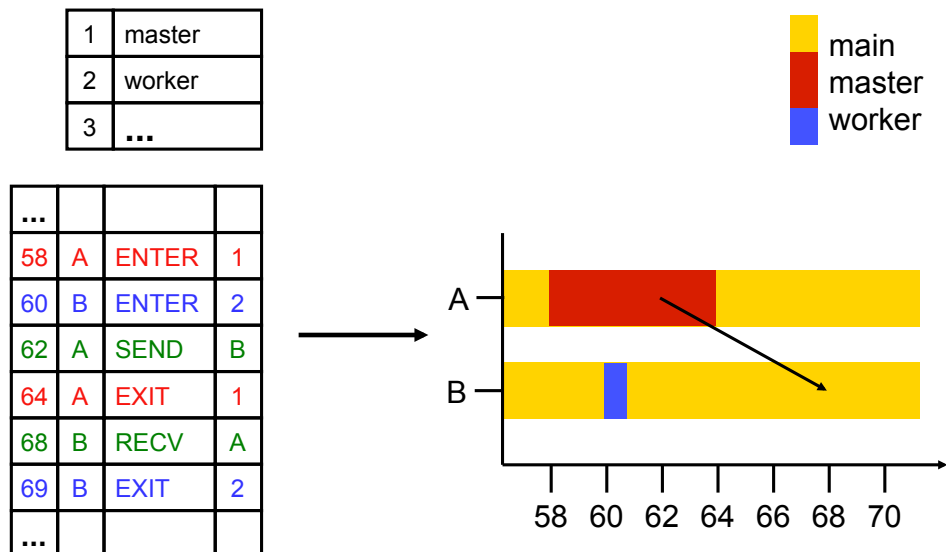
Step IV: Convert traces to Jumpshot, Vampir (OTF), ParaVer or other formats

```
% tau2slog2 tau.trc tau.edf -o app.slog2; jumpshot app.slog2
% tau_convert -paraver tau.trc tau.edf app.prv; paraver app.prv
% tau2otf tau.trc tau.edf app.otf; vampir app.otf
```

Generating event traces



Tracing Analysis and Visualization



Profiling / Tracing Comparison

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

Event Trace Visualization

Trace Visualization

- Alternative and supplement to automatic analysis
- Show dynamic run-time behavior visually
- Provide statistics and performance metrics
 - global timeline for parallel processes/threads
 - process timeline plus performance counters
 - statistic summary display
 - communication statistics, more ...
- Interactive browsing, zooming, selecting
 - adapt statistics to zoom level (time interval)
 - also for very large and highly parallel traces

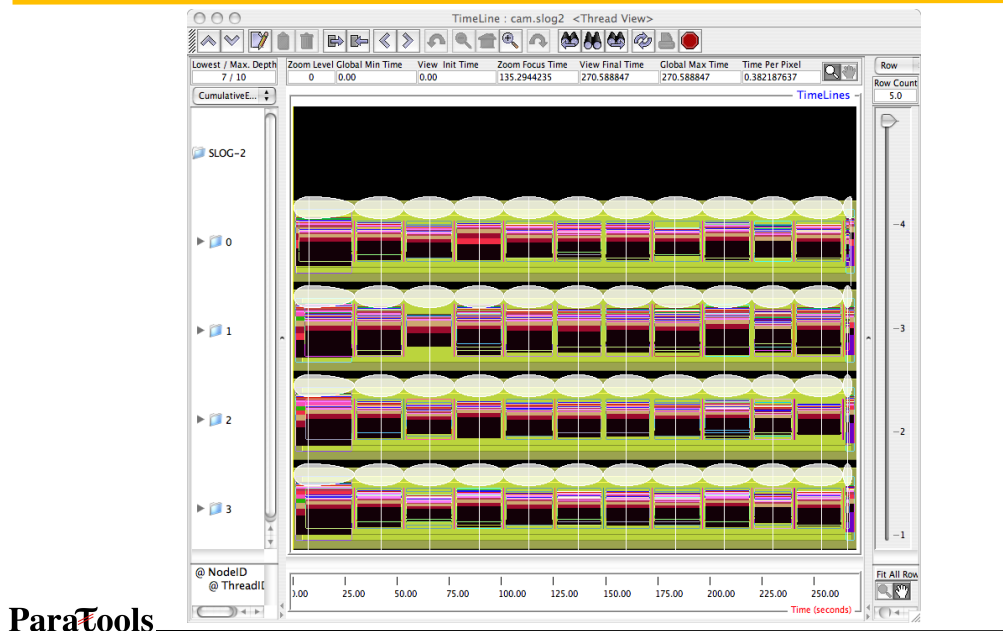
Trace Formats

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

Jumpshot

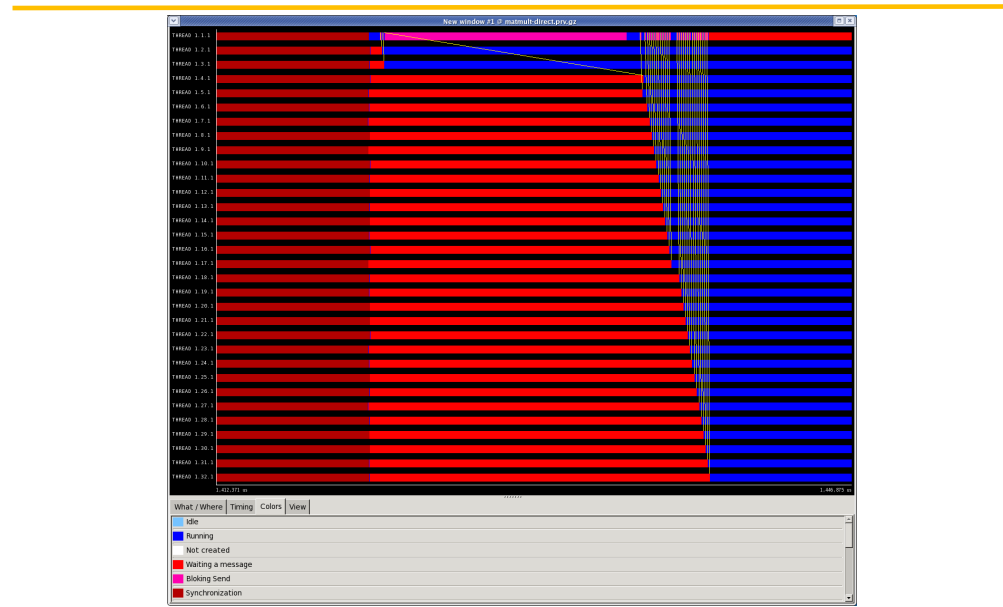
- <http://www-unix.mcs.anl.gov/perfvis/software/viewers/index.htm>
- Developed at Argonne National Laboratory as part of the MPICH project
 - Also works with other MPI implementations
 - Jumpshot is bundled with the TAU package
- Java-based tracefile visualization tool for postmortem performance analysis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

Jumpshot

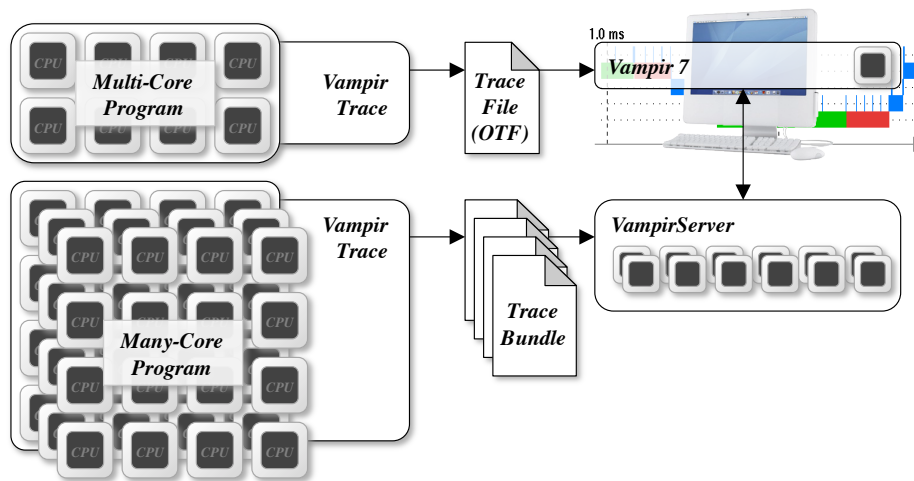


145

ParaVer [<http://www.bsc.es/paraver>]



Vampir Toolset Architecture [T.U. Dresden]



ParaTools

147

Vampir Displays

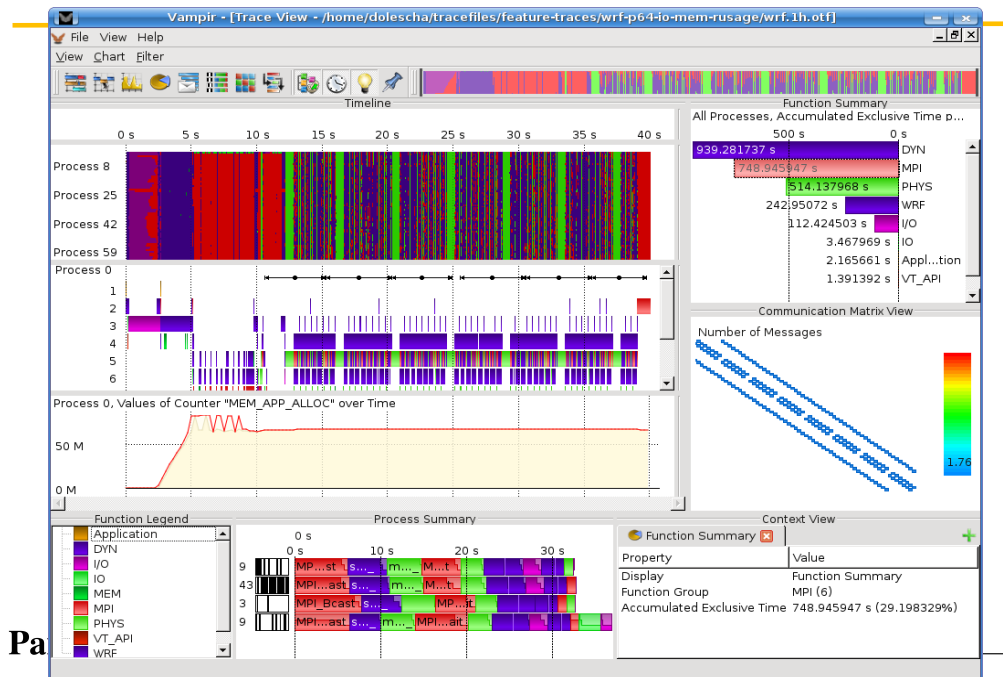
The main displays of Vampir:

- Master Timeline
- Process and Counter Timeline
- Function Summary
- Message Summary
- Process Summary
- Communication Matrix
- Call Tree

