Workload Characterization using the TAU Performance System

Sameer S. Shende, Allen D. Malony, and Alan Morris

Performance Research Laboratory,
Department of Computer and Information Science
University of Oregon, Eugene, OR, USA,
{sameer,malony,amorris}@cs.uoregon.edu

Abstract. Workload characterization is an important technique that helps us understand the performance of parallel applications and the demands they place on the system. Each application run is profiled using instrumentation at the MPI library level. Characterizing the performance of the MPI library based on the sizes of messages helps us understand how the performance of an application is affected based on messages of different sizes. Partitioning of the time spent in MPI routines based on the type of MPI operation and the message size involved requires a two level mapping of performance data. This paper describes how performance mapping is implemented in the TAU performance system to support workload characterization.
Keywords: Performance mapping, measurement, instrumentation, performance evaluation, workload characterization

1 Introduction

Technology for empirical performance evaluation of parallel programs is driven by the increasing complexity of high performance computing environments and programming methodologies. To keep pace with the growing complexity of large scale parallel supercomputers, performance tools must provide for the effective instrumentation of complex software and the correlation of runtime performance data with system characteristics. Workload characterization is an important tool for understanding the the nature and performance of the workload submitted to a parallel system. Understanding the workload characteristics helps in correlating the effects of architectural features on workload behavior. It helps us in planning system capacity based on an assessment of the demands placed on the system. It helps us identify which components in a system may need to be upgraded. Performance data is collected for each application. Performance profiles contain valuable information such as the time spent in each message communication routine based on the message size. Profiling tools that focus their attention on capturing aggregate performance data over all invocations of message communication and I/O routines ignore the performance variation for small and large buffer sizes. In this paper, we describe the techniques for measuring the performance of a parallel application based on message buffer sizes. When this

information is gathered from several applications and stored in a performance database, we can classify the performance of the entire system using histograms that show the time spent in inter-process communication and I/O routines based on buffer sizes. We discuss the improvements that we made to the TAU performance system [3] in the areas of instrumentation, measurement and analysis to support workload characterization. Section §2 describes the related work in this area, Section §3 describes the TAU performance system, and describes how performance mapping is applied to characterize the performance of MPI routines based on the message sizes. Section §5 concludes the paper and we discuss future work in this section.

2 Related Work

IPM [1] is an integrated performance monitoring system that helps in workload characterization. IPM can characterize the application performance based on message sizes. IPM uses library preloading mechanism for instrumenting an application under Linux. The performance data is stored in a performance data repository which can be queried for application characteristics based on a number of parameters such as dates, and MPI performance data. PerfSuite [7] is another performance toolkit that builds a performance data repository based on execution time and hardware performance counters [11] to characterize the performance of an application and the system.

3 Mapping Performance Data

TAU[2] is an integrated, configurable, and portable profiling and tracing toolkit. It provides support for portable instrumentation, measurement, and analysis. Instrumentation calls can be inserted in TAU using a source-to-source translator tool *tau_instrumentor* based on the Program Database Toolkit (PDT)[8], using the MPI profiling interface, using DyninstAPI[12] for runtime instrumentation and re-writing the executable image, or using JVMPI[6] for instrumentation of Java programs. The TAU performance system provides an API for mapping low-level performance data to higher modes of abstraction. It uses the SEAA model [5] of mapping that provides support for both embedded and external associations. External associations use an external map (implemented as a hash table) to access performance data using a user specified key. The performance data is typically for begin/end timers or atomic events that are triggered at a certain place in the source code. We have built TAU's callpath[2] and phase profiling abstractions[4] using this mapping technology. Context events that map atomic events to the currently executing application callstack, are also implemented using TAU's mapping capabilities.

TAU's MPI wrapper interposition library helps us track the time spent in each MPI call. It defines each MPI routine and invokes timer calls before and after each name-shifted MPI interface call. It can access arguments that flow through the MPI routines. So, it can maintain user defined events that track

the sizes of messages for each MPI call. We have extended this wrapper to use a two level mapping involving the identifier of the MPI call and the size of the message buffer as a key to access a timer that stores the performance data of that call. By using these two fields, we can determine if a given message buffer size and call have occured in the past. If not, a new timer is created with a name that embeds the nature of the MPI call as well as the buffer size. Thus, we can track the time spent in an MPI call using mapping techniques to characterize the performance based on individual message sizes.

