

# TAU Performance System®

TAU Tutorial Bldg. 880/A1N  
Sandia National Laboratories, NM  
Sameer Shende  
[sameer@paratools.com](mailto:sameer@paratools.com)  
<http://www.paratools.com/sandia11>

## ParaTools

---

### Outline - Brief tutorial on the TAU toolset (morning)

	Slide #
• Background information, application examples	5
• Overview of different methods of instrumenting applications	16
• Custom profiling	57
• Techniques for manual instrumentation of individual routines	65
• Generating event traces	67
• Running the application; generation of performance data	78
• Analyzing performance data with ParaProf, PerfExplorer	85
• Throttling effect of frequently called small subroutines	122
• Observing I/O bandwidth and volume	126
• PAPI hardware counters	137
• Estimation of tool intrusiveness	153
• Hands-on training with sample codes (provided)	157

## ParaTools

---

## Hands-on analysis of own benchmark application (afternoon)

---

- Review of instrumentation process, address any problems with individual applications
- Determination of routines requiring further investigation, custom profiling if needed
- Running on the RedSky, 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

## Workshop Goals

---

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

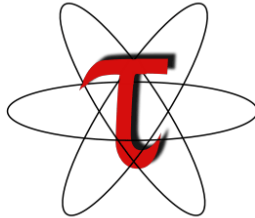
**ParaTools**

---

4

---

# Background information, application examples

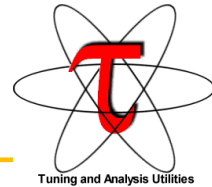


ParaTools

---

5

## 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

---

6

## 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

---

7

## 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

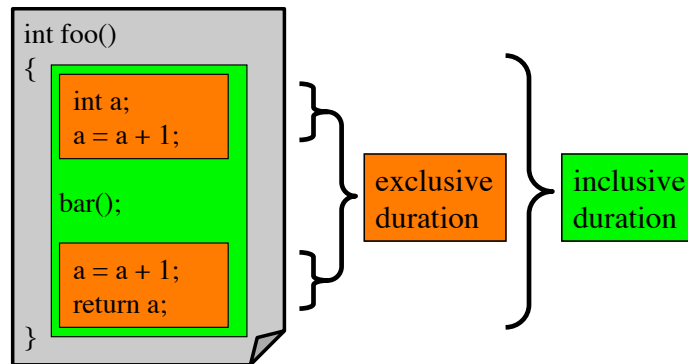
Paratools

---



## 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

```

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 1x%

```

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

Atomic events  
(trigger with  
value)

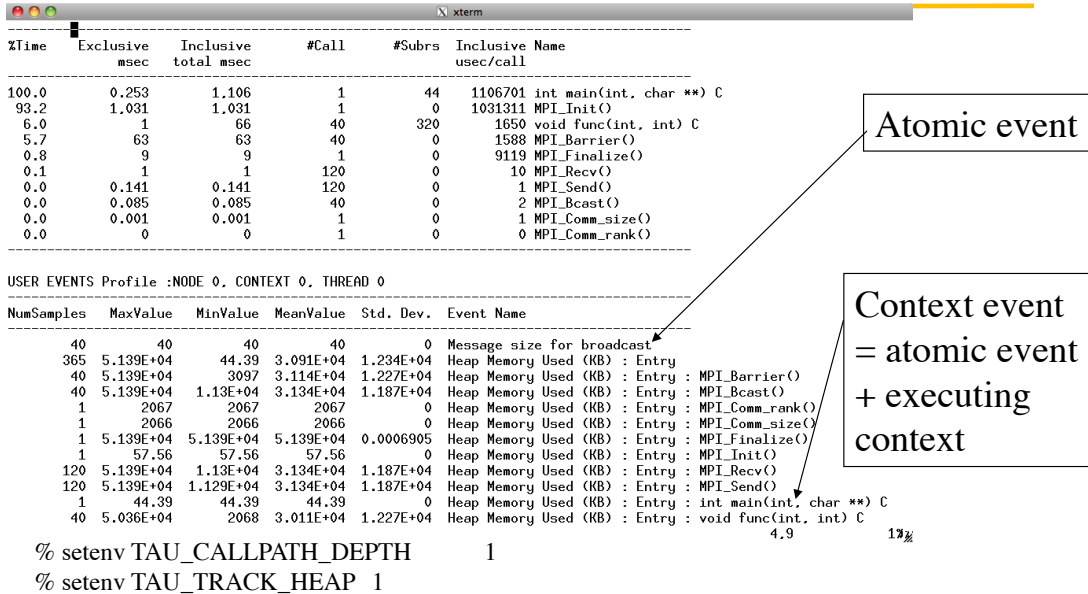
```

% setenv TAU_CALLPATH_DEPTH 0
% setenv TAU_TRACK_HEAP 1

