TAU

David Gunter

High Performance Computing Division

Los Alamos National Laboratory

TAU: Motivation

Efficient HPC application development is aided by performance analysis:

- **Identification of bottlenecks (communication, memory)**
- **Optimize process communication (parallel and threaded codes)**
- **Tune runtime behavior**

TAU is a tool framework developed to fulfill this goal.

- **Integrated, scalable, portable, and flexible**
- **Supports Fortran, C, C++, Java, and Python for parallel, multi-threaded, and hybrid codes**
- **Provides:**
  - Automatic instrumentation
  - Highly configurable measurement system supporting many flavors of profiling and tracing
  - Contains analysis and visualization tools

TAU: Tuning & Analysis Utilities

- **Three levels of code instrumentation**
  - Dynamic binary rewriting
    - Uses DynInst
    - Avoids having to recompile large codes
  - Dynamic instrumentation through compiler directives
  - Automated static source rewriting using PDT (included with TAU)

Method	Recompile Required	PDT Required	MPI Events	Routine-level Events	Low level events (loops, phases, etc.)	Throttling to reduce overhead	Exclude files	Exclude code regions
Binary Rewriting			✔			✔	✔	
Compiler	✔		✔	✔		✔	✔	
Source (PDT)	✔	✔	✔	✔	✔	✔	✔	✔

TAU: Tuning & Analysis Utilities

What will this instrumentation give you?

- **Profiling: Shows you *how much time* was spent in each routine**
- **Tracing: Shows you *when* the events take place in each process along a timeline**

Basic Steps

- **Collect 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

TAU Availability

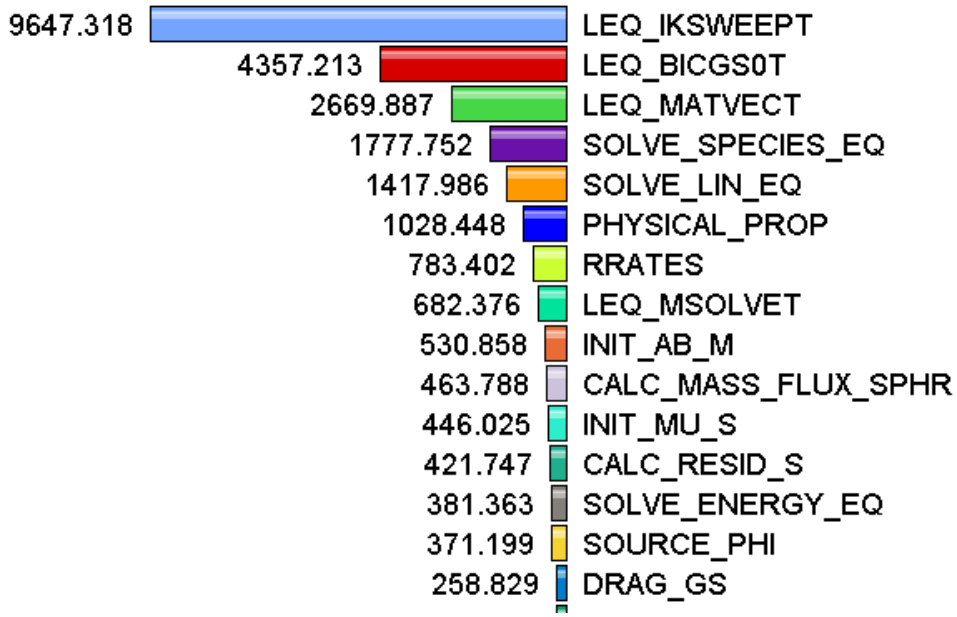
Platform	Version	Usage	Documentation	POC
LANL/Lobo	2.19.1	module load friendly-testing module load tau/2.19.1	TAU website	David Gunter dog@lanl.gov
SNL	TBD	TBD	TAU website	TBD
LLNL	TBD	TBD	TBD	TBD

Usage Scenarios: Dynamic Routine-level Profile

Goal: What routines account for the most time? How much?

