Demo: Using TAU for Performance Evaluation of Scientific Software

Sameer Shende
Performance Research Laboratory
University of Oregon
Eugene, OR 97403, USA
sameer@cs.uoregon.edu

Allen D. Malony
Department of Computer and Information Science
University of Oregon
Eugene, OR 97403, USA
malony@cs.uoregon.edu

***Abstract*—This paper presents the demonstration of the TAU Performance System for performance evaluation of Scientific Software written in C++, C, and Fortran.**

***Index Terms*—TAU, instrumentation, performance analysis, PDT, measurement.**

I. Introduction

The ability of performance technology to keep pace with the growing complexity of parallel and distributed systems depends on robust performance framew[1]orks that can at once provide system-specific performance capabilities and support high-level performance problem solving. Flexibility and portability in empirical methods and processes are influenced primarily by the strategies available for instrumentation and measurement, and how effectively they are integrated and composed. This demo will present the TAU (Tuning and Analysis Utilities) parallel performance system [1] and describes how it addresses diverse requirements for performance engineering of scientific software.

II. Performance Evaluation

Given the diversity of performance problems, evaluation methods, and types of events and metrics, the instrumentation and measurement mechanisms needed to support performance observation must be flexible, to give maximum opportunity for configuring performance experiments, and portable, to allow consistent cross-platform performance problem solving. In general, flexibility in empirical performance evaluation implies freedom in experiment design, and choices in selection and control of experiment mechanisms. Using tools that otherwise limit the type and structure of performance methods will restrict evaluation scope. Portability, on the other hand, looks for common abstractions in performance methods and how these can be supported by reusable and consistent techniques across different computing environments (software and hardware).

The TAU parallel performance system is the product of over two decades of development to create a robust, flexible, portable, and integrated framework and toolset for performance instrumentation, measurement, analysis, and visualization of large-scale parallel computer systems and applications. The architecture of TAU is shown in Fig. 1.

III. Demonstration

The demo will highlight the instrumentation of MPI programs on the NSF XSEDE system, Stampede, at TACC. It will demonstrate how TAU may be used to insert instrumentation in the source code using the C, C++, and Fortran parsers from the Program Database Toolkit (PDT) with TAU compiler scripts that may be used in place of compiler scripts provided by MPI. It will show to execute programs on the Intel® Xeon PhiTM systems and generate profiles that will be loaded in TAU's ParaProf 3D browser as shown in Fig. 2. These profiles may be stored in TAUdb, a performance database and analyzed using TAU's PerfExplorer tool [2] for cross-platform scalability studies and performance data mining. TAU uses PAPI [3] internally to access low-level hardware performance counters such as floating point instructions, level 1 and 2 data cache misses, and vector instructions executed in the code. Using these counters, TAU can show the extent of loop vectorization as shown in Fig. 3. TAU's ParaProf browser can show the time spent in each routine on all threads in its main window as shown in Fig. 4. It can also show the communication matrix as shown in Fig. 5 and a thread statistics window as shown in Fig. 6. TAU can support automatic instrumentation for code written in C, C++, Fortran, Java, and Python. It can be easily integrated in the build system of application frameworks and be enabled at compile-time using specially designed compiler scripts. TAU also supports instrumentation during program execution using preloading of TAU's Dynamic Shared Object (DSO) in the address space of the executing application. Using *tau_exec*, a user may evaluate the performance of an un-instrumented application. This includes memory, I/O, communication performance as well as event-based sampling to show the contribution at the statement level. TAU supports a variety of runtime systems used in HPC including OpenSHMEM, MPI, MPC, OpenMP, pthread, OpenCoArrays, CUDA, OpenCL, and OpenACC. The demo will show the use of TAU for performance engineering of software used in HPC.

IV. Conclusion

The TAU performance system addresses performance technology problems at three levels: instrumentation, measurement, and analysis. The TAU framework supports the configuration and integration of these layers to target specific performance problem solving needs. However, effective exploration of performance will necessarily require prudent