```

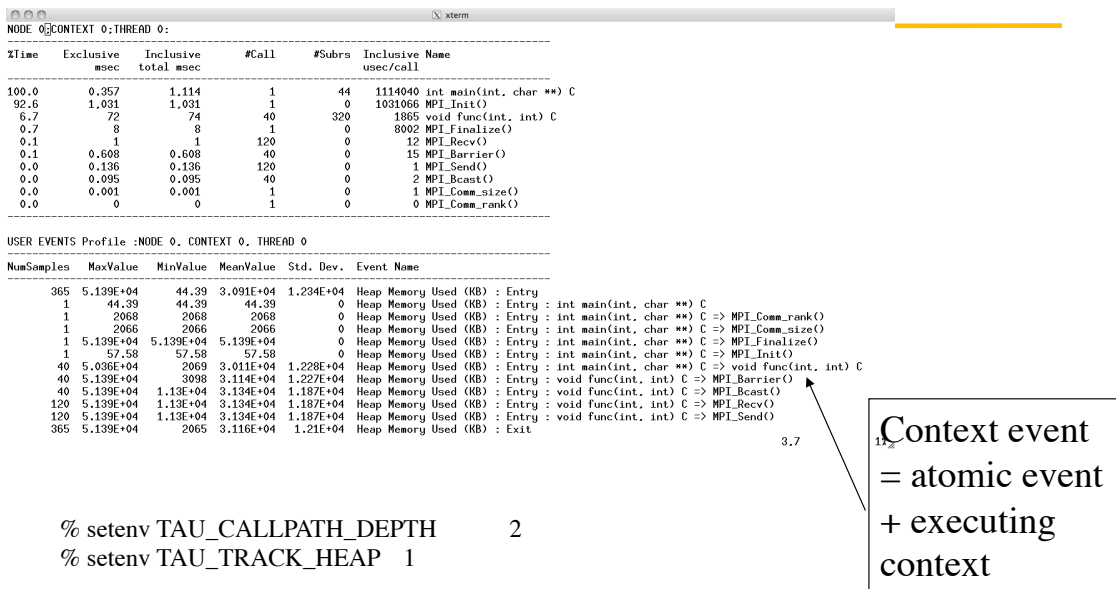
## ParaTools

# Atomic Events, Context Events



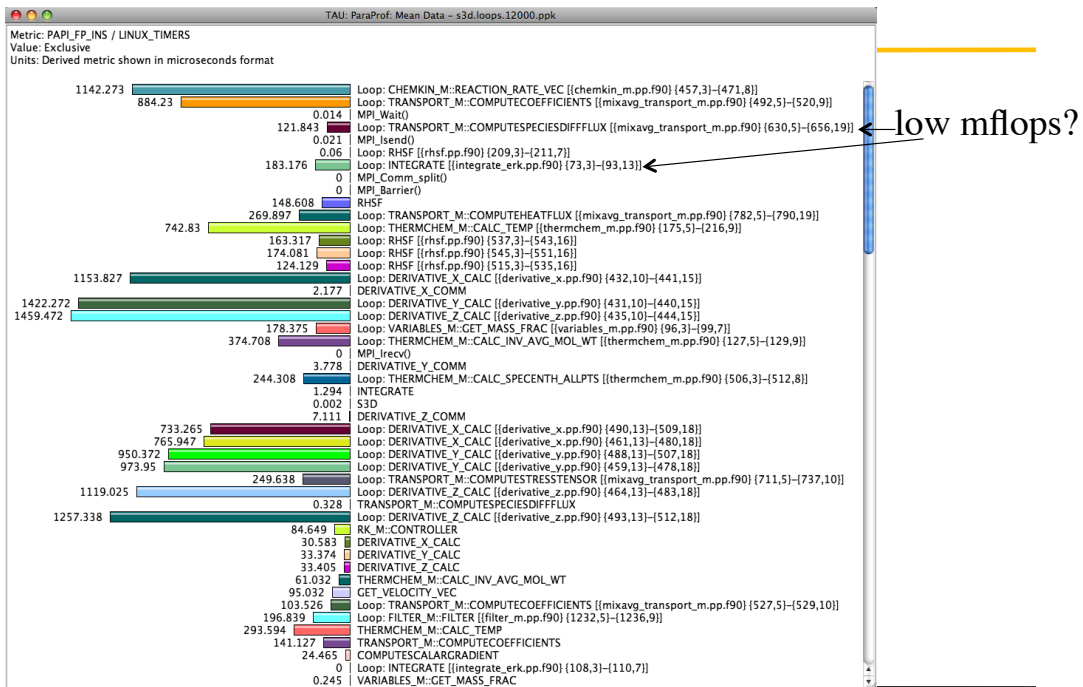
## Paratools

# Context Events (Default)

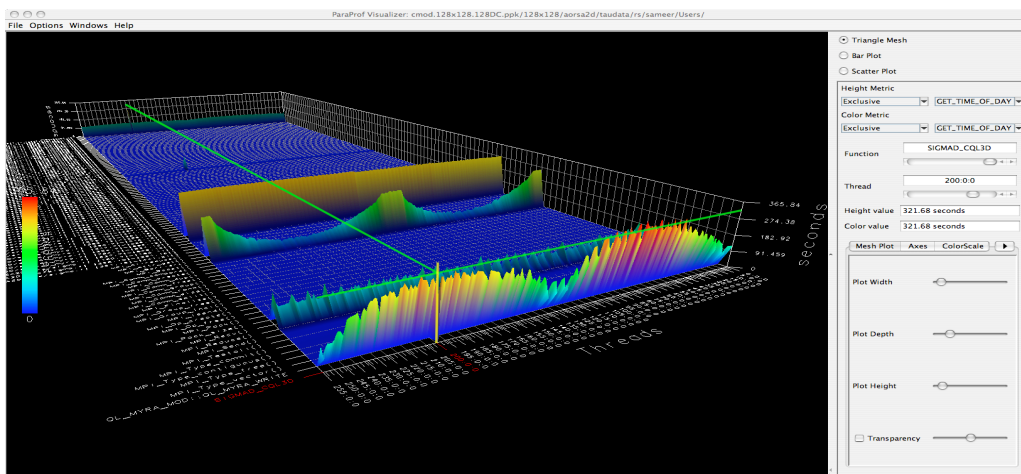


## Paratools

# ParaProf: Mflops Sorted by Exclusive Time



# Parallel Profile Visualization: ParaProf



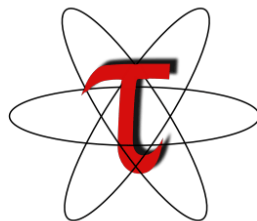
## Steps of Performance Evaluation

---

- 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

ParaTools

---

17

## Instrumentation Techniques

---

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

ParaTools

---

18

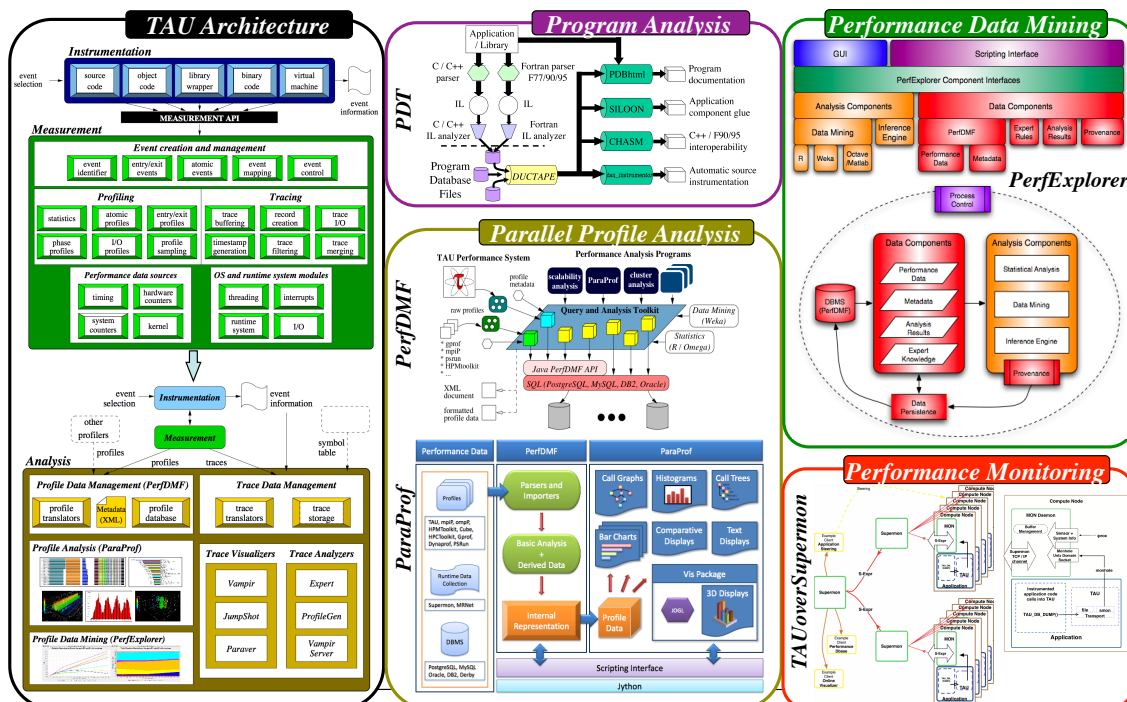
# Instrumentation Techniques

- 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

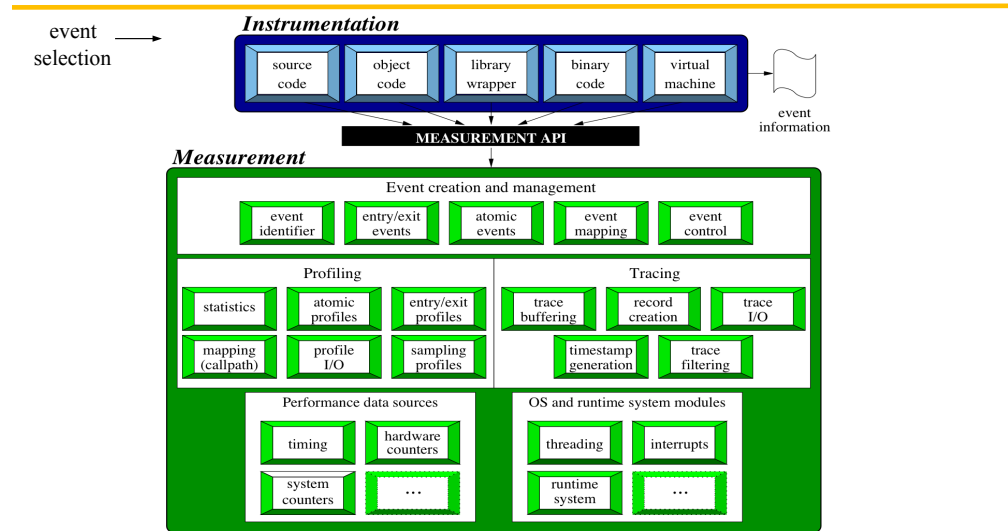
## ParaTools

19

## TAU Performance System Components

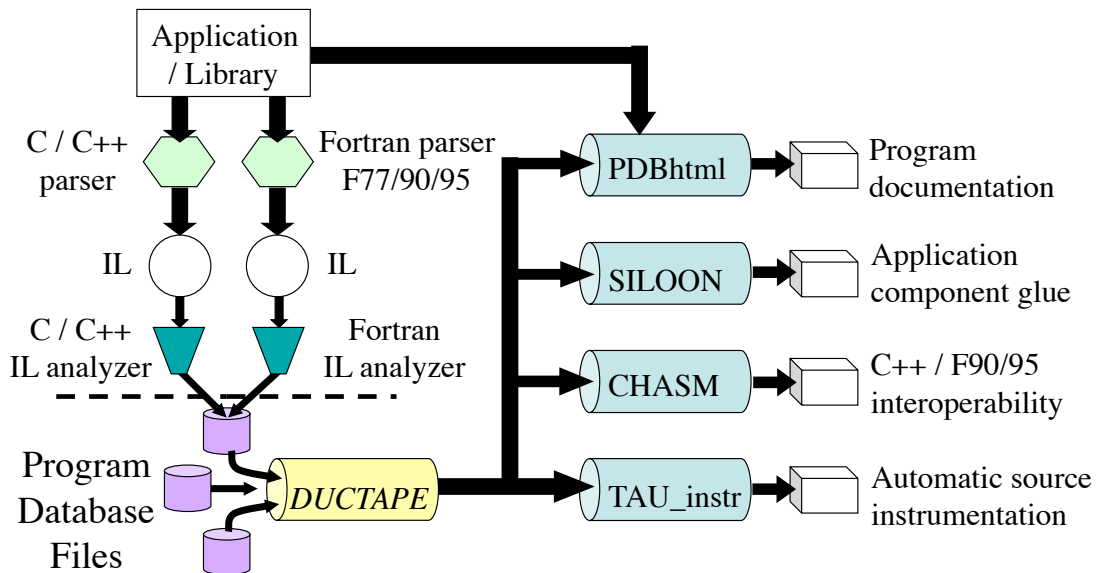


# TAU Performance System Architecture



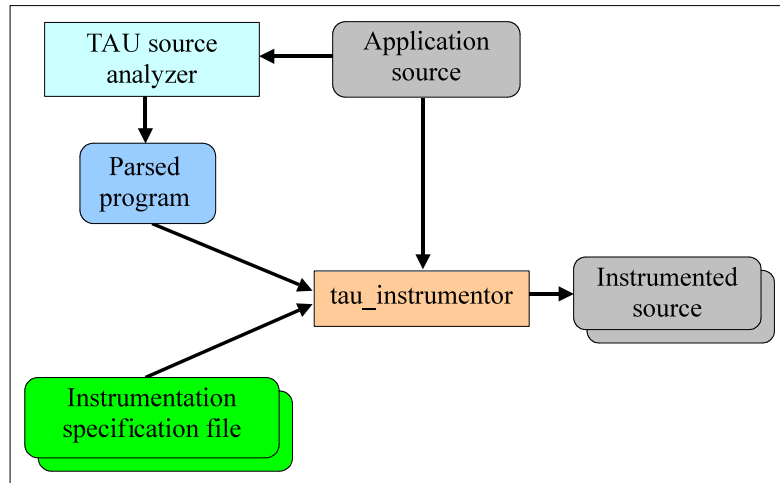
ParaTools

# Program Database Toolkit (PDT)



ParaTools

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

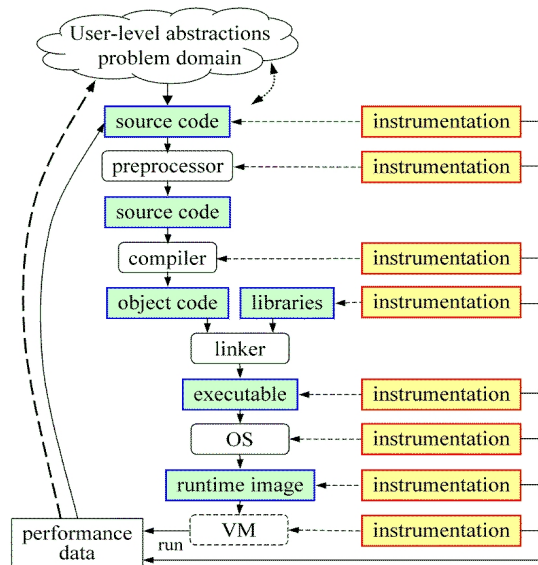


ParaTools

23

## Direct Observation: Mapping

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



ParaTools

24



## 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 /projects/tau/tau.bashrc (or .cshrc)  
% export TAU\_MAKEFILE=\$TAU\_ROOT/lib/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:  
% mpif90 foo.f90  
changes to  
% tau\_f90.sh foo.f90
- Execute application and analyze performance data:
  - % pprof (for text based profile display)
  - % paraprof (for GUI)

## ParaTools

---

25

## TAU Measurement Configuration

---

```
% cd $TAU_ROOT/lib; 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_ROOT/lib/Makefile.tau-mpi-pdt-pgi  
– % tau_f90.sh matrix.f90 -o matrix
```

## ParaTools

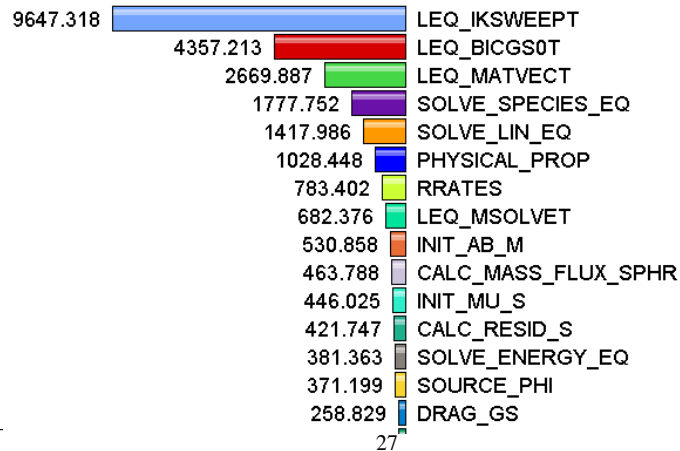
---

26

## Usage Scenarios: Routine Level Profile

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

Metric: P\_VIRTUAL\_TIME  
Value: Exclusive  
Units: seconds



ParaTools

27

## Solution: Generating a flat profile with MPI

```
% export TAU_MAKEFILE=$TAU_ROOT/lib/Makefile.tau-mpi-pdt-pgi
% export PATH=$TAU_ROOT/bin:$PATH
OR
% source /projects/tau/tau.bashrc
% tau_f90.sh matmult.f90 -o matmult
(Or edit Makefile and change F90=tau_f90.sh)

% srun -n 4 -p specops./matmult
% module load gcc
% pprof
% paraprof &
OR
% paraprof --pack app.ppk
  Move the app.ppk file to your desktop.

% paraprof app.ppk
% paraprof &
```

ParaTools

28

## Automatic Instrumentation

---

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

```
Before
CXX = CC
F90 = ftn
CFLAGS =
LIBS = -lm
OBSJ = f1.o f2.o f3.o ... fn.o

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

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

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

ParaTools

29

## TAU\_COMPILER Commandline Options

---

- See `<taudir>/<arch>/bin/tau_compiler.sh -help`
- Compilation:
  - % `ftn -c foo.f90`
  - Changes to
    - % `gfpase foo.f90 $(OPT1)`
    - % `tau_instrumentor foo.pdb foo.f90 -o foo.inst.f90 $(OPT2)`
    - % `ftn -c foo.f90 $(OPT3)`
- Linking:
  - % `ftn foo.o bar.o -o app`
  - Changes to
    - % `ftn foo.o bar.o -o app $(OPT4)`
- Where options `OPT[1-4]` default values may be overridden by the user:
  - `F90 = tau_f90.sh`

ParaTools

30

# 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

...

# 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_OPS:PAPI_NATIVE_<event>)

## Paratools

# Compiling Fortran Codes with TAU

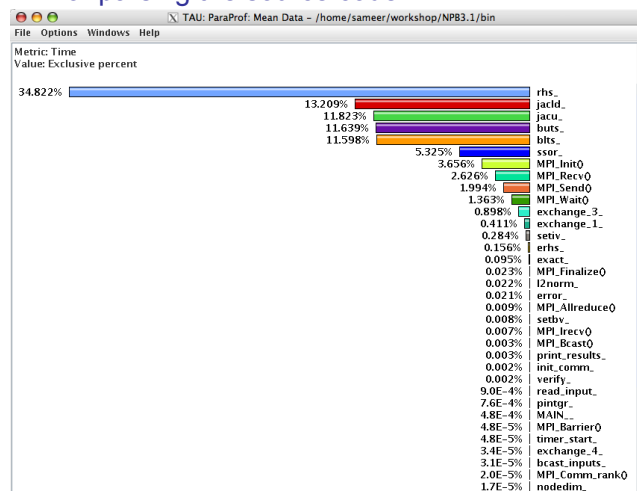
- If your Fortran code uses free format in .f files (fixed is default for .f), you may use:  
`% export TAU_OPTIONS='-optPdtF95Opts="-R free" -optVerbose '`
- To use the compiler based instrumentation instead of PDT (source-based):  
`% export TAU_OPTIONS='-optCompInst -optVerbose'`
- If your Fortran 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`

## ParaTools

33

# Usage Scenarios: Compiler-based Instrumentation

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



## ParaTools

34

## Use Compiler-Based Instrumentation

---

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

% srun -n 4 -p specops ./a.out
% paraprof --pack app.ppk
    Move the app.ppk file to your desktop.
% paraprof app.ppk
```

## ParaTools

---

35

## 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

---

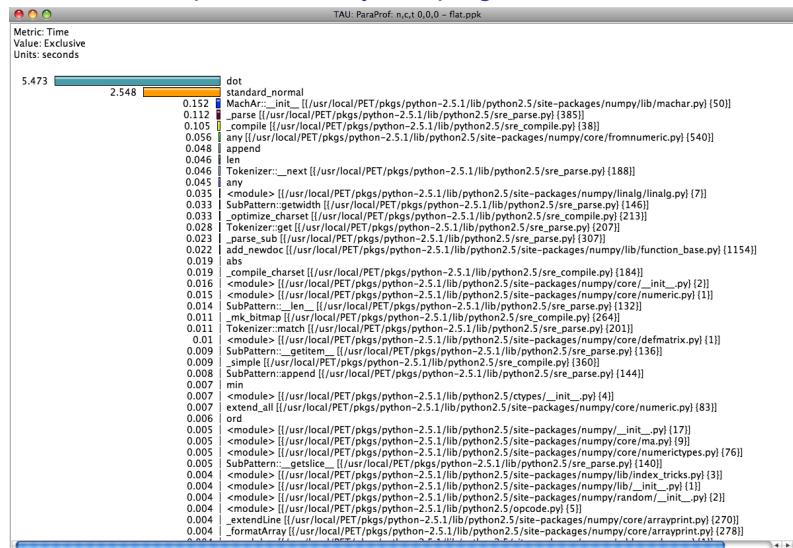
# tau\_run with NAS PBS

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

3  
7

## 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**

39

## 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)
% export LD_LIBRARY_PATH=<taudir>/x86_64/lib/bindings-python-pdt:
$LD_LIBRARY_PATH
% ./wrapper.py

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

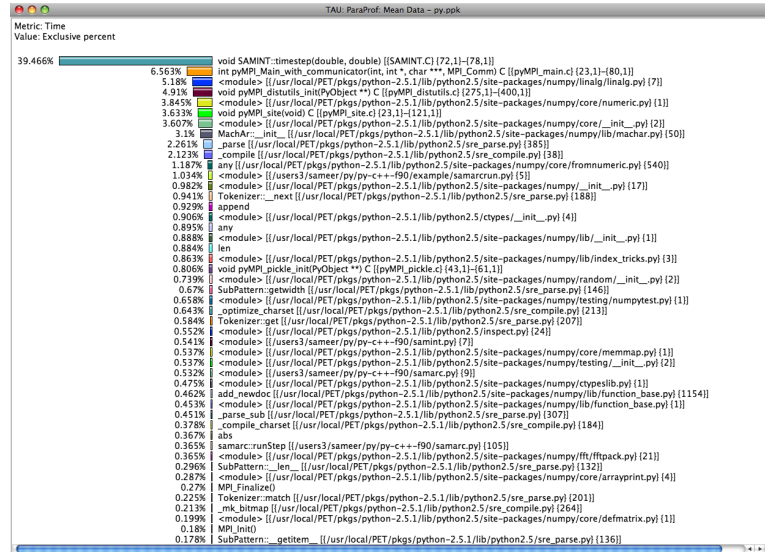
**ParaTools**

40



# Usage Scenarios: Mixed Python+F90+C+pyMPI

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



## Paratools

41

## 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 <pkgs>/pyMPI-2.5b0-TAU/bin/pyMPI
./wrapper.py (Instrumented pyMPI with wrapper.py)
```

42

## 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)

---

- 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

---

## 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
- 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

Paratools wrapper has TAU measurement code

45

## HDF5 Library Wrapping

```
[sameer@zorak]$ 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)
• group (hdf5)
• tau_exec loads libhdf5_wrap.so shared library using -loadlib=<libwrap_pkg.so>
• creates the wrapper/ directory with -opt