4 Performance Experimentation

The use of the TAU performance system involves the coordination of several steps: instrumentation selection, measurement configuration, compilation and linking with the application, application execution and generation of performance data on the target platform, and performance data storage for analysis. We describe the sequence of these steps as a performance experiment. We use the term experiment generally to denote a specific choice of instrumentation and measurement for a specific application code, but what this means exactly should be left to the user. We define the term trial to mean an instance of an experiment. A trial might either repeat an experiment run (e.g., to determine performance variation) or modify an experiment run parameter (e.g., number of processors), which would not represent such a significant change as to constitute a new experiment.

The performance data gathered from executing the application is stored in TAU's performance database, PerfDMF [9] which is then queried by the ParaProf profile browser and other analysis tools such as PerfExplorer [10] for performance data mining operations. The performance data stored in PerDMF is multi-variate and multi-dimensional, both within single trials and experiments as well as across experiments, applications, and platforms. PerfExplorer is a framework for parallel performance data mining and knowledge discovery – finding out new performance facts and relationships as the outcome of searching and analyzing the stored performance data.

5 Conclusion

In the process of workload characterization for high performance parallel systems, it is important to have portable and configurable tools that can target the different performance features and experiments of interest. Presently, the TAU performance system has such capabilities for steps in this process, from common event instrumentation, profile and trace measurements and data analysis to meet workload characterization objectives. However, there are aspects of experiment automation and knowledge discovery that require innovation and performance technology and tools. Tool integration is also important. The full paper will describe in more detail how workload characterization requirements can be met by TAU and other systems.

6 Acknowledgments

Research at the University of Oregon is sponsored by contracts (DE-FG02-05ER25663, DE-FG02-05ER25680) from the MICS program of the U.S. Dept. of Energy, Office of Science.

References

1. J. Borrill, J. Carter, L. Oliker, D. Skinner, R. Biswas, "Integrated Performance Monitoring of a Cosmology Application on Leading HEC Platforms," In Proc. of International Conference on Parallel Processing (ICPP 2005), pp. 119–128, IEEE, 2005.
2. S. Shende, and A. Malony, "The TAU Parallel Performance System," In International Journal of High Performance Computing Applications, ACTS Collection Special Issue, Summer 2006.
3. A. Malony, S. Shende, "Performance Technology for Complex Parallel and Distributed Systems," In G. Kotsis, P. Kacsuk (eds.), *Distributed and Parallel Systems, From Instruction Parallelism to Cluster Computing, Third Workshop on Distributed and Parallel Systems (DAPSYS 2000)*, Kluwer, pp. 37–46, 2000.
4. A. D. Malony, S. Shende, and A. Morris, "Phase-Based Parallel Performance Profiling," In Proceedings of the PARCO 2005 conference, 2005.
5. S. Shende, "The Role of Instrumentation and Mapping in Performance Measurement," Ph.D. Dissertation, University of Oregon, August 2001.
6. S. Shende, and A. D. Malony, "Integration and Application of TAU in Parallel Java Environments," In Concurrency and Computation: Practice and Experience, Volume 15(3-5), Mar-Apr 2003, Wiley, pp. 501–519, 2003.
7. R. Kufrin, "PerfSuite: An Accessible, Open Source Performance Analysis Environment for Linux," In Proceedings of the 6th International Conference on Linux Clusters: The HPC Revolution 2005 (LCI-05), 2005.
8. K. Lindlan, J. Cuny, A. Malony, S. Shende, B. Mohr, R. Rivenburgh, C. Rasmussen, "A Tool Framework for Static and Dynamic Analysis of Object-Oriented Software with Templates," SC 2000 conference, 2000.
9. K. A. Huck, A. D. Malony, R. Bell, and A. Morris, "Design and Implementation of a Parallel Performance Data Management Framework," In Proceedings of International Conference on Parallel Processing (ICPP 2005), IEEE Computer Society, 2005.
10. K. A. Huck, and A. D. Malony, "PerfExplorer: A Performance Data Mining Framework for Large-Scale Parallel Computing," In Proceedings of SC 2005 conference, ACM, 2005.
11. S. Browne, J. Dongarra, N. Garner, G. Ho, and P. Mucci, "A Portable Programming Interface for Performance Evaluation on Modern Processors," *International Journal of High Performance Computing Applications*, **14**(3):189–204, Fall 2000.
12. B. Buck and J. Hollingsworth, "An API for Runtime Code Patching", *Journal of High Performance Computing Applications*, pp. 317–329, 14(4), 2000.