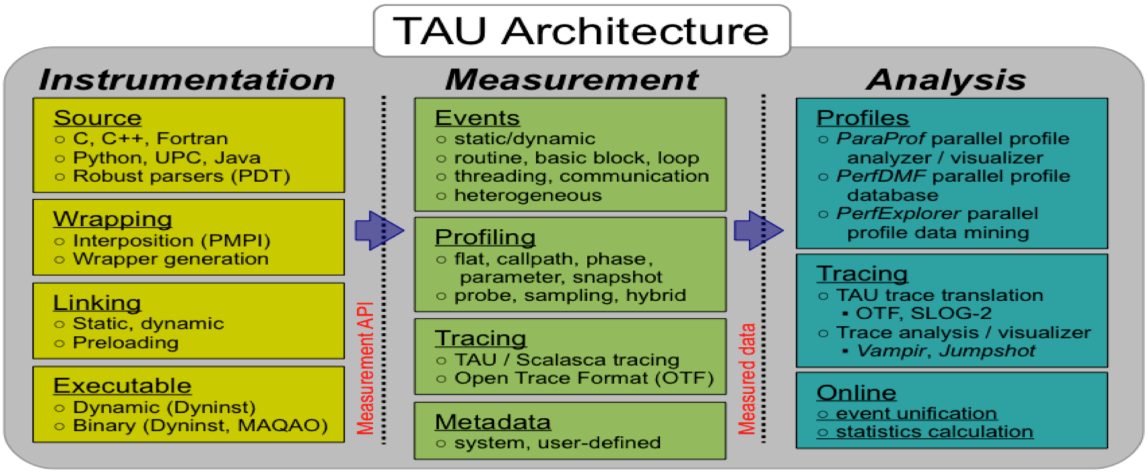


Figure 1. Architecture of TAU.

selection from the range of alternative methods TAU provides to assemble meaningful performance experiments that sheds light on the relevant performance properties. To this end, the TAU performance system offers support to the performance analysis in various ways, including powerful selective and multi-level instrumentation, profile and trace measurement modalities, interactive performance analysis analysis, and performance data management.

ACKNOWLEDGMENT

This work was supported by the National Science Foundation (NSF) grant number ACI-1450471. This work used the Extreme Science and Discovery Environment (XSEDE) that is supported by the NSF grant number ACI-1053575 and used allocation TG-ASC090010.

REFERENCES

[1] S. Shende and A. D. Malony, "The TAU Parallel Performance System," IJPCA, Vol 20, No. 2, pp. 287-311, 2006. http://tau.uoregon.edu.

[2] K. Huck and A. D. Malony, "PerfExplorer: A Performance Data Mining Framework for Large-Scale Parallel Computing," Proc. SC'2005, ACM, IEEE, 2005.

[3] U. Tennessee, Performance Application Programming Interface, http://icl.cs.utk.edu/papi, 2016.

[4] M. Geimer, S. Shende, B. Wesarg, and B. Wylie, "Practical Hybrid Parallel Application Performance Engineering," Tutorial, SC'15,Austin,TX, http://sc15.supercomputing.org/schedule/event_detail-evid=tut117.html, Nov. 16, 2015.

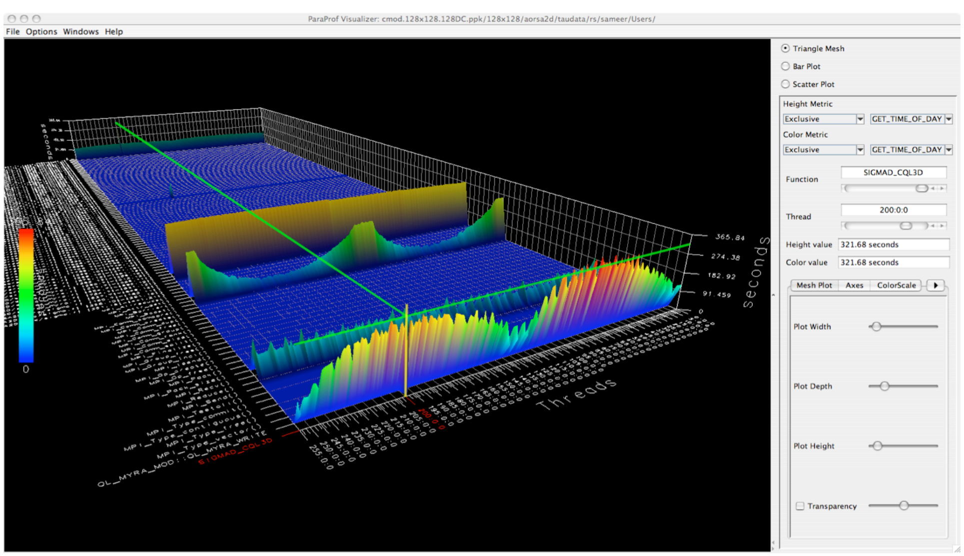


Figure 2. TAU's ParaProf 3D Profile Browser.