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()
  0.1    0.001        0.001        2          0           0 herr_t H5open()
  0.0    0            0            1          0           0 herr_t H5Tclose()
```

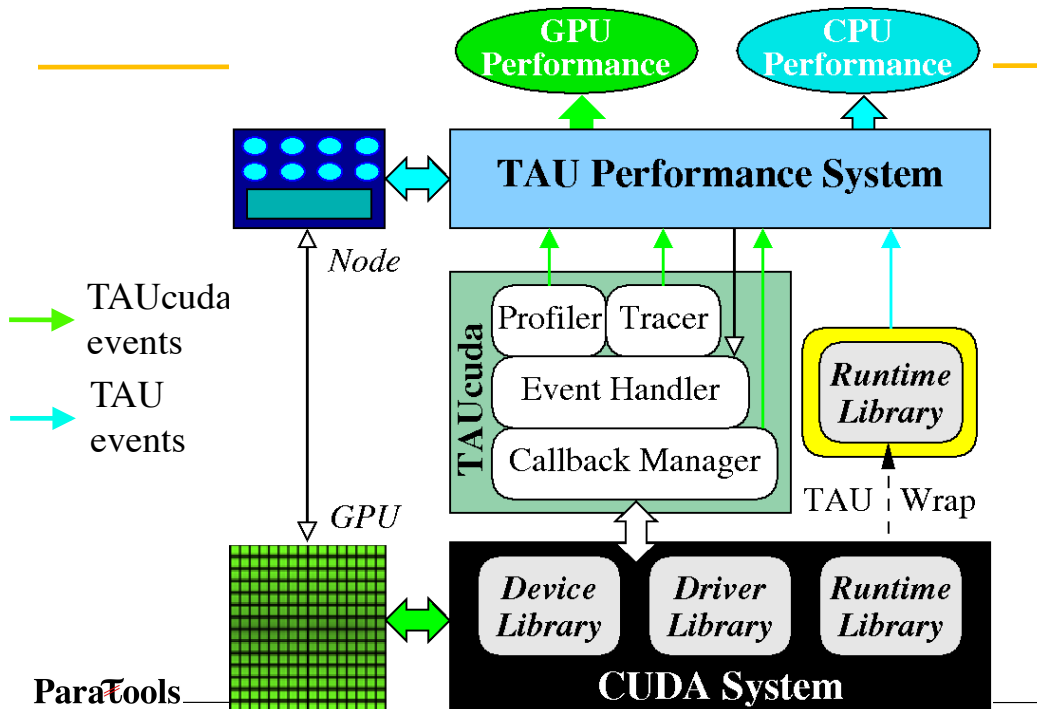
46

## Profiling GPGPU Executions

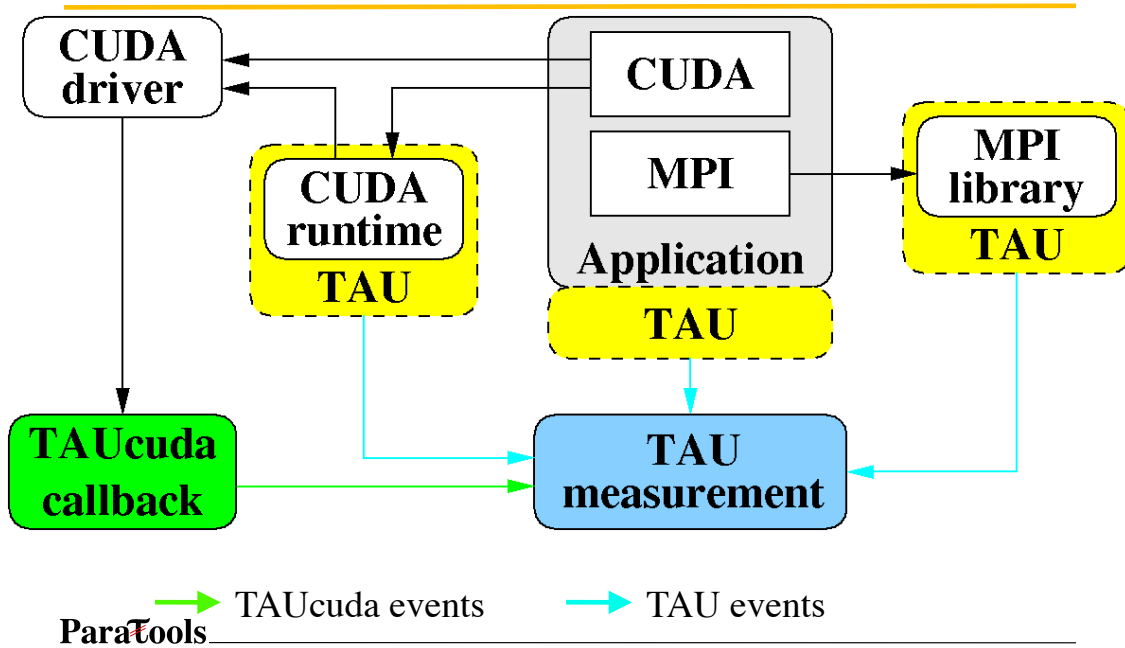
- GPGPU compilers (e.g., CAPS hmpp and PGI) can now automatically generate GPGPU code using manual annotation of loop-level constructs and routines (hmpp)
- The loops (and routines for HMPP) are transferred automatically to the GPGPU
- TAU intercepts the runtime library routines and examines the arguments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

## ParaTools

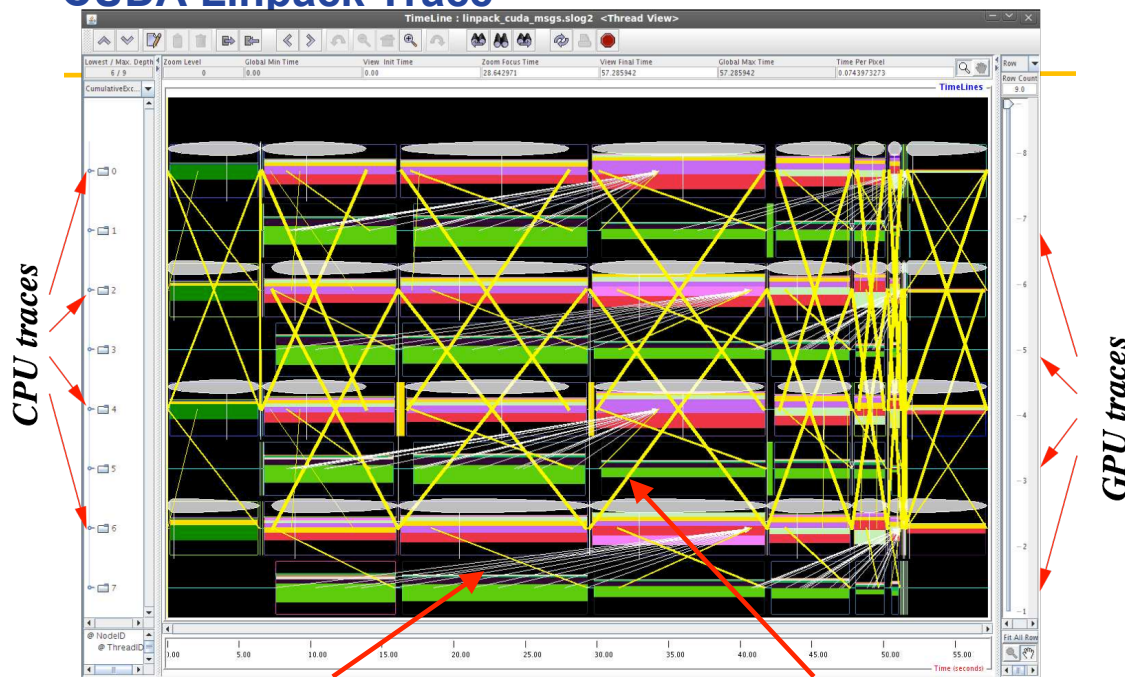
### TAUcuda Architecture



# TAUcuda Instrumentation



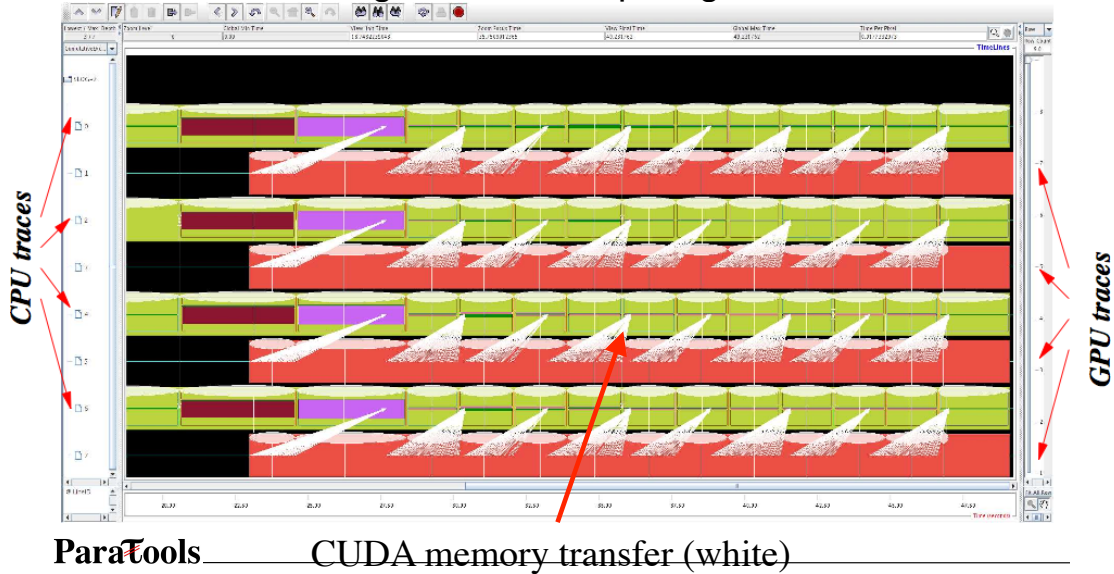
# CUDA Linpack Trace



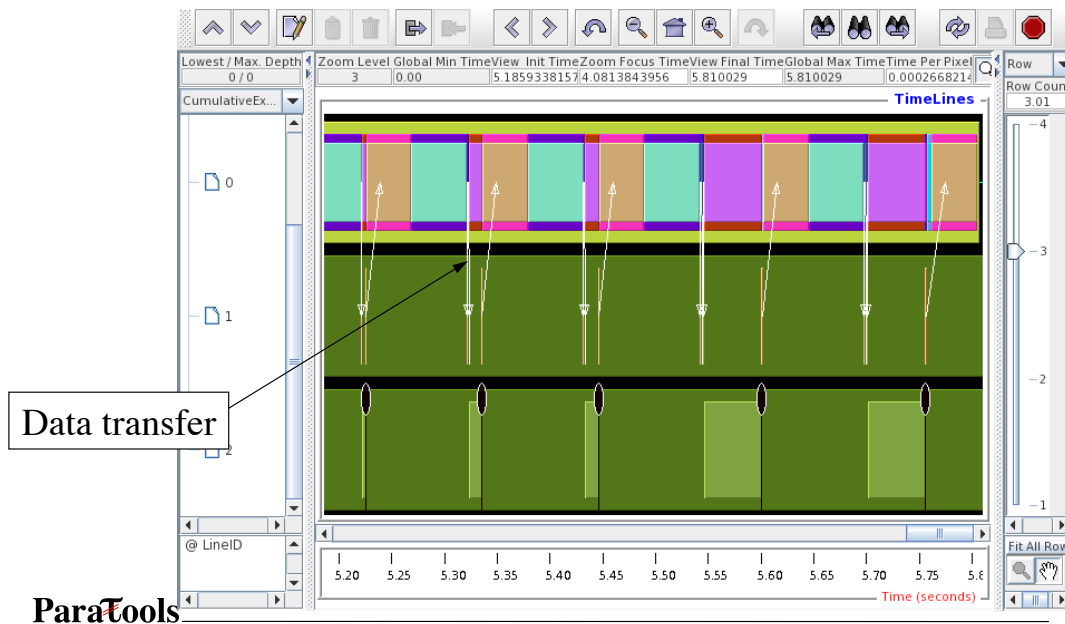
Paratools memory transfer (white) MPI communication (yellow)

## SHOC Stencil2D (512 iterations, 4 CPUxGPU)

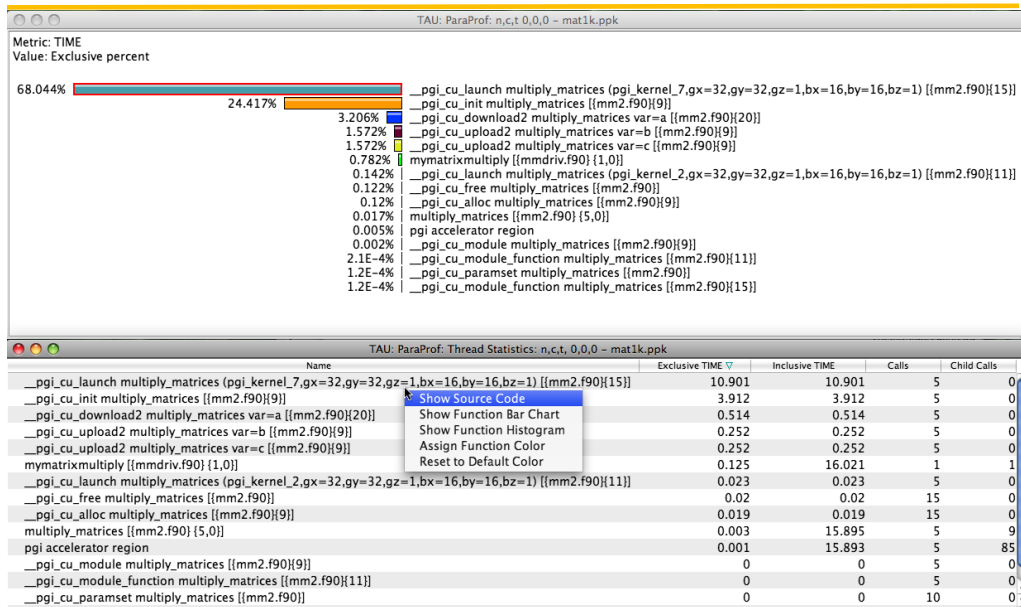
- Scalable Heterogeneous Computing benchmark suite



## Scaling NAMD with CUDA (Jumpshot with TAU)

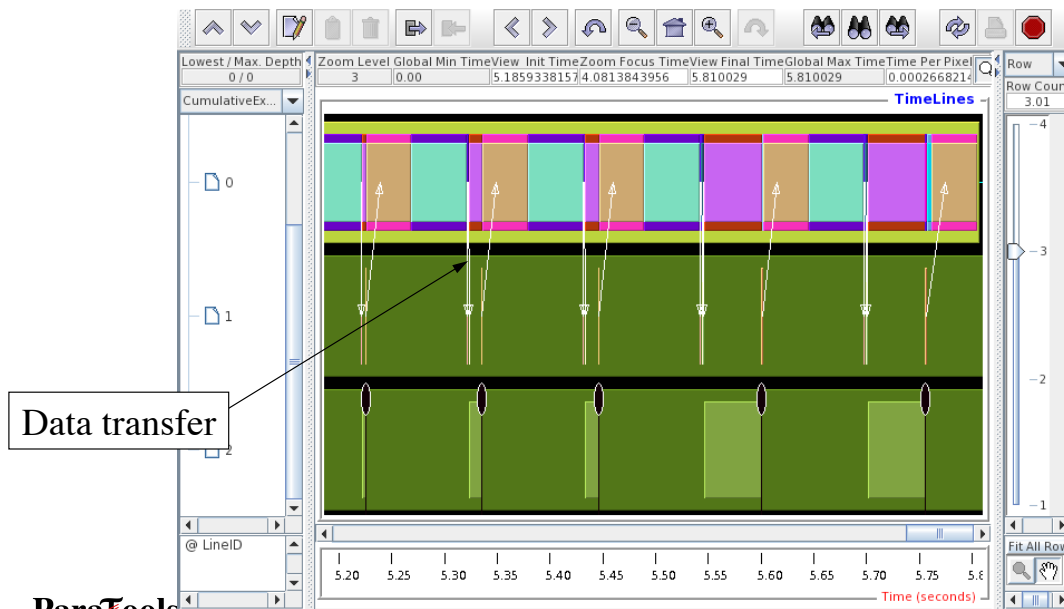


# Measuring Performance of PGI GPGPU Accelerated Code



ParaTools

# Scaling NAMD with CUDA (Jumpshot with TAU)



ParaTools

# Using TAU with Java Applications

Step I: Sun JDK 1.6+ [download from [www.javasoft.com](http://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
```

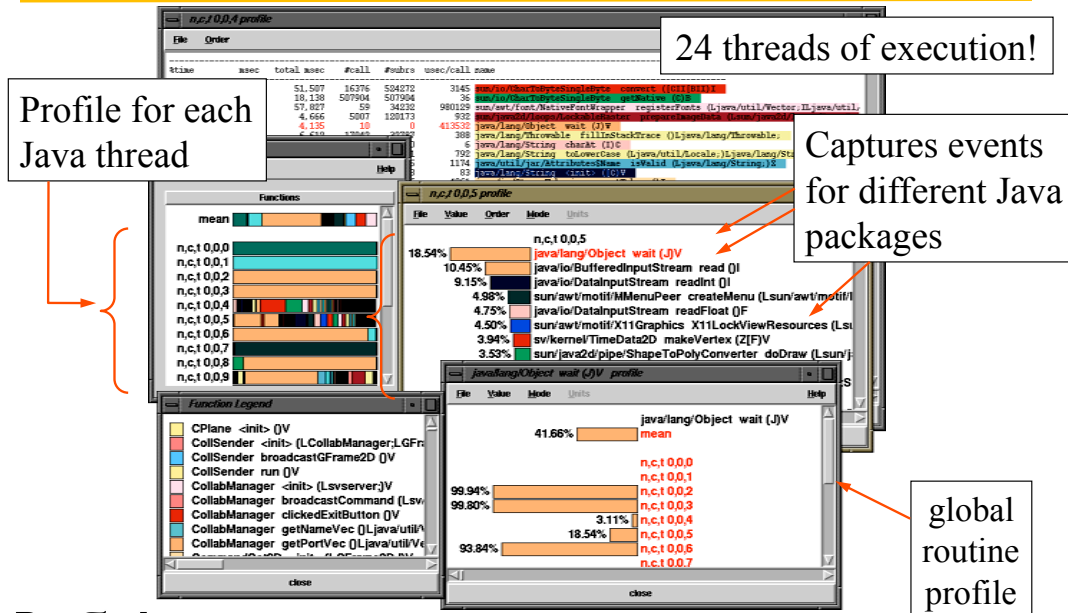
```
% tau_java -tau:agentlib=<different_libTAU.so> -tau:include=<item>
-tau:exclude=<item> application
```

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

ParaTools

55

## TAU Profiling of Java Application (SciVis)



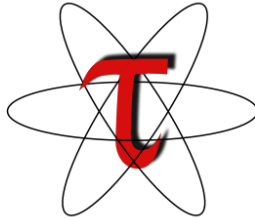
ParaTools

56



---

# Custom profiling



---

## 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
```

## ParaTools

---

59

## 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

---

60

# 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

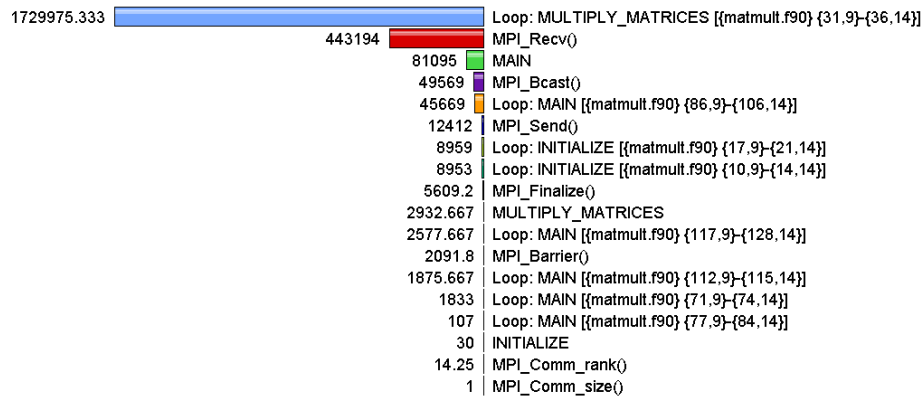
BEGIN_FILE_INCLUDE_LIST
Main.cpp
Foo?.c
*.C
END_FILE_INCLUDE_LIST
# Instruments routines in Main.cpp, Foo?.c and *.C files only
# Use BEGIN_[FILE]_INCLUDE_LIST with END_[FILE]_INCLUDE_LIST
```

61

## Usage Scenarios: Loop Level Instrumentation

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

Metric: GET\_TIME\_OF\_DAY  
Value: Exclusive  
Units: microseconds



Par

# Solution: Generating a loop level profile

```
% export TAU_MAKEFILE=$TAU_ROOT/lib/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 F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)
% srun -n 4 -p specops./a.out
% paraprof --pack app.ppk
Move the app.ppk file to your desktop.

% paraprof app.ppk
```

## Paratools

63

## ParaProf's Source Browser: Loop Level Instrumentation

The image displays three function data windows from ParaProf's Source Browser, each showing a horizontal bar chart of execution time for different compiler optimization levels (n.ct) and a table of mean, standard deviation, and maximum values.

**Window 1: 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

Value	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	
114.581	n.ct 5,0	
114.837	n.ct 6,0	
114.789	n.ct 7,0	

**Window 2: 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

Value	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	
12.339%	n.ct 4,0	
12.241%	n.ct 5,0	
12.221%	n.ct 6,0	
12.226%	n.ct 7,0	

**Window 3: 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

Value	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 Instrumentation:**

```
606 grad_mlxMW(:, :, :n) = grad_mlxMW(:, :, :n)*avnolwt(:, :, :)
```

```
607 end do
```

```
608
```

```
609 ! compute grad_P
```

```
610 if (baro_switch) then
```

```
611   allocate(grad_P(nx,ny,nz,3))
```

```
612   grad_P = 0.0
```

```
613   if (vary_in_x == 1) then
```

```
614     call derivative_x(nx,ny,nz, Press, grad_P(:, :, :, 1), scale_ix, 1)
```

```
615   endif
```

```
616   if (vary_in_y == 1) then
```

```
617     call derivative_y(nx,ny,nz, Press, grad_P(:, :, :, 2), scale_jy, 1)
```

```
618   endif
```

```
619   if (vary_in_z == 1) then
```

```
620     call derivative_z(nx,ny,nz, Press, grad_P(:, :, :, 3), scale_iz, 1)
```

```
621   endif
```

```
622 endif
```

```
623
```

```
624 ! Changed by Rananan - 01/24/05
```

```
625 ! Ds_mlxavg is now vhrb
```

```
626 !
```

```
627 !grad_P/press and avnolwt*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 m=1,3
```

```
631   SPECIES: do n=1,n_spec
```

```
632
```

```
633   if (baro_switch) then
```

```
634     ! driving force includes gradient in mole fraction and baro-diffusion:
```

```
635     diffFlux(:, :, :n, n) = - Ds_mlxavg(:, :, :n) * ( grad_Ys(:, :, :n, n) &
```

```
636       + Ys(:, :, :n) * ( grad_mlxMW(:, :, :n) &
```

```
637         + (1 - molwt(n)/Paromolwt) * grad_P(:, :, :n, Press) )
```

```
638   else
```

```
639     ! driving force is just the gradient in mole fraction:
```

```
640     diffFlux(:, :, :n, n) = - Ds_mlxavg(:, :, :n) * ( grad_Ys(:, :, :n, n) &
```

```
641       + Ys(:, :, :n) * grad_mlxMW(:, :, :n) )
```

```
642   endif
```

```
643
```

```
644 ! Add thermal diffusion:
```

```
645 if (thermoDiff_switch) then
```

```
646   diffFlux(:, :, :n, n) = diffFlux(:, :, :n, n) &
```

```
647     - Ds_mlxavg(:, :, :n) * Rs_therm_diff(:, :, :n) * molwt(n) &
```

```
648     * avnolwt * grad_T(:, :, :n) / 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
```

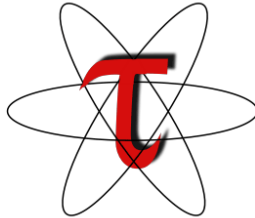
```
668 subroutine computeStressTensor( grad_u)
```

```
669
```

```
670
```

---

## Techniques for manual instrumentation of individual routines



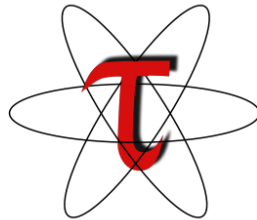
### Instrumenting a C code

```
#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.
```

---

# Generating event traces



ParaTools

---

67

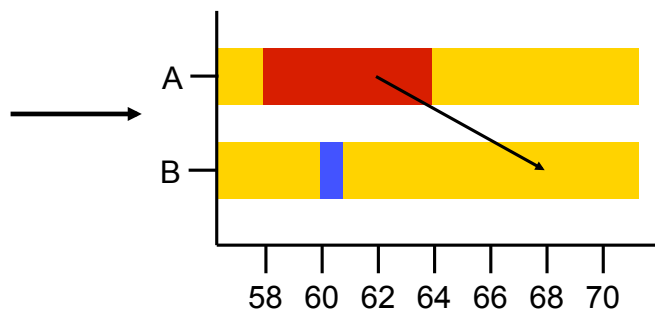
## Tracing Analysis and Visualization

---

1	master
2	worker
3	...



