HTTP://TAU.UOREGON.EDU

TAU PERFORMANCE ANALYSIS

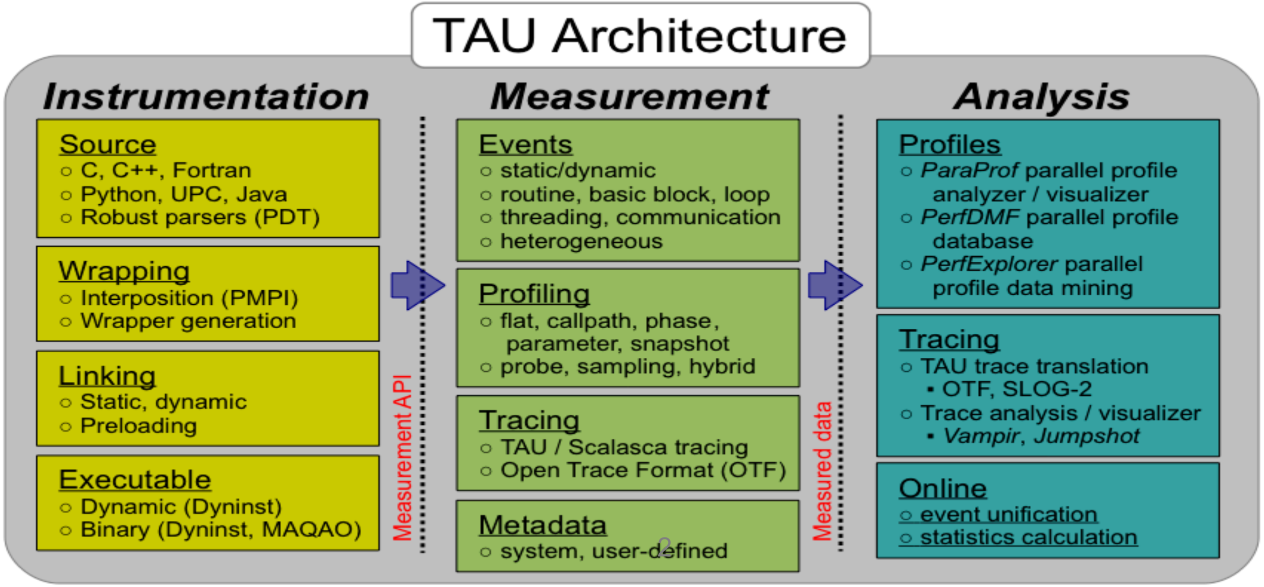
SAMEER SHENDE
Director, Performance Research Lab,
University of Oregon
ParaTools, Inc.
sameer@cs.uoregon.edu

Tuning and Analysis Utilities

5/3/2017 10am – 10:45am. ALCF, Building 240 conference room, ANL

TAU PERFORMANCE SYSTEM®

- Parallel performance framework and toolkit
  - Supports all HPC platforms, compilers, runtime system
  - Provides portable instrumentation, measurement, analysis



TAU Architecture

Instrumentation

Source
- C, C++, Fortran
- Python, UPC, Java
- Robust parsers (PDT)

Wrapping
- Interposition (PMPI)
- Wrapper generation

Linking
- Static, dynamic
- Preloading

Executable
- Dynamic (Dyninst)
- Binary (Dyninst, MAQAO)

Measurement API

Measurement

Events
- static/dynamic
- routine, basic block, loop
- threading, communication
- heterogeneous

Profiling
- flat, callpath, phase, parameter, snapshot
- probe, sampling, hybrid

Tracing
- TAU / Scalasca tracing
- Open Trace Format (OTF)

Metadata
- system, user-defined

Measured data

Analysis

Profiles
- *ParaProf* parallel profile analyzer / visualizer
- *PerfDMF* parallel profile database
- *PerfExplorer* parallel profile data mining

Tracing
- TAU trace translation
  - OTF, SLOG-2
- Trace analysis / visualizer
  - *Vampir*, *Jumpshot*

Online
- event unification
- statistics calculation

TAU PERFORMANCE SYSTEM

- Instrumentation
  - Fortran, C++, C, UPC, Java, Python, Chapel, Spark
  - Automatic instrumentation
- Measurement and analysis support
  - MPI, OpenSHMEM, ARMCI, PGAS, DMAPP
  - pthreads, OpenMP, OMPT interface, hybrid, other thread models
  - GPU, CUDA, OpenCL, OpenACC
  - Parallel profiling and tracing
  - Use of Score-P for native OTF2 and CUBEX generation
- Analysis
  - Parallel profile analysis (ParaProf), data mining (PerfExplorer)
  - Performance database technology (TAUdb)
  - 3D profile browser

APPLICATION PERFORMANCE ENGINEERING USING TAU

- How much time is spent in each application routine and outer *loops*? Within loops, what is the contribution of each *statement*? What is the time spent in OpenMP loops?
- How many instructions are executed in these code regions?
  Floating point, Level 1 and 2 *data cache misses*, hits, branches taken? What is the extent of vectorization for loops on Intel MIC?
- What is the memory usage of the code? When and where is memory allocated/de-allocated? Are there any memory leaks? What is the memory footprint of the application? What is the memory high water mark?
- How much energy does the application use in Joules? What is the peak power usage?
- What are the I/O characteristics of the code? What is the peak read and write *bandwidth* of individual calls, total volume?
- What is the contribution of each *phase* of the program? What is the time wasted/spent waiting for collectives, and I/O operations in Initialization, Computation, I/O phases?
- How does the application *scale*? What is the efficiency, runtime breakdown of performance across different core counts?

INSTRUMENTATION

Add hooks in the code to perform measurements

- **Source instrumentation using a preprocessor**
  - Add timer start/stop calls in a copy of the source code.
  - Use Program Database Toolkit (PDT) for parsing source code.
  - Requires recompiling the code using TAU shell scripts (tau_cc.sh, tau_f90.sh)
  - Selective instrumentation (filter file) can reduce runtime overhead and narrow instrumentation focus.
- **Compiler-based instrumentation**
  - Use system compiler to add a special flag to insert hooks at routine entry/exit.
  - Requires recompiling using TAU compiler scripts (tau_cc.sh, tau_f90.sh…)
- **Runtime preloading of TAU's Dynamic Shared Object (DSO)**
  - No need to recompile code! Use **aprun tau_exec ./app** with options.
  - Requires dynamic executable (link using –**dynamic** on Theta).

SIMPLIFYING TAU’S USAGE (TAU_EXEC)

- Uninstrumented execution
  - % mpirun -np 64 ./a.out
- Track MPI performance
  - % mpirun -np 64 tau_exec ./a.out
- Use event based sampling (compile with –g)
  - % mpirun –np 64 tau_exec –ebs ./a.out
  - Also –ebs_source=<PAPI_COUNTER> -ebs_period=<overflow_count>
- Track POSIX I/O and MPI performance (MPI enabled by default)
  - % mpirun -np 64 tau_exec –T mpi,pdt,papi –io ./a.out
- Track OpenMP runtime routines
  - % mpirun −np 64 tau_exec −T ompt,pdt,mpi −ompt ./a.out
- Track memory operations
  - % export TAU_TRACK_MEMORY_LEAKS=1
  - % mpirun –np 64 tau_exec –memory_debug ./a.out (bounds check)
- Load wrapper interposition library
  - % mpirun –np 64 tau_exec –loadlib=<path/libwrapper.so> ./a.out

RUNTIME PRELOADING

- Injects TAU DSO in the executing application
- Requires dynamic executables
- We must compile with −dynamic −g
- Use tau_exec while launching the application