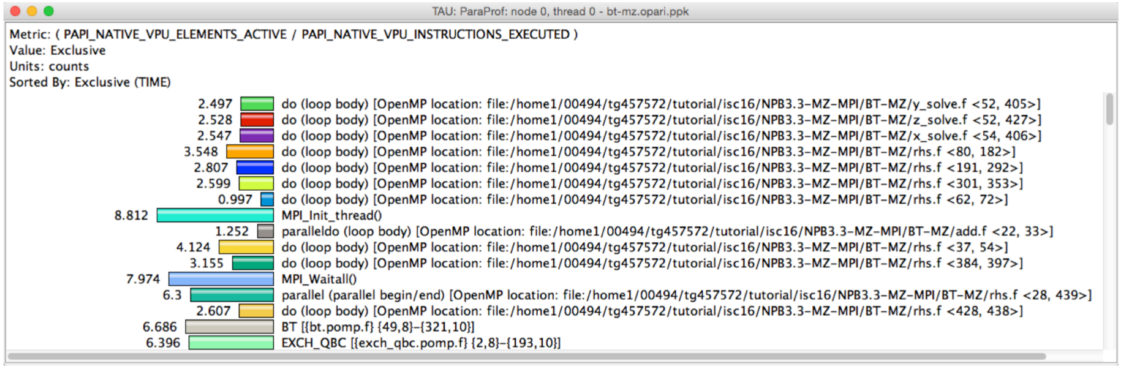


Figure 3. Vectorization Intensity sorted by exclusive time in TAU’s ParaProf Profile Browser.