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



ParaTools

---

68

## 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

## ParaTools

---

69

## 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 interoperation between tracing tools

## ParaTools

---

70

## Generate a Trace File

---

```
% export TAU_MAKEFILE=$TAU_ROOT/lib/Makefile.tau-mpi-pdt-pgi
% export PATH=$TAU_ROOT/bin:$PATH
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)
%
% export TAU_TRACE=1
% srun -n 4 -p specops ./a.out
% tau_treemerge.pl
(merges binary traces to create tau.trc and tau.edf files)
JUMPSHOT:
% tau2slog2 tau.trc tau.edf -o app.slog2
% jumpshot app.slog2
OR
VAMPIR:
% tau2otf tau.trc tau.edf app.otf -n 4 -z
(4 streams, compressed output trace)
% vampir app.otf
OR
PARAVER:
% tau_convert -paraver tau.trc tau.edf app.prv
% paraver app.prv
```

ParaTools

71

## Jumpshot

---

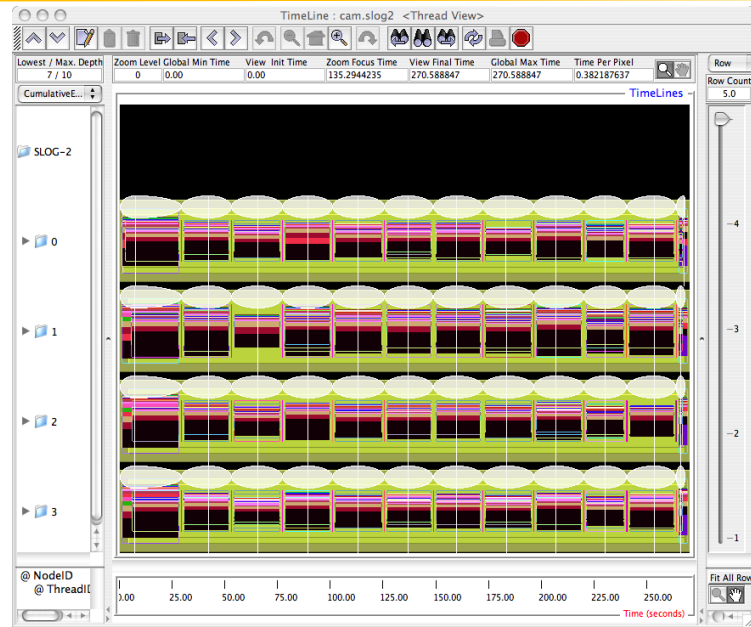
- <http://www-unix.mcs.anl.gov/perfvis/software/viewers/index.htm>
- Developed at Argonne National Laboratory as part of the MPICH project
  - Also works with other MPI implementations
  - Jumpshot is bundled with the TAU package
- Java-based tracefile visualization tool for postmortem performance analysis 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

ParaTools

72



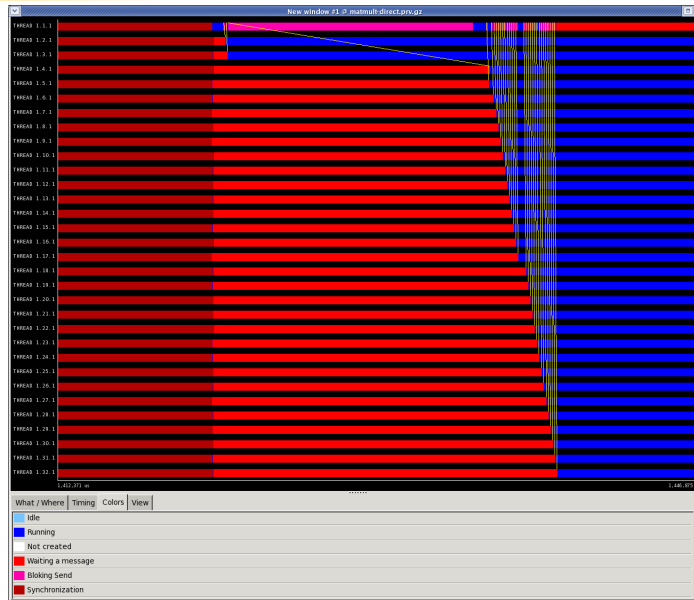
# Jumpshot



ParaTools

73

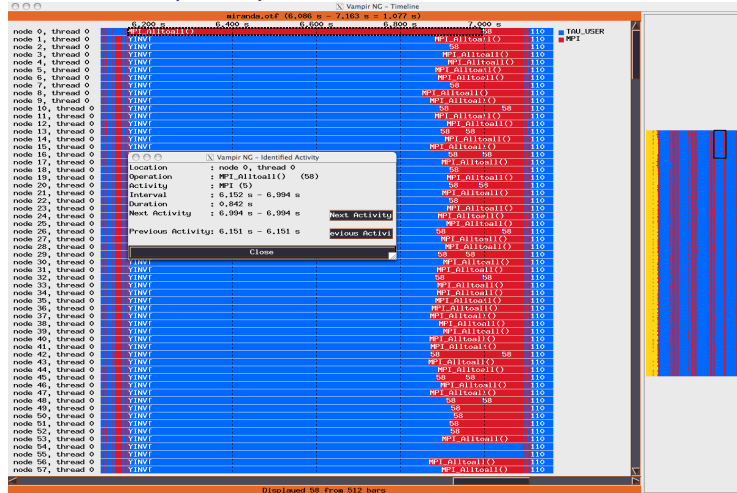
# ParaVer [<http://www.bsc.es/paraver>]



ParaTools

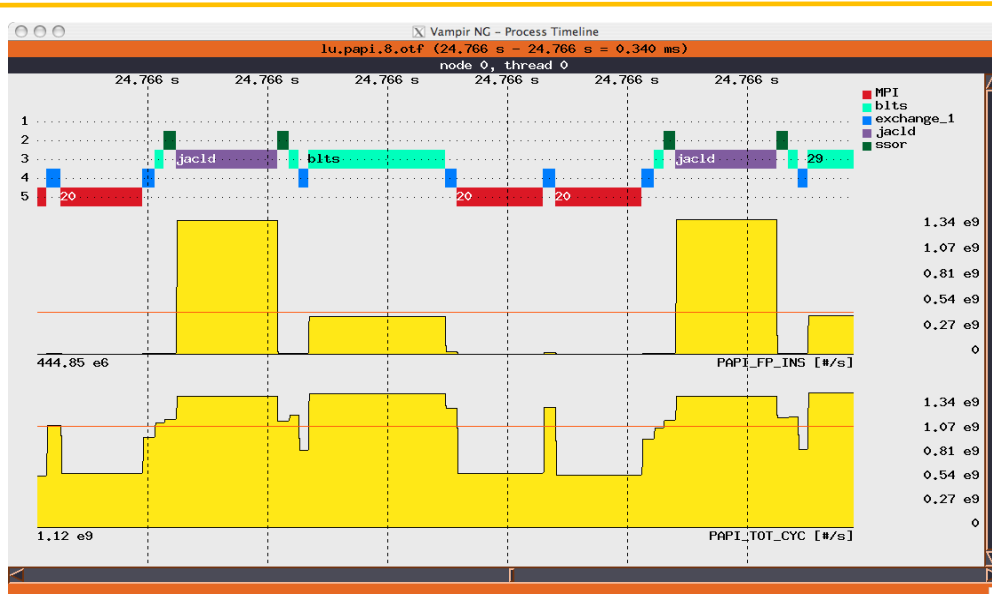
# Usage Scenarios: Generating a Trace File

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



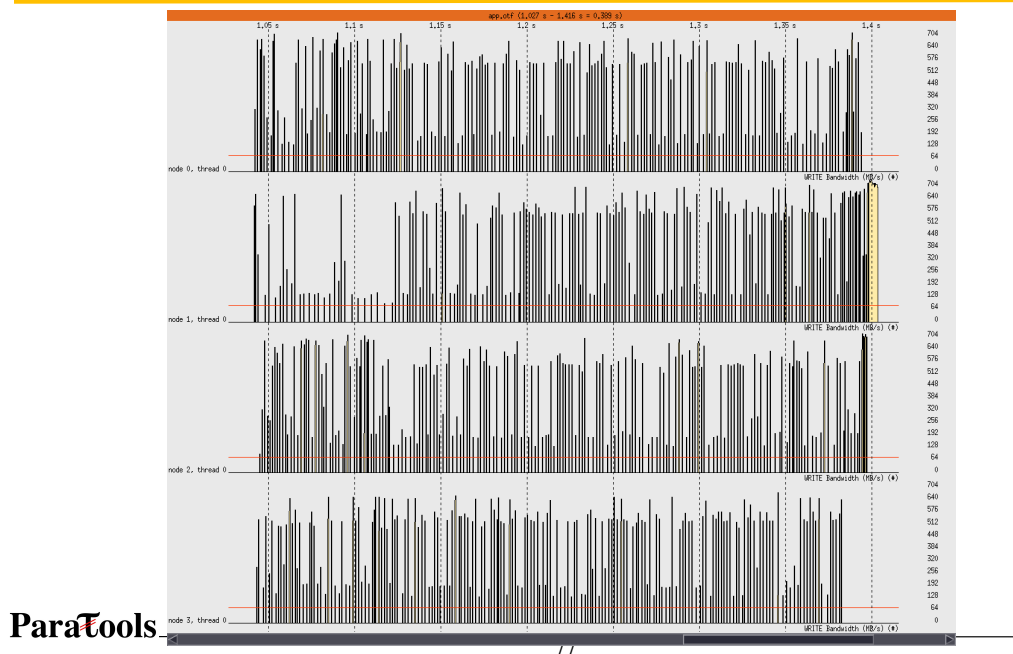
ParaTools

# VNG Process Timeline with PAPI Counters

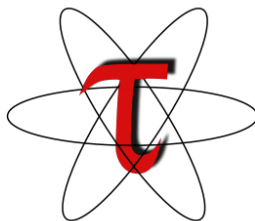


ParaTools

## Vampir Counter Timeline Showing I/O BW



Running the application, generation of performance data



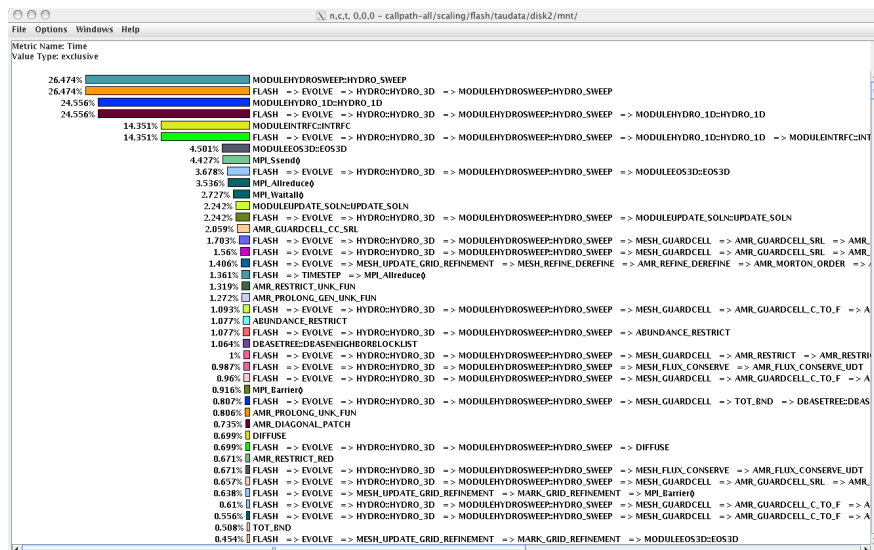
# Environment Variables in TAU

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_OPS:PAPI_NATIVE_<event>)

Paratools

## Usage Scenarios: Generating Callpath Profile

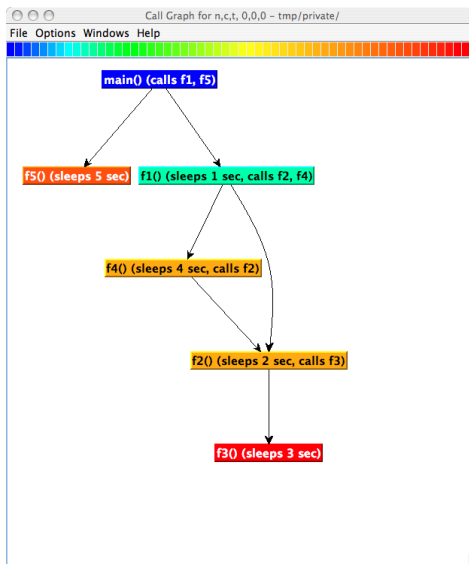
- Callpath profile for a given callpath depth:



Paratools

# Callpath Profile

- Generates program callgraph



Paratools

81

# Communication Matrix

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

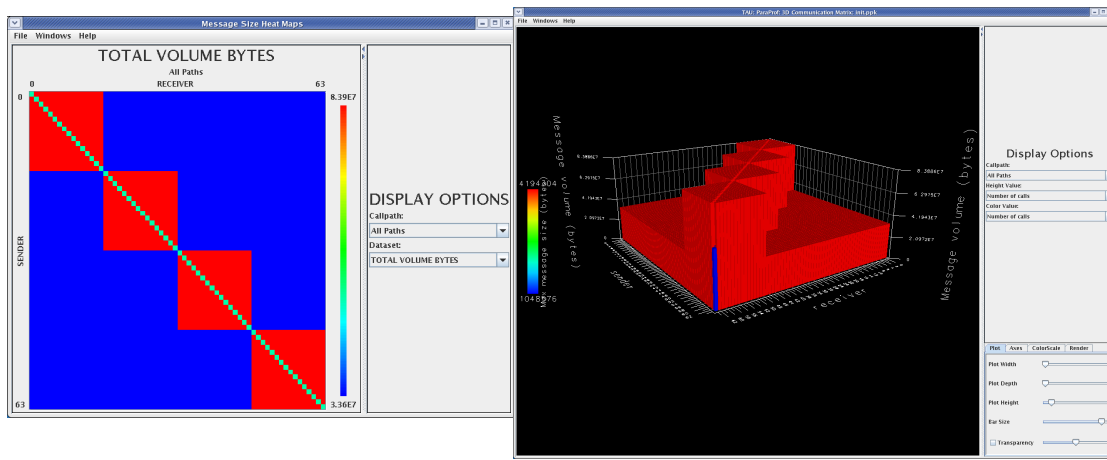
%
% export TAU_COMM_MATRIX=1
%srun -n 4 -p specops./a.out (setting the environment variables)