HANDS-ON

NPB 3.3 MZ

- Setup preferred program environment compilers
  - Default set Intel Compilers with Intel MPI. You must compile with −**dynamic -g**

```
% mkdir /lus/theta-fs0/projects/Comp_Perf_Workshop/$USER
% cd !$; tar zxf /soft/perftools/tau/workshop.tgz
% module load tau
% cd MZ-NPB3.3-MPI; cat README
% make clean
% make suite
% cd bin
In a second window:
% qsub -I -n 1 -A Comp_Perf_Workshop  -t 50 -q cache-quad
% cd bin; module load tau
% export OMP_NUM_THREADS=4
% aprun -n 16 ./bt-mz.B.16
% aprun -n 16 tau_exec -T ompt,mpi,pdt -ompt -ebs ./bt-mz.B.16
% paraprof --pack ex1.ppk
In the first window:
% paraprof ex1.ppk &
```

NPB-MZ-MPI SUITE

- The NAS Parallel Benchmark suite (MPI+OpenMP version)
  - Available from:
    http://www.nas.nasa.gov/Software/NPB
  - 3 benchmarks in Fortran77
  - Configurable for various sizes & classes

```
% ls
bin/     common/   jobscript/   Makefile   README.install    SP-MZ/
BT-MZ/   config/   LU-MZ/       README     README.tutorial   sys/
```

- Subdirectories contain source code for each benchmark
  - plus additional configuration and common code
- The provided distribution has already been configured for the tutorial, such that it's ready to “make” one or more of the benchmarks and install them into a (tool-specific) “bin” subdirectory

NPB-MZ-MPI / BT (BLOCK TRIDIAGONAL SOLVER)

- What does it do?
  - Solves a discretized version of the unsteady, compressible Navier-Stokes equations in three spatial dimensions
  - Performs 200 time-steps on a regular 3-dimensional grid
- Implemented in 20 or so Fortran77 source modules
- Uses MPI & OpenMP in combination
  - 16 processes each with 4 threads should be reasonable
  - bt-mz.B.16 should take around 1 minute

NPB-MZ-MPI / BT: CONFIG/MAKE.DEF

```
#                SITE- AND/OR PLATFORM-SPECIFIC DEFINITIONS.
#
#---------------------------------------------------------------------------

#---------------------------------------------------------------------------
# Configured for generic MPI with GCC compiler
#---------------------------------------------------------------------------
#OPENMP  = -fopenmp      # GCC compiler
OPENMP = -qopenmp –extend-source        # Intel compiler

...
#---------------------------------------------------------------------
# The Fortran compiler used for MPI programs
#---------------------------------------------------------------------

F77 = ftn # Intel compiler

# Alternative variant to perform instrumentation

...
```

Default (no instrumentation)

BUILDING AN NPB-MZ-MPI BENCHMARK

```
% make
  ===========================================
  =      NAS PARALLEL BENCHMARKS 3.3        =
  =      MPI+OpenMP Multi-Zone Versions     =
  =      F77                                =
  ===========================================

  To make a NAS multi-zone benchmark type

         make <benchmark-name> CLASS=<class> NPROCS=<nprocs>

  where <benchmark-name> is “bt-mz”, “lu-mz”, or “sp-mz”
        <class>          is “S”, “W”, “A” through “F”
        <nprocs>         is number of processes

  [...]

 ***************************************************************
 * Custom build configuration is specified in config/make.def  *
 * Suggested tutorial exercise configuration for HPC systems:  *
 *       make bt-mz CLASS=C NPROCS=8                           *
 ***************************************************************
```

- Type “make” for instructions
- make suite

TAU_EXEC

```
$ tau_exec

Usage: tau_exec [options] [--] <exe> <exe options>

Options:
        -v            Verbose mode
        -s            Show what will be done but don't actually do anything (dryrun)
        -qsub         Use qsub mode (BG/P only, see below)
        -io           Track I/O
        -memory       Track memory allocation/deallocation
        -memory_debug Enable memory debugger
        -cuda         Track GPU events via CUDA
        -cupti        Track GPU events via CUPTI (Also see env. variable TAU_CUPTI_API)
        -opencl       Track GPU events via OpenCL
        -openacc      Track GPU events via OpenACC (currently PGI only)
        -ompt         Track OpenMP events via OMPT interface
        -armci        Track ARMCI events via PARMCI
        -ebs          Enable event-based sampling
        -ebs_period=<count> Sampling period (default 1000)
        -ebs_source=<counter> Counter (default itimer)
        -um          Enable Unified Memory events via CUPTI
        -T <DISABLE,GNU,ICPC,MPI,OMPT,OPENMP,PAPI,PDT,PROFILE,PTHREAD,SCOREP,SERIAL> : Specify TAU tags
        -loadlib=<file.so>   : Specify additional load library
        -XrunTAUsh-<options> : Specify TAU library directly
        -gdb          Run program in the gdb debugger

Notes:
               Defaults if unspecified: -T MPI
               MPI is assumed unless SERIAL is specified
```

- Tau_exec preloads the TAU wrapper libraries and performs measurements.

No need to recompile the application!

TAU_EXEC EXAMPLE (CONTINUED)

```
Example:
    mpirun -np 2 tau_exec -T icpc,ompt,mpi  -ompt ./a.out
    mpirun -np 2 tau_exec -io ./a.out
Example - event-based sampling with samples taken every 1,000,000 FP instructions
    mpirun -np 8 tau_exec -ebs -ebs_period=1000000 -ebs_source=PAPI_FP_INS ./ring
Examples - GPU:
    tau_exec -T serial,cupti -cupti ./matmult (Preferred for CUDA 4.1 or later)
    tau_exec -openacc ./a.out
    tau_exec -T serial -opencl ./a.out (OPENCL)
    mpirun -np 2 tau_exec -T mpi,cupti,papi -cupti -um ./a.out (Unified Virtual Memory in CUDA 6.0+)

qsub mode (IBM BG/Q only):
    Original:
      qsub -n 1 --mode smp -t 10 ./a.out
    With TAU:
      tau_exec -qsub -io -memory -- qsub -n 1 … -t 10 ./a.out

Memory Debugging:
    -memory option:
      Tracks heap allocation/deallocation and memory leaks.
    -memory_debug option:
      Detects memory leaks, checks for invalid alignment, and checks for
      array overflow.  This is exactly like setting TAU_TRACK_MEMORY_LEAKS=1
      and TAU_MEMDBG_PROTECT_ABOVE=1 and running with -memory
```

- tau_exec can enable event based sampling while launching the executable using env TAU_SAMPLING=1 or tau_exec -ebs

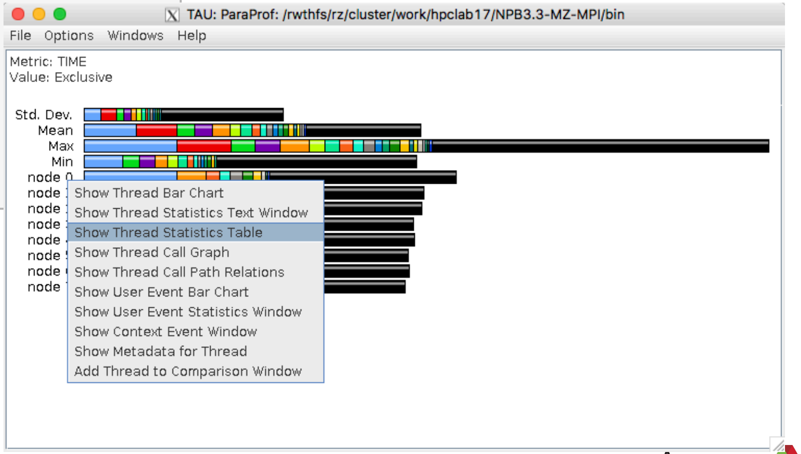
EVENT BASED SAMPLING WITH TAU

- Launch paraprof

```
% cd MZ-NPB3.3-MPI; cat README
% make clean;
% make suite
% cd bin
% qsub -I -n 1 -A Comp_Perf_Workshop  -t 50 -q cache-quad
% export OMP_NUM_THREADS=4
% aprun -n 16 tau_exec -T ompt -ebs ./bt-mz.B.16
% On head node:
% module load tau
% paraprof
```