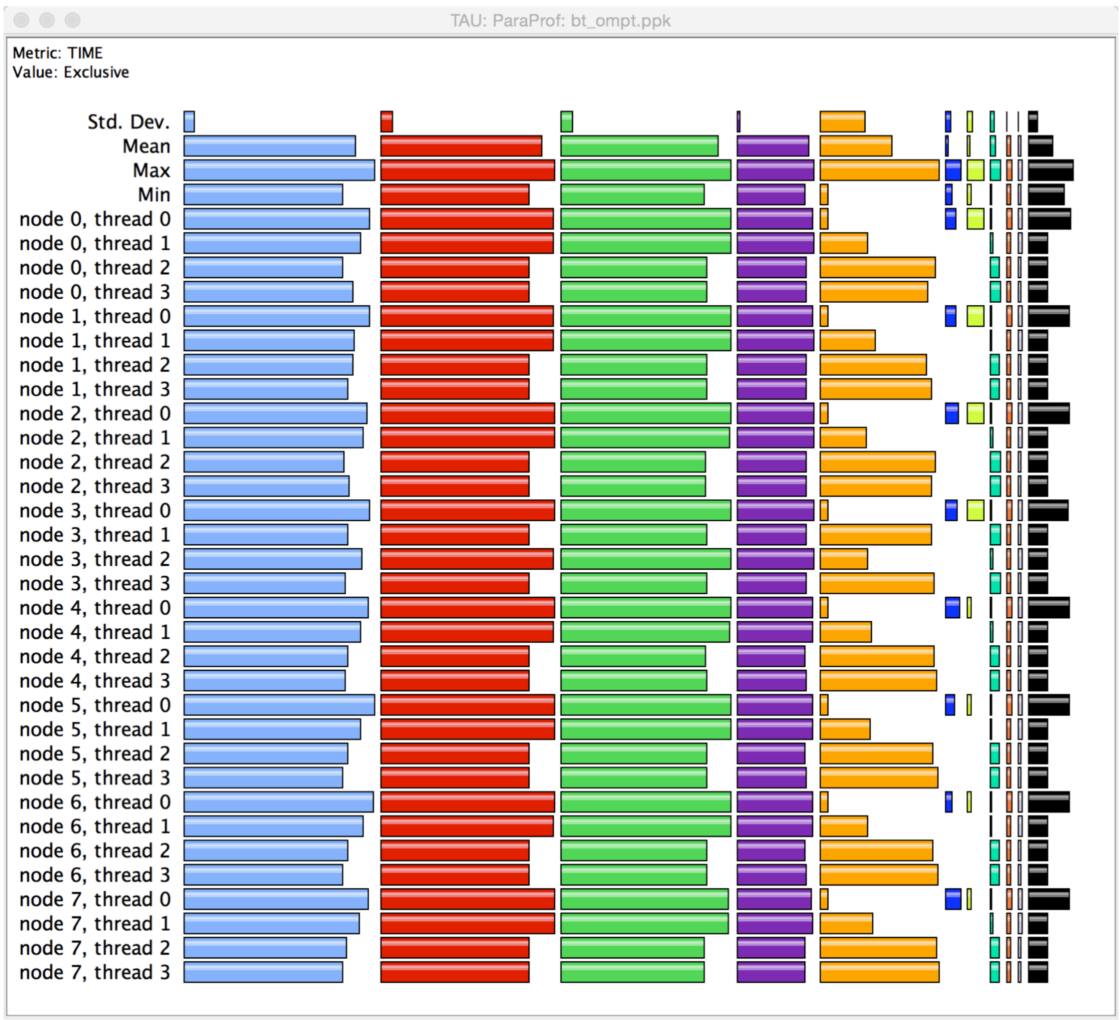


Figure 4. TAU's ParaProf Main Window.

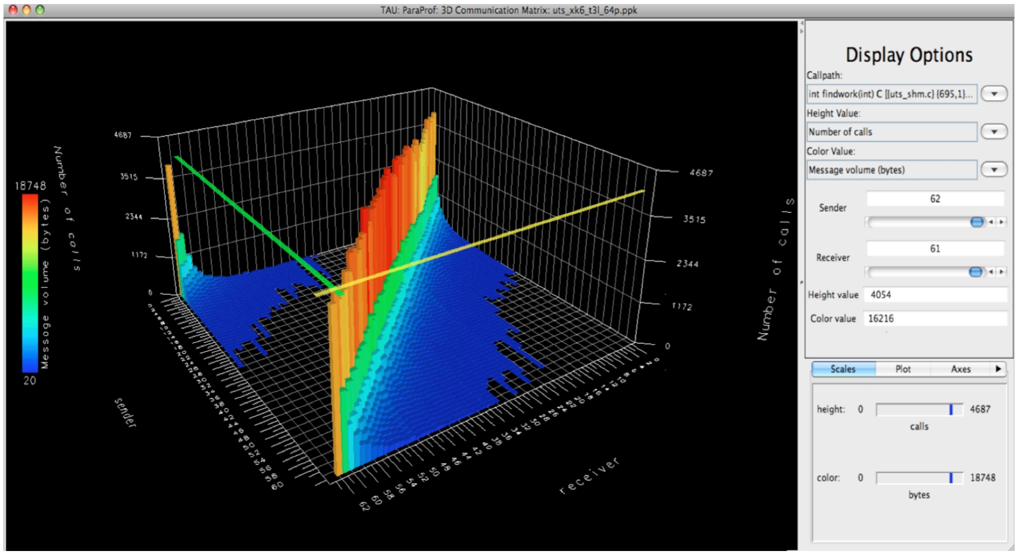
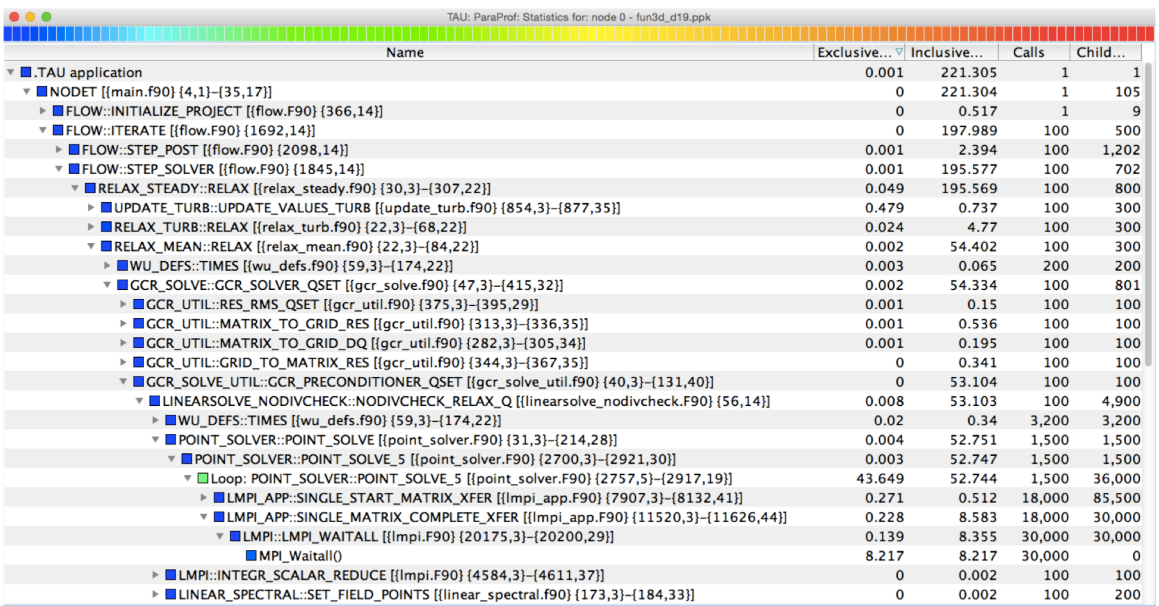


Figure 6. Communication matrix window in TAU's ParaProf.