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

Paratools

82

# ParaProf: Communication Matrix Display



## ParaTools

# Generate a Callpath Profile

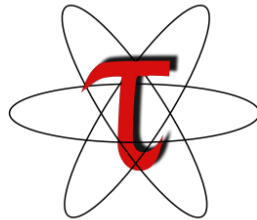
```
% export TAU_MAKEFILE=$TAU_ROOT/lib/Makefile.tau-mpi-pdt
% export PATH=$TAU_ROOT/bin:$PATH
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)

%
% export TAU_CALLPATH=1
% export TAU_CALLPATH_DEPTH=100

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

## ParaTools

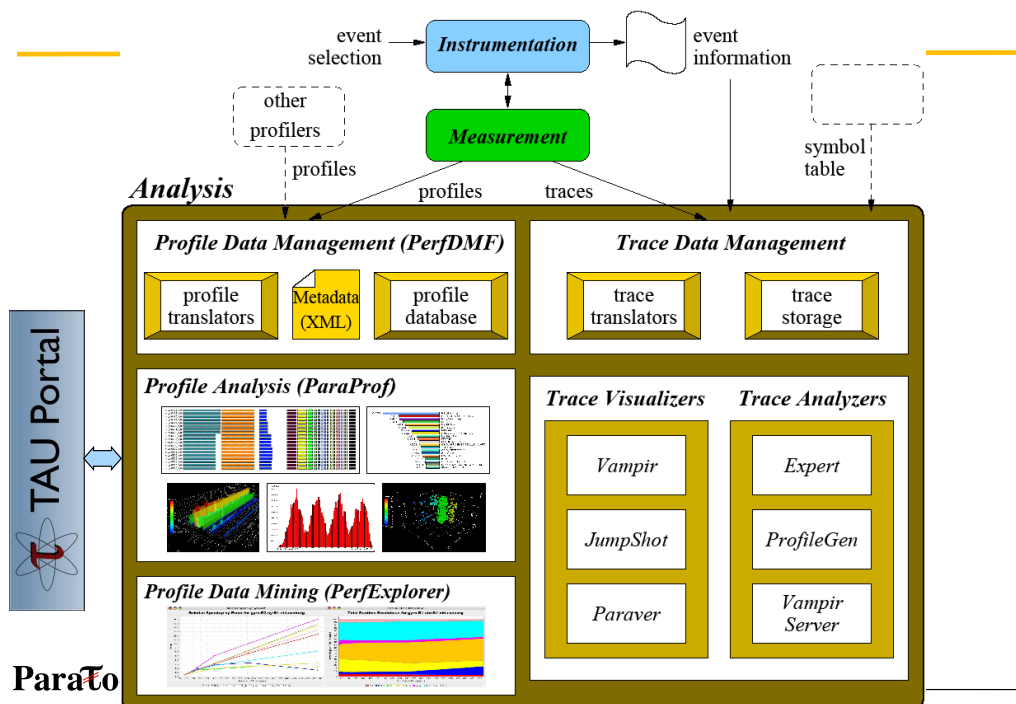
# Analyzing performance data with ParaProf, PerfExplorer



ParaTools

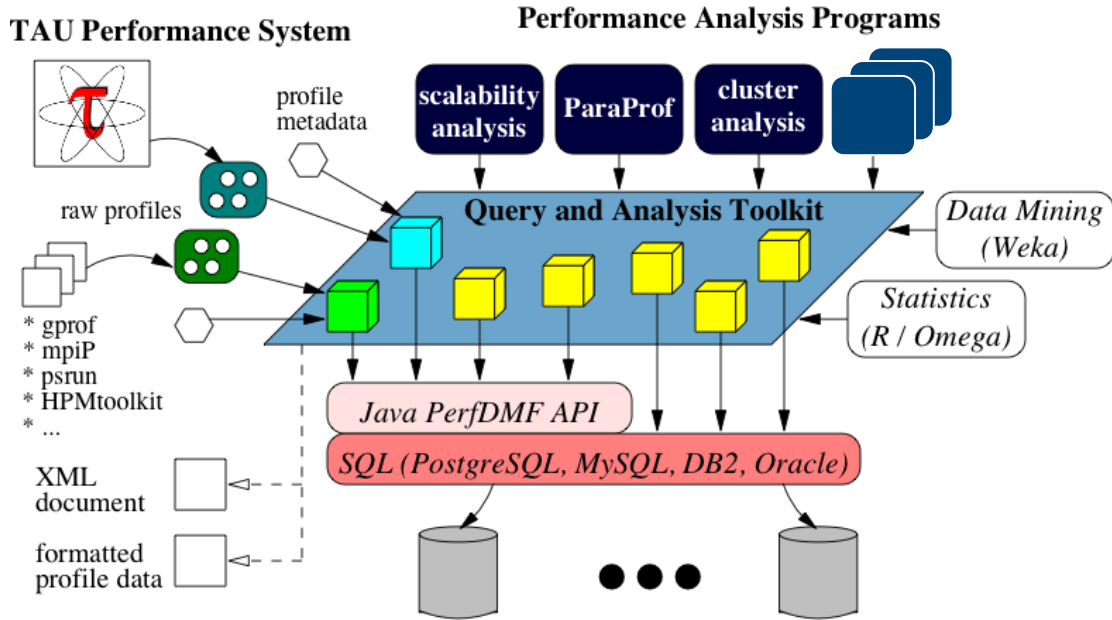
85

## TAU Performance System Architecture



ParaTools

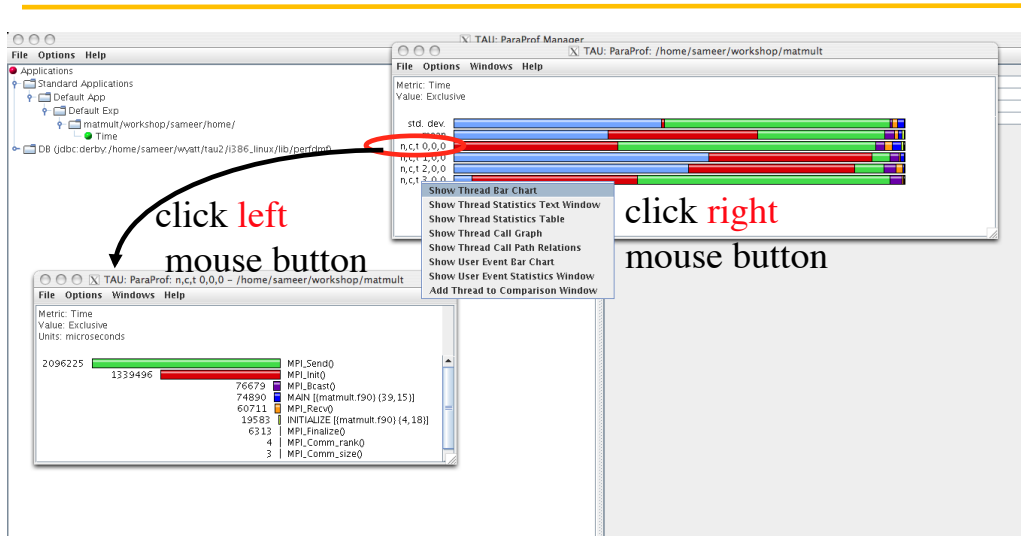
# PerfDMF: Performance Data Mgmt. Framework



## ParaTools

87

## ParaProf Main Window



% paraprof matmult.ppk

## ParaTools

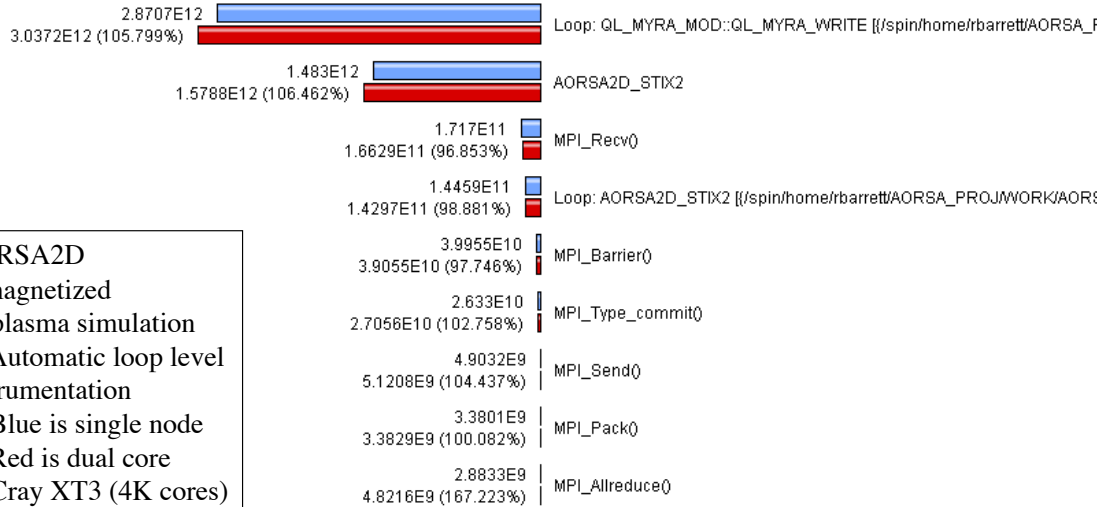
88



# Comparing Effects of Multi-Core Processors

Metric: PAPI\_RES\_STL  
 Value: Exclusive  
 Units: counts

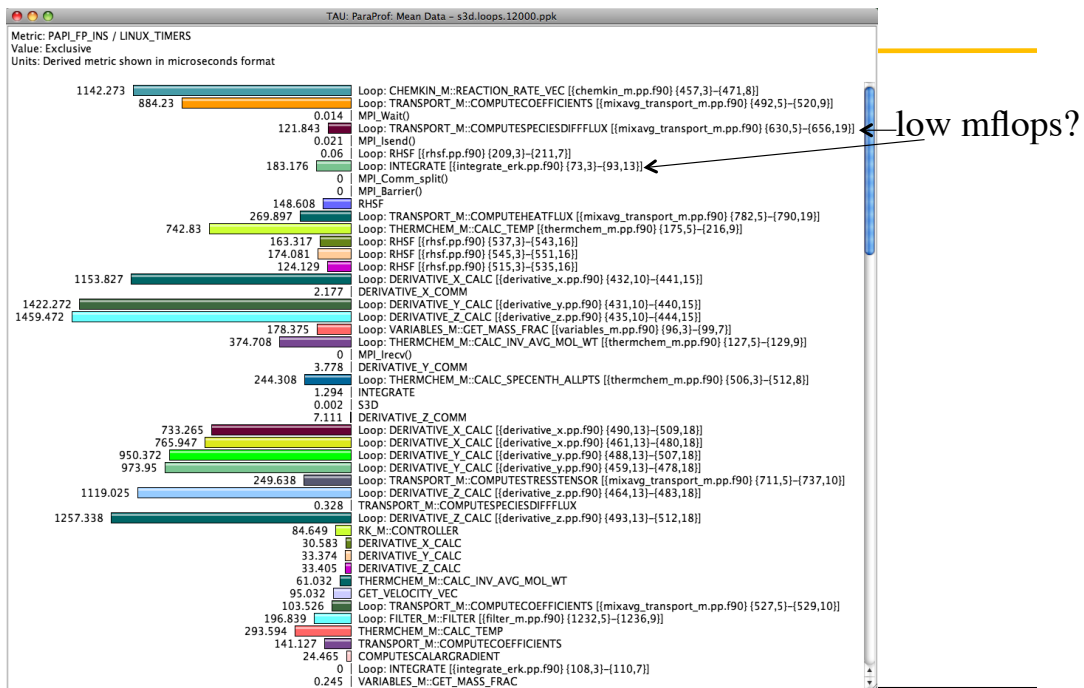
■ C:\iter.350x350.4096pes.sn.loops.BARRIER.ppk - Mean  
■ C:\iter.350x350.2048pes.dc.loops.BARRIER.ppk - Mean



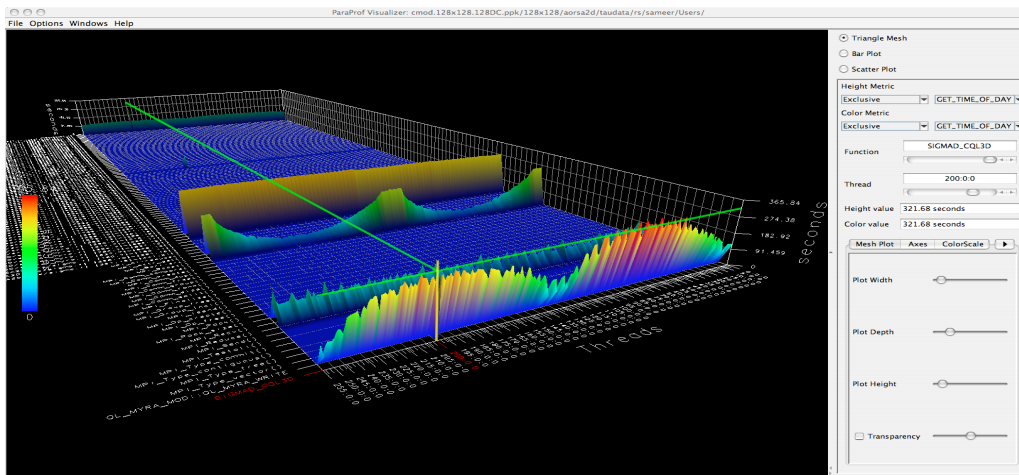
AORSA2D  
 Magnetized plasma simulation  
 Automatic loop level instrumentation  
 Blue is single node  
 Red is dual core  
 Cray XT3 (4K cores)

## Paratools

# ParaProf: Mflops Sorted by Exclusive Time

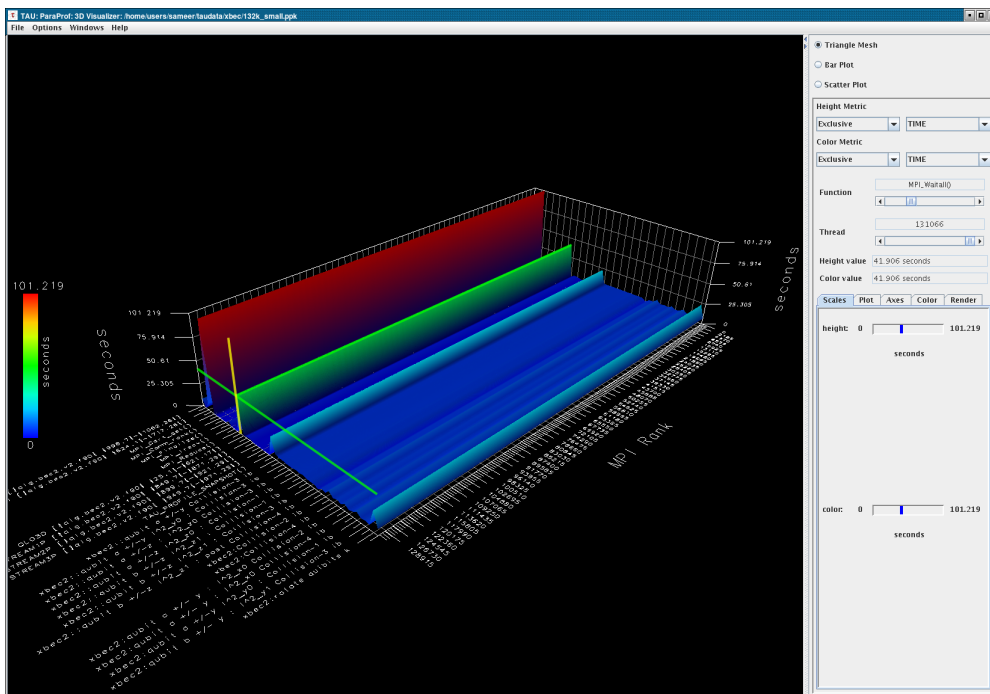


# Parallel Profile Visualization: ParaProf

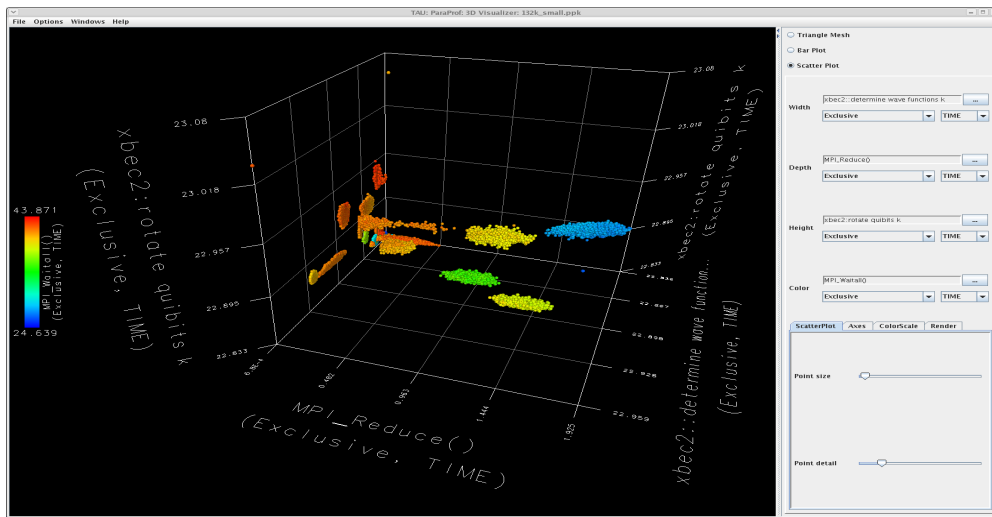


## ParaTools

# Scalable Visualization: ParaProf (128k cores)

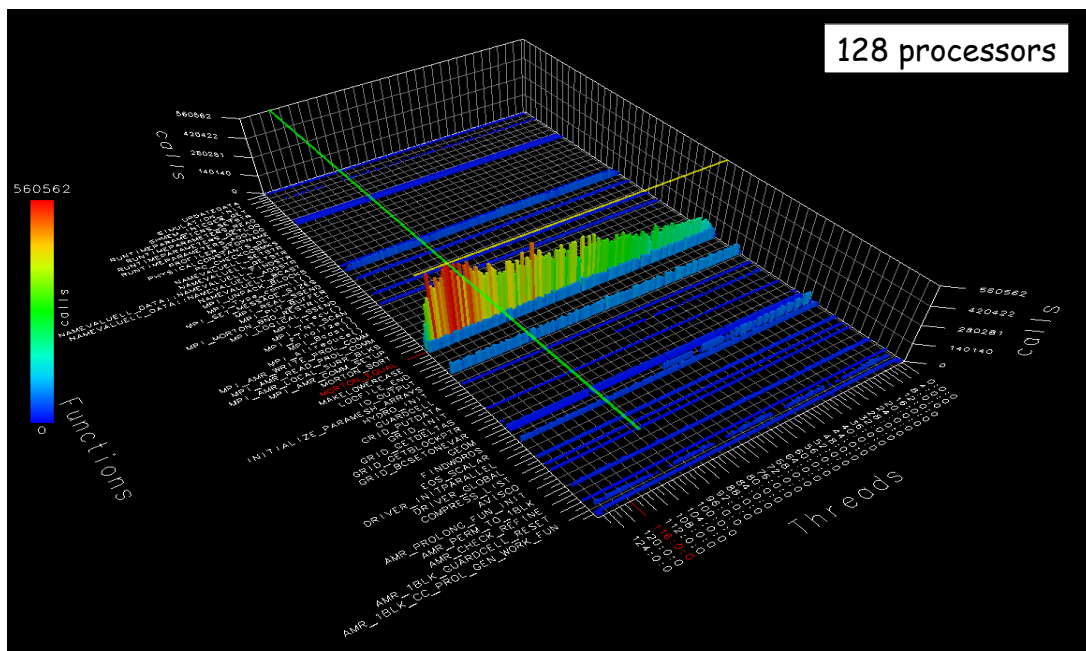


## Scatter Plot: ParaProf (128k cores)

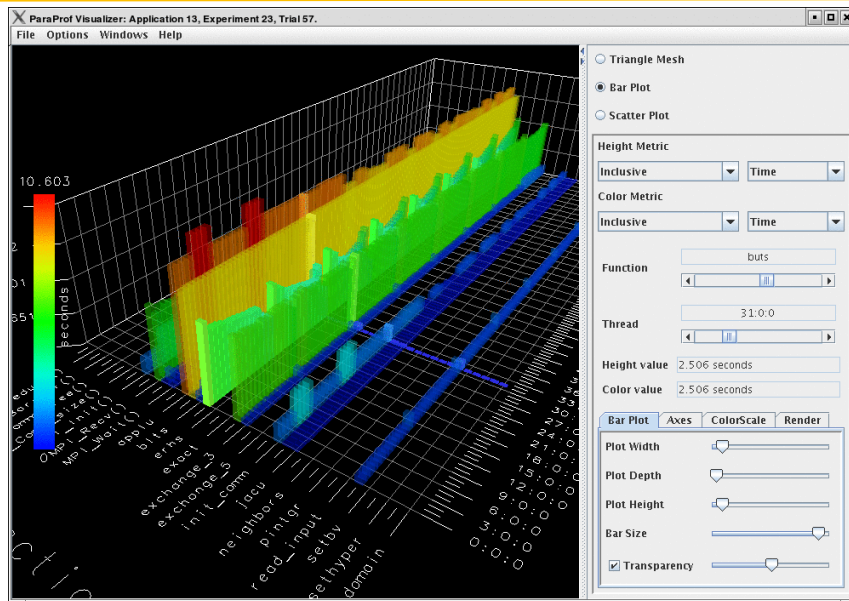


## ParaTools

## ParaProf – 3D Full Profile Bar Plot (Flash)



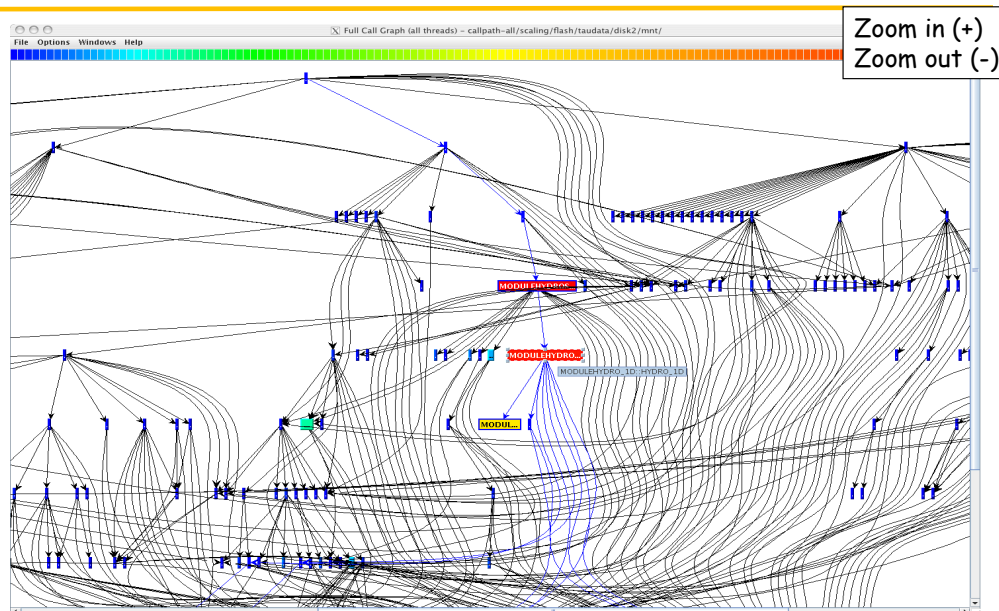
## ParaProf Bar Plot (Zoom in/out +/-)



ParaTools

95

## ParaProf – Callgraph Zoomed (Flash)



ParaTools

96

# 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_Init()	0.056	0.054	1	1
GSI SUB	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
GLB SOI	5,212.171	0.024	1	12
MPL_Finalize()	1.004	1.004	1	0
OBS_PARA	3.635	0.181	1	56
JFUNC::CREATE_JFUNC	0.142	0.142	1	0
GUESS_GRIDS::CREATE_GES_BIAS_GRIDS	0.059	0.059	1	0
READ_GUESS	1,406.412	0.023	1	8
READ_OBS	3,770.188	0.016	1	6
MPL_Allreduce()	3,725.802	3,725.802	3	0
READ_BUFRTOV5	44.369	0.254	1	871,535
SATTHIN::MAKEGVALS	0	0	1	0
W3FS21	0	0	1	1
BINARY_FILE_UTILITY::OPEN_BINARY_FILE	0.025	0.012	1	3
INITIALIZE::INITIALIZE_RTM	0.099	0.001	1	2
GUESS_GRIDS::CREATE_SFC_GRIDS	0	0	1	0
M_FVANAGRID::ALLGETLIST	30.582	0	1	10
ERROR_HANDLER::DISPLAY_MESSAGE	0	0	1	0
JFUNC::SET_POINTER	0	0	1	0
OZINFO::OZINFO_READ	0.016	0.016	1	0
DETER_SUBDOMAIN	0.008	0.008	1	0
GRIDMOD::CREATE_MAPPING	0.005	0.005	1	0
INIT_COMMVARS	0.004	0.004	1	0
M_FVANAGRID::ALLGETLIST	10.711	0	1	1
GRIDMOD::CREATE_GRID_VARS	0	0	1	0

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	DPRODMOD::DPRODX[66]
0.33	0.33	104/430	INITALLMOD::INITALL[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	SETUPRESALL[1900]
20.069	20.069	208/430	STPCALCMOD::STPCALC[1942]
--> 3964.18	3964.18	430	MPI_Allreduce() [1762]
2.6E-4	30.582	1/15	GLB SOI[93]
0.007	0.036	1/15	GR1[107]
2.7E-4	10.111	1/15	GR1UB[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	1575.595	15	M_FVANAGRID::ALLGETLIST [1773]
6.2E-5	6.2E-5	1/1	BALMOD::CREATE_BALANCE_VARS[7]
4.6E-5	4.6E-5	1/1	BALMOD::DESTROY_BALANCE_VARS[8]
3.494	3.494	1/1	BALMOD::PREBAL[9]
0.017	0.017	1/1	BERROR::CREATE_BERROR_VARS[11]
2.0E-4	2.0E-4	1/1	BERROR::DESTROY_BERROR_VARS[12]
8.6E-5	8.6E-5	1/1	BERROR::SET_PREDICTORS_VAR[16]
5.7E-5	5.7E-5	1/1	COMPACT_DIFFS::CREATE_CDIFP_COEFS[34]
4.9E-5	4.9E-5	1/1	COMPACT_DIFFS::DESTROY_CDIFP_COEFS[35]
0.015	0.042	1/1	COMPACT_DIFFS::HEISPR[41]
0.052	0.196	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]
9.1E-5	9.1E-5	1/1	GRIDMOD::DESTROY_MAPPING[99]
0.169	0.169	3/3	GUESS_GRIDS::CREATE_ARM_GRIDS[1692]
3.3E-4	3.3E-4	3/3	GUESS_GRIDS::DESTROY_ARM_GRIDS[1695]