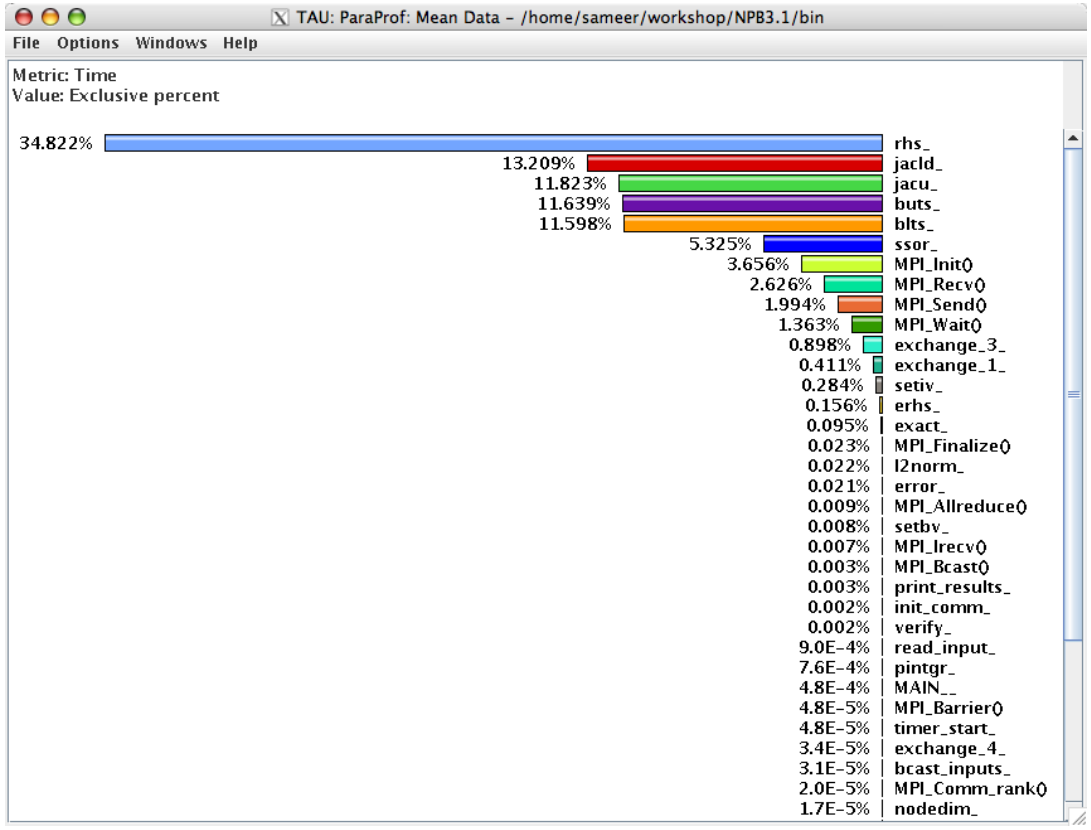
Flat profile with wallclock time:

Metric: P_VIRTUAL_TIME
Value: Exclusive
Units: seconds



Usage Scenarios: Compiler-based Instrumentation

Goal: Easily generate routine-level performance data via compiler directives



Usage Scenarios: Loop-level Instrumentation

Goal: What loops account for the most time? How much?