- Right Click on Node 0 and choose

Show Thread Statistics Table

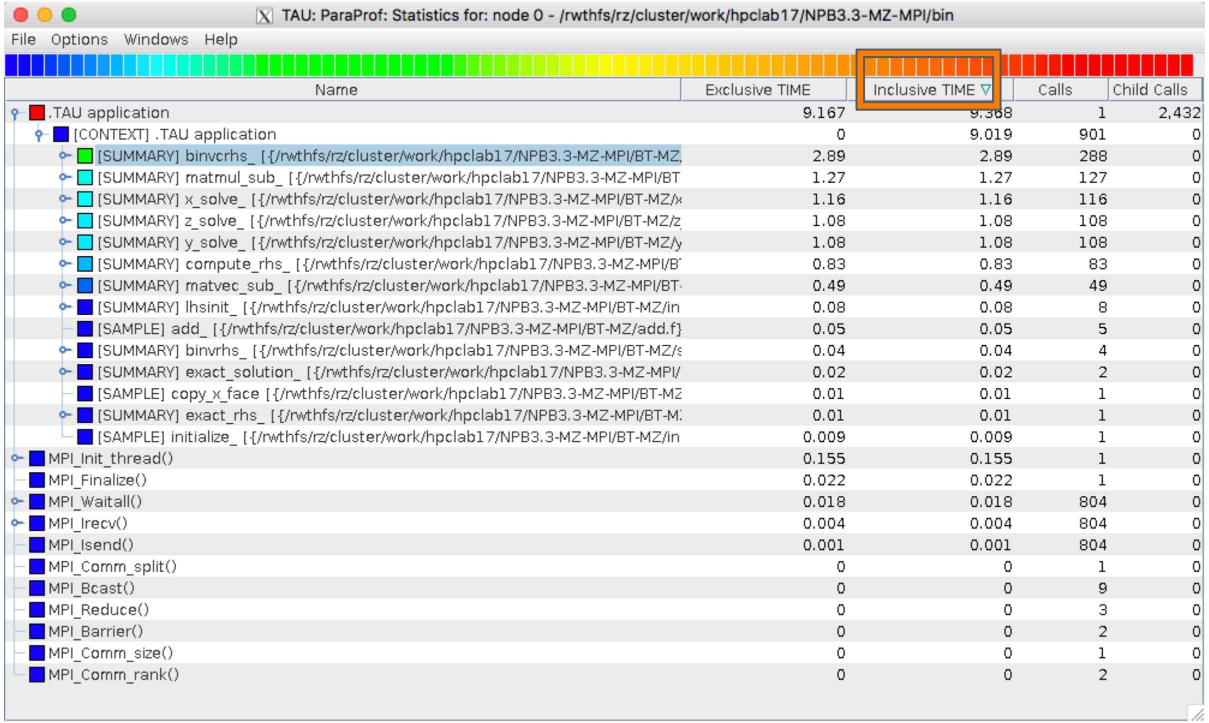

TAU: ParaProf: /rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/bin
File
Options
Windows
Help
Std. Dev.
Mean
Max
Min
node 0
node
node
node
node
node
node
node
Show Thread Bar Chart
Show Thread Statistics Text Window
Show Thread Statistics Table
Show Thread Call Graph
Show Thread Call Path Relations
Show User Event Bar Chart
Show User Event Statistics Window
Show Context Event Window
Show Metadata for Thread
Add Thread to Comparison Window

Metric: TIME
Value: Exclusive

PARAPROF

- Click on Columns: to sort by incl time

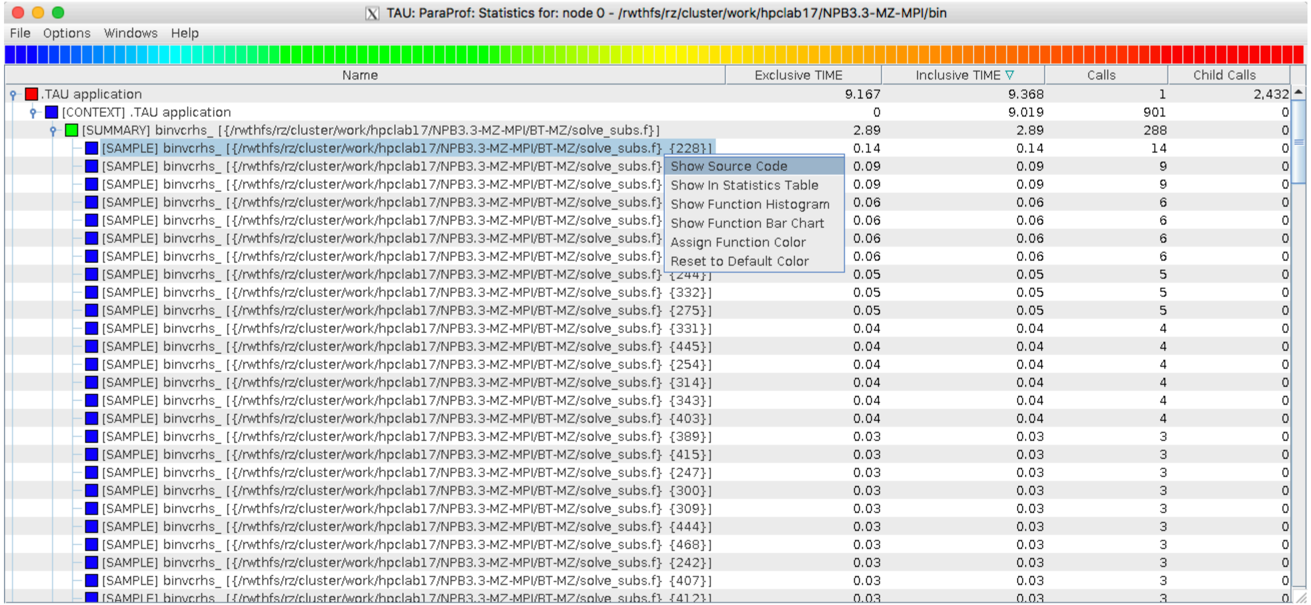
- Open binvcrhs
- Click on Sample



TAU: ParaProf: Statistics for: node 0 - /rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/bin
File
Options
Windows
Help