ParaTools

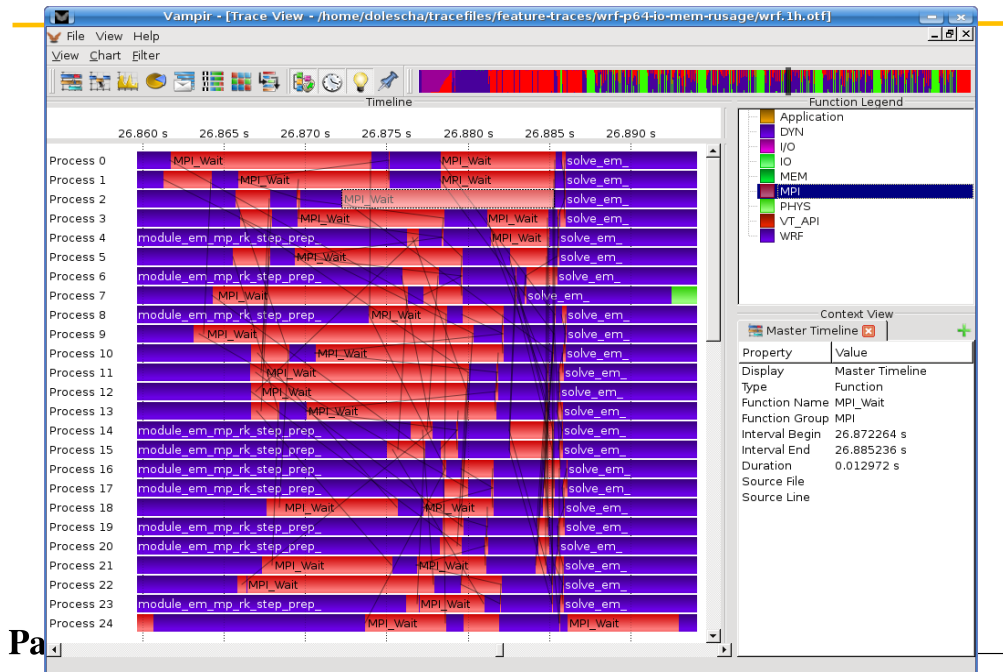
148 148

Vampir 7 Display Overview



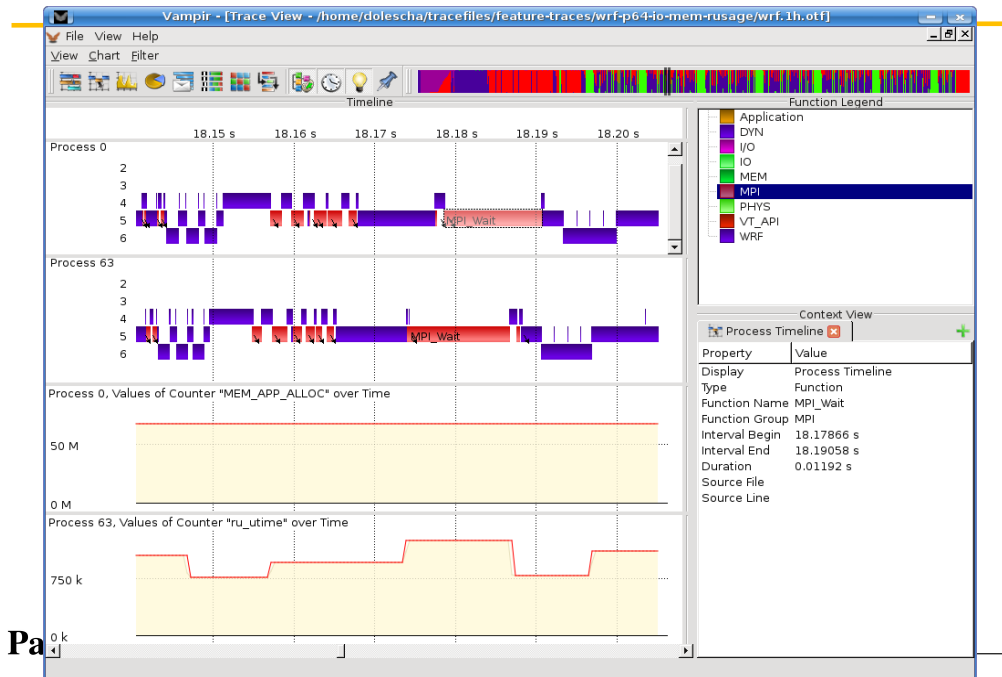
Pa

Master Timeline



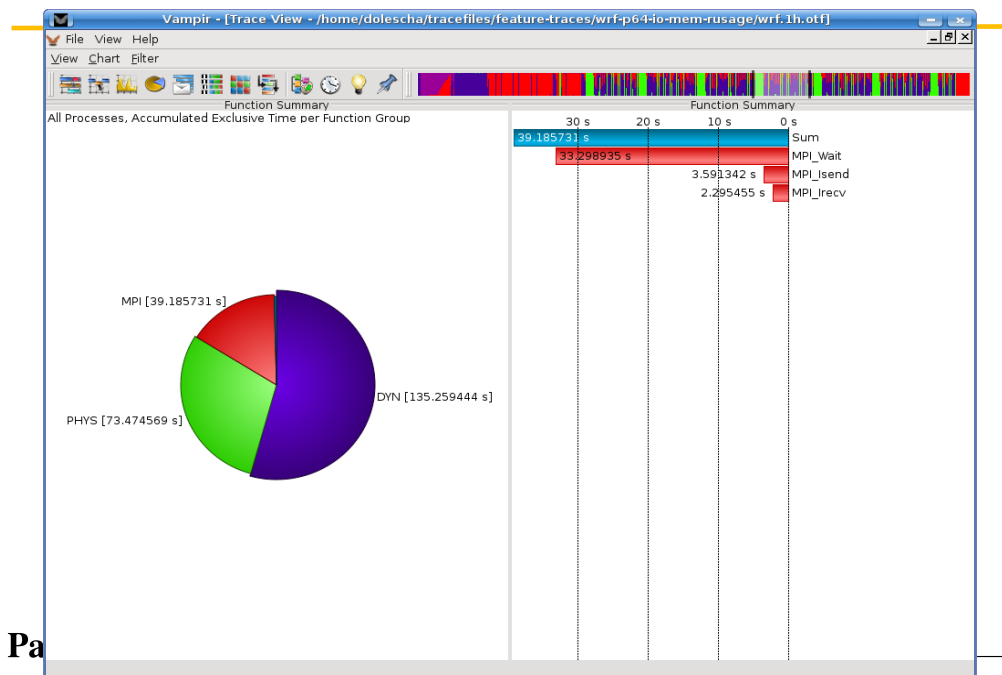
Pa

Process and Counter Timelines



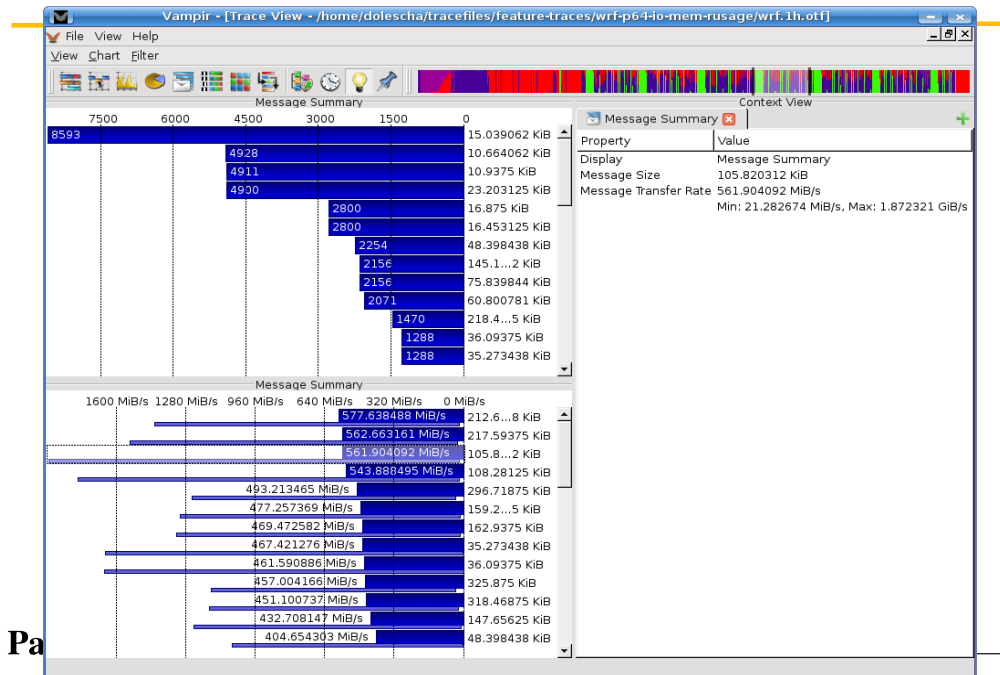
Pa

Function Summary



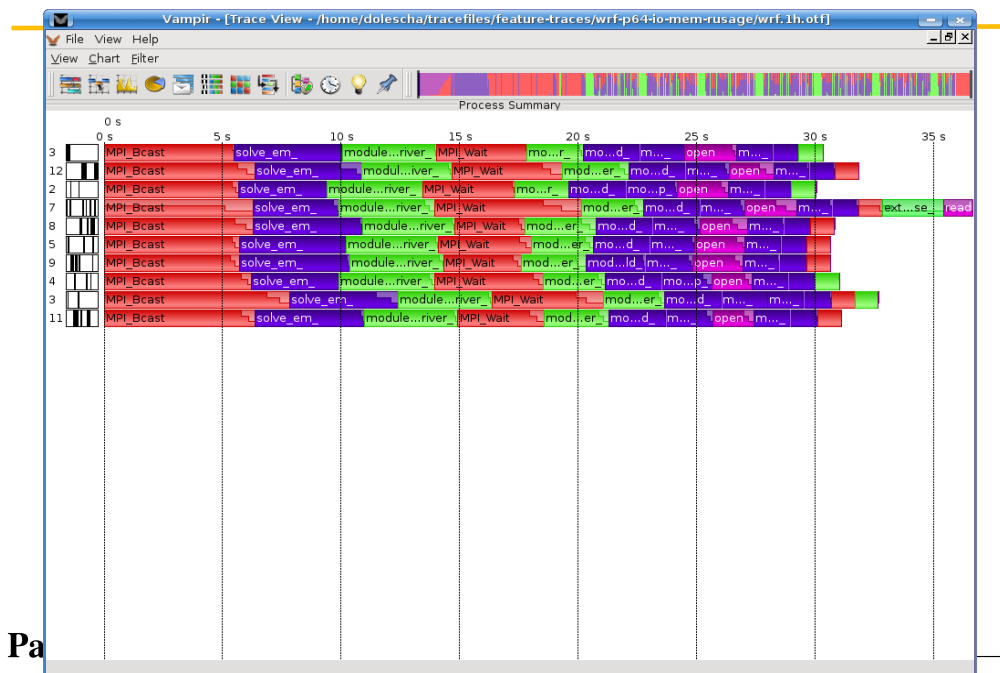
Pa

Message Summary



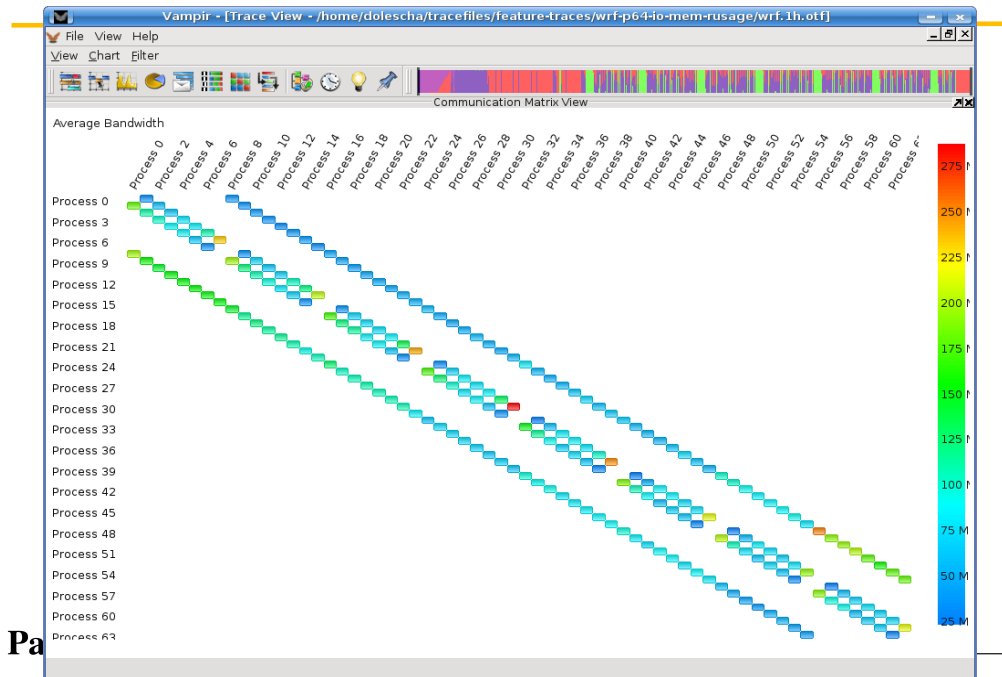
Pa

Process Summary



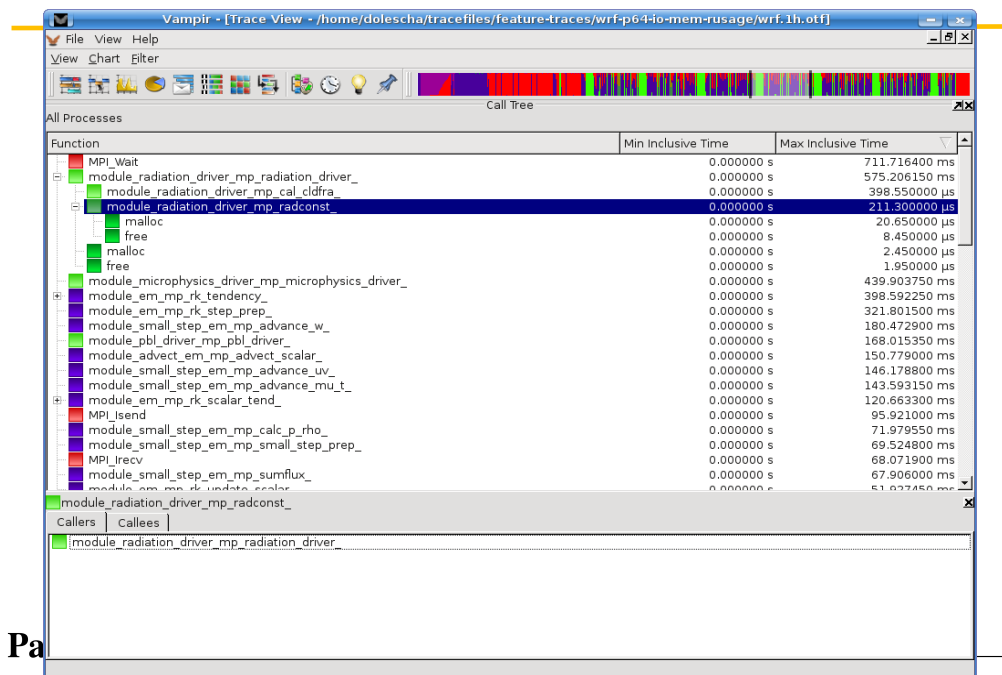
Pa

Communication Matrix



Pa

Call Tree



Pa

Profiling and Tracing

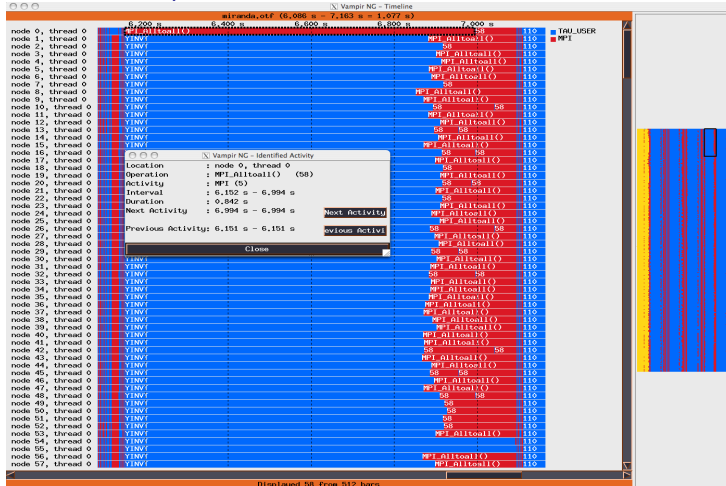
- Tracing Advantages
 - preserve temporal and spatial relationships
 - allow reconstruction of dynamic behavior on any required abstraction level
 - profiles can be calculated from trace
- Tracing Disadvantages
 - traces can become very large
 - may cause perturbation
 - instrumentation and tracing is complicated (event buffering, clock synchronization, ...)

Common Event Types

- Enter/leave of function/routine/region
 - time, process/thread, function ID
- Send/receive of P2P message (MPI)
 - time, sender, receiver, length, tag, comm.
- Collective communication (MPI)
 - time, process, root, communicator, # bytes
- Hardware performance counter values
 - time, process, counter ID, value

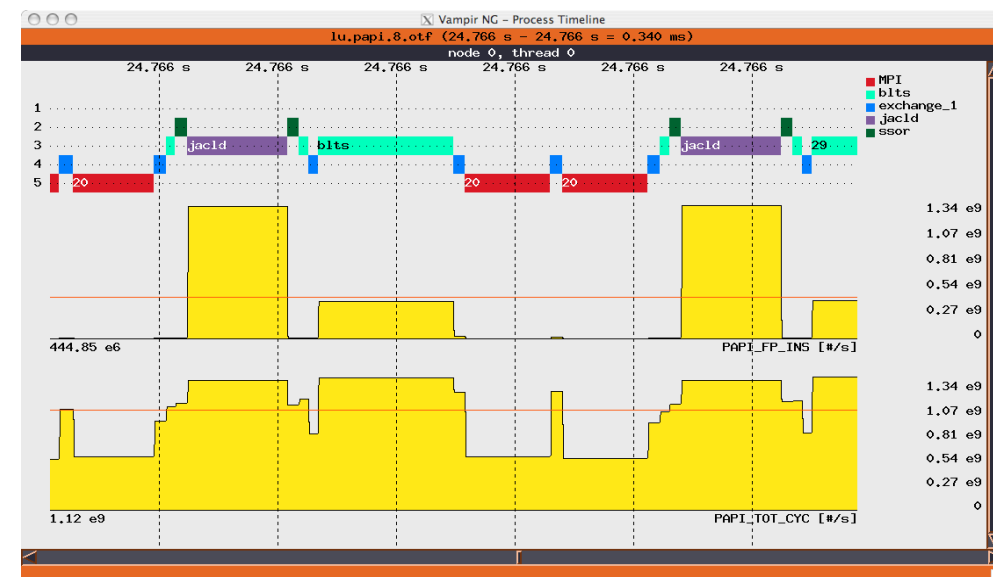
Using TAU with Vampir

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



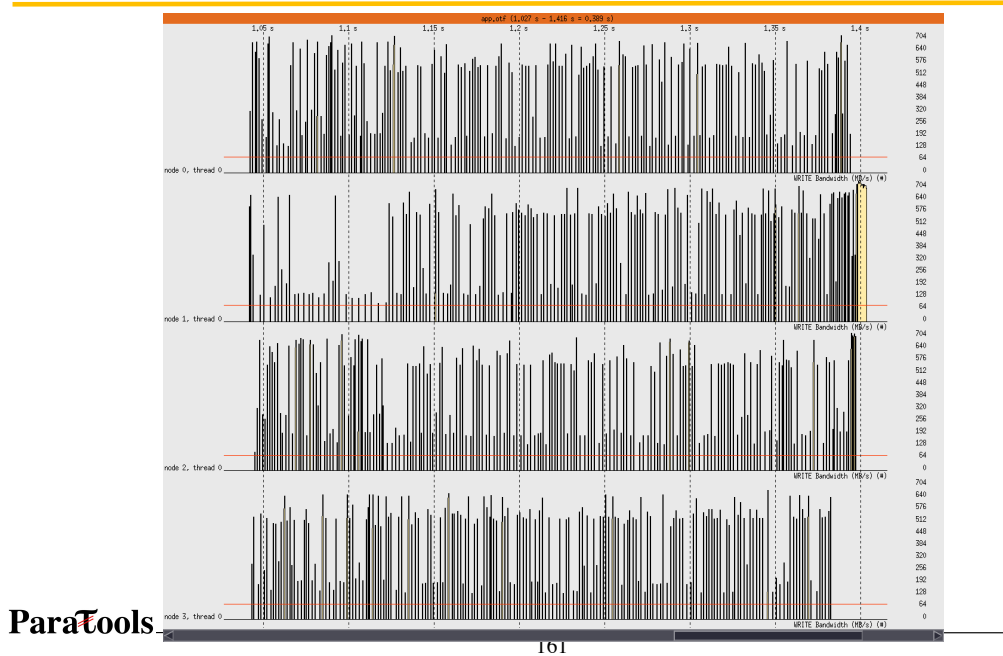
ParaTools

Vampir Process Timeline with PAPI Counters



ParaTools

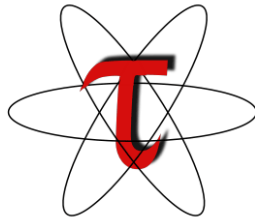
Vampir Counter Timeline Showing I/O BW



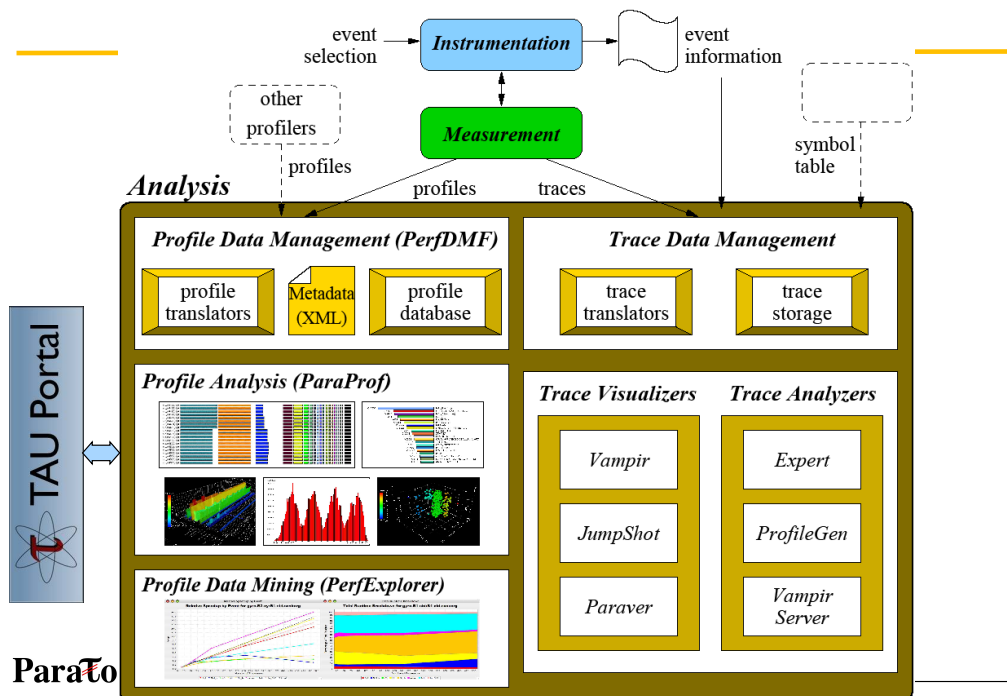
Generate a Trace File

```
% export TAU_MAKEFILE=$TAU/Makefile.tau-mpi-pdt-pgi
% export PATH=$TAU_ROOT/bin:$PATH
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)
%
% export TAU_TRACE=1
% aprun -n 4 ./a.out
% tau_multimerge
JUMPSHOT:
% tau2slog2 tau.trc tau.edf -o app.slog2
% jumpshot app.slog2
OR
VAMPIR:
% tau2otf tau.trc tau.edf app.otf -n 4 -z
(4 streams, compressed output trace)
% vampir app.otf
OR
PARAVER:
% tau_convert -paraver tau.trc tau.edf app.prv
% paraver app.prv
```