2.2E-4	2.2E-4	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_RM[1705]
5.8E-5	5.8E-5	1/1	JFUNC::DESTROY_JFUNC[1739]
0.003	0.003	1/430	MPI_Allreduce() [1762]
0.017	0.017	68/116	MPI_Recv() [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 the "DB (jdbc:derby:databases/perf.mf)" folder. On the right is a table of metadata for a selected application.

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

A "Load Trial" dialog box is also visible, showing a list of trial types including "Tau profiles", "HPMToolkit", "Gprof", "PSRun", "ParaProf Packed Profile", "Cube", and "HPCToolkit".

Paratools

# Performance Database: Storage of MetaData

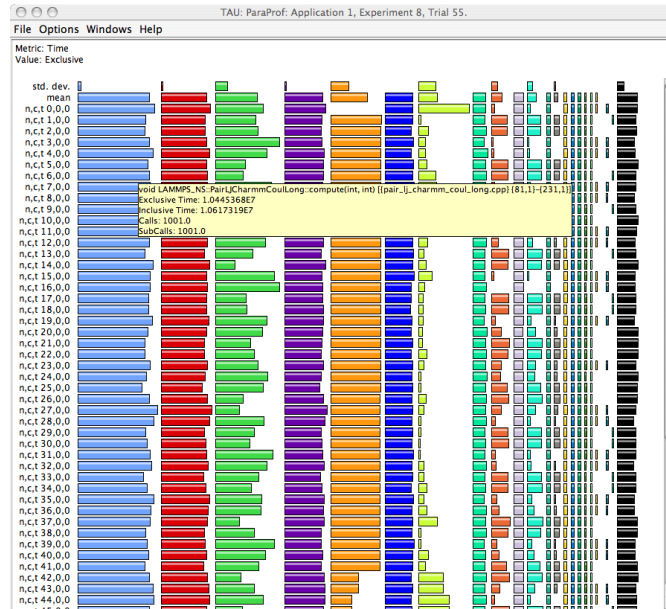
The screenshot shows the ParaProf Manager interface. On the left is a tree view of applications, with a callout box labeled "performance database" pointing to the "DB Applications" folder. On the right is a table of metadata for a selected application.

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

A "Load Trial" dialog box is also visible, showing a "Trial Type" dropdown set to "Tau profiles" and a "Select Directory" field containing "/Users/sameer/rs/taudata/s3d".

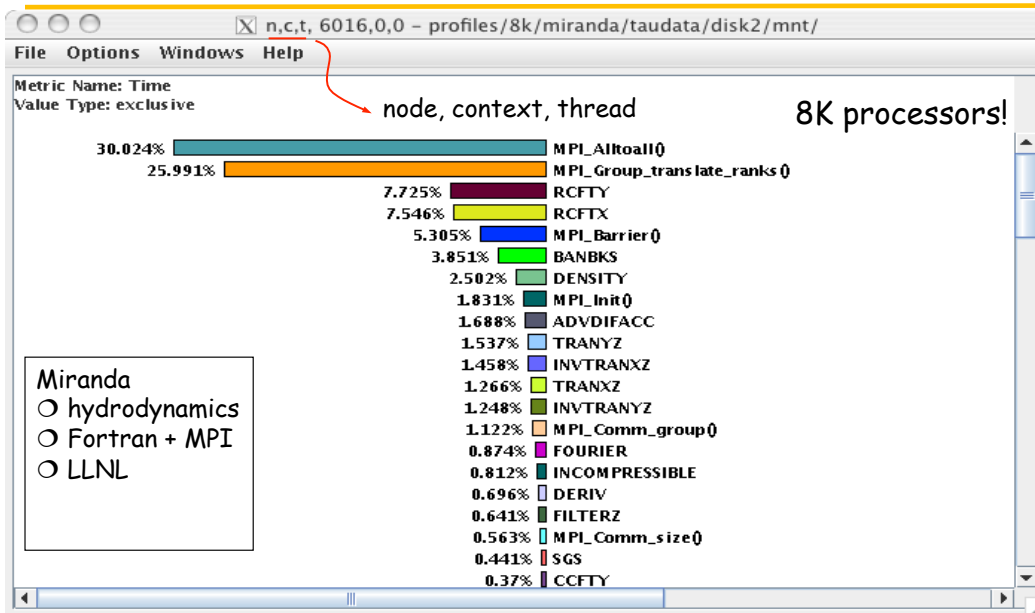
Paratools

# ParaProf Main Window (Lammps)



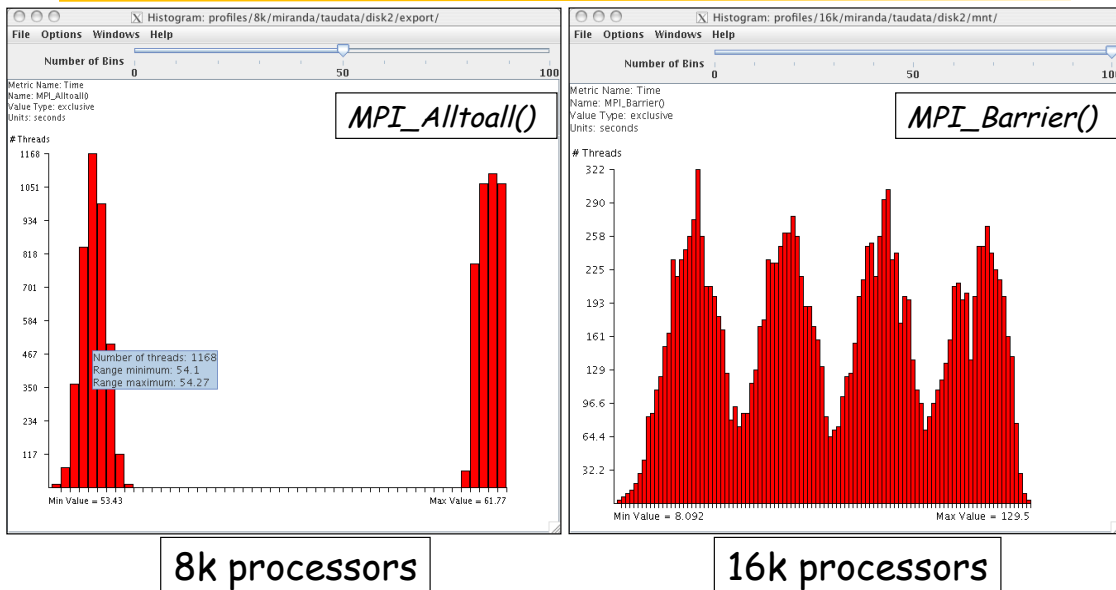
ParaTools

# ParaProf – Flat Profile (Miranda)



ParaTools

## ParaProf – Histogram View (Miranda)



ParaTools

103

## 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

ParaTools

104



## 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)

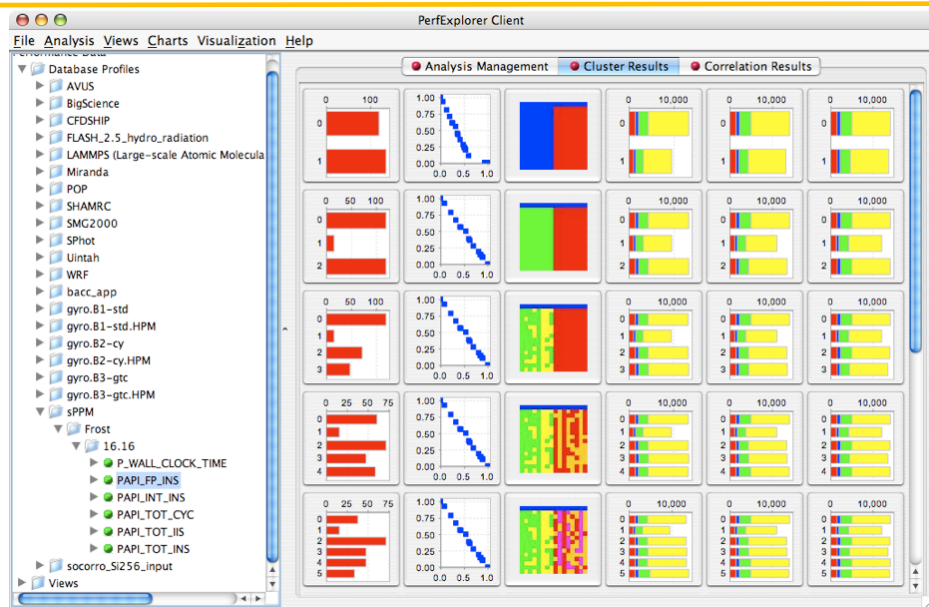
## PerfExplorer - Cluster Analysis

- 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

## ParaTools

107

## PerfExplorer - Cluster Analysis (sPPM)

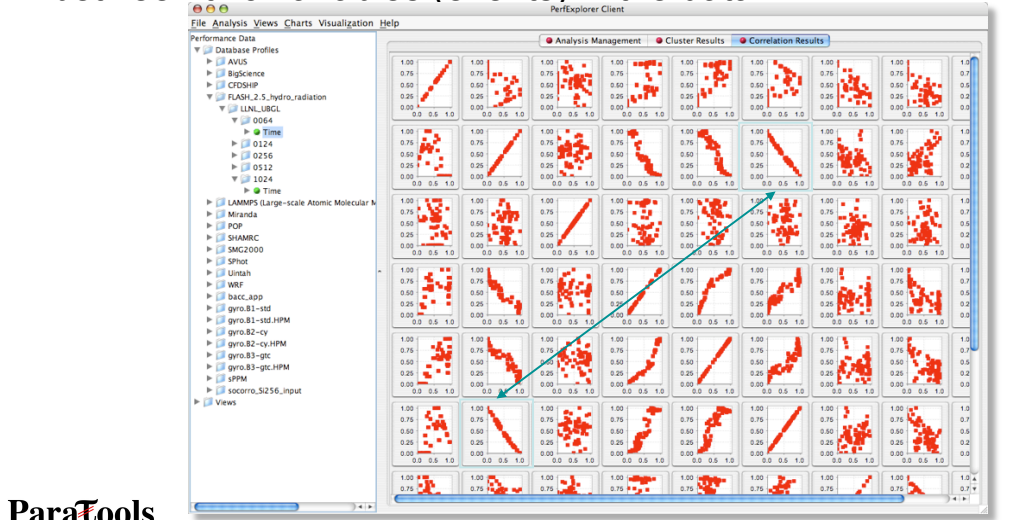


## ParaTools

108

## PerfExplorer - Correlation Analysis (Flash)

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

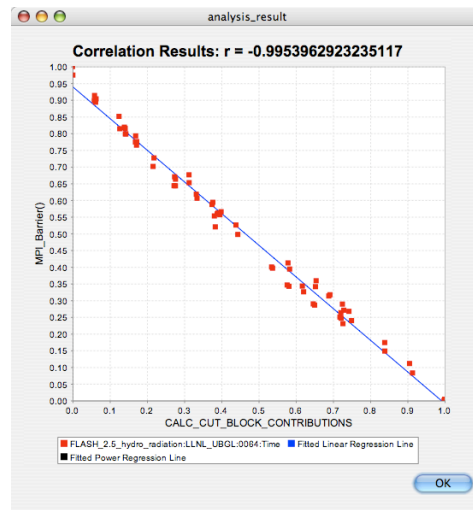


ParaTools

109

## PerfExplorer - Correlation Analysis (Flash)

- -0.995 indicates strong, negative relationship
- As CALC\_CUT\_BLOCK\_CONTRIBUTIONS increases in execution time, MPI\_Barrier() decreases



ParaTools

110

# 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

111

## PerfExplorer - Interface

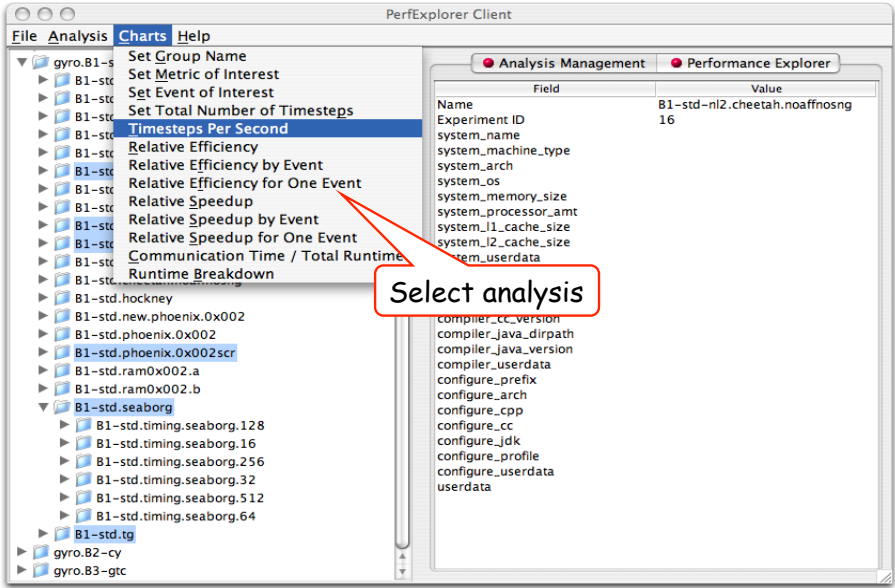
The screenshot shows the PerfExplorer Client interface. On the left is a file tree under 'gyro.B1-std' with various subfolders like 'B1-std-hwpc.phoenix.0x002', 'B1-std-inst.phoenix.0x002', etc. On the right is a table with 'Field' and 'Value' columns. The table contains fields like 'Name', 'Experiment ID', 'system\_name', etc. Red callout boxes are present: one pointing to the file tree with the text 'Select experiments and trials of interest', one pointing to the table with 'Experiment metadata', and a larger one at the bottom with 'Data organized in application, experiment, trial structure (will allow arbitrary in future)'.

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

## ParaTools

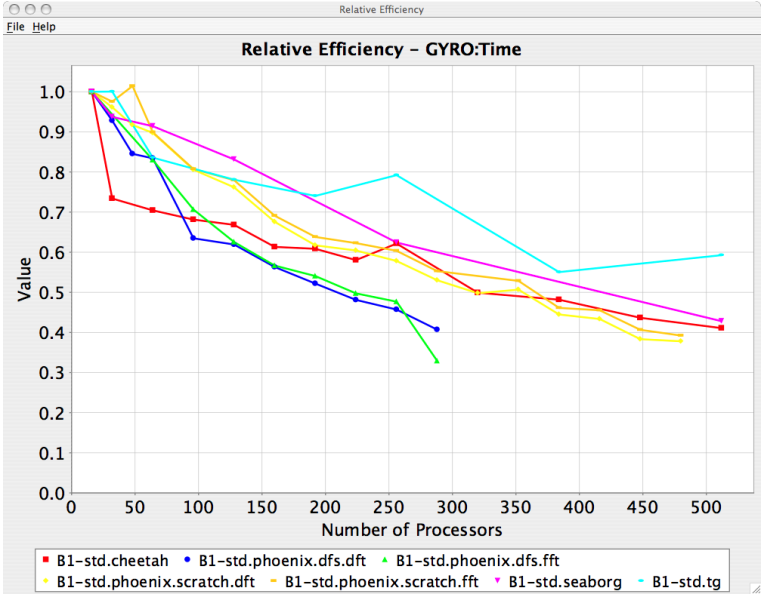
112

# PerfExplorer - Interface



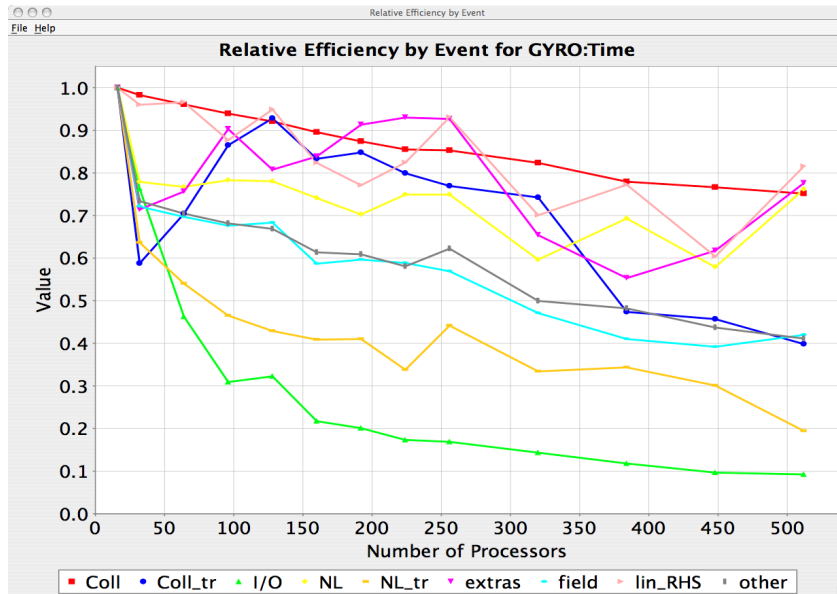
ParaTools

# PerfExplorer - Relative Efficiency Plots



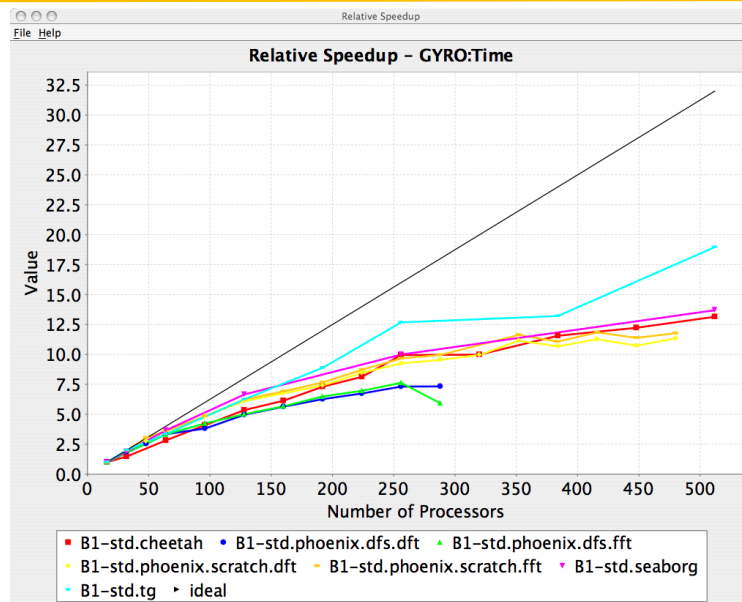
ParaTools

## PerfExplorer - Relative Efficiency by Routine



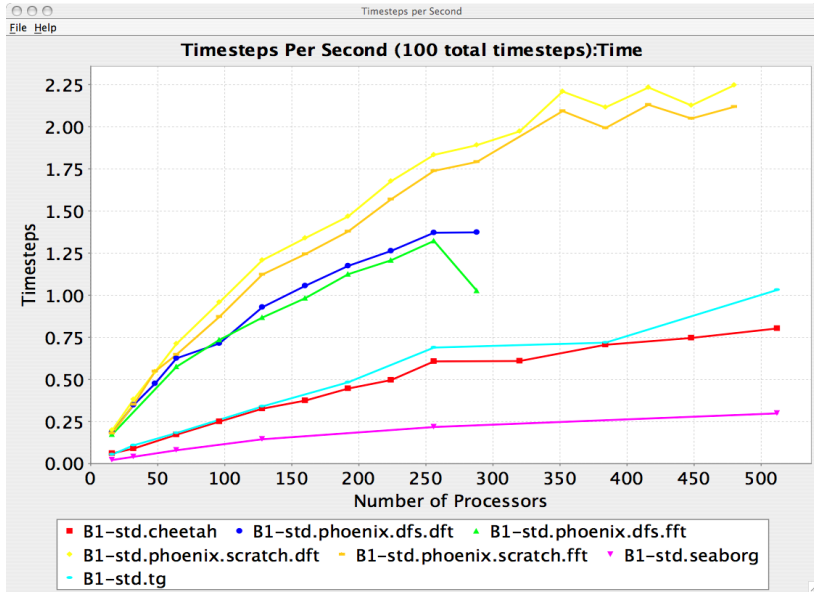
ParaTools

## PerfExplorer - Relative Speedup



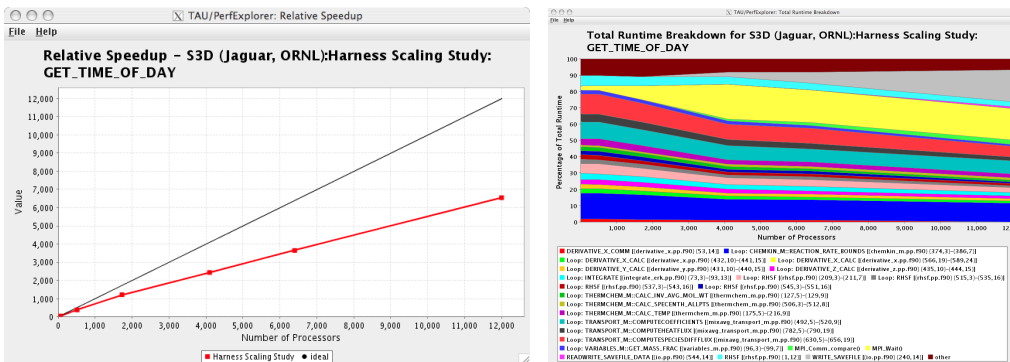
ParaTools

# PerfExplorer - Timesteps Per Second

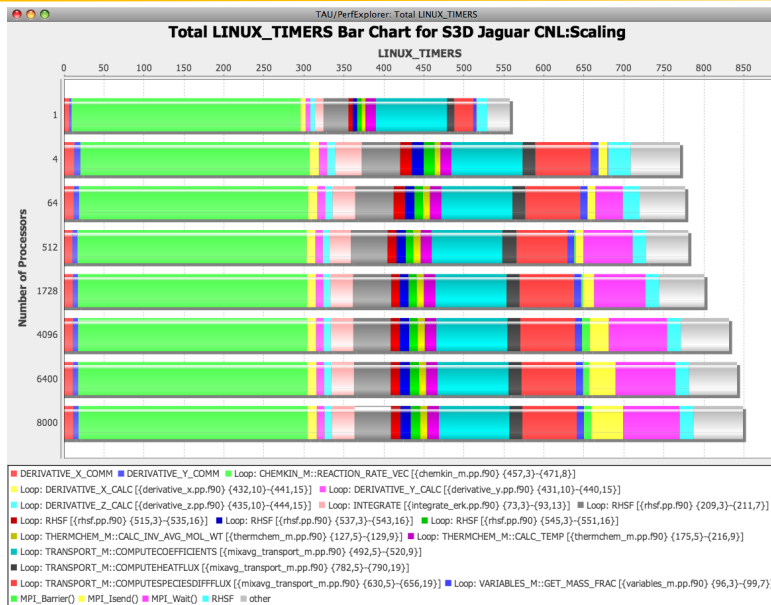


# Usage Scenarios: Evaluate Scalability

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

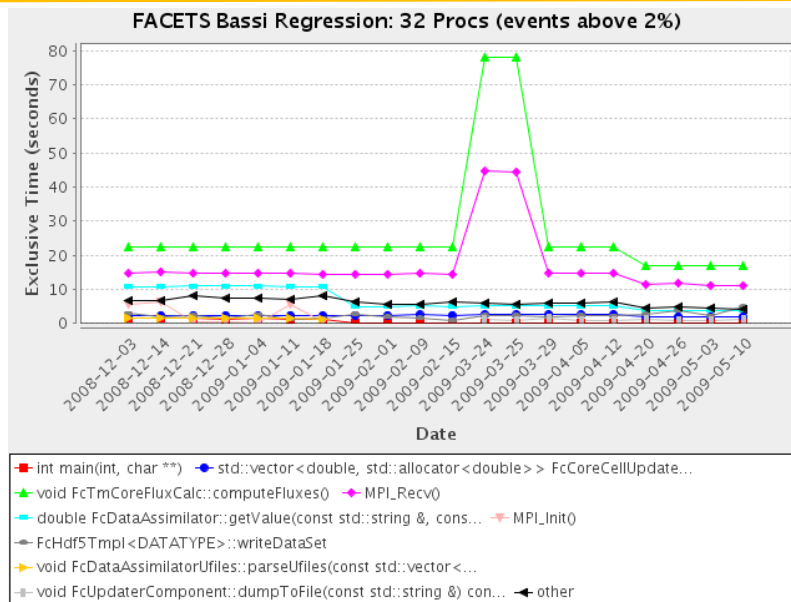


# Usage Scenarios: Evaluate Scalability



ParaTools

# Performance Regression Testing



ParaTools



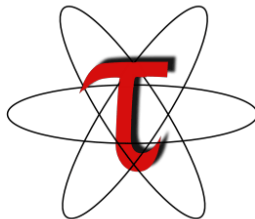
## Evaluate Scalability using PerfExplorer Charts

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

ParaTools

121

## Throttling effect of frequently called small routines



ParaTools

122