Name	Exclusive TIME	Inclusive TIME ▽	Calls	Child Calls
🟥 .TAU application	9.167	9.368	1	2,432
🟦 [CONTEXT] .TAU application	0	9.019	901	0
🟩 [SUMMARY] binvcrhs_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ,	2.89	2.89	288	0
🟦 [SUMMARY] matmul_sub_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT	1.27	1.27	127	0
🟦 [SUMMARY] x_solve_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/x	1.16	1.16	116	0
🟦 [SUMMARY] z_solve_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/z	1.08	1.08	108	0
🟦 [SUMMARY] y_solve_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/y	1.08	1.08	108	0
🟦 [SUMMARY] compute_rhs_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT	0.83	0.83	83	0
🟦 [SUMMARY] matvec_sub_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-	0.49	0.49	49	0
🟦 [SUMMARY] lhsinit_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/in	0.08	0.08	8	0
🟦 [SAMPLE] add_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/add.f}	0.05	0.05	5	0
🟦 [SUMMARY] binvrhs_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/s	0.04	0.04	4	0
🟦 [SUMMARY] exact_solution_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/	0.02	0.02	2	0
🟦 [SAMPLE] copy_x_face [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ	0.01	0.01	1	0
🟦 [SUMMARY] exact_rhs_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-M	0.01	0.01	1	0
🟦 [SAMPLE] initialize_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/in	0.009	0.009	1	0
🟦 MPI_Init_thread()	0.155	0.155	1	0
🟦 MPI_Finalize()	0.022	0.022	1	0
🟦 MPI_Waitall()	0.018	0.018	804	0
🟦 MPI_Irecv()	0.004	0.004	804	0
🟦 MPI_Isend()	0.001	0.001	804	0
🟦 MPI_Comm_split()	0	0	1	0
🟦 MPI_Bcast()	0	0	9	0
🟦 MPI_Reduce()	0	0	3	0
🟦 MPI_Barrier()	0	0	2	0
🟦 MPI_Comm_size()	0	0	1	0
🟦 MPI_Comm_rank()	0	0	2	0
]

PARAPROF



TAU: ParaProf: Statistics for: node 0 - /rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/bin

Name	Exclusive TIME	Inclusive TIME ▽	Calls	Child Calls
🟥 .TAU application	9.167	9.368	1	2,432
🟦 [CONTEXT] .TAU application	0	9.019	901	0
🟩 [SUMMARY] binvcrhs_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f}]	2.89	2.89	288	0
🟦 [SAMPLE] binvcrhs_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} {228}]	0.14	0.14	14	0
🟦 [SAMPLE] binvcrhs_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f}	0.09	0.09	9	0
🟦 [SAMPLE] binvcrhs_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f}	0.09	0.09	9	0
🟦 [SAMPLE] binvcrhs_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f}	0.06	0.06	6	0
🟦 [SAMPLE] binvcrhs_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f}	0.06	0.06	6	0
🟦 [SAMPLE] binvcrhs_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f}	0.06	0.06	6	0
🟦 [SAMPLE] binvcrhs_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f}	0.06	0.06	6	0
🟦 [SAMPLE] binvcrhs_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} {244}]	0.05	0.05	5	0
🟦 [SAMPLE] binvcrhs_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} {332}]	0.05	0.05	5	0
🟦 [SAMPLE] binvcrhs_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} {275}]	0.05	0.05	5	0
🟦 [SAMPLE] binvcrhs_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} {331}]	0.04	0.04	4	0
🟦 [SAMPLE] binvcrhs_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} {445}]	0.04	0.04	4	0
🟦 [SAMPLE] binvcrhs_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} {254}]	0.04	0.04	4	0
🟦 [SAMPLE] binvcrhs_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} {314}]	0.04	0.04	4	0
🟦 [SAMPLE] binvcrhs_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} {343}]	0.04	0.04	4	0
🟦 [SAMPLE] binvcrhs_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} {403}]	0.04	0.04	4	0
🟦 [SAMPLE] binvcrhs_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} {389}]	0.03	0.03	3	0
🟦 [SAMPLE] binvcrhs_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} {415}]	0.03	0.03	3	0
🟦 [SAMPLE] binvcrhs_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} {247}]	0.03	0.03	3	0
🟦 [SAMPLE] binvcrhs_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} {300}]	0.03	0.03	3	0
🟦 [SAMPLE] binvcrhs_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} {309}]	0.03	0.03	3	0
🟦 [SAMPLE] binvcrhs_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} {444}]	0.03	0.03	3	0
🟦 [SAMPLE] binvcrhs_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} {468}]	0.03	0.03	3	0
🟦 [SAMPLE] binvcrhs_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} {242}]	0.03	0.03	3	0
🟦 [SAMPLE] binvcrhs_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} {407}]	0.03	0.03	3	0
🟦 [SAMPLE] binvcrhs_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f} {412}]	0.03	0.03	3	0

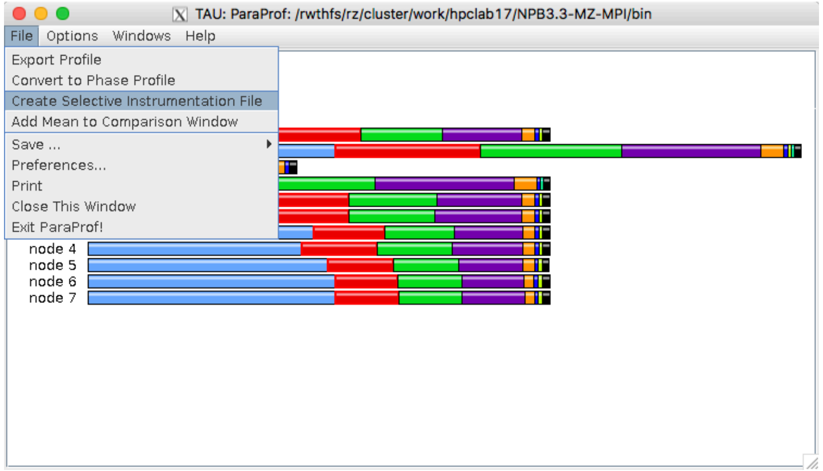
TAU SOURCE INSTRUMENTATION

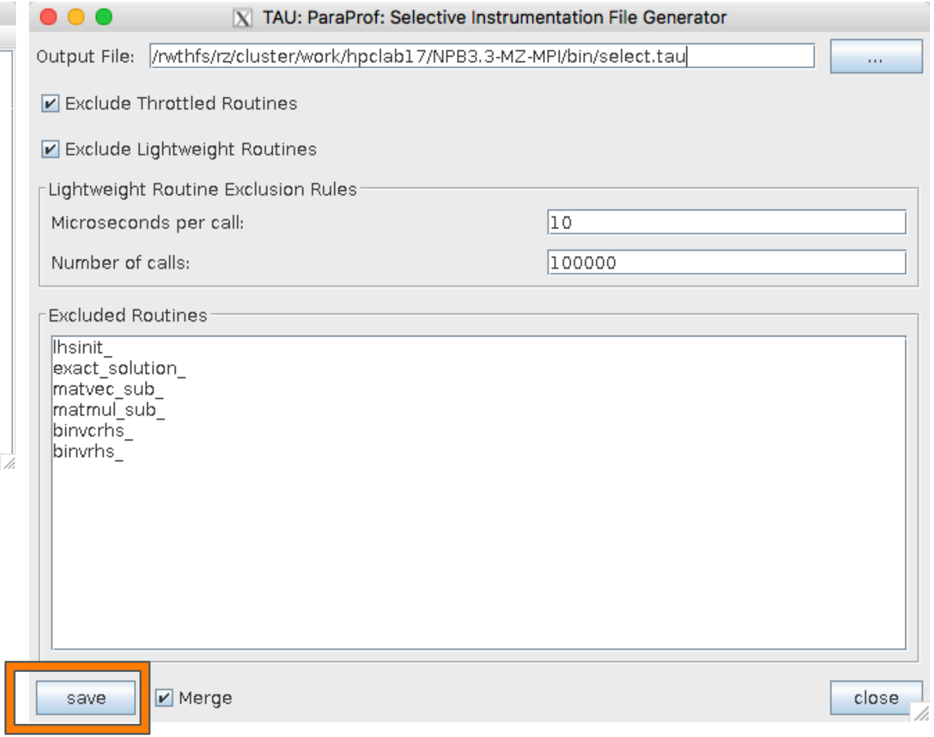
- Choose TAU configuration

```
module load tau;
export TAU_MAKEFILE=$TAU/Makefile.tau-intel-papi-ompt-mpi-pdt-openmp
```