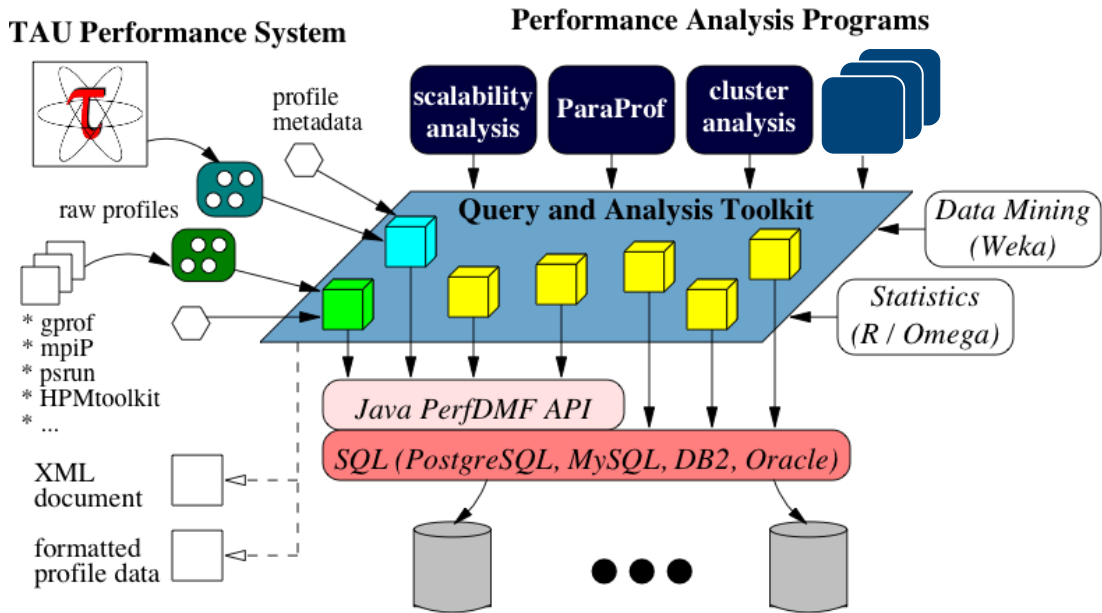
Analyzing performance data with ParaProf, PerfExplorer



TAU Performance System Architecture

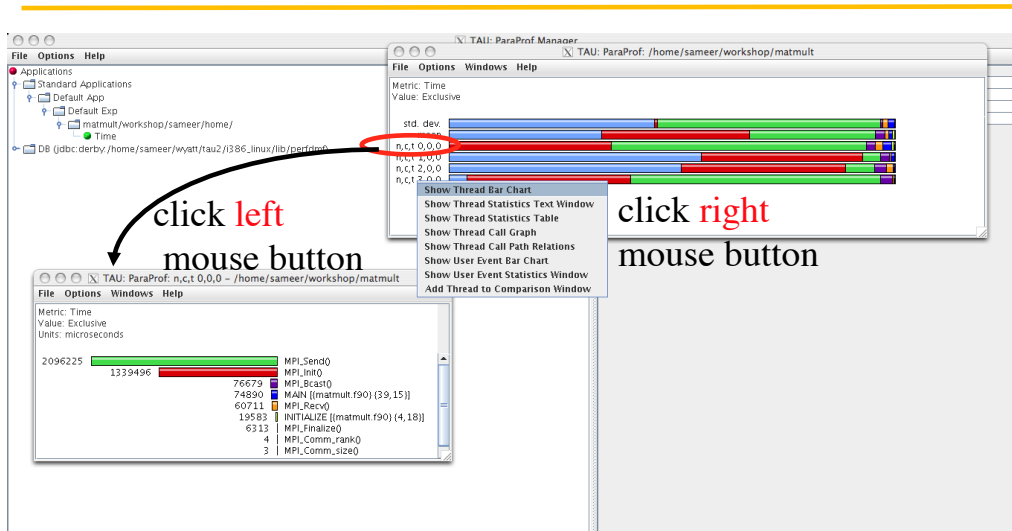


PerfDMF: Performance Data Mgmt. Framework



ParaTools

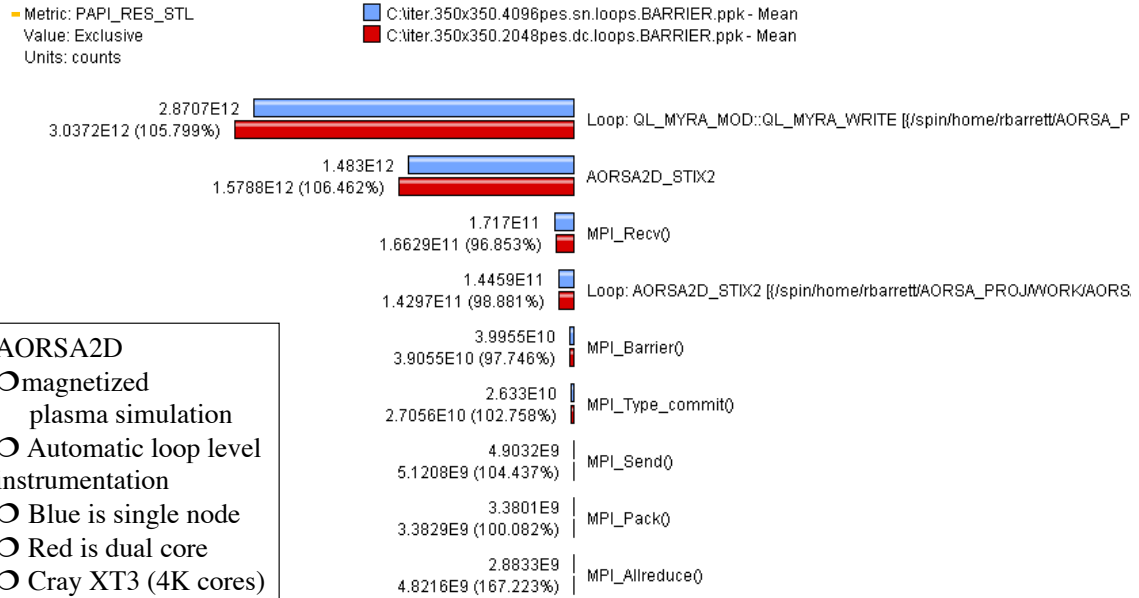
ParaProf Main Window



% paraprof matmult.ppk

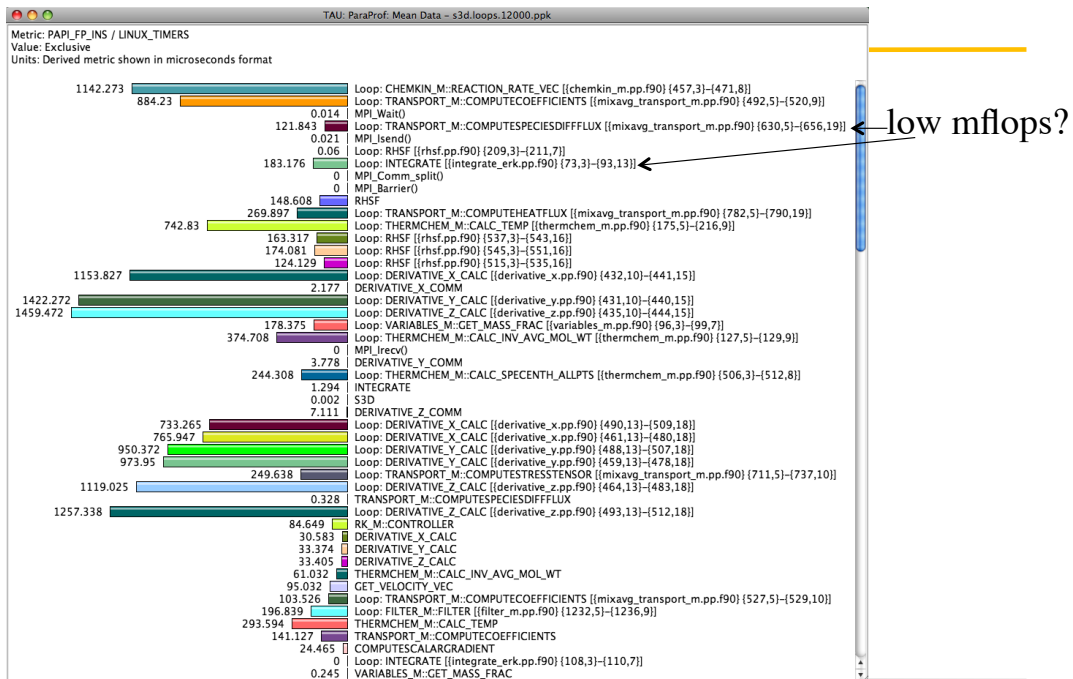
ParaTools

Comparing Effects of Multi-Core Processors

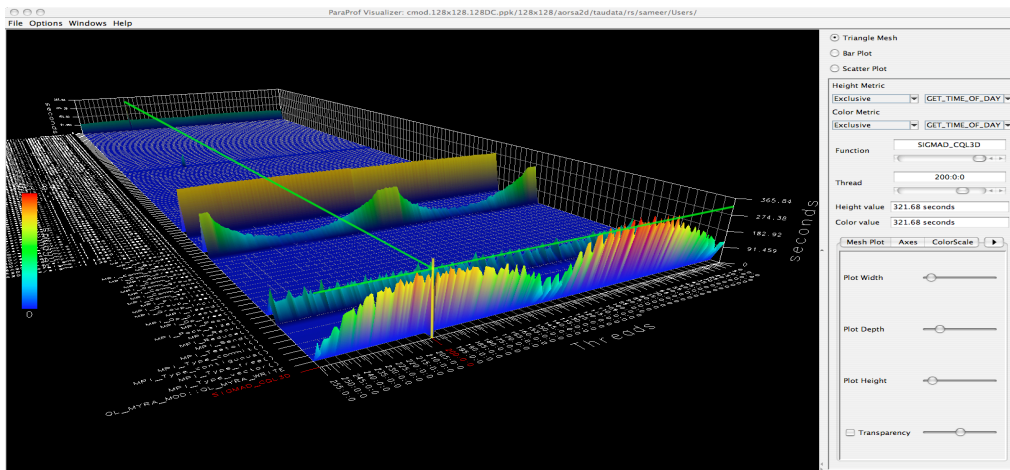


ParaTools

ParaProf: Mflops Sorted by Exclusive Time

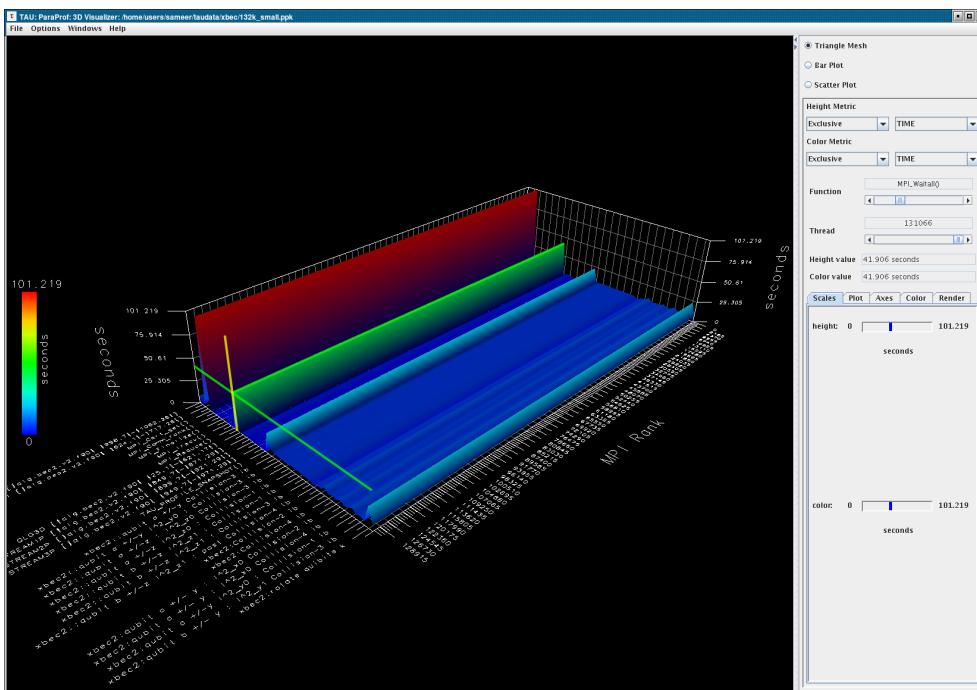


Parallel Profile Visualization: ParaProf

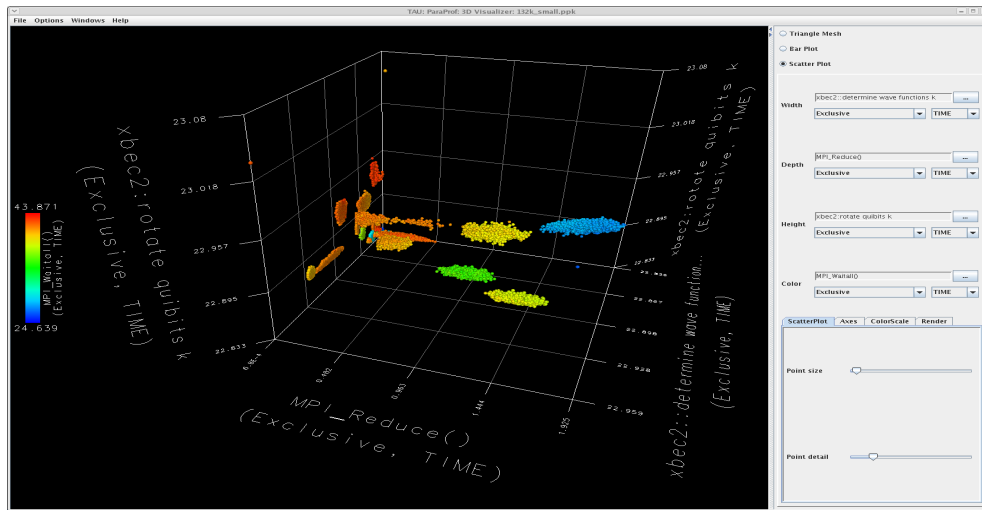


ParaTools

Scalable Visualization: ParaProf (128k cores)

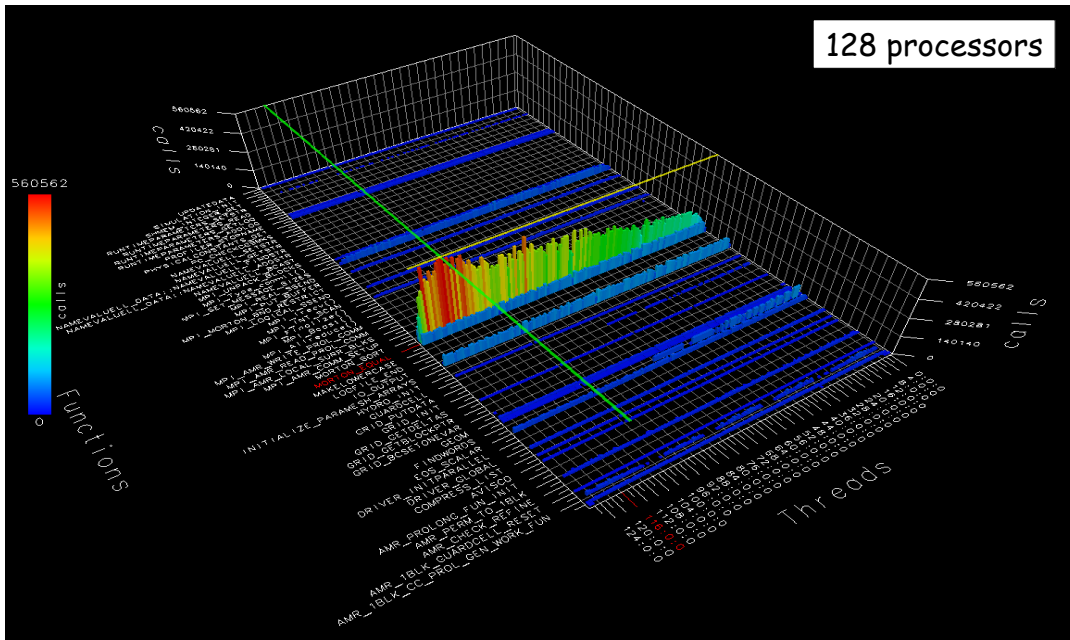


Scatter Plot: ParaProf (128k cores)