Flat profile with wallclock time and loop instrumentation

Metric: GET_TIME_OF_DAY
Value: Exclusive
Units: microseconds

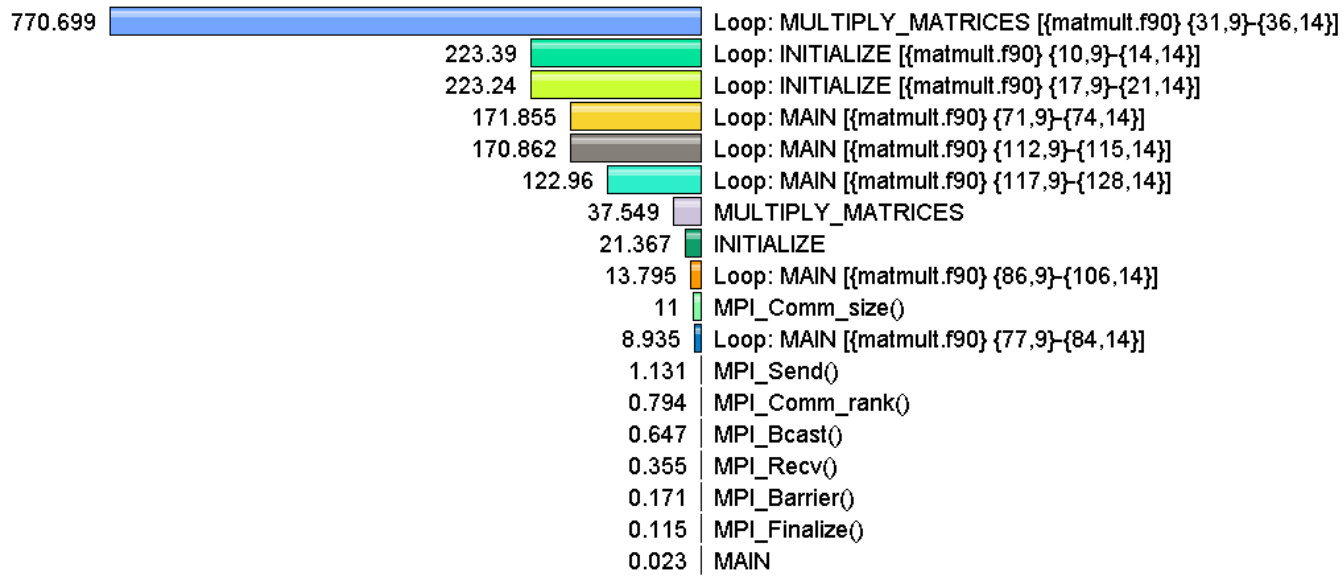
1729975.333	Loop: MULTIPLY_MATRICES [{matmult.f90} {31,9}-{36,14}]
443194	MPI_Recv()
81095	MAIN
49569	MPI_Bcast()
45669	Loop: MAIN [{matmult.f90} {86,9}-{106,14}]
12412	MPI_Send()
8959	Loop: INITIALIZE [{matmult.f90} {17,9}-{21,14}]
8953	Loop: INITIALIZE [{matmult.f90} {10,9}-{14,14}]
5609.2	MPI_Finalize()
2932.667	MULTIPLY_MATRICES
2577.667	Loop: MAIN [{matmult.f90} {117,9}-{128,14}]
2091.8	MPI_Barrier()
1875.667	Loop: MAIN [{matmult.f90} {112,9}-{115,14}]
1833	Loop: MAIN [{matmult.f90} {71,9}-{74,14}]
107	Loop: MAIN [{matmult.f90} {77,9}-{84,14}]
30	INITIALIZE
14.25	MPI_Comm_rank()
1	MPI_Comm_size()

Usage Scenarios: Calculate mflops in loops

Goal: MFlops in all loops

Flat profile with PAPI_FP_INS/OPS and loop instrumentation

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



Usage Scenarios: Generating Callpath Profile

Goal: Who calls MPI_Barrier()? Where?

Callpath profile for a given callpath depth:

n,c,t, 0,0,0 - callpath-all/scaling/flash/taudata/disk2/mnt/

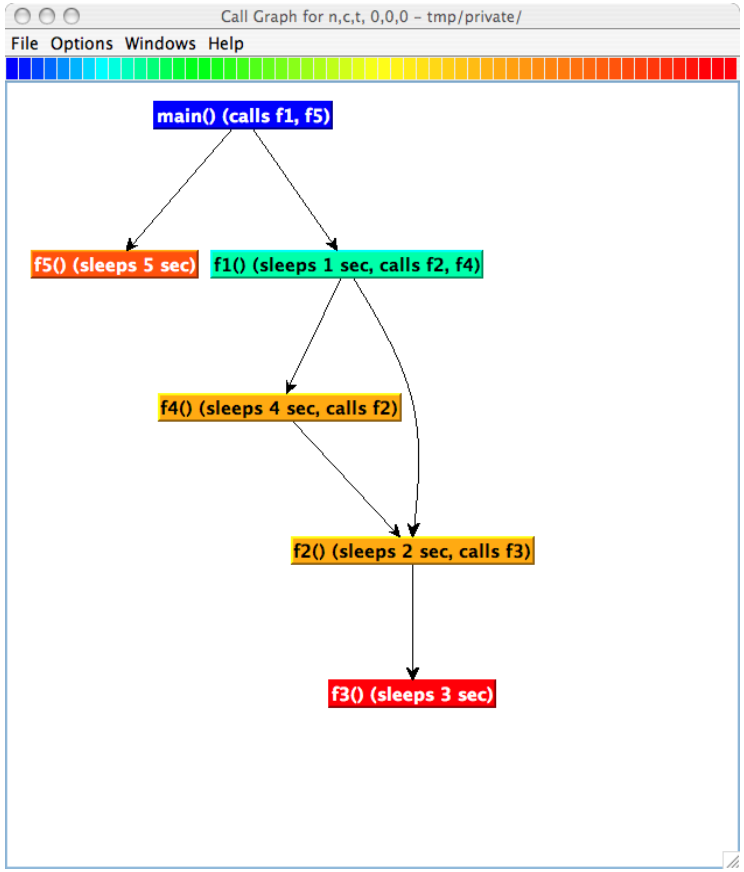
File Options Windows Help

Metric Name: Time
Value Type: exclusive

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

Usage Scenarios: Generating Callpath Profile (cont'd)

Generates program callgraph:



Usage Scenarios: Detect Memory Leaks

TAU: ParaProf: Mean Context Events - mem.ppk

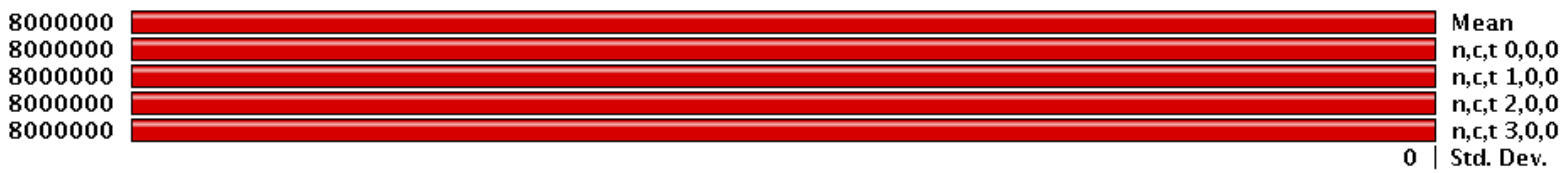
File Windows Help

Name	NumSamples	MaxValue	MinValue	MeanValue	Std. Dev.
MAIN [{matrix.f90} {141,7}-{146,22}]					
MATRICES::ALLOCATE_MATRICES [{matrix.f90} {10,7}-{13,38}]					
MEMORY LEAK! malloc size <file=matrix.f90, variable=C, line=11>	1	8,000,000	8,000,000	8,000,000	0
malloc size <file=matrix.f90, variable=A, line=11>	1	8,000,000	8,000,000	8,000,000	0
malloc size <file=matrix.f90, variable=B, line=11>	1	8,000,000	8,000,000	8,000,000	0
malloc size <file=matrix.f90, variable=C, line=11>	1	8,000,000	8,000,000	8,000,000	0
MATRICES::DEALLOCATE_MATRICES [{matrix.f90} {14,7}-{17,40}]					
free size <file=matrix.f90, variable=A, line=15>	1	8,000,000	8,000,000	8,000,000	0
free size <file=matrix.f90, variable=B , line=15>	1	8,000,000	8,000,000	8,000,000	0