- Edit config/make.def to adjust build configuration
  - Uncomment specification of compiler/linker: F77 = tau_f77.sh or use make F77=tau_f77.sh
- Make clean and build new tool-specific executable
- Change to the directory containing the new executable before running it with the desired tool configuration

CREATE A SELECTIVE INSTRUMENTATION FILE, RE-INSTRUMENT, RE-RUN




TAU: ParaProf: Selective Instrumentation File Generator

Output File: /rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/bin/select.tau

☑ Exclude Throttled Routines

☑ Exclude Lightweight Routines

Lightweight Routine Exclusion Rules

Microseconds per call: 10

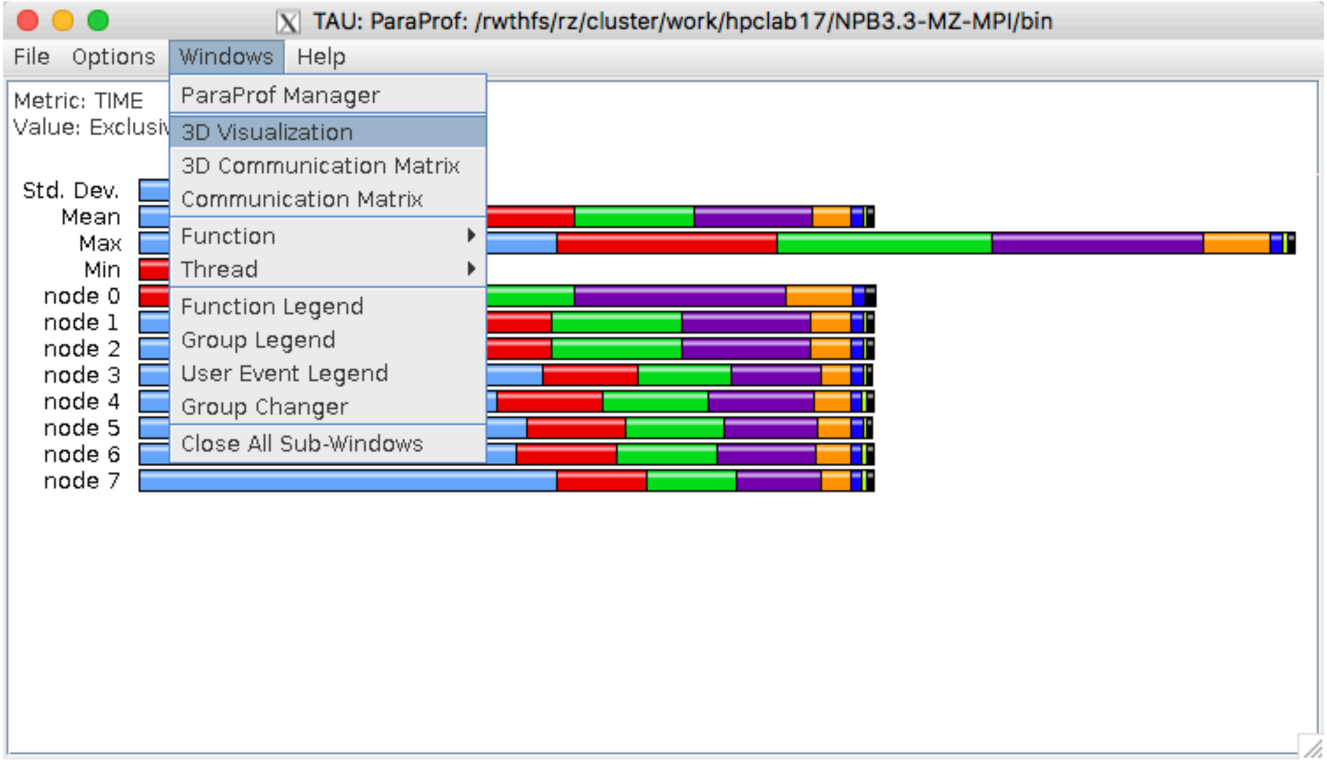
Number of calls: 100000

Excluded Routines

```
lhsinit_
exact_solution_
matvec_sub_
matmul_sub_
binvcrhs_
binvrhs_
```

☑ Merge

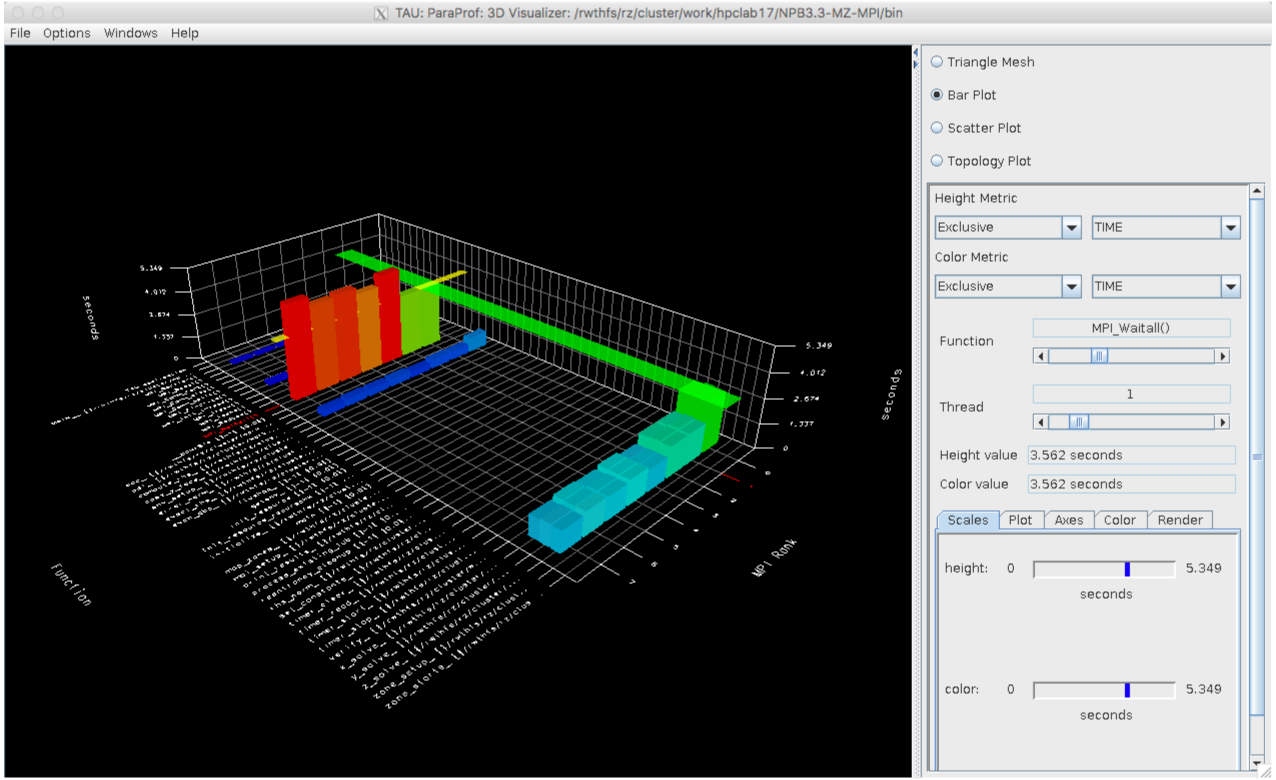
PARAPROF WITH OPTIMIZED INSTRUMENTATION



TAU: ParaProf: /rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/bin

Metric: TIME
Value: Exclusiv

3D VISUALIZATION WITH PARAPROF



TAU: ParaProf: 3D Visualizer: /rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/bin