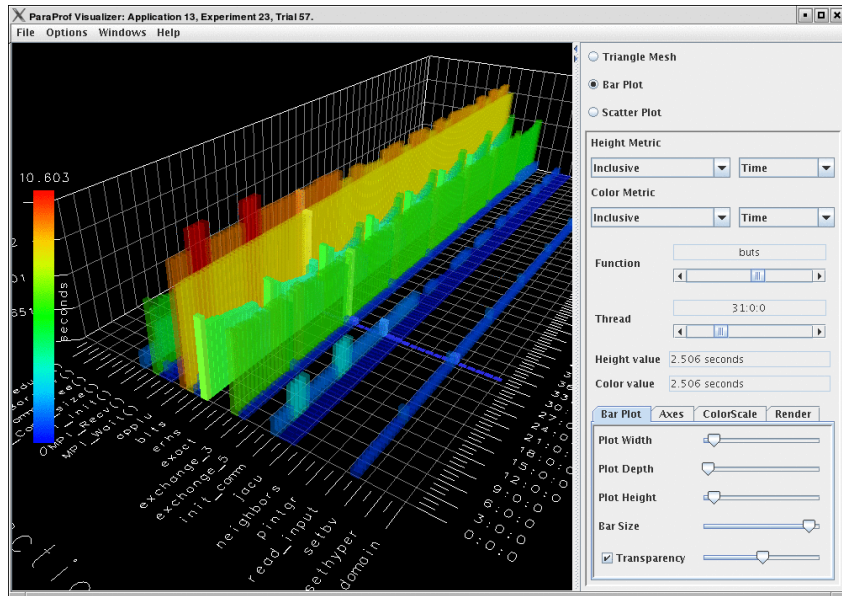


ParaTools

ParaProf – 3D Full Profile Bar Plot (Flash)



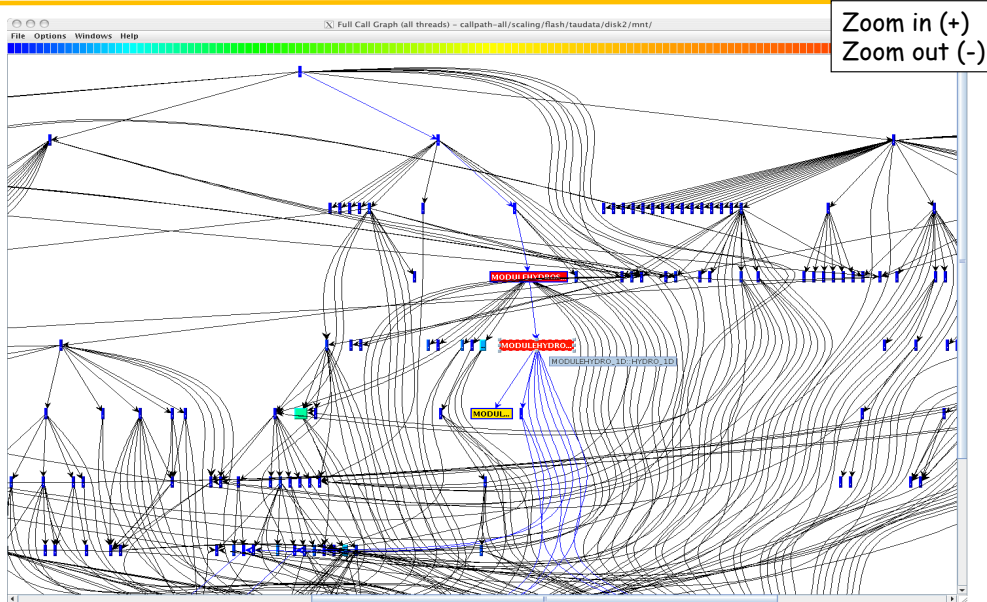
ParaProf Bar Plot (Zoom in/out +/-)



ParaTools

173

ParaProf – Callgraph Zoomed (Flash)



ParaTools

174

ParaProf - Thread Statistics Table (GSI)

Name	Inclusive Time	Exclusive Time	Calls	Child Calls
GSI	5,223.564	0.098	1	30
SPECMOD::INIT_SPEC_VARS	0.26	0.26	1	0
MPL_Init0	0.056	0.054	1	1
CSISUB	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
GLBIOI	5,212.171	0.024	1	12
MPL_Finalize0	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
MPL_Allreduce0	3,725.802	3,725.802	3	0
READ_BUFRTOV0	44.369	0.254	1	871,535
SATTHIN::MAKECVALS	0	0	1	1
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

ParaTools

ParaProf - Callpath Thread Relations Window

Exclusive	Inclusive	Calls/Tot.Calls	Name[14]
0.023	0.023	3/430	COMPUTE_DERIVED[55]
2.02	2.02	104/430	DPRODXMOD::DPRODX[66]
0.33	0.33	104/430	INFALLMOD::INFALL[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	SETUPHSHALL[1900]
20.069	20.069	208/430	STPCALCMOD::STPCALC[1942]
3964.18	3964.18	430	MPL_Allreduce0 [1762]
2.6E-4	30.582	1/15	GLBIOI[93]
0.007	0.036	1/15	GRI[107]
2.7E-4	10.711	1/15	GRIEUB[1690]
31.273	1347.703	3/15	M_FVANAGRID::ALLGETLIST_ [1773]
0.412	0.412	1/15	PREWQ[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	1575.995	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_CDIFC_CORFS[34]
4.9E-5	4.9E-5	1/1	COMPACT_DIFFS::DESTROY_CDIFC_CORFS[35]
0.015	0.042	1/1	COMPACT_DIFFS::IHSFW[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]
3.2E-4	3.2E-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	MPL_Allreduce0 [1762]
0.017	0.017	88/116	MPI_Bcast [1764]
0.004	0.004	297/409	MPI_Comm_rank [1765]

ParaTools



ParaProf – Manager Window

The screenshot shows the ParaProf Manager interface. On the left is a tree view of applications, with a callout box labeled "performance database" pointing to it. On the right is a table of metadata fields and values. A callout box labeled "metadata" points to this table. A "Load Trial" dialog box is also visible, showing a list of trial types.

Field	Value
Name	64 CPU
Application ID	4
Experiment ID	25
Trial ID	85
DATE	
COLLECTORID	
NODE_COUNT	64
CONTEXTS_PER_NODE	1
THREADS_PER_CONTEXT	1

ParaTools

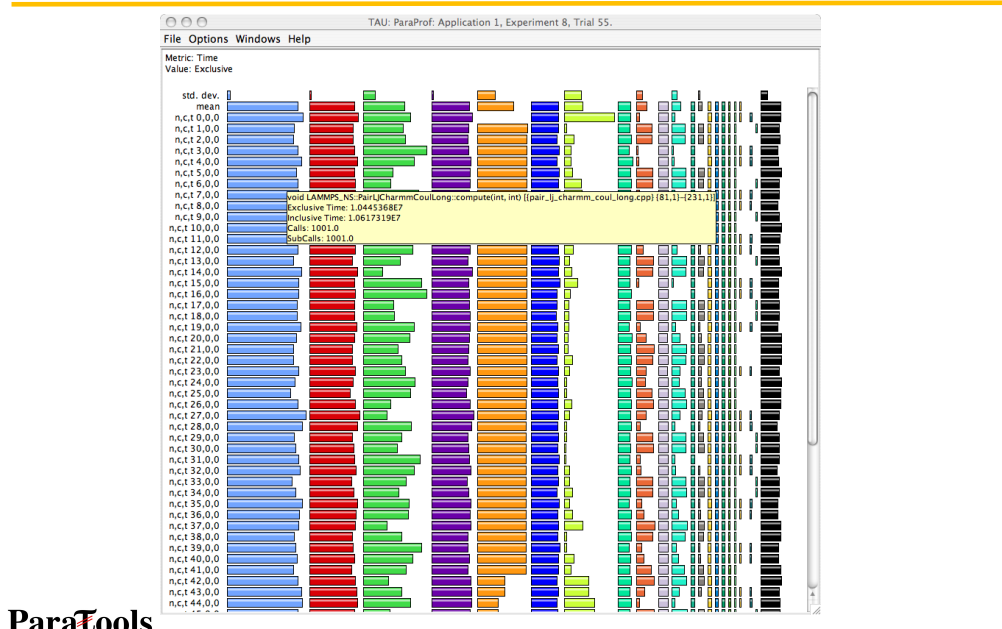
Performance Database: Storage of MetaData

The screenshot shows the ParaProf Manager interface. On the left is a tree view of applications, with a callout box labeled "performance database" pointing to it. On the right is a table of metadata fields and values. A "Load Trial" dialog box is also visible, showing a list of trial types.

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, 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

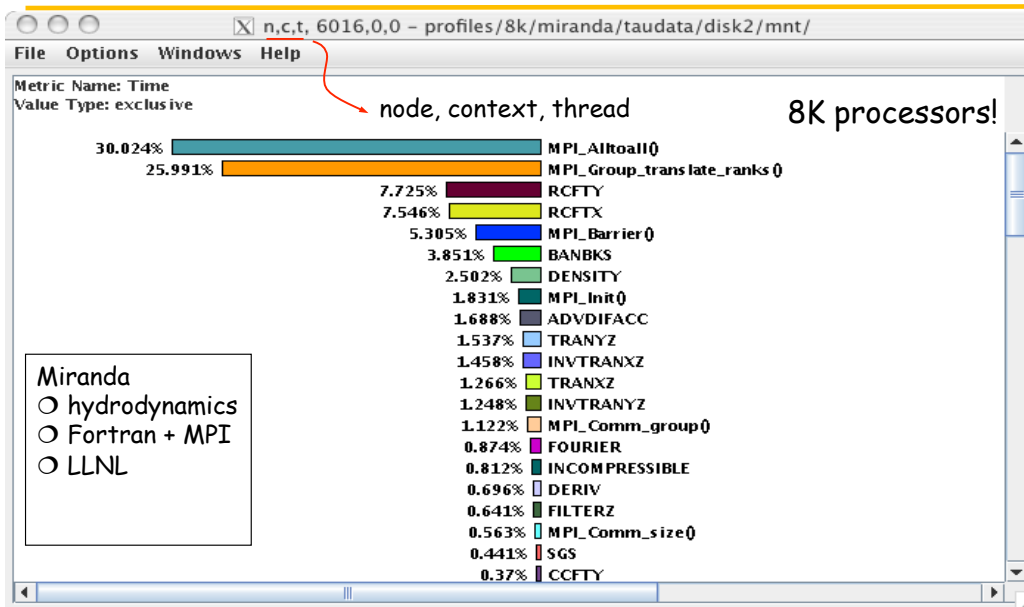
ParaTools

ParaProf Main Window (Lammps)



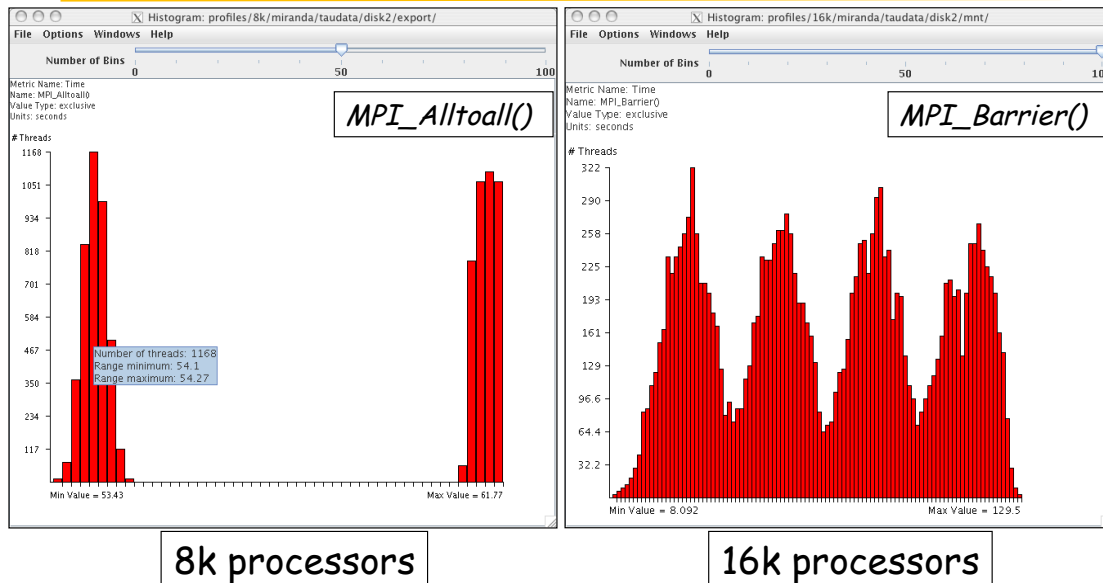
ParaTools

ParaProf – Flat Profile (Miranda)



ParaTools

ParaProf – Histogram View (Miranda)



ParaTools

181

Performance Problem Solving Goals

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

ParaTools

182

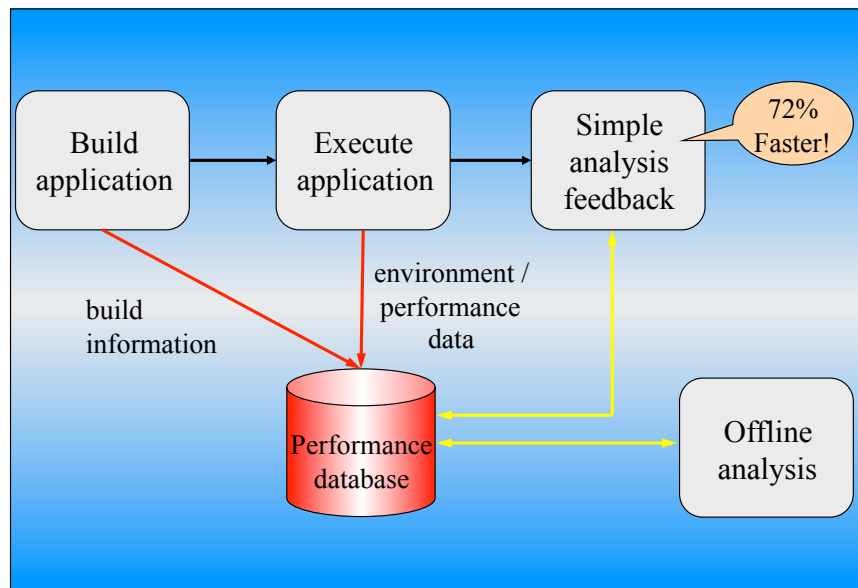
Performance Analysis Questions

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

ParaTools

183

Automatic Performance Analysis



ParaTools

184

Performance Data Management

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

Metadata Collection

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

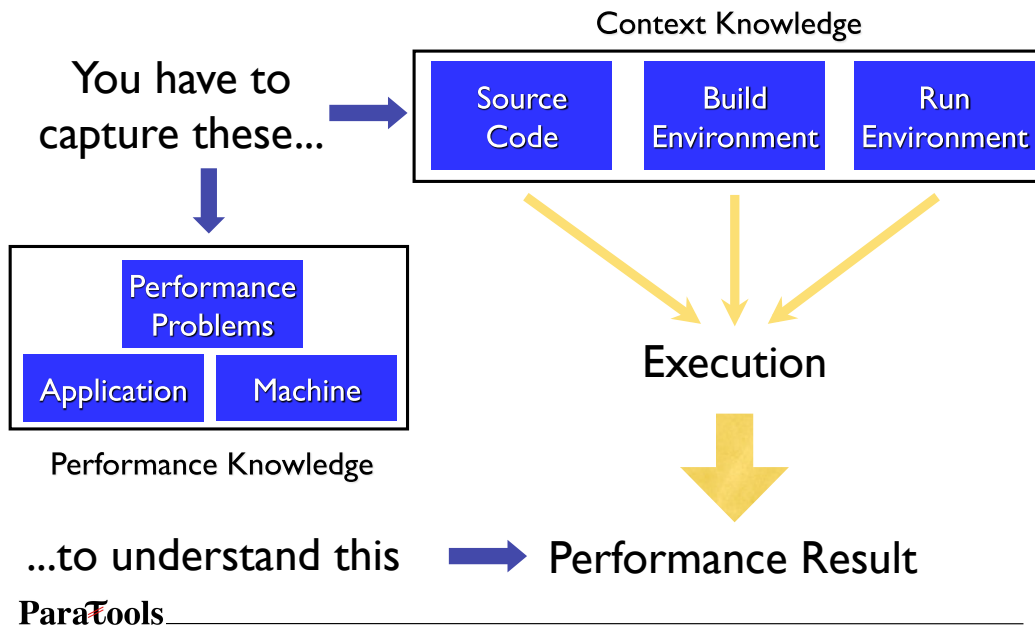
Performance Data Mining

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

How to explain performance?