## 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

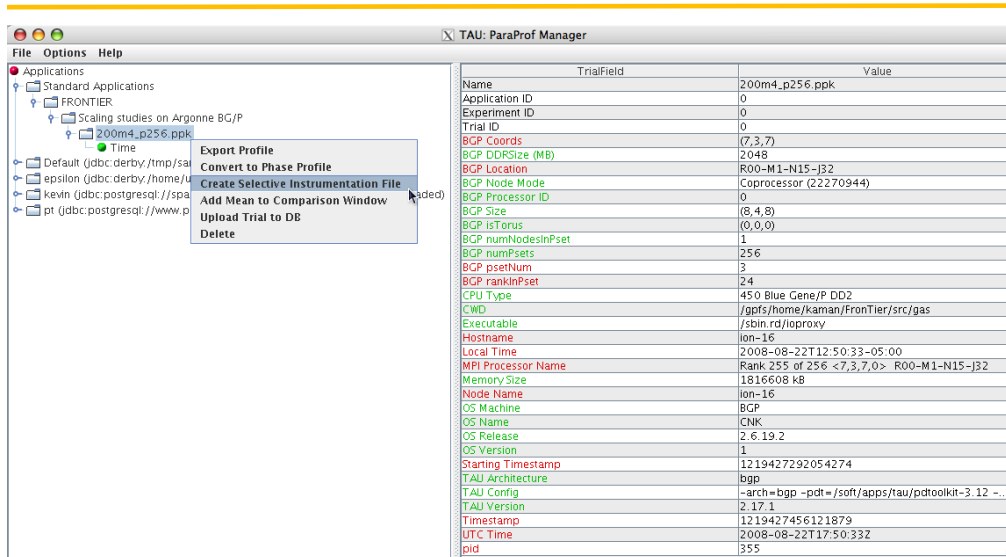
```
% tau_instrumentor [options] -f <file> OR
% export TAU_OPTIONS='-optTauSelectFile=tau.txt'
```
- Compensation of local instrumentation overhead

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

## ParaTools

123

## ParaProf: Creating Selective Instrumentation File

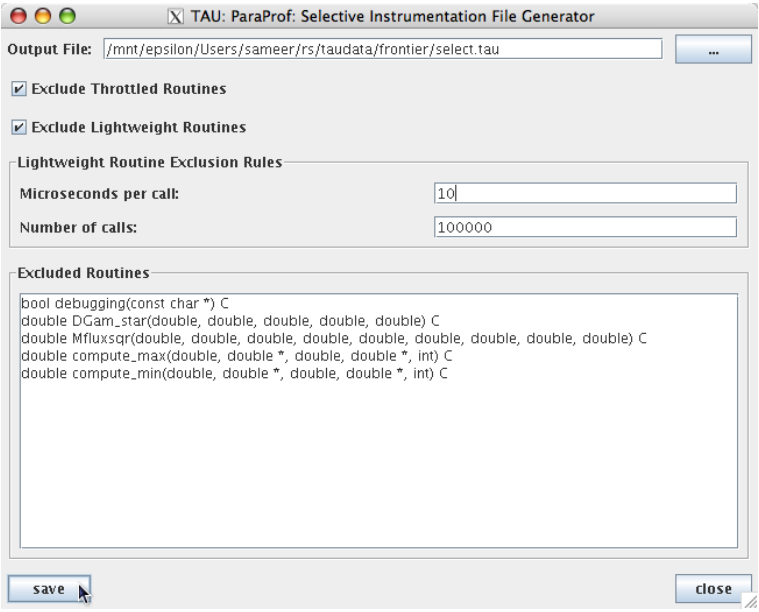


TrialField	Value
Name	200m4_p256.ppk
Application ID	0
Experiment ID	0
Trial ID	0
BCP Coords	(7,3,7)
BCP DDIRSize (MB)	2048
BCP Location	R00-M1-N15-J32
BCP Node Mode	Coprocessor (22270944)
BCP Processor ID	0
BCP Size	(8,4,8)
BCP isTorus	(0,0,0)
BCP numNodesInPset	1
BCP numPsets	256
BCP psetNum	3
BCP rankInPset	24
CPU Type	45 0 Blue Gene/P DD2
CWD	/gpfs/home/kaman/FrontTier/src/gas
Executable	/sbin.rd/ioproxy
Hostname	lon-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	lon-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

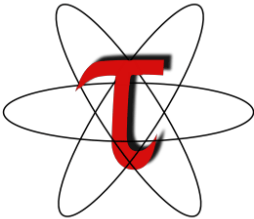
124

# Choosing Rules for Excluding Routines



ParaTools

# Observing I/O bandwidth and volume



ParaTools

## 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

## 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

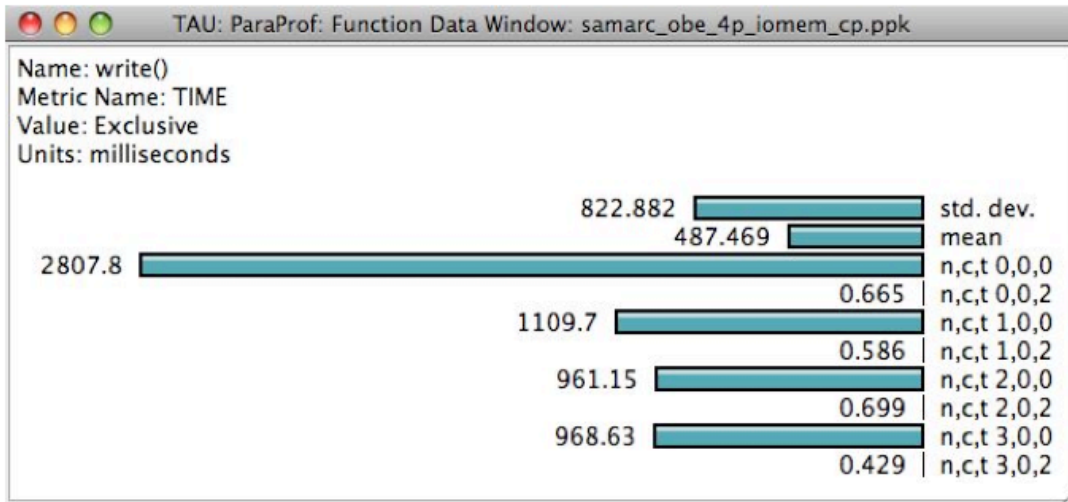
Unbuffered I/O	Buffered I/O	Communication	Control	Asynchronous I/O
open	fopen	socket	fcntl	aio_read
open64	fopen64	pipe	rewind	aio_write
close	fdopen	socketpair	lseek	aio_suspend
read	freopen	bind	lseek64	aio_cancel
write	fclose	accept	fseek	aio_return
readv	fprintf	connect	dup	lio_listio
writew	fscanf	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	535	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/gids/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/uverbs/abi_version">	8	1	8	8	8	0
Bytes Read <file="/sys/class/infiniband_verbs/uverbs/device">	24	3	8	8	8	0
Bytes Read <file="/sys/class/infiniband_verbs/uverbs/device/vendor">	24	3	8	8	8	0
Bytes Read <file="/sys/class/infiniband_verbs/uverbs/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/gids/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/uverbs/abi_version">	4	1	4	4	4	0
READ Bandwidth (MB/s) <file="/sys/class/infiniband_verbs/uverbs/device">	16	3	8	4	5.333	1.886
READ Bandwidth (MB/s) <file="/sys/class/infiniband_verbs/uverbs/device/vendor">	20	3	8	4	6.667	1.886
READ Bandwidth (MB/s) <file="/sys/class/infiniband_verbs/uverbs/ibdev">	32	1	32	32	32	0

## Time Spent in POSIX I/O write()



ParaTools

133

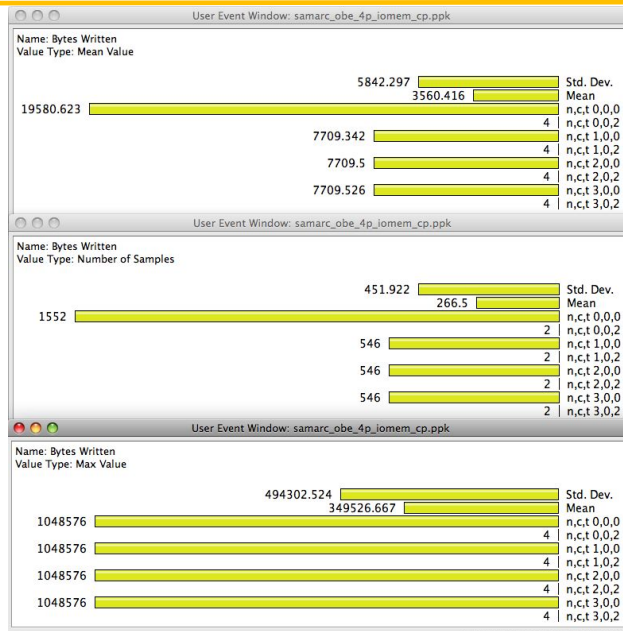
## Volume of I/O by File, Memory

Name	Total	MeanValue	NumSamples	MinValue	MaxValue	Std. Dev.
TAU application						
read()						
fopen64()						
fclose()						
OurMain()						
malloc size	25,235	1,097.174	23	11	12,032	2,851.143
free size	22,707	1,746.692	13	11	12,032	3,660.642
OurMain [(wrapper.py)[3]]						
read()						
malloc size	3,877	323.083	12	32	981	252.72
free size	1,536	219.429	7	32	464	148.122
fopen64()						
fclose()						
<module> [(obe.py)[8]]						
samarcWriteRestartData [(samarcInterface.py)[145]]						
write()						
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
open64()						

ParaTools

134

# Bytes Written



ParaTools

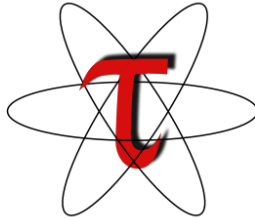
# Memory Leaks in MPI

Name	Total	MeanValue	NumSamples	MaxValue	MinValue	Std. Dev.
TAU application						
MPI_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
MPI_Comm_free()						
free size	10,952	195.571	56	1,024	48	255.353
read()						
MPI_Type_free()						
MPI_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
OurMain0						
fclose()						



---

## PAPI hardware counters



## 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 micro-processors.
- Countable events are defined in two ways:
  - Platform-neutral *preset* events
  - Platform-dependent native events
- Presets can be **derived** from multiple *native events*
- All events are referenced by name and collected in EventSets

## ParaTools

139

## 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 (Notes)
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|
-----
```