○ Triangle Mesh

● Bar Plot

○ Scatter Plot

○ Topology Plot

Height Metric: Exclusive, TIME

Color Metric: Exclusive, TIME

Function: MPI_Waitall()

Thread: 1

Height value: 3.562 seconds

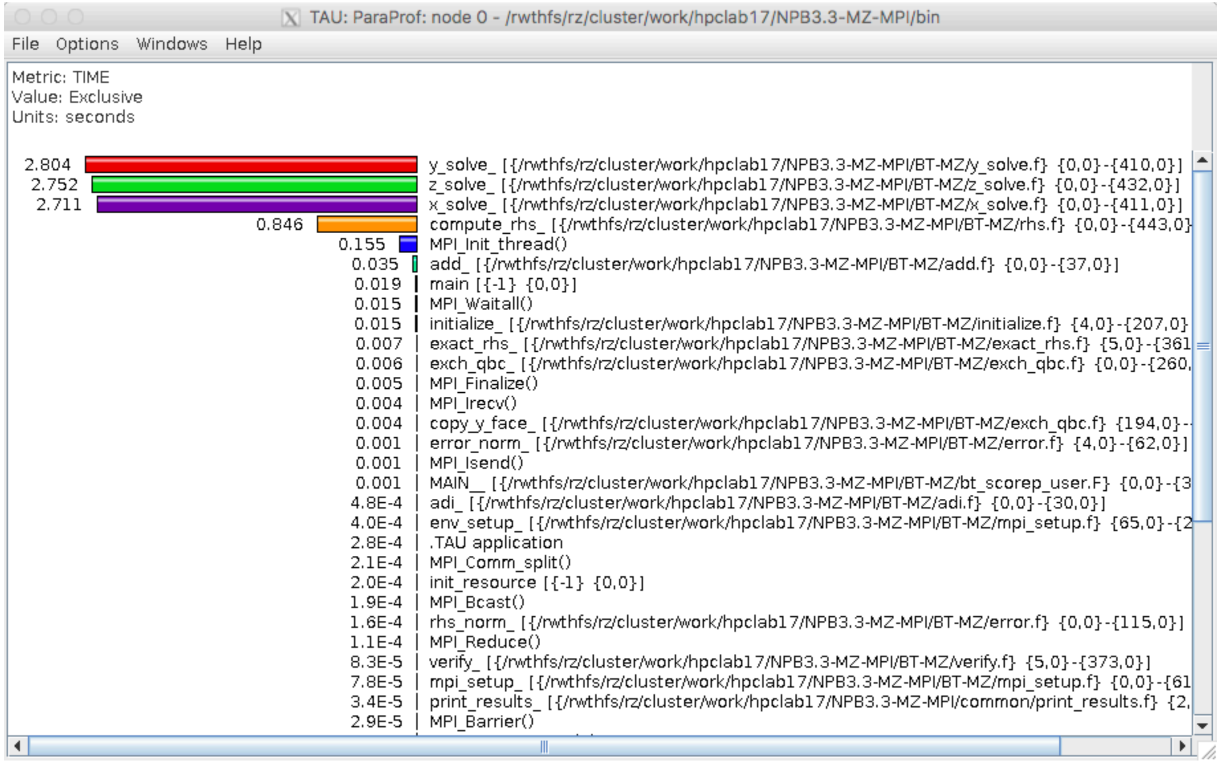
Color value: 3.562 seconds

height: 0 – 5.349 seconds

color: 0 – 5.349 seconds

PARAPROF: NODE 0

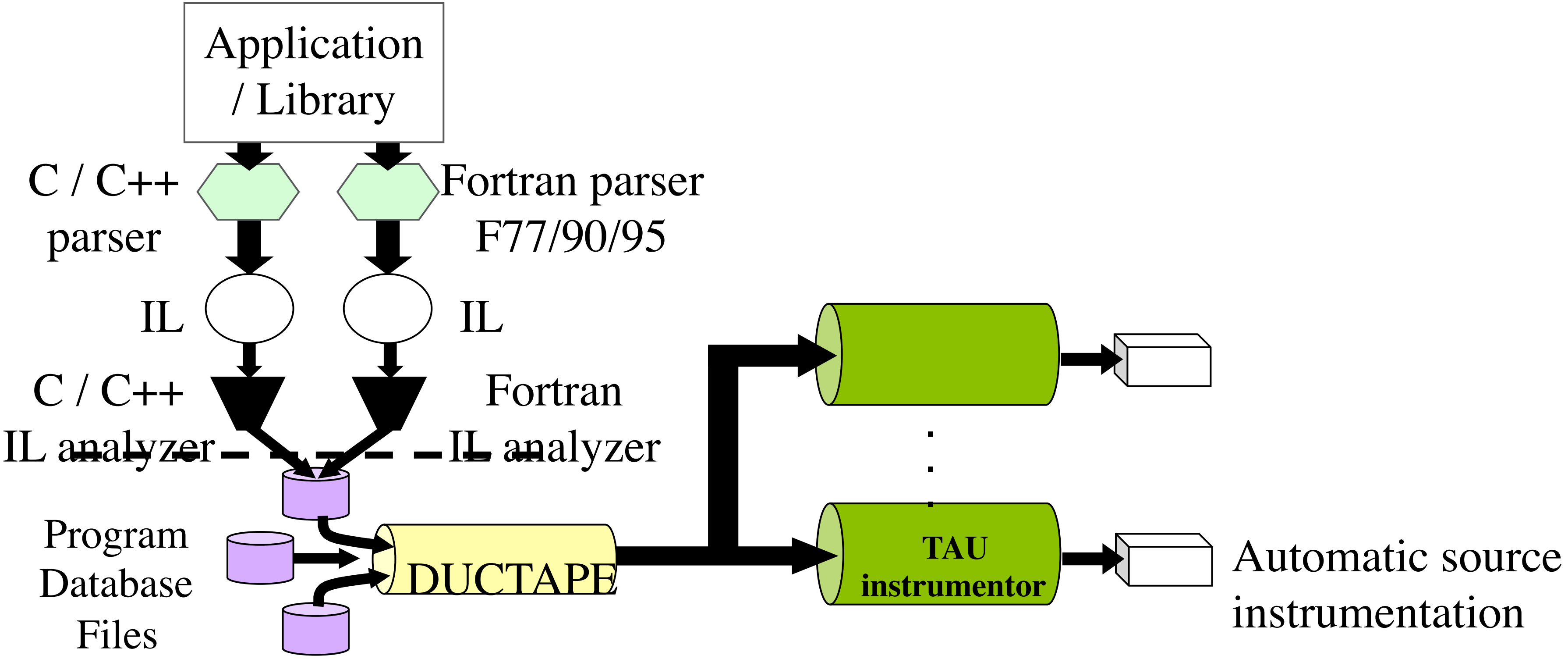
- Optimized instrumentation!