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

Metadata and Knowledge Role



189

Performance Optimization Process

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

Using Performance Database (PerfDMF)

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

PerfDMF and the TAU Portal

- **Development of the TAU portal**
 - Common repository for collaborative data sharing
 - Profile uploading, downloading, user management
 - Paraprof, PerfExplorer can be launched from the portal using Java Web Start (no TAU installation required)
- **Portal URL**
 - <http://tau.nic.uoregon.edu>

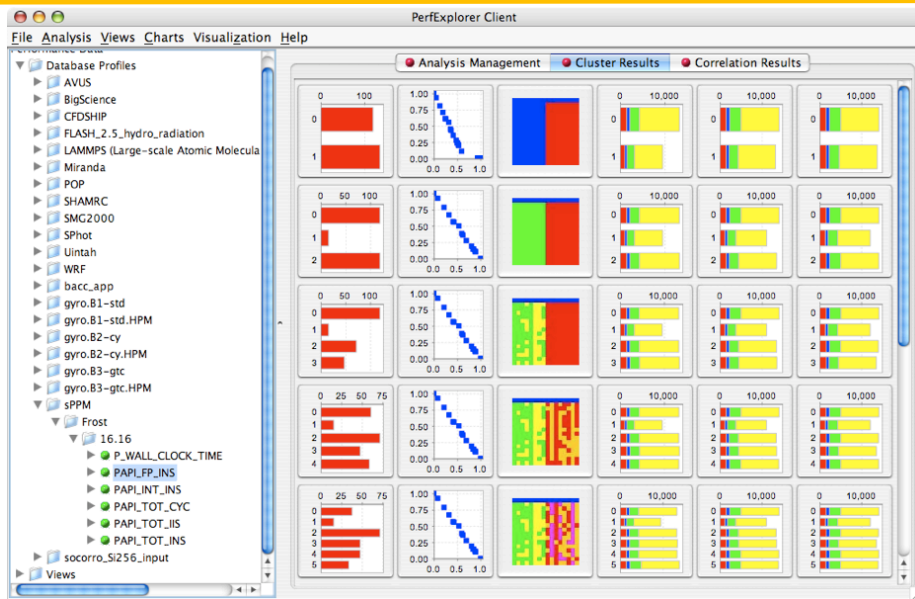
Performance Data Mining (PerfExplorer)

- 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)

PerfExplorer - Cluster Analysis

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

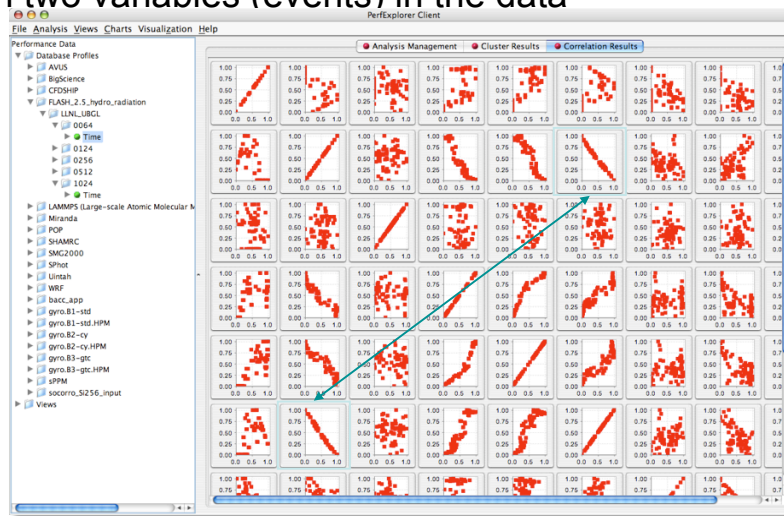
PerfExplorer - Cluster Analysis (sPPM)



ParaTools

PerfExplorer - Correlation Analysis (Flash)

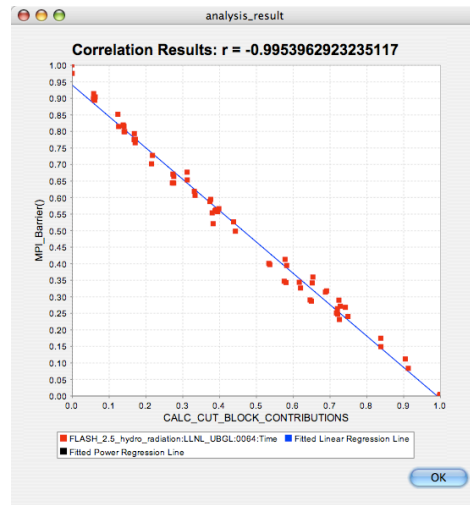
- Describes strength and direction of a linear relationship between two variables (events) in the data



ParaTools

PerfExplorer - Correlation Analysis (Flash)

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



ParaTools

197

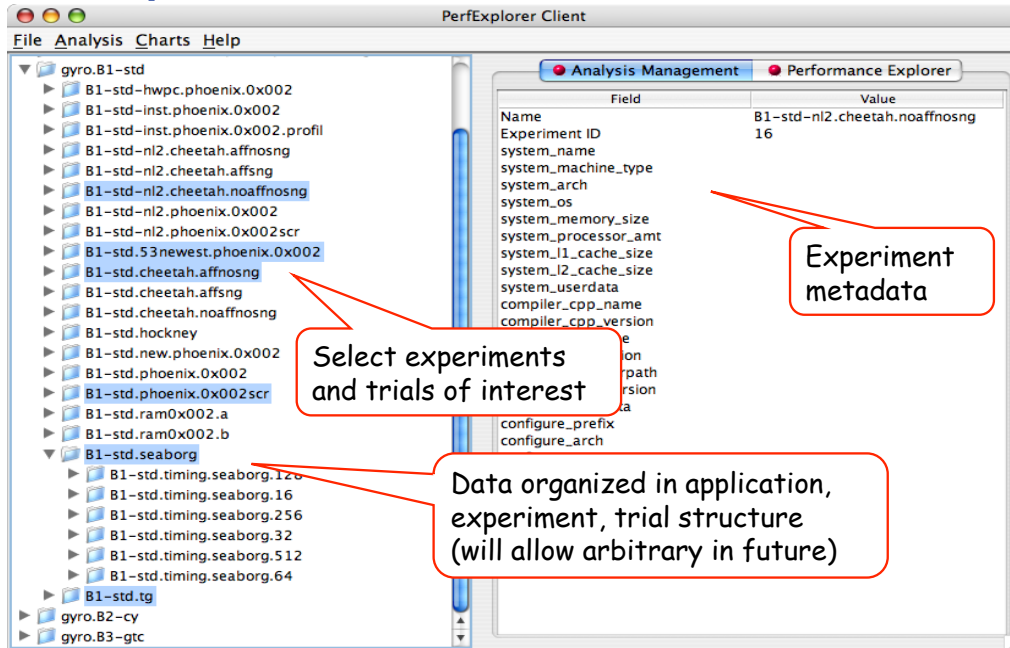
PerfExplorer - Comparative Analysis

- 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

ParaTools

198

PerfExplorer - Interface

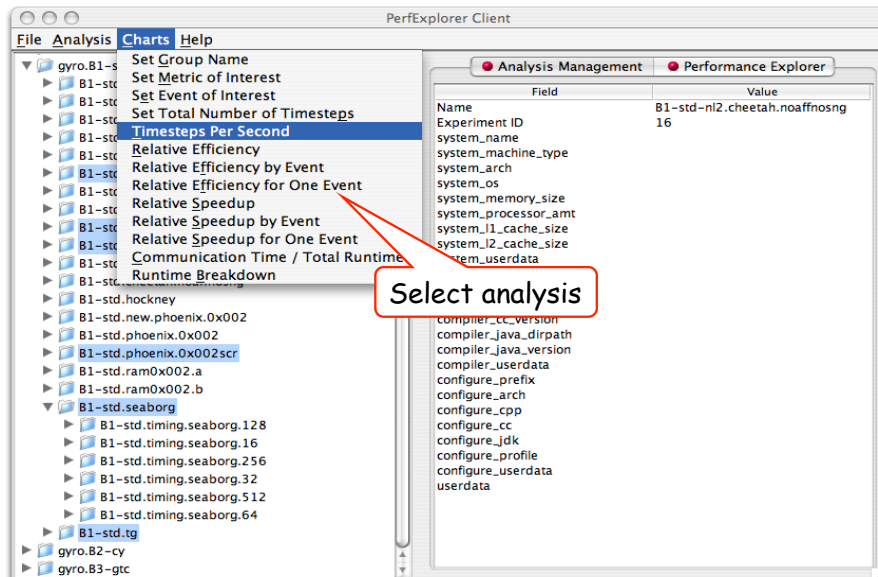


Select experiments and trials of interest

Experiment metadata

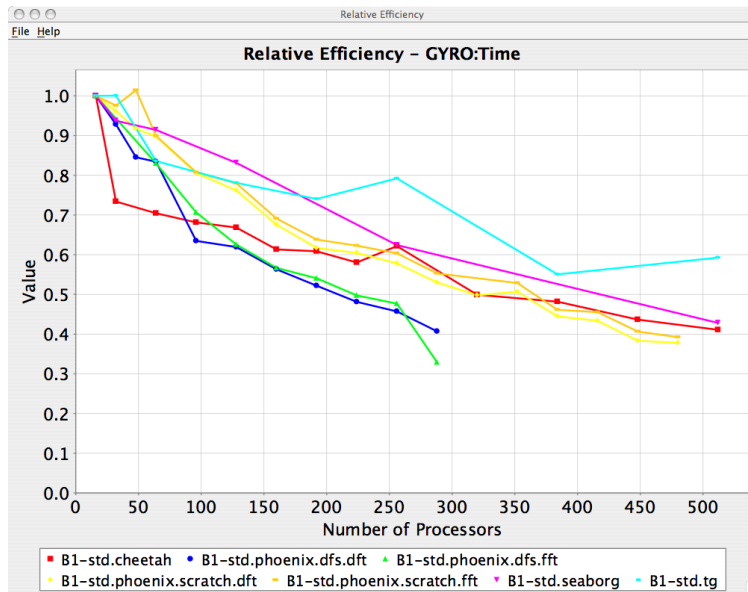
Data organized in application, experiment, trial structure (will allow arbitrary in future)

PerfExplorer - Interface



Select analysis

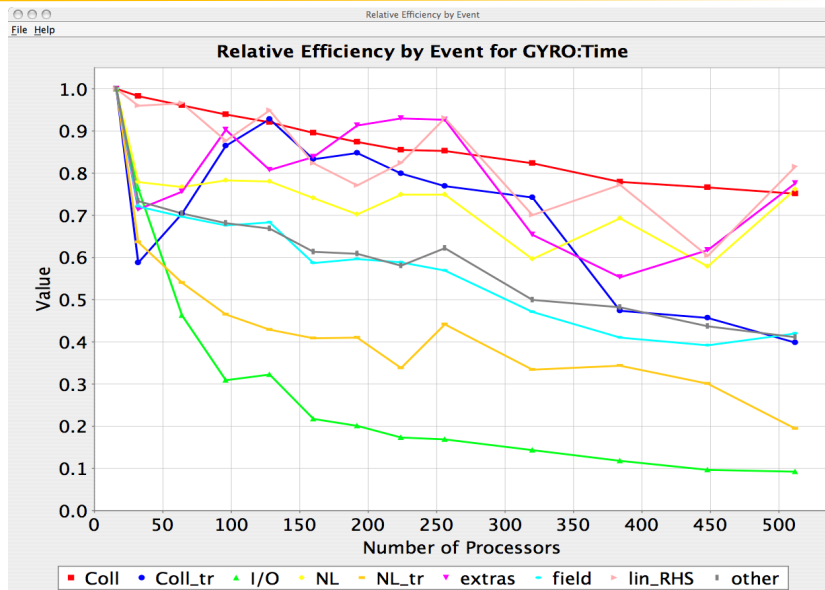
PerfExplorer - Relative Efficiency Plots



ParaTools

201

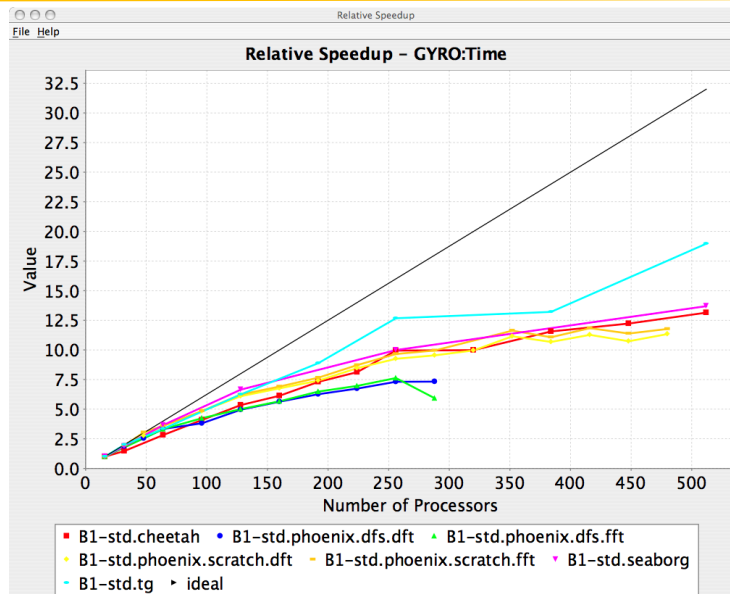
PerfExplorer - Relative Efficiency by Routine



ParaTools

202

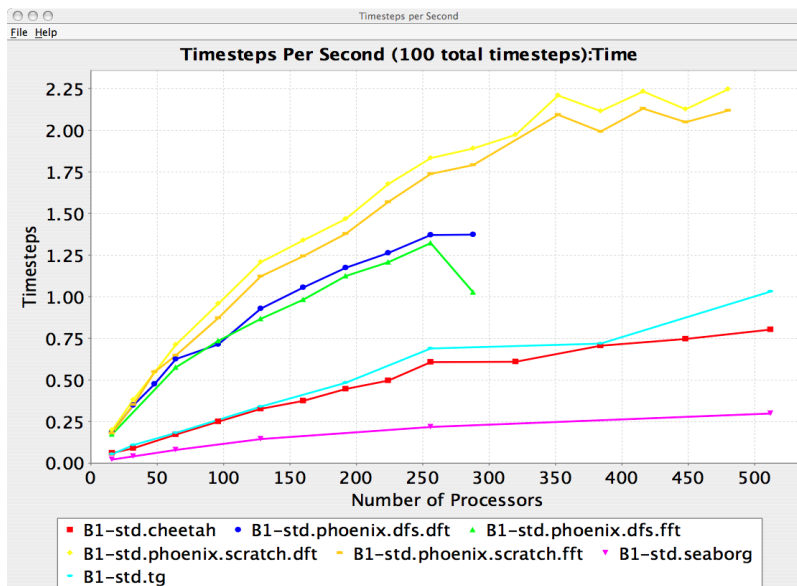
PerfExplorer - Relative Speedup



ParaTools

203

PerfExplorer - Timesteps Per Second

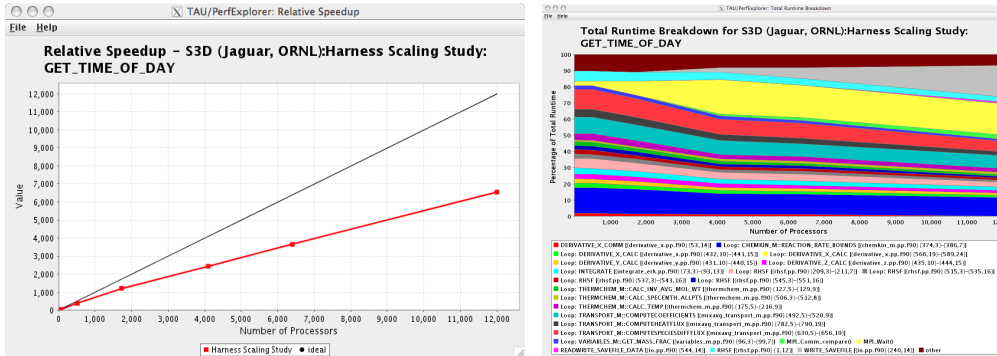


ParaTools

204

Usage Scenarios: Evaluate Scalability

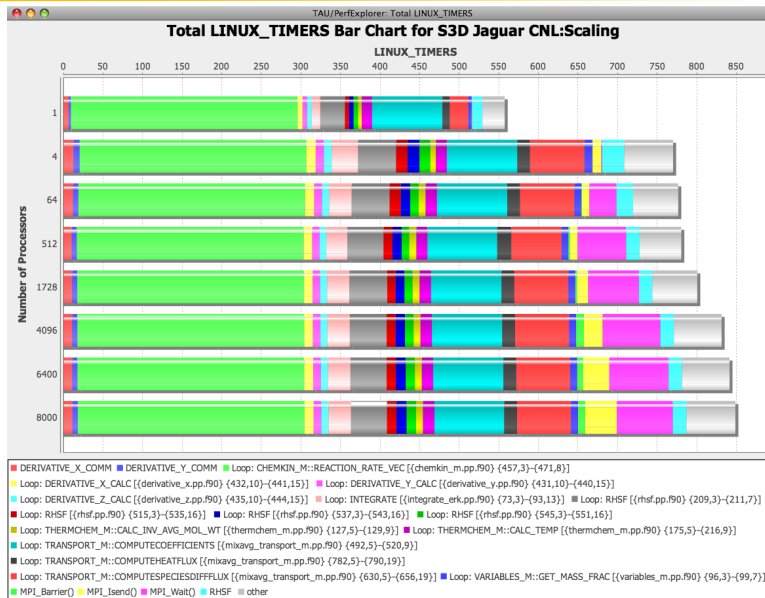
- Goal: How does my application scale? What bottlenecks occur at what core counts?
- Load profiles in PerfDMF database and examine with PerfExplorer