User Event Window: mem.ppk

File Options Windows Help

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



Usage Scenarios: Mixed Language Profiling

Goal: Generate multi-level profiling for Python/MPI/C/F90/C++

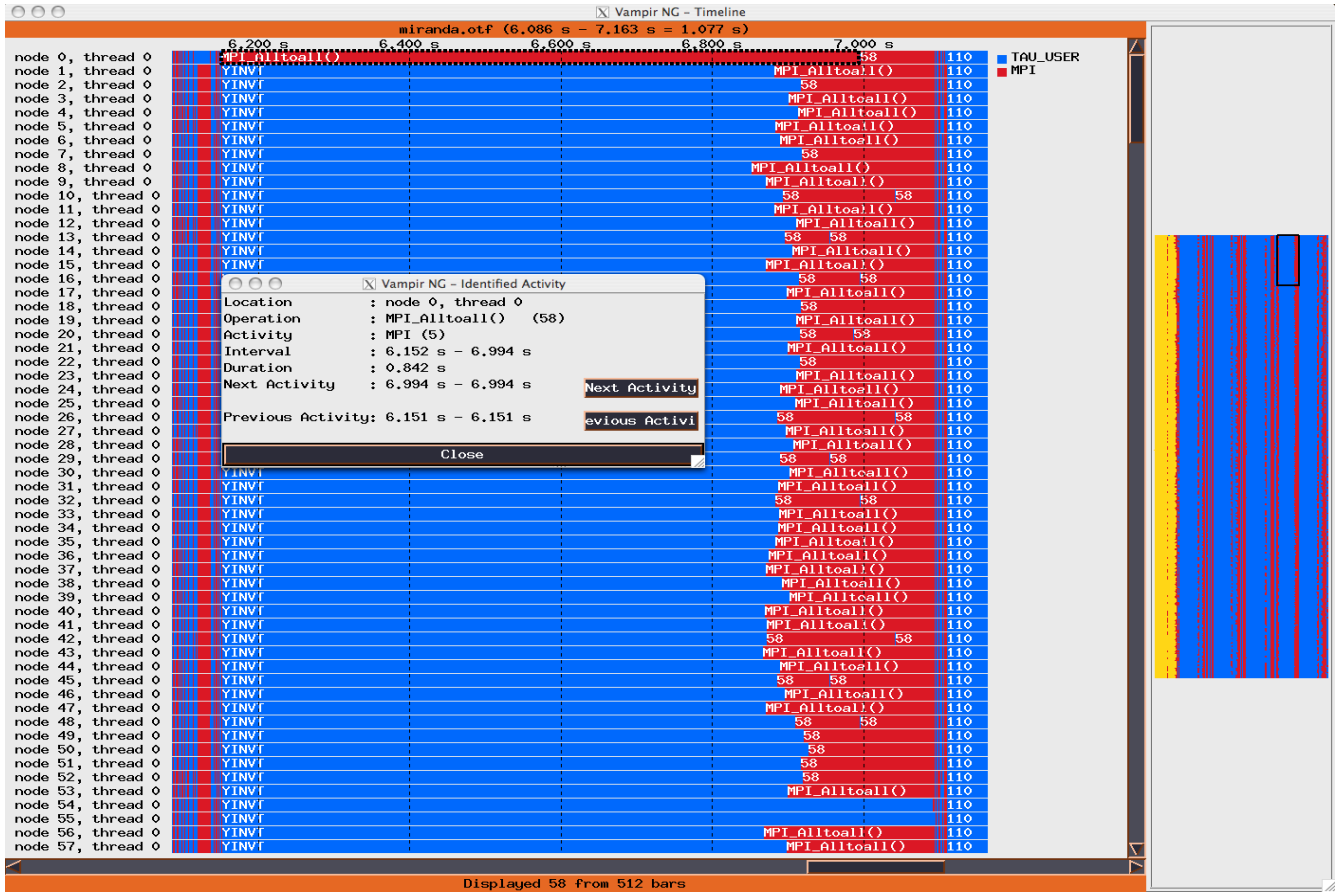
TAU: ParaProf: Mean Data – py.ppk

Metric: Time
Value: Exclusive percent

39.466%	void SAMINT::timestep(double, double) [{SAMINT.C} {72,1}-{78,1}]
6.563%	int pyMPI_Main_with_communicator(int, int *, char ***, MPI_Comm) C [{pyMPI_main.c} {23,1}-{80,1}]
5.18%	<module> [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/site-packages/numpy/linalg/linalg.py} {7}]
4.91%	void pyMPI_distutils_init(PyObject **) C [{pyMPI_distutils.c} {275,1}-{400,1}]
3.845%	<module> [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/site-packages/numpy/core/numeric.py} {1}]
3.633%	void pyMPI_site(void) C [{pyMPI_site.c} {23,1}-{121,1}]
3.607%	<module> [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/site-packages/numpy/core/__init__.py} {2}]
3.1%	MachAr::__init__ [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/site-packages/numpy/lib/machar.py} {50}]
2.261%	_parse [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/sre_parse.py} {385}]
2.123%	_compile [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/sre_compile.py} {38}]
1.187%	any [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/site-packages/numpy/core/fromnumeric.py} {540}]
1.034%	<module> [{/users3/sameer/py/py-c++-f90/example/samarcrun.py} {5}]
0.982%	<module> [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/site-packages/numpy/__init__.py} {17}]
0.941%	Tokenizer::__next [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/sre_parse.py} {188}]
0.929%	append
0.906%	<module> [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/ctypes/__init__.py} {4}]
0.895%	any
0.888%	<module> [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/site-packages/numpy/lib/__init__.py} {1}]
0.884%	len
0.863%	<module> [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/site-packages/numpy/lib/index_tricks.py} {3}]
0.806%	void pyMPI_pickle_init(PyObject **) C [{pyMPI_pickle.c} {43,1}-{61,1}]
0.739%	<module> [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/site-packages/numpy/random/__init__.py} {2}]