TAU: ParaProf: node 0 - /rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/bin
File
Options
Windows
Help

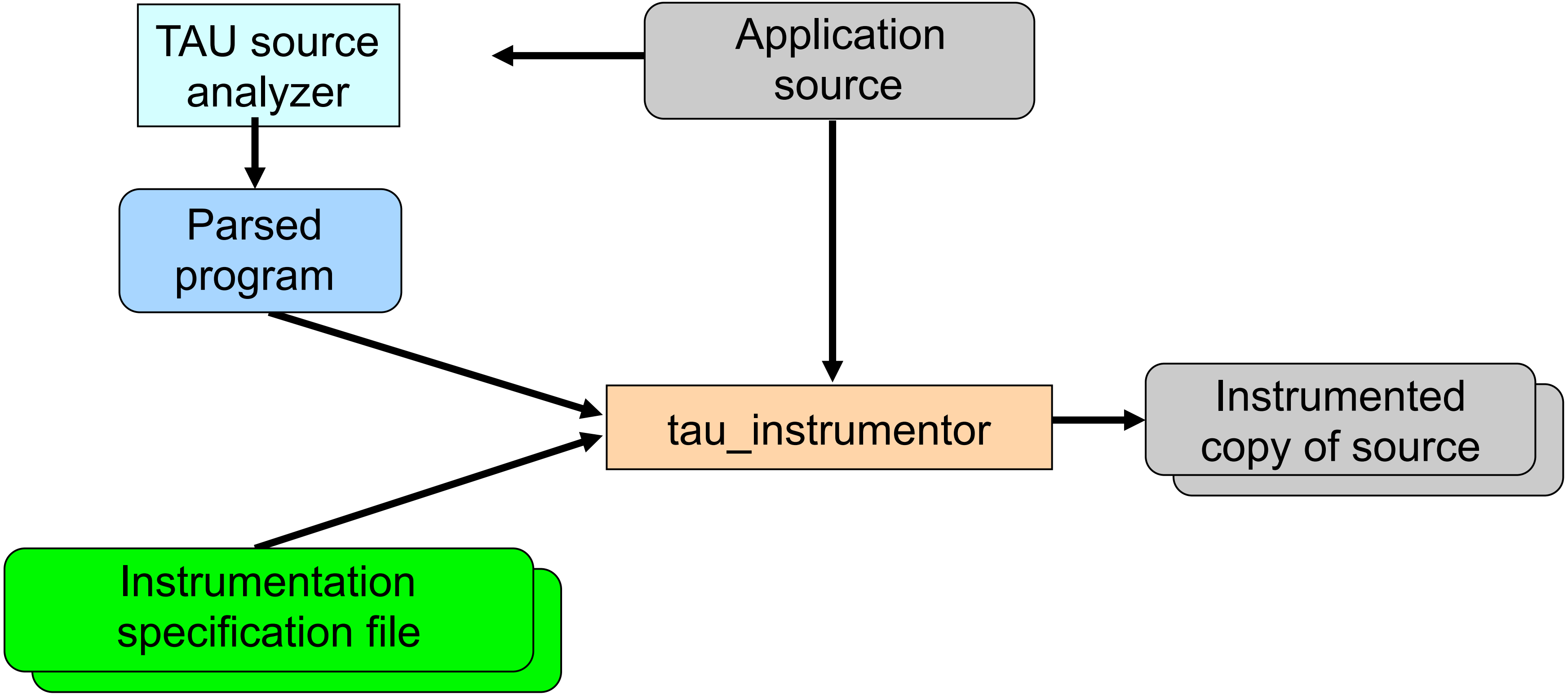
```
Metric: TIME
Value: Exclusive
Units: seconds

2.804  y_solve_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/y_solve.f} {0,0}-{410,0}]
2.752  z_solve_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/z_solve.f} {0,0}-{432,0}]
2.711  x_solve_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/x_solve.f} {0,0}-{411,0}]
0.846  compute_rhs_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/rhs.f} {0,0}-{443,0}
0.155  MPI_Init_thread()
0.035  add_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/add.f} {0,0}-{37,0}]
0.019  main [{-1} {0,0}]
0.015  MPI_Waitall()
0.015  initialize_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/initialize.f} {4,0}-{207,0}
0.007  exact_rhs_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/exact_rhs.f} {5,0}-{361
0.006  exch_qbc_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/exch_qbc.f} {0,0}-{260,
0.005  MPI_Finalize()
0.004  MPI_Irecv()
0.004  copy_y_face_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/exch_qbc.f} {194,0}-
0.001  error_norm_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/error.f} {4,0}-{62,0}]
0.001  MPI_Isend()
0.001  MAIN__ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/bt_scorep_user.F} {0,0}-{3
4.8E-4  adi_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/adi.f} {0,0}-{30,0}]
4.0E-4  env_setup_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/mpi_setup.f} {65,0}-{2
2.8E-4  .TAU application
2.1E-4  MPI_Comm_split()
2.0E-4  init_resource [{-1} {0,0}]
1.9E-4  MPI_Bcast()
1.6E-4  rhs_norm_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/error.f} {0,0}-{115,0}]
1.1E-4  MPI_Reduce()
8.3E-5  verify_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/verify.f} {5,0}-{373,0}]
7.8E-5  mpi_setup_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/BT-MZ/mpi_setup.f} {0,0}-{61
3.4E-5  print_results_ [{/rwthfs/rz/cluster/work/hpclab17/NPB3.3-MZ-MPI/common/print_results.f} {2,
2.9E-5  MPI_Barrier()
```

SOURCE INSTRUMENTATION

TAU' S STATIC ANALYSIS SYSTEM: PROGRAM DATABASE TOOLKIT (PDT)



PDT: AUTOMATIC SOURCE INSTRUMENTATION



USING SOURCE INSTRUMENTATION IN TAU

- TAU supports several compilers, measurement, and thread options
  - Intel compilers, profiling with hardware counters using PAPI, MPI library, OpenMP…
  - Each measurement configuration of TAU corresponds to a unique stub makefile (configuration file) and library that is generated when you configure it
- To instrument source code automatically using PDT
  - Choose an appropriate TAU stub makefile in <arch>/lib:

```
% module load UNITE tau

% export TAU_MAKEFILE=$TAU/Makefile.tau-intel-papi-mpi-pdt
% export TAU_OPTIONS=‘-optVerbose …’ (see tau_compiler.sh )
```

  - Use tau_f90.sh, tau_cxx.sh, tau_upc.sh, or tau_cc.sh as F90, C++, UPC, or C compilers respectively:

```
% ftn          foo.f90      changes to
% tau_f90.sh foo.f90
```

- Set runtime environment variables, execute application and analyze performance data:

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

INSTALLING TAU

- Installing PDT:
  - wget http://tau.uoregon.edu/pdt_lite.tgz
  - ./configure –prefix=<dir>; make ; make install
- Installing TAU on Theta:
  - wget http://tau.uoregon.edu/tau.tgz
  - ./configure −arch=craycnl -mpi −pdt=<dir> -bfd=download −unwind=download −iowrapper;
  - make install
  - For x86_64 clusters running Linux
  - ./configure -c++=mpicxx −cc=mpicc −fortran=mpif90 −pdt=<dir> -bfd=download −unwind=download
  - make install
- Using TAU:
  - export TAU_MAKEFILE=<taudir>/x86_64/lib/Makefile.tau-<TAGS>
  - make CC=tau_cc.sh CXX=tau_cxx.sh F90=tau_f90.sh

INSTALLING TAU ON LAPTOPS

- Installing TAU under Mac OS X:
  - Download Java
  - http://tau.uoregon.edu/java.dmg
  - Install java.dmg
  - wget http://tau.uoregon.edu/tau.dmg
  - Install tau.dmg
- Installing TAU under Windows
  - http://tau.uoregon.edu/tau.exe
- Installing TAU under Linux
  - http://tau.uoregon.edu/tau.tgz
  - ./configure; make install
  - export PATH=<taudir>/x86_64/bin:$PATH