ParaTools

205

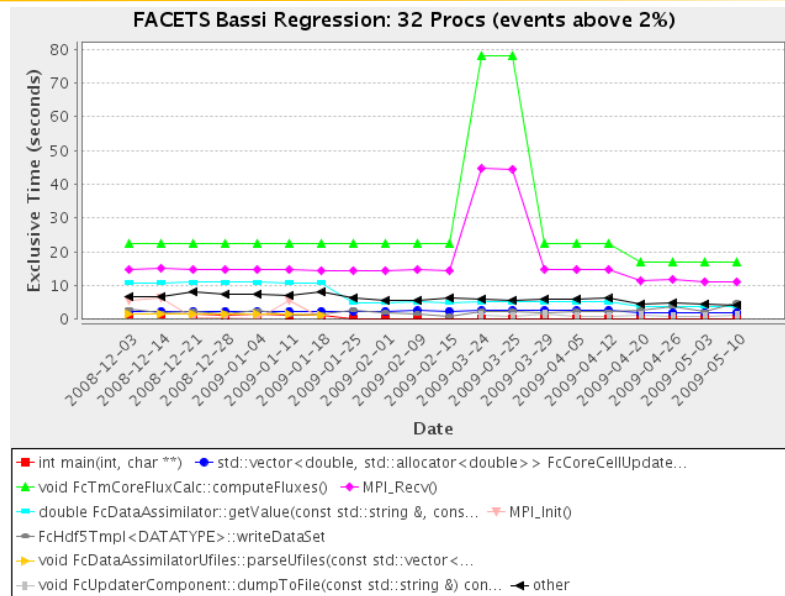
Usage Scenarios: Evaluate Scalability



ParaTools

206

Performance Regression Testing



ParaTools

207

Evaluate Scalability using PerfExplorer Charts

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

ParaTools

208



Overview

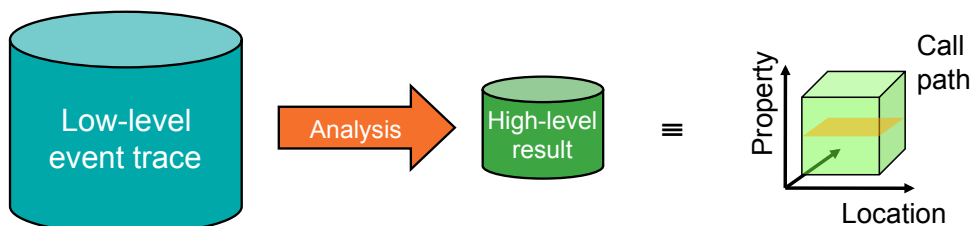
- Introduction
 - Motivation for automatic trace analysis
- Scalasca components and usage
 - instrumentation
 - measurement collection & automated analysis
 - analysis report exploration
- Demonstration
- Summary

Motivation

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

Automatic Trace Analysis

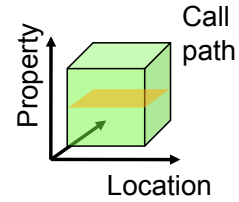
- Idea:
 - Automatic search for *patterns* of inefficient behaviour
 - Classification of behaviour
 - Quantification of significance



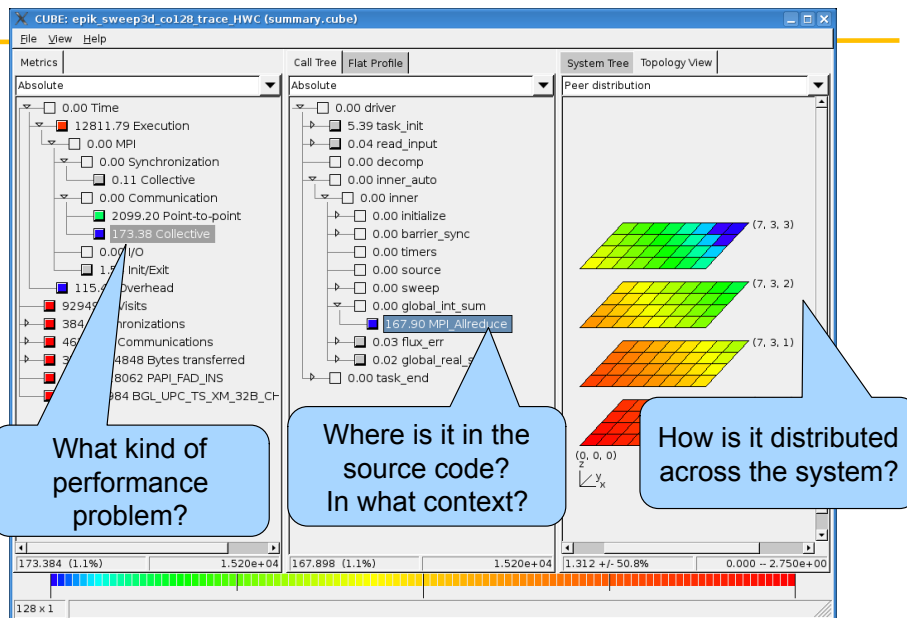
- Quicker than manual analysis
- Guaranteed to cover the entire event trace

CUBE Result Browser

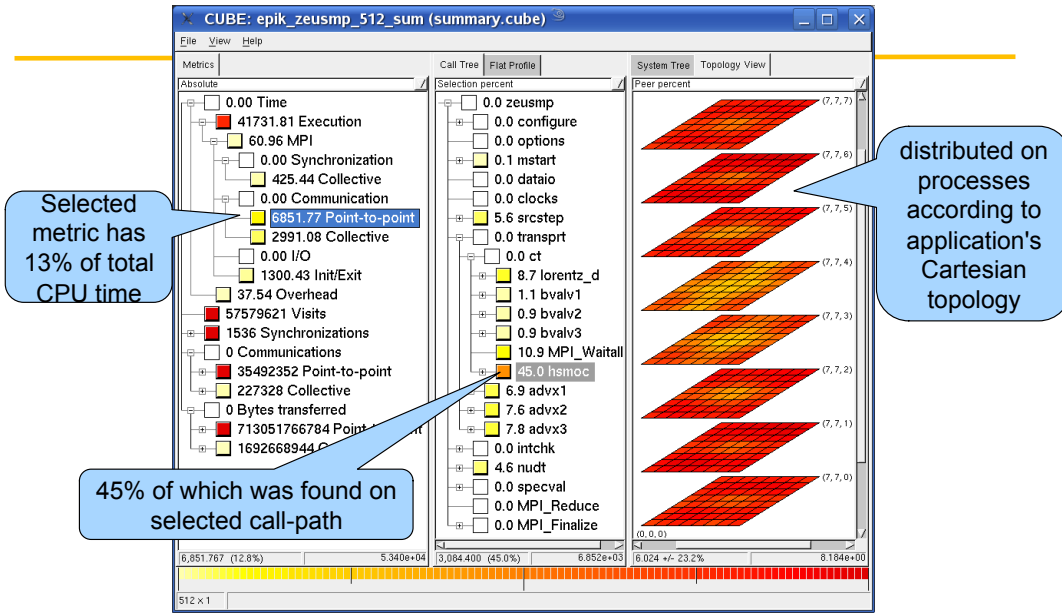
- Representation of results (severity matrix) along three hierarchical axes
 - Performance property
 - Call tree path
 - System location
- Three coupled tree browsers
- Each node displays severity
 - As colour: for easy identification of hotspots
 - As value: for precise comparison
 - Inclusive value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



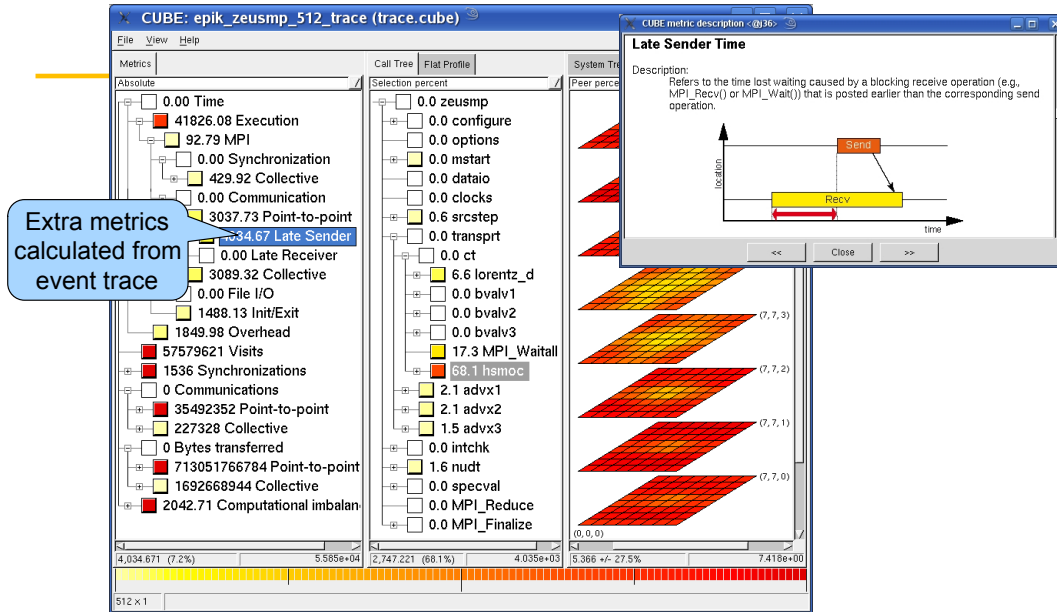
Basic Analysis Presentation



Summary Profile Analysis



Trace Pattern Analysis



Analysis Methodology

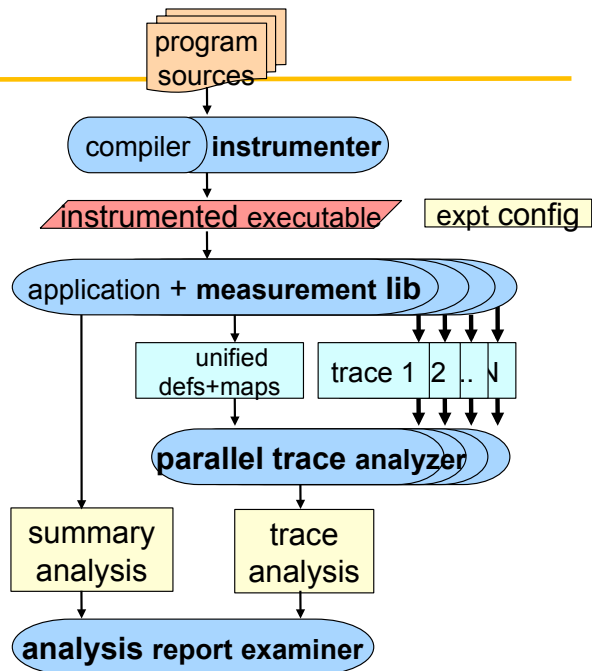
- Instrumentation of application executable and libraries
 - automatic MPI, OpenMP and function instrumentation
 - complementary manual region and phase instrumentation
- Execution of instrumented executable under control of configurable measurement collection & analysis nexus
 - commence from scalable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

ParaTools

217

Scalasca Components

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



ParaTools

218

Scalasca unified command: scalasca

- Run without action argument for basic usage info

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

Scalasca instrumenter: skin

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

Scalasca collector & analyzer: scan

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

Scalasca analysis report explorer: square

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

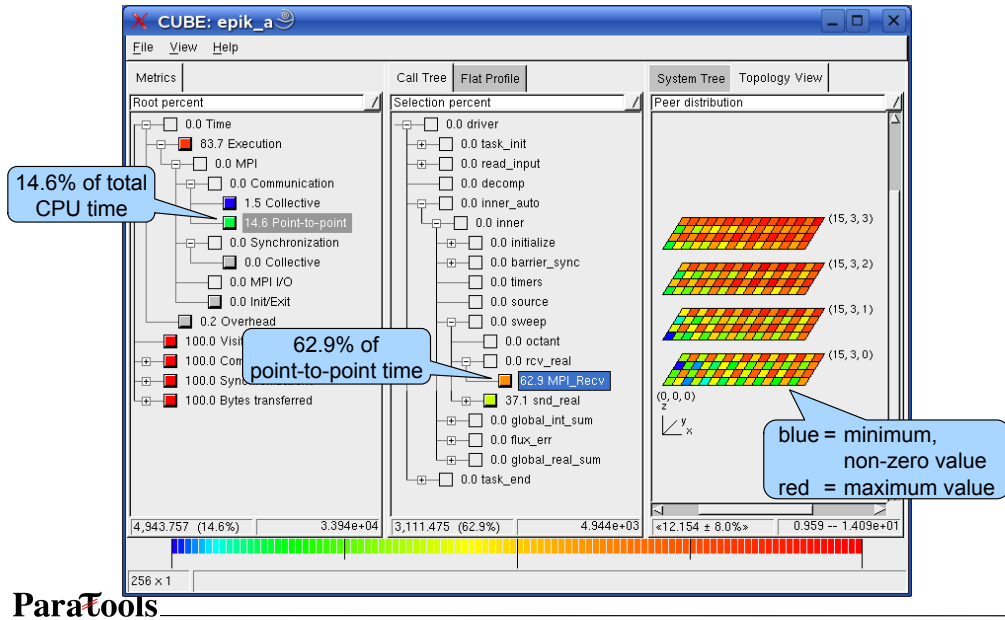
EPIK experiment archive

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

Display modes (system tree/topology only)

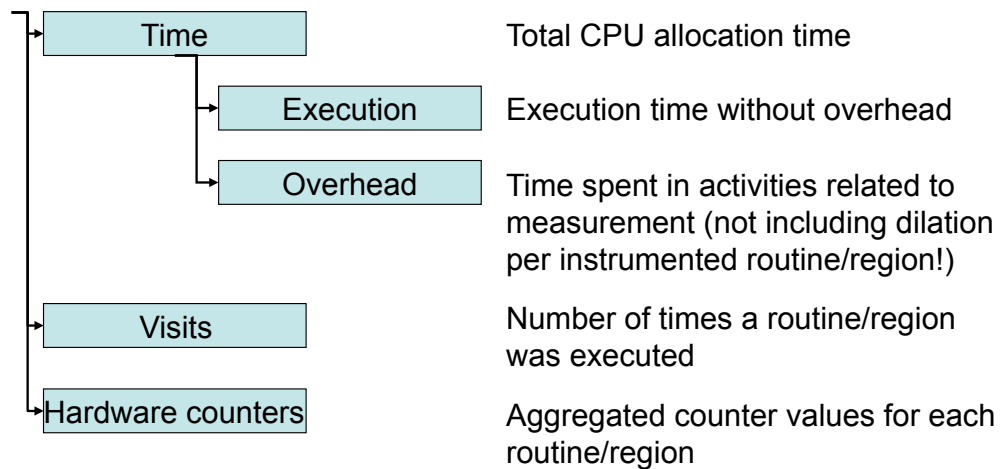
- Peer percent
 - Percentage relative to maximum of peer values (all values of the current leaf level)
- Peer distribution
 - Percentage relative to maximum and non-zero minimum of peer values

Display mode example

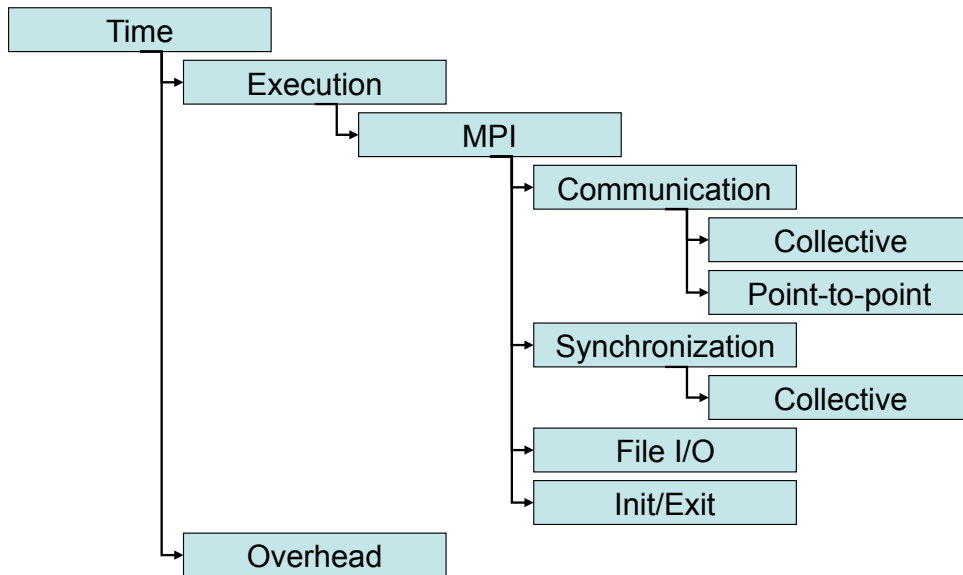


225

Generic metrics



MPI Time hierarchy



ParaTools

227

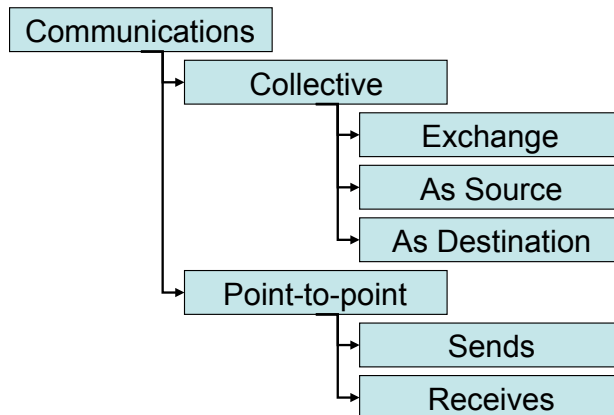
MPI Time hierarchy (cont.)