TAU: ParaProf: Statistics for: node 0 - fun3d_d19.ppk

Name	Exclusive...	Inclusive...	Calls	Child...
.TAU application	0.001	221.305	1	1
NODET [{main.f90} {4,1}-{35,17}]	0	221.304	1	105
FLOW::INITIALIZE_PROJECT [{flow.F90} {366,14}]	0	0.517	1	9
FLOW::ITERATE [{flow.F90} {1692,14}]	0	197.989	100	500
FLOW::STEP_POST [{flow.F90} {2098,14}]	0.001	2.394	100	1,202
FLOW::STEP_SOLVER [{flow.F90} {1845,14}]	0.001	195.577	100	702
RELAX_STEADY::RELAX [{relax_steady.f90} {30,3}-{307,22}]	0.049	195.569	100	800
UPDATE_TURB::UPDATE_VALUES_TURB [{update_turb.f90} {854,3}-{877,35}]	0.479	0.737	100	300
RELAX_TURB::RELAX [{relax_turb.f90} {22,3}-{68,22}]	0.024	4.77	100	300
RELAX_MEAN::RELAX [{relax_mean.f90} {22,3}-{84,22}]	0.002	54.402	100	300
WU_DEFS::TIMES [{wu_defs.f90} {59,3}-{174,22}]	0.003	0.065	200	200
GCR_SOLVE::GCR_SOLVER_QSET [{gcr_solve.f90} {47,3}-{415,32}]	0.002	54.334	100	801
GCR_UTIL::RES_RMS_QSET [{gcr_util.f90} {375,3}-{395,29}]	0.001	0.15	100	100
GCR_UTIL::MATRIX_TO_GRID_RES [{gcr_util.f90} {313,3}-{336,35}]	0.001	0.536	100	100
GCR_UTIL::MATRIX_TO_GRID_DQ [{gcr_util.f90} {282,3}-{305,34}]	0.001	0.195	100	100
GCR_UTIL::GRID_TO_MATRIX_RES [{gcr_util.f90} {344,3}-{367,35}]	0	0.341	100	100
GCR_SOLVE_UTIL::GCR_PRECONDITIONER_QSET [{gcr_solve_util.f90} {40,3}-{131,40}]	0	53.104	100	100
LINEARSOLVE_NODIVCHECK::NODIVCHECK_RELAX_Q [{linearsolve_nodivcheck.F90} {56,14}]	0.008	53.103	100	4,900
WU_DEFS::TIMES [{wu_defs.f90} {59,3}-{174,22}]	0.02	0.34	3,200	3,200
POINT_SOLVER::POINT_SOLVE [{point_solver.F90} {31,3}-{214,28}]	0.004	52.751	1,500	1,500
POINT_SOLVER::POINT_SOLVE_5 [{point_solver.F90} {2700,3}-{2921,30}]	0.003	52.747	1,500	1,500
Loop: POINT_SOLVER::POINT_SOLVE_5 [{point_solver.F90} {2757,5}-{2917,19}]	43.649	52.744	1,500	36,000
LMPI_APP::SINGLE_START_MATRIX_XFER [{lmpi_app.F90} {7907,3}-{8132,41}]	0.271	0.512	18,000	85,500
LMPI_APP::SINGLE_MATRIX_COMPLETE_XFER [{lmpi_app.F90} {11520,3}-{11626,44}]	0.228	8.583	18,000	30,000
LMPI::LMPI_WAITALL [{lmpi.F90} {20175,3}-{20200,29}]	0.139	8.355	30,000	30,000
MPI_Waitall()	8.217	8.217	30,000	0
LMPI::INTEGR_SCALAR_REDUCE [{lmpi.F90} {4584,3}-{4611,37}]	0	0.002	100	100
LINEAR_SPECTRAL::SET_FIELD_POINTS [{linear_spectral.f90} {173,3}-{184,33}]	0	0.002	100	200

Figure 5. A callpath profile shown in a thread statistics window in ParaProf.