0.67%	SubPattern::getwidth [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/sre_parse.py} {146}]
0.658%	<module> [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/site-packages/numpy/testing/numpytest.py} {1}]
0.643%	_optimize_charset [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/sre_compile.py} {213}]
0.584%	Tokenizer::get [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/sre_parse.py} {207}]
0.552%	<module> [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/inspect.py} {24}]
0.541%	<module> [{/users3/sameer/py/py-c++-f90/samint.py} {7}]
0.537%	<module> [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/site-packages/numpy/core/memmap.py} {1}]
0.537%	<module> [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/site-packages/numpy/testing/__init__.py} {2}]
0.532%	<module> [{/users3/sameer/py/py-c++-f90/samarc.py} {9}]
0.475%	<module> [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/site-packages/numpy/ctypeslib.py} {1}]
0.462%	add_newdoc [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/site-packages/numpy/lib/function_base.py} {1154}]
0.453%	<module> [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/site-packages/numpy/lib/function_base.py} {1}]
0.451%	_parse_sub [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/sre_parse.py} {307}]
0.378%	_compile_charset [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/sre_compile.py} {184}]
0.367%	abs
0.365%	samarc::runStep [{/users3/sameer/py/py-c++-f90/samarc.py} {105}]
0.365%	<module> [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/site-packages/numpy/fft/fftpack.py} {21}]
0.296%	SubPattern::__len__ [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/sre_parse.py} {132}]
0.287%	<module> [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/site-packages/numpy/core/arrayprint.py} {4}]
0.27%	MPI_Finalize()
0.225%	Tokenizer::match [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/sre_parse.py} {201}]
0.213%	_mk_bitmap [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/sre_compile.py} {264}]
0.199%	<module> [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/site-packages/numpy/core/defmatrix.py} {1}]
0.18%	MPI_Init()
0.178%	SubPattern::__getitem__ [{/usr/local/PET/pkgs/python-2.5.1/lib/python2.5/sre_parse.py} {136}]