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

ParaTools

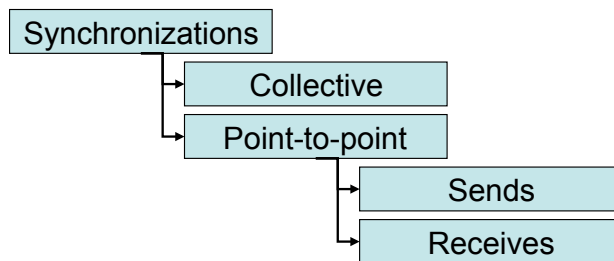
228

MPI Communications hierarchy



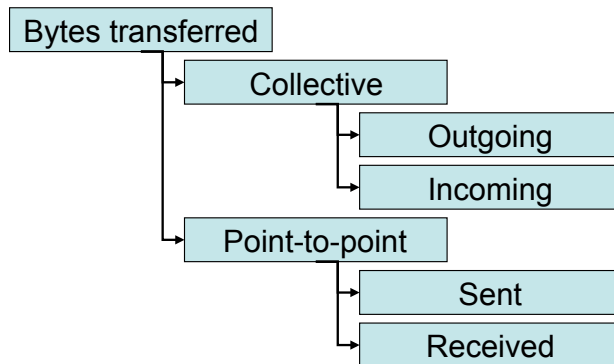
- Provides the number of calls to an MPI communication function of the corresponding class
- Zero-sized message transfers are considered *synchronization!*

MPI Synchronizations hierarchy



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

MPI Bytes transferred hierarchy



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

Combined trace collection & analysis

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

Trace analysis output example

SCOUT

Analyzing experiment archive ./epik_sweep3d_co32_trace

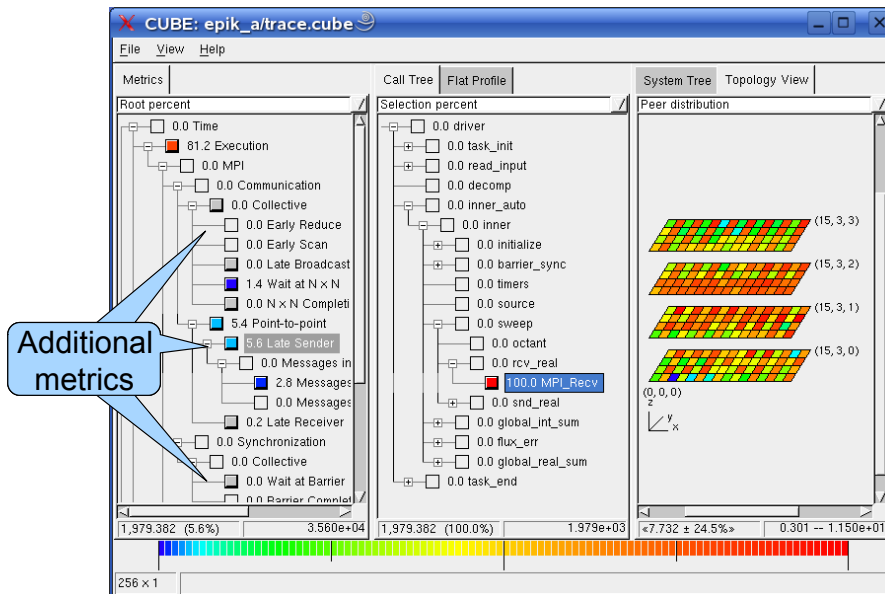
```
Reading definition files ... done
Reading event trace files ... done
Preprocessing ... done
Analyzing event traces ... done
Writing report ... done
```

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

ParaTools

233

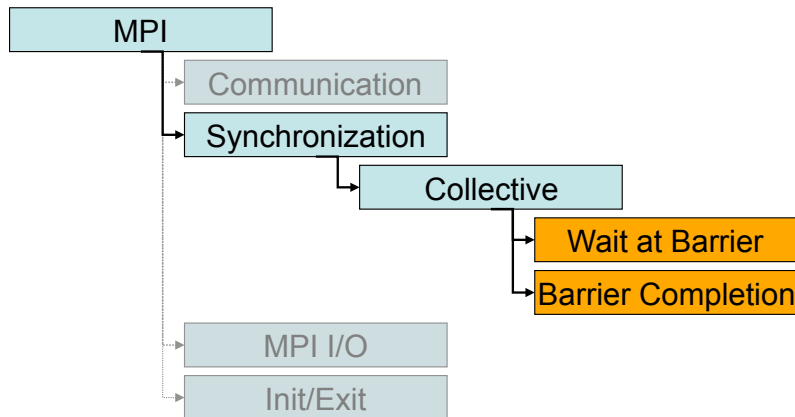
Trace analysis result



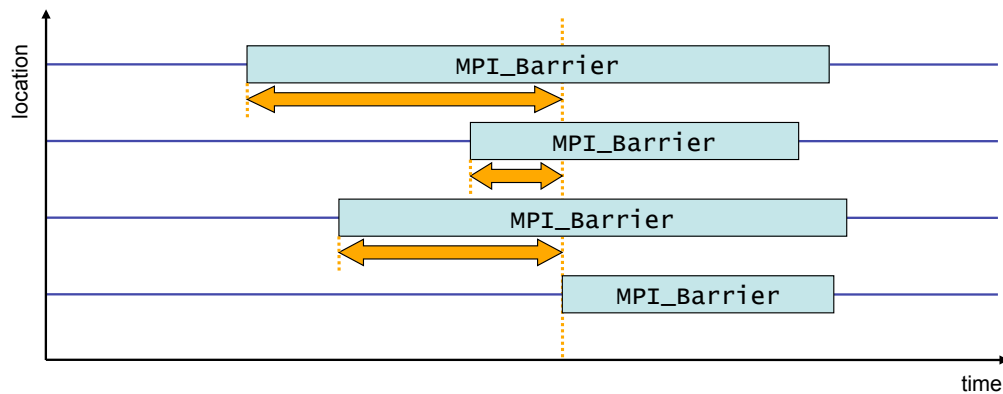
ParaTools

234

MPI collective synchronization time

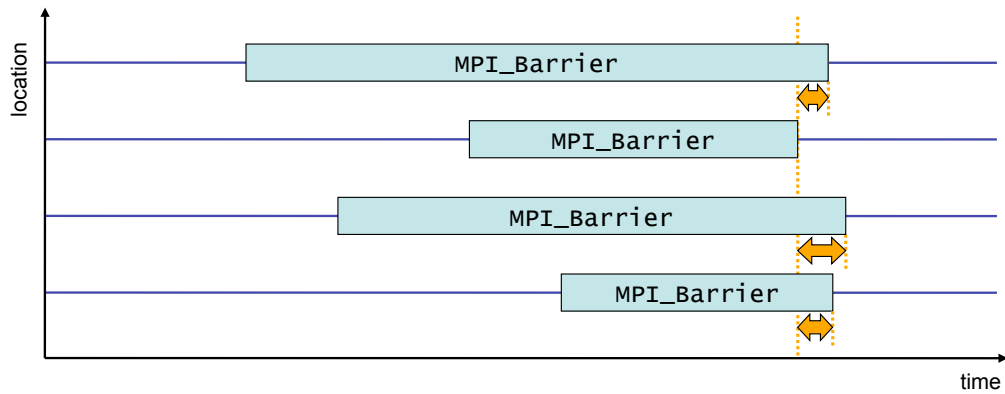


Wait at Barrier = Early Barrier



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

Barrier Completion

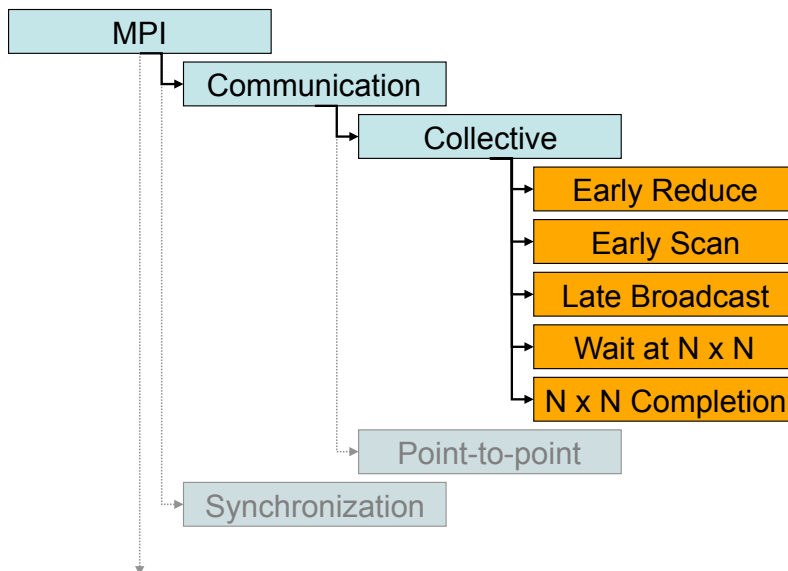


- Time spent in barrier after the first process has left the operation
- Applies to: MPI_Barrier()

ParaTools

237

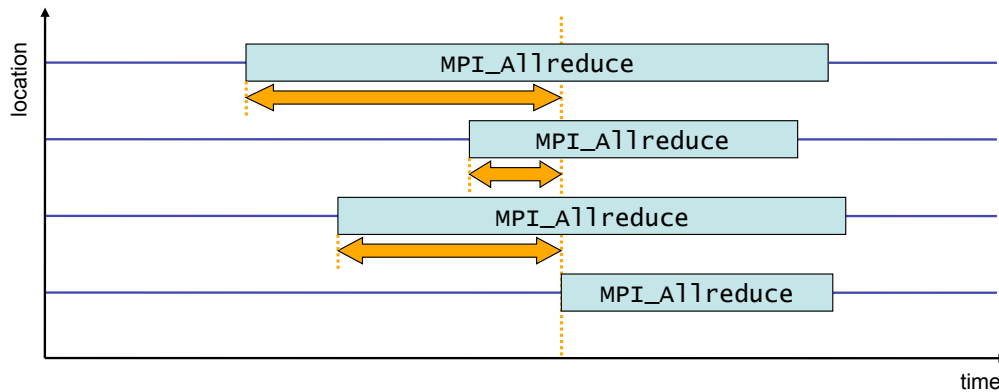
MPI collective communication time



ParaTools

238

Wait at N x N = Early N x N

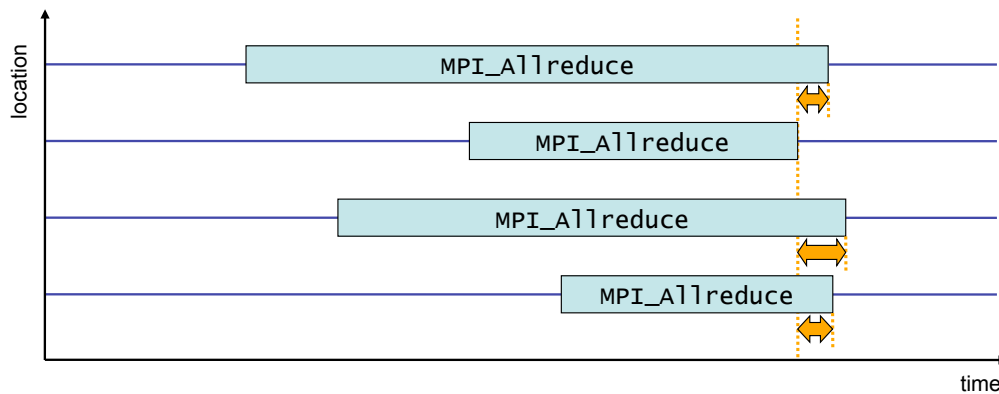


- Time spent waiting in front of a synchronizing collective operation call until the last process reaches the operation
- Applies to: MPI_Allreduce(), MPI_Alltoall(), MPI_Alltoallv(), MPI_Allgather(), MPI_Allgatherv(), MPI_Reduce_scatter()

ParaTools

239

N x N Completion

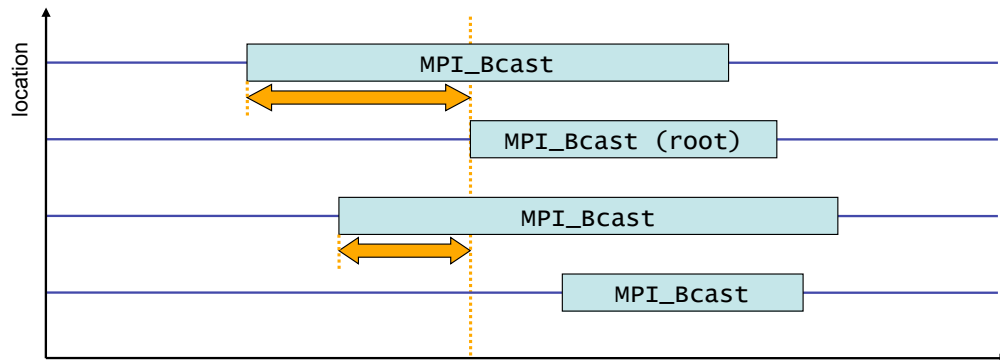


- Time spent in synchronizing collective operations after the first process has left the operation
- Applies to: MPI_Allreduce(), MPI_Alltoall(), MPI_Alltoallv(), MPI_Allgather(), MPI_Allgatherv(), MPI_Reduce_scatter()