DIFFERENT MAKEFILES FOR TAU COMPILER

```
%  module load tau
% ls $TAU/Makefile.*
/soft/perftools/tau/tau-2.26.1/craycnl/lib/Makefile.tau-intel-papi-mpi-pdt
/soft/perftools/tau/tau-2.26.1/craycnl/lib/Makefile.tau-intel-papi-mpi-pdt-openmp-opari
/soft/perftools/tau/tau-2.26.1/craycnl/lib/Makefile.tau-intel-papi-mpi-pthread-pdt
/soft/perftools/tau/tau-2.26.1/craycnl/lib/Makefile.tau-intel-papi-ompt-mpi-pdt-openmp
/soft/perftools/tau/tau-2.26.1/craycnl/lib/Makefile.tau-intel-papi-ompt-pdt-openmp
/soft/perftools/tau/tau-2.26.1/craycnl/lib/Makefile.tau-intel-papi-pdt
/soft/perftools/tau/tau-2.26.1/craycnl/lib/Makefile.tau-intel-papi-pdt-openmp-opari
/soft/perftools/tau/tau-2.26.1/craycnl/lib/Makefile.tau-intel-papi-pthread-pdt
```

For an MPI+OpenMP+F90 application with Intel MPI, you may choose

`Makefile.tau-intel-papi-ompt-mpi-pdt-openmp`

- Supports MPI instrumentation & PDT for automatic source instrumentation

```
% export TAU_MAKEFILE=$TAU/Makefile.tau-intel-papi-ompt-mpi-pdt-openmp
% tau_f90.sh app.f90 -o app; aprun -n 256 ./app; paraprof
```

COMPILE-TIME OPTIONS

Optional parameters for the TAU_OPTIONS environment variable:
% tau_compiler.sh

-optVerbose	Turn on verbose debugging messages
-optCompInst	Use compiler based instrumentation
-optNoCompInst	Do not revert to compiler instrumentation if source instrumentation fails.
-optTrackIO	Wrap POSIX I/O call and calculates vol/bw of I/O operations (configure TAU with –*iowrapper*)
-optTrackGOMP	Enable tracking GNU OpenMP runtime layer (used without –opari)
-optMemDbg	Enable runtime bounds checking (see TAU_MEMDBG_* env vars)
-optKeepFiles	Does not remove intermediate .pdb and .inst.* files
-optPreProcess	Preprocess sources (OpenMP, Fortran) before instrumentation
-optTauSelectFile=”<file>"	Specify selective instrumentation file for *tau_instrumentor*
-optTauWrapFile=”<file>"	Specify path to *link_options.tau* generated by *tau_gen_wrapper*
-optHeaderInst	Enable Instrumentation of headers
-optTrackUPCR	Track UPC runtime layer routines (used with tau_upc.sh)
-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) …

COMPILE-TIME OPTIONS (CONTD.)

- Optional parameters for the TAU_OPTIONS environment variable:

% tau_compiler.sh

-optMICOffload Links code for Intel MIC offloading, requires both host and MIC TAU libraries

-optShared Use TAU’s shared library (libTAU.so) instead of static library (default)

-optPdtCxxOpts=“” Options for C++ parser in PDT (cxxparse).

-optPdtF90Parser=“” Specify a different Fortran parser

-optPdtCleanscapeParser Specify the Cleanscape Fortran parser instead of GNU gfparser

-optTau=“” Specify options to the tau_instrumentor

-optTrackDMAPP Enable instrumentation of low-level DMAPP API calls on Cray

-optTrackPthread Enable instrumentation of pthread calls

See tau_compiler.sh for a full list of TAU_OPTIONS.

…

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=select.tau -optVerbose -optPreProcess’
% cat select.tau
BEGIN_INSTRUMENT_SECTION
loops  routine=“#”
# this statement instruments all outer loops in all routines. # is wildcard as well as comment in first column.
END_INSTRUMENT_SECTION
```

SELECTIVE INSTRUMENTATION FILE WITH PROGRAM DATABASE TOOLKIT (PDT)

To use an instrumentation specification file for source instrumentation:

```
% export TAU_OPTIONS= ‘-optTauSelectFile=/path/to/select.tau –optVerbose ’
% cat select.tau
 BEGIN_EXCLUDE_LIST
 BINVCRHS
 MATMUL_SUB
 MATVEC_SUB
 EXACT_SOLUTION
 BINVRHS
 LHS#INIT
 TIMER_#
 END_EXCLUDE_LIST
```

NOTE: paraprof can create this file from an earlier execugtion for you.
File -> Create Selective Instrumentation File -> save

RUNTIME ENVIRONMENT VARIABLES

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_FOOTPRINT	0	Setting to 1 turns on tracking memory usage by sampling periodically the resident set size and high water mark of memory usage
TAU_TRACK_POWER	0	Tracks power usage by sampling periodically.
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_SAMPLING	1	Setting to 1 enables event-based sampling.
TAU_TRACK_SIGNALS	0	Setting to 1 generate debugging callstack info when a program crashes
TAU_COMM_MATRIX	0	Setting to 1 generates communication matrix display using context events
TAU_THROTTLE	1	Setting to 0 turns off throttling. Throttles 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_CALLSITE	0	Setting to 1 enables callsite profiling that shows where an instrumented function was called. Also compatible with tracing.
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., ENERGY,TIME,P_VIRTUAL_TIME,PAPI_FP_INS,PAPI_NATIVE_<event>:<subevent>)

RUNTIME ENVIRONMENT VARIABLES (CONTD.)

Environment Variable	Default	Description
TAU_TRACK_MEMORY_LEAKS	0	Tracks allocates that were not de-allocated (needs –optMemDbg or tau_exec –memory)
TAU_EBS_SOURCE	TIME	Allows using PAPI hardware counters for periodic interrupts for EBS (e.g., TAU_EBS_SOURCE=PAPI_TOT_INS when TAU_SAMPLING=1)
TAU_EBS_PERIOD	100000	Specifies the overflow count for interrupts
TAU_MEMDBG_ALLOC_MIN/MAX	0	Byte size minimum and maximum subject to bounds checking (used with TAU_MEMDBG_PROTECT_*)
TAU_MEMDBG_OVERHEAD	0	Specifies the number of bytes for TAU’s memory overhead for memory debugging.
TAU_MEMDBG_PROTECT_BELOW/ABOVE	0	Setting to 1 enables tracking runtime bounds checking below or above the array bounds (requires –optMemDbg while building or tau_exec –memory)
TAU_MEMDBG_ZERO_MALLOC	0	Setting to 1 enables tracking zero byte allocations as invalid memory allocations.
TAU_MEMDBG_PROTECT_FREE	0	Setting to 1 detects invalid accesses to deallocated memory that should not be referenced until it is reallocated (requires –optMemDbg or tau_exec –memory)
TAU_MEMDBG_ATTEMPT_CONTINUE	0	Setting to 1 allows TAU to record and continue execution when a memory error occurs at runtime.
TAU_MEMDBG_FILL_GAP	Undefined	Initial value for gap bytes
TAU_MEMDBG_ALINGMENT	Sizeof(int)	Byte alignment for memory allocations
TAU_EVENT_THRESHOLD	0.5	Define a threshold value (e.g., .25 is 25%) to trigger marker events for min/max

Download TAU from U. Oregon

Tuning and Analysis Utilities

http://www.hpclinux.com [OVA file]

http://tau.uoregon.edu

for more information

Free download, open source, BSD license

PRL, UNIVERSITY OF OREGON, EUGENE

www.uoregon.edu

SUPPORT ACKNOWLEDGMENTS

- US Department of Energy (DOE)
  - ANL
  - Office of Science contracts
  - SciDAC, LBL contracts
  - LLNL-LANL-SNL ASC/NNSA contract
  - Battelle, PNNL and ORNL contract
- Department of Defense (DoD)
  - PETTT, HPCMP
- National Science Foundation (NSF)
  - SI2-SSI, Glassbox
- NASA
- Partners:
  - University of Oregon
  - The Ohio State University
  - ParaTools, Inc.
  - University of Tennessee, Knoxville
  - T.U. Dresden, GWT
  - Jülich Supercomputing Center

THANK YOU! QUESTIONS?

www.anl.gov

Argonne NATIONAL LABORATORY