Usage Scenarios: MPI Trace File

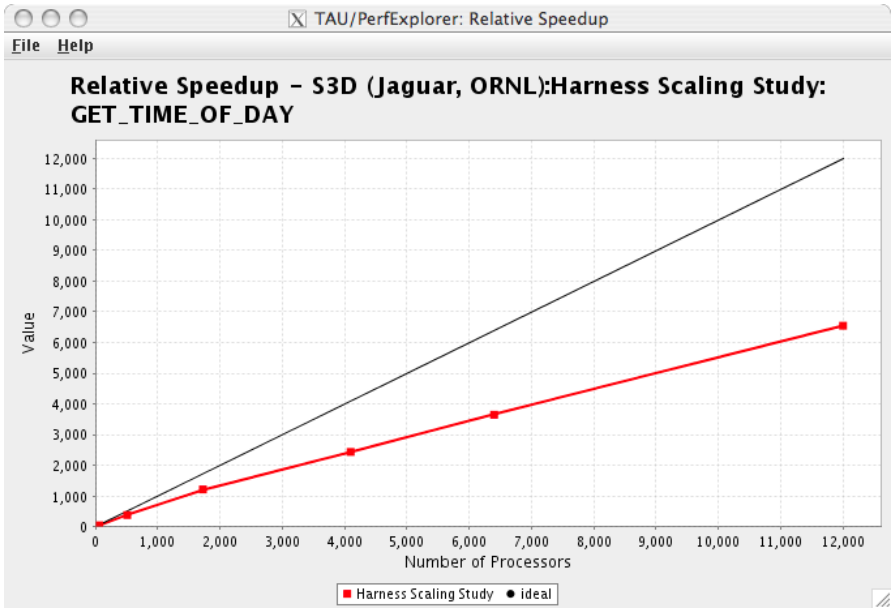
Goal: MPI Event Tracing

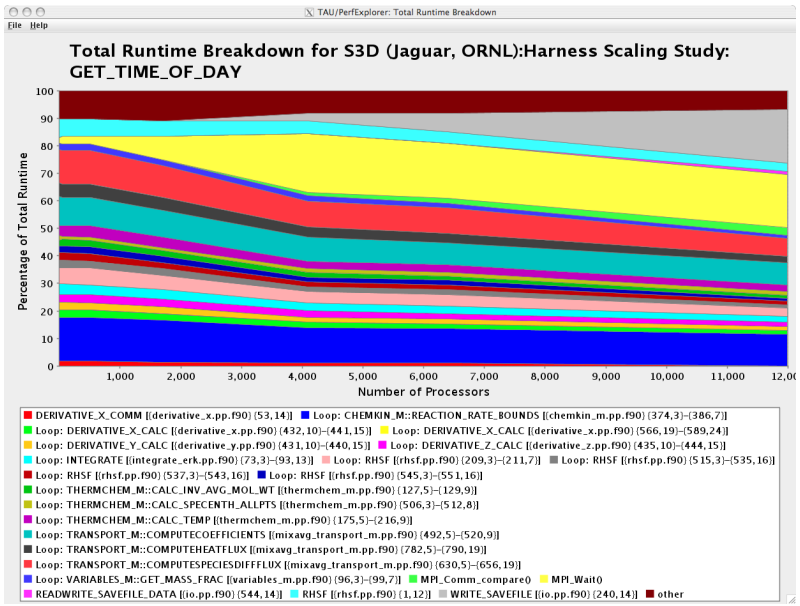


Usage Scenarios: Evaluate Scalability

Goal: Determine how an application scales? What bottlenecks occur at what core counts?

Load profiles into PerfDMF database and examine with PerfExplorer





TAU References

- TAU is a joint product of the University of Oregon, LANL, and Research Centre Jülich, ZAM, Germany.
- TAU is developed for the DOE Office of Science, ASC initiatives at LLNL, the ZeptoOS project at ANL, and for LANL.
- TAU Users Guide and papers available from the TAU website: http://tau.uoregon.edu/
- A list of projects using TAU and projects related to TAU may be found at: http://www.cs.uoregon.edu/research/tau/links.php