ParaTools

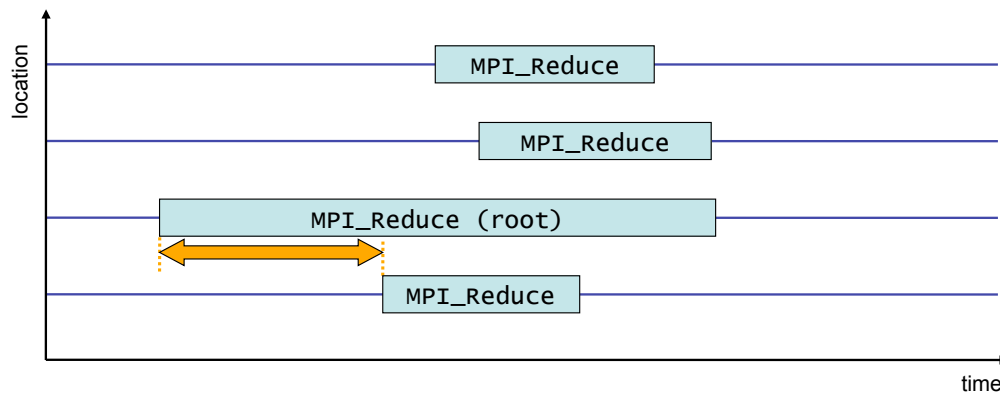
240

Late Broadcast = Early Broadcast



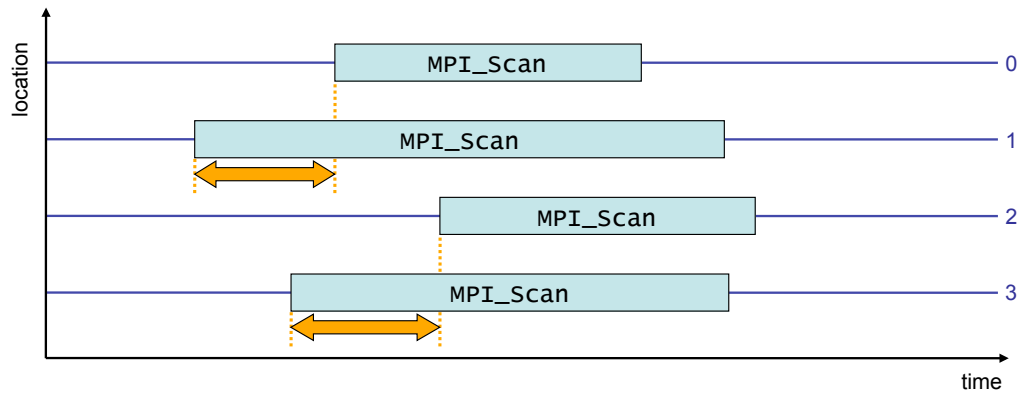
- Waiting times of the destination processes of a collective 1-to-N communication operation which enter the operation earlier than the source process (root)
 - Late Broadcast by source = Early Broadcast by destinations
- Applies to: MPI_Bcast(), MPI_Scatter(), MPI_Scatterv()

Early Reduce



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

Early Scan

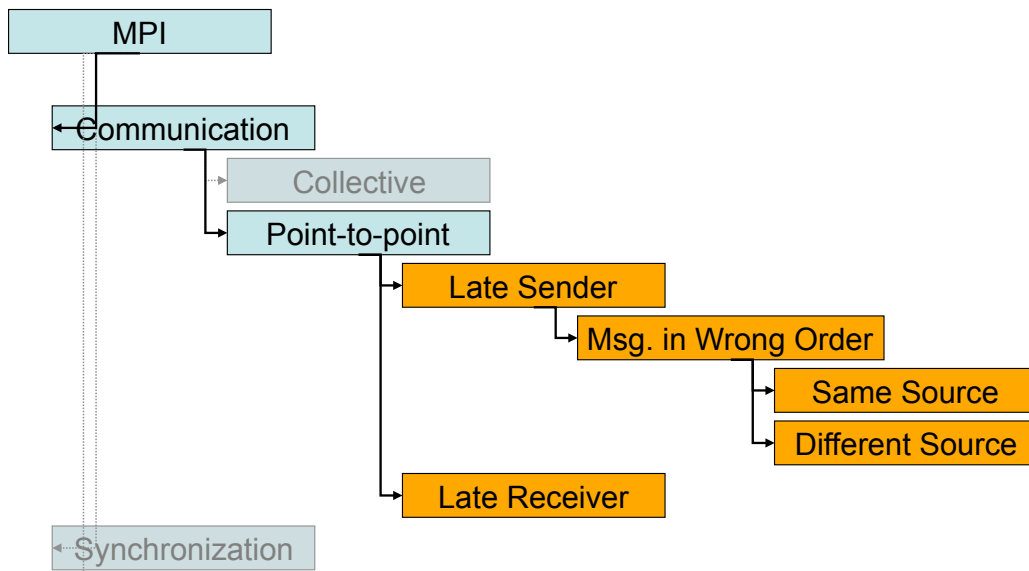


- Waiting time if process n enters a prefix reduction operation earlier than its sending counterparts (i.e., ranks $0..n-1$)
- Applies to: MPI_Scan()

ParaTools

243

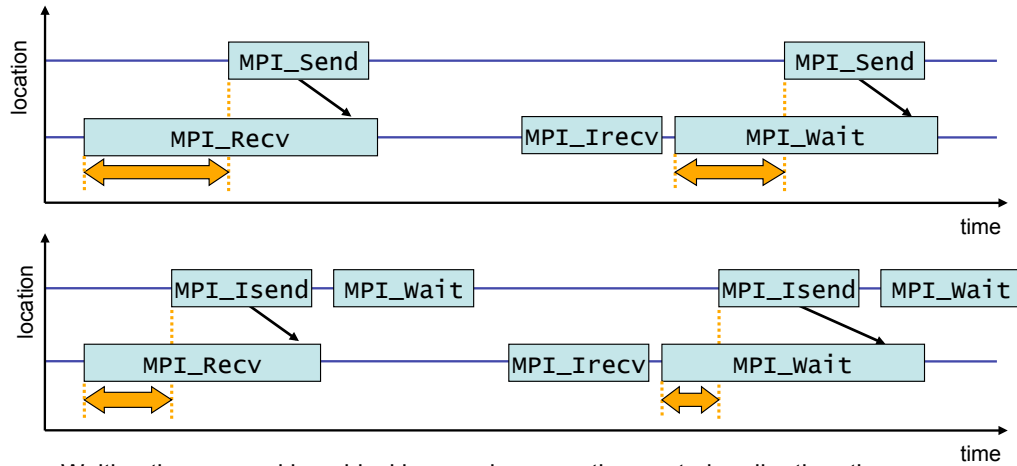
MPI point-to-point communication time



ParaTools

244

Late Sender = Early Receive

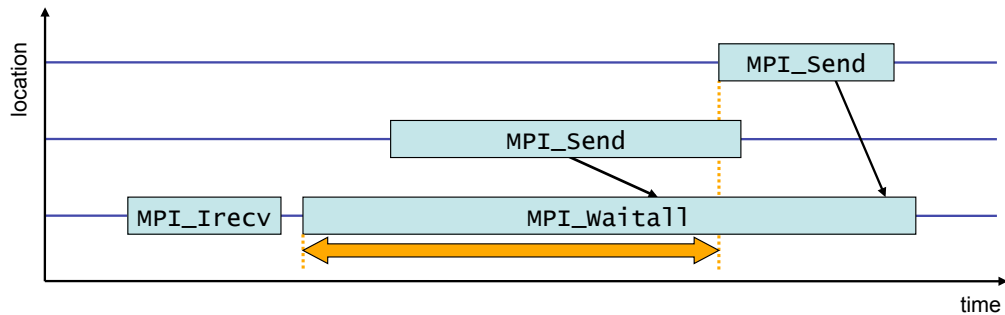


- Waiting time caused by a blocking receive operation posted earlier than the corresponding send operation
- Applies to blocking as well as non-blocking communication

ParaTools

245

Late Sender = Early Receive (cont.)

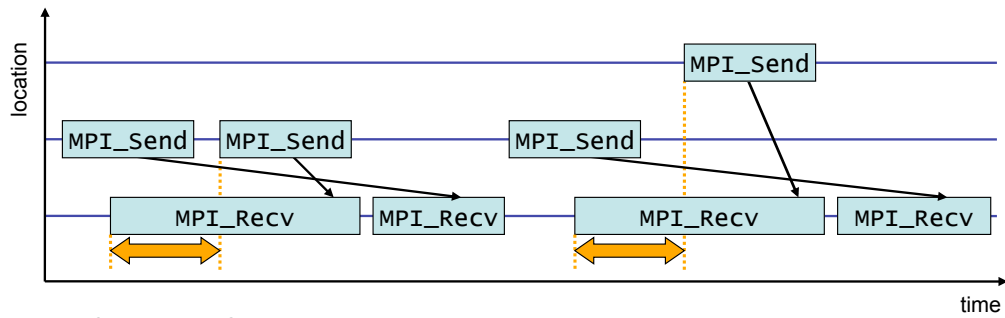


- While waiting for several messages, the maximum waiting time is accounted
- Applies to: `MPI_waitall()`, `MPI_waitsome()`

ParaTools

246

Late Sender, Messages in Wrong Order

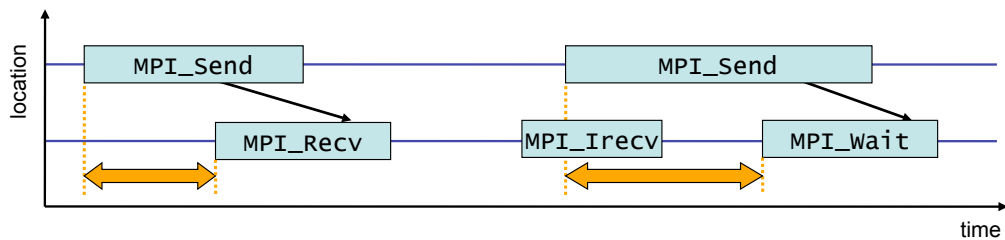


- Refers to Late Sender situations which are caused by messages received in wrong order
 - Early receive of message out of order
- Comes in two flavours:
 - Messages sent from same source location
 - Messages sent from different source locations

ParaTools

247

Late Receiver = Early Send

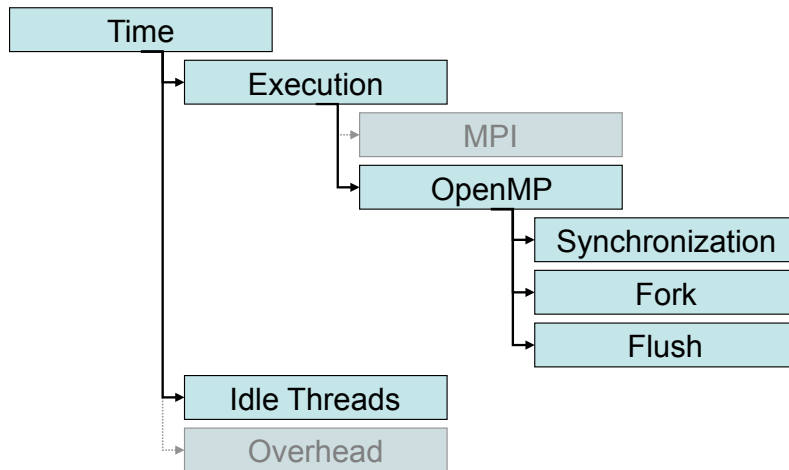


- Waiting time caused by a blocking send operation posted earlier than the corresponding receive operation
- Does not apply to non-blocking sends

ParaTools

248

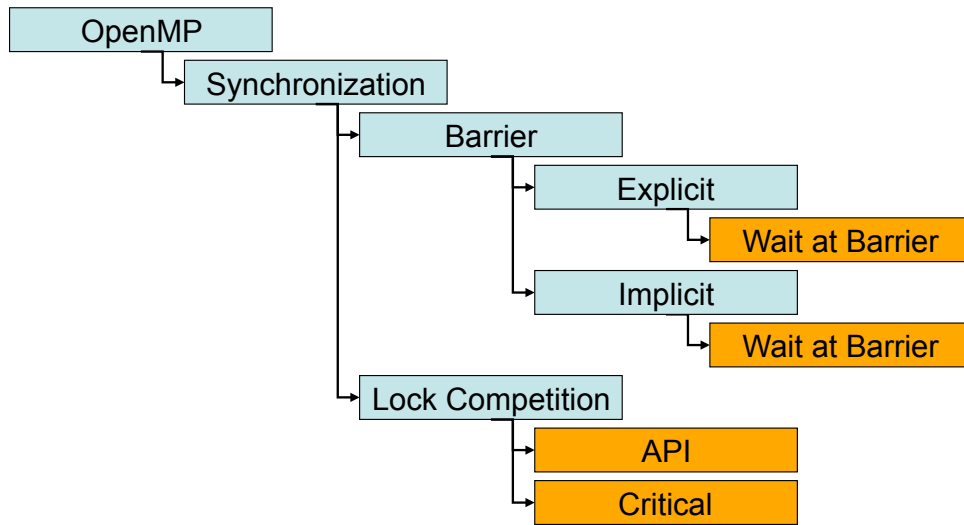
OpenMP Time hierarchy



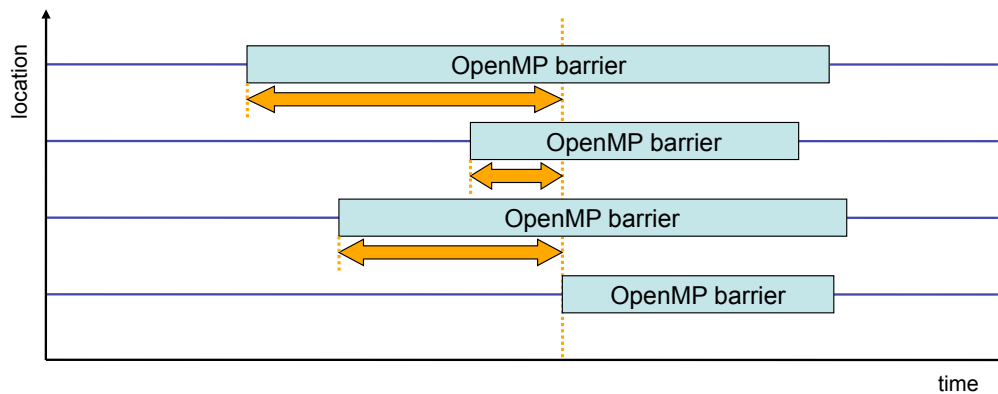
OpenMP Time hierarchy details

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

OpenMP synchronization time

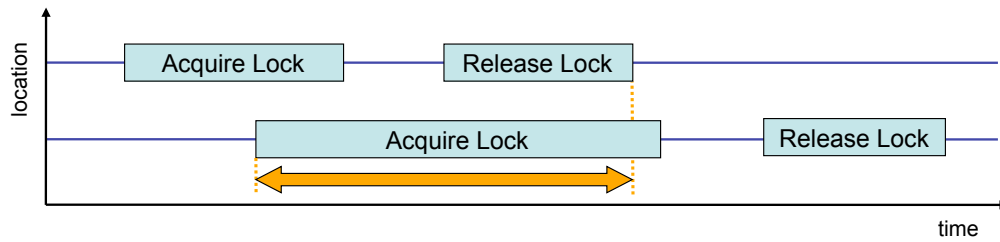


Wait at Barrier = Early Barrier



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

Lock competition

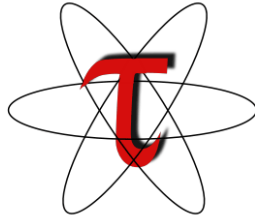


- Time a thread spends waiting for a lock that is held by other threads until it is released and can be acquired by this thread
- Applies to: critical sections, OpenMP lock API

Other metrics

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

Hands-on training with sample codes



ParaTools

255

Labs!



Lab: PAPI, and TAU

ParaTools

256

Lab Instructions

Get `workshop.tar.gz` using:

```
% wget
  http://www.paratools.com/lanl11/workshop.tar.gz
```

Or

```
% cp /usr/projects/packages/tau/cielito/tar/
  workshop.tar.gz;
  tar xzf workshop.tar.gz
```

And follow the instructions in the README file.

```
http://tau.uoregon.edu/point.iso LiveDVD
```

For LiveDVD, see `~/workshop-point/README` and follow.

Use `/usr/projects/packages/tau/cielito/_latest` as the TAU directory on Cielito, LANL and craycnl as the architecture directory. For Yellowrail or Turing, please use `x86_64` as the architecture directory.

ParaTools

257

Lab Instructions

To profile a code using TAU:

1. Choose TAU stub makefile

```
% source /usr/projects/packages/tau/cielito//tau.bashrc
  [ or .cshrc]
% export TAU_MAKEFILE=
  $TAU/Makefile.tau-[options]
```
2. Change the compiler name to `tau_cxx.sh`, `tau_f90.sh`, `tau_cc.sh`:

```
% make CC=tau_cc.sh CXX=tau_cxx.sh F90=tau_f90.sh
```
3. Allocate nodes to run jobs:

```
% qsub -I -q debug -l size=12 -l walltime=0:59:00
```
4. If stub makefile has `-papi` in its name, set the `TAU_METRICS` environment variable:

```
% export TAU_METRICS=TIME:PAPI_L2_DCM:PAPI_TOT_CYC...
```
5. Execute the application:

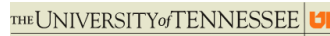
```
% aprun -n 8 ./a.out
```
6. Build and run workshop examples, then run `pprof/paraprof`

ParaTools

258

Support Acknowledgements

- Department of Energy (DOE)
 - Office of Science contracts
 - ANL, ORNL, PNNL contracts
 - LLNL-LANL-SNL ASC/NNSA Level 3 contract
- Department of Defense (DoD)
 - HPCMO, PETTT, HPTi
- National Science Foundation (NSF)
 - POINT, SI2
- University of Oregon
 - Dr. A. Malony, W. Spear, S. Biersdorff, S. Millstein, Dr. C. Lee
- University of Tennessee, Knoxville
 - Dr. Shirley Moore
- T.U. Dresden, GWT
 - Dr. Wolfgang Nagel and Dr. Andreas Knupfer
- Research Centre Juelich
 - Dr. Bernd Mohr, Dr. Felix Wolf



ParaTools