## ParaTools

## 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|
```

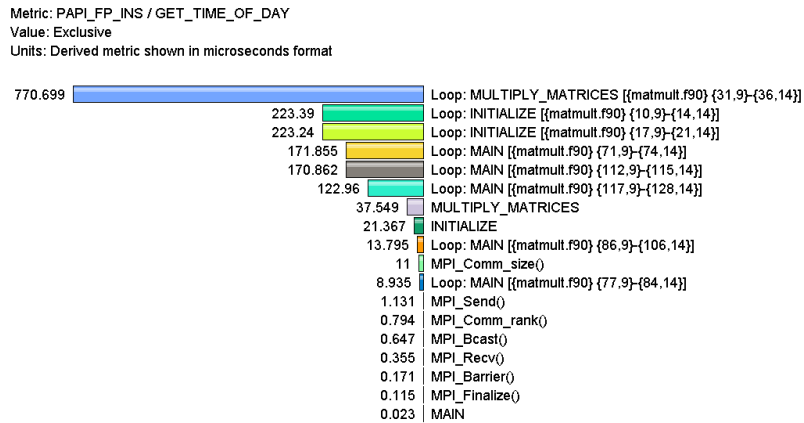
```
0x40008028 :NO_OTHER |Bus cycles when core is active and the other is halted|
-----
```

```
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:



Par

149

## Generate a PAPI profile with 2 or more counters

```
% export TAU_MAKEFILE=$TAU_ROOT/lib/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
% srun -n 4 -p specops ./a.out
% paraprof --pack app.ppk
Move the app.ppk file to your desktop.
% paraprof app.ppk
Choose Options -> Show Derived Panel -> "PAPI_FP_INS", click "/", "TIME", click "Apply"
choose.
```

Paratools

150

# Derived Metrics in ParaProf

The screenshot shows the TAU: ParaProf Manager interface. It features a sidebar with a file tree and a main window displaying a table of system and application parameters. Below the table is an expression input field and an 'Apply' button.

TrialField	Value
Trial ID	0
CPU Cores	6
CPU MHz	2600.093
CPU Type	Six-Core AMD Opteron(tm) Processor 8435
CPU Vendor	AuthenticAMD
CWD	/root/#90
Cache Size	512 KB
Executable	/root/#90/ring.1
File Type Index	1
File Type Name	Tau profiles
Hostname	b2e455.corning.com
Local Time	2010-04-18T02:27:58-04:00
NPI Processor Name	b2e455.corning.com
Memory Size	66006592 KB
Node Name	b2e455.corning.com
OS Machine	x86_64
OS Name	Linux
OS Release	2.6.18-128.el5.perfctr
OS Version	#1 SMP Mon Jun 29 12:32:22 PDT 2009
Starting Timestamp	1271572078767980
TAU Architecture	x86_64
TAU Config	-c++-g++ -cc=gcc -fortran=gfortran -mpilib=/usr/nic/dev/intel/mpi/3.2.2.006/lib64 -mpilnc=/usr/nic/dev/intel/mpi/3.2.2.006/in...
TAU Makefile	/usr/corning/apps/paratools/tau-2.19.1/x86_64/lib/Makefile.tau-intelmpi-papi-mpi-pdt
TAU Version	tau-2.19.1
TAU_CALLPATH	off
TAU_CALLPATH_DEPTH	2
TAU_COMM_MATRIX	off
TAU_COMPENSATE	off
TAU_PROFILE	on
TAU_PROFILE_FORMAT	profile
TAU_THROTTLE	on
TAU_THROTTLE_NUMCALLS	100000
TAU_THROTTLE_PERCALL	10
TAU_TRACE	off
TAU_TRACK_HEADROOM	off
TAU_TRACK_HEAP	off
TAU_TRACK_MESSAGE	off
Timestamp	1271572078835079

Expression: "PAPL\_FP\_INS"/"PAPL\_L1\_DCM" Clear

## Paratools

151

# ParaProf's Source Browser: Loop Level Instrumentation

The screenshot displays three function data windows and a source code browser. The function data windows show performance metrics for different loops, and the source code browser shows the corresponding code with line numbers.

**Function Data Window 1:** Loop: TRANSPORT\_Mc:COMPUTESPECIESDIFFFLUX. Metric Name: PAPL\_FP\_INS / GET\_TIME\_OF\_DAY. Value: Exclusive. Units: Derived metric shown in microseconds format.

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

**Function Data Window 2:** Loop: TRANSPORT\_Mc:COMPUTESPECIESDIFFFLUX. Metric Name: GET\_TIME\_OF\_DAY. Value: Exclusive percent.

Value	Mean	std. dev.
12.206%	0.91%	
11.931%	n.ct 0,0,0	
12.19%	n.ct 1,0,0	
12.248%	n.ct 2,0,0	
12.258%	n.ct 3,0,0	
12.339%	n.ct 4,0,0	
12.241%	n.ct 5,0,0	
12.221%	n.ct 6,0,0	
12.226%	n.ct 7,0,0	

**Function Data Window 3:** Loop: TRANSPORT\_Mc:COMPUTESPECIESDIFFFLUX. Metric Name: PAPL\_L1\_DCM. Value: Exclusive. Units: counts.

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

**Source Code Browser:** /mnt/epsilon/Users/sameer/rs/taudata/s3d/harness/flat/papi8

```

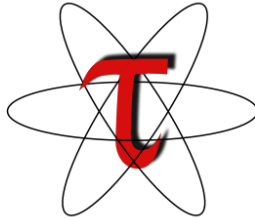
606 grad_nixNW(:, :, :n) = grad_nixNW(:, :, :n)*avolwt(:, :, :);
607 end do
608
609 ! compute grad_P
610 if (baro_switch) then
611   allocate(grad_P(nv,ny,nz,3))
612   grad_P = 0.0
613   !f (vary_in_x == 1) then
614     call derivative_x(nv,ny,nz, Press, grad_P(:, :, :, 1), scale_ix, 1)
615   end if
616   !f (vary_in_y == 1) then
617     call derivative_y(nv,ny,nz, Press, grad_P(:, :, :, 2), scale_jy, 1)
618   end if
619   !f (vary_in_z == 1) then
620     call derivative_z(nv,ny,nz, Press, grad_P(:, :, :, 3), scale_jz, 1)
621   end if
622
623 ! Changed by Rananan - 01/24/05
624 ! ds_nixavg is now vherb
625 !
626 !grad_P/press and avolwt*grad_T/Temp can be optimized by division before the loop.
627 ! compute diffusive flux for species n in direction n.
628 diffFlux(:, :, :n, spec_n) = 0.0
629 DIRECTION: do m=1,3
630   SPECIES: do n=1,n_spec-1
631
632     !f (baro_switch) then
633       ! driving force includes gradient in mole fraction and baro-diffusion:
634       diffFlux(:, :, :n, n) = - ds_nixavg(:, :, :n) * ( grad_Ys(:, :, :n, n) &
635         + Ys(:, :, :n) * ( grad_nixNW(:, :, :n) &
636           + (1 - molwt(n)/Paromol) * grad_P(:, :, :n, Press) ) )
637     else
638       ! driving force is just the gradient in mole fraction:
639       diffFlux(:, :, :n, n) = - ds_nixavg(:, :, :n) * ( grad_Ys(:, :, :n, n) &
640         + Ys(:, :, :n) * grad_nixNW(:, :, :n) )
641     end if
642
643     ! Add thermal diffusion:
644     !f (thermoDiffSwitch) then
645       diffFlux(:, :, :n, n) = diffFlux(:, :, :n, n) &
646         - ds_nixavg(:, :, :n) * Rs_therm_diff(:, :, :n) * molwt(n) &
647         + avolwt * grad_T(:, :, :n) / Temp
648     end if
649
650     ! compute contribution to nth species diffusive flux
651     ! this will ensure that the sum of the diffusive fluxes is zero:
652     diffFlux(:, :, :n, spec_n) = diffFlux(:, :, :n, spec_n) - diffFlux(:, :, :n, n)
653
654   enddo SPECIES
655 enddo DIRECTION
656
657 if (baro_switch) then
658   deallocate(grad_P)
659 end if
660
661 return
662 end subroutine computeSpeciesDiffFlux
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```



---

## 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...

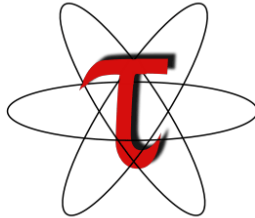
Total cost for PAPI_read(2 counters) over 1000000 iterations
min cycles   : 288
max cycles   : 102429
mean cycles  : 301.000000
std deviation: 144.694053
cost.c                               PASSED
```

## PAPI Utilities: *papi\_cost*

```
Cost distribution profile
63:***** 999969 counts *****
153:
243:
[...]
1863:
1953:*****
2043:
2133:*****
2223:
2313:
2403:*****
2493:*****
2583:*****
2673:*****
2763:*****
2853:*****
2943:
3033:*****
3123:*****
3213:*****
3303:
3393:
3483:
3573:
3663:*****
```

---

## Hands-on training with sample codes



ParaTools

---

157

## Labs!

---



Lab: PAPI, and TAU

ParaTools

---

158

## Lab Instructions (for OCF systems)

---

Get workshop.tar.gz using:

```
% wget
  http://www.paratools.com/sandia11/workshop.tar.gz
```

Or

```
% cp /projects/tau/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.

---

## ParaTools

159

## Lab Instructions

---

To profile a code using TAU:

1. Change the compiler name to tau\_cxx.sh, tau\_f90.sh, tau\_cc.sh:  
F90 = tau\_f90.sh
2. Choose TAU stub makefile  
% source /projects/tau/tau.bashrc [ or .cshrc]  
% export TAU\_MAKEFILE=  
\$TAU\_ROOT/lib/Makefile.tau-[options]
3. If stub makefile has -papi in its name, set the TAU\_METRICS environment variable:  
% export  
TAU\_METRICS=TIME:PAPI\_L2\_DCM:PAPI\_TOT\_CYC...
4. Run: srun -n 4 -p specops ./a.out
5. Build and run workshop examples, then run pprof/  
paraprof

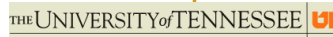
---

## ParaTools

160

# Support Acknowledgements

- Department of Energy (DOE)
  - Office of Science 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