

Workload Characterization using the TAU Performance System

Sameer 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. It can be used to describe performance effects due to application parameters, compiler options, and platform configurations. In this paper, workload characterization features in the TAU parallel performance system are demonstrated for elucidating the performance of the MPI library based on the sizes of messages. Such characterization partitions the time spent in the MPI routines used by an application based on the type of MPI operation and the message size involved. It requires a two-level mapping of performance data, a unique feature implemented in TAU. Results from the NPB LU benchmark are presented. We also discuss the use of mapping for memory consumption 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, and in identifying which components in a system may need to be upgraded. This is a *systems perspective* on workload characterization. There is also an *application perspective* that characterizes application-specific performance behavior in the context of workload and platform aspects. For instance, in this paper, we use workload characterization techniques recently implemented in the TAU performance system [1] to study message communication performance.

Workload characterization methods collect performance data for each application in the workload set. For instance, performance profiles can contain statistics on performance in application regions (e.g., routines) and with respect to specific behaviors, such as message communication 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. It is this ability to expose application features and observe their performance effects that we are interested in supporting as part of a workload characterization methodology.

In this paper, we describe the techniques for measuring the performance of a parallel application's message communication 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 [1] 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 reports on our experience with message communication characterization of the NPB LU benchmark. We have also applied performance mapping to memory usage characterization. Brief discussion is given to workload characterization of memory consumption. Section §6 concludes the paper and we discuss future work in this section.

2 Related Work

Workload characterization is a rich area in performance evaluation research. Our specific interest is in workload characterization for high-performance computing. There are two projects of related interest to our work.

The OpenWLC [2] system is a scalable, integrated environment for systematically collecting the monitored data and applying workload characterization techniques to raw data produced by monitoring application programs. OpenWLC's framework employs a component-based, multi-tier, architecture to cope with large amounts of monitored data during collection, storage, visualization and analysis stages.

IPM [3] is an integrated performance monitoring system developed at the Lawrence Berkeley Laboratory (LBL) for use at the National Energy Research Supercomputing Center (NERSC). IPM is in active use for application performance analysis and workload characterization. Specific to our interests, IPM can characterize the application performance based on message sizes. It uses library preloading mechanisms for instrumenting an application under Linux and on other platforms where preloading of shared libraries is available. 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 execution

```

#include <stdio.h>
#include <mpi.h>

int main(int argc, char **argv) {
    int rank, size, i, j;
    int buffer[16*1024*1024];
    MPI_Init(&argc, &argv);
    MPI_Comm_size( MPI_COMM_WORLD, &size );
    MPI_Comm_rank( MPI_COMM_WORLD, &rank );
    for (i=0;i<1000;i++)
        for (j=1;j<16*1024*1024;j*=2) {
            if (rank == 0) {
                MPI_Send(buffer, j, MPI_INT, 1, 42, MPI_COMM_WORLD);
            } else {
                MPI_Status status;
                MPI_Recv(buffer, j, MPI_INT, 0, 42, MPI_COMM_WORLD, &status);
            }
        }
    MPI_Finalize();
}

```

Fig. 1. Message Size Characterization Instrumentation

date and MPI performance data. LBL has implemented a web-based interface for this purpose.

Certainly, other application performance measurement tools can be applied to workload characterization. However, the ability to store multi-experiment performance data, including metadata about compiler and system parameters, is important criteria for workload characterization support. PerfSuite [4] is a performance toolkit that builds a performance data repository based on execution time and hardware performance counters [5] to characterize the performance of an application and the system. TAU can work with PerfSuite and other tools to integrate performance results across applications and platforms.

3 Workload Characterization and Performance Mapping

Workload characterization analyzes the effects of application execution in a system context. Application measurements could be made of total performance, such as total execution time, but finer granularity measurements can better identify workload effects specific to program properties. However, certain properties require a measurement system that can observe execution parameters and characterize application performance based on unique parameters instances. The general concept is one of *performance mapping*, wherein an association can be established between low-level performance data and high-level measurement abstractions, specialized by program semantics. The TAU performance system is able to support performance mapping for workload characterization.

TAU[1] is an integrated, configurable, and portable profiling and tracing toolkit. It provides support for portable instrumentation, measurement, and

```
% icc mpi.c -lmpi
% mpirun -np 2 tau_load.sh -XrunTAU-icpc-mpi-pdt a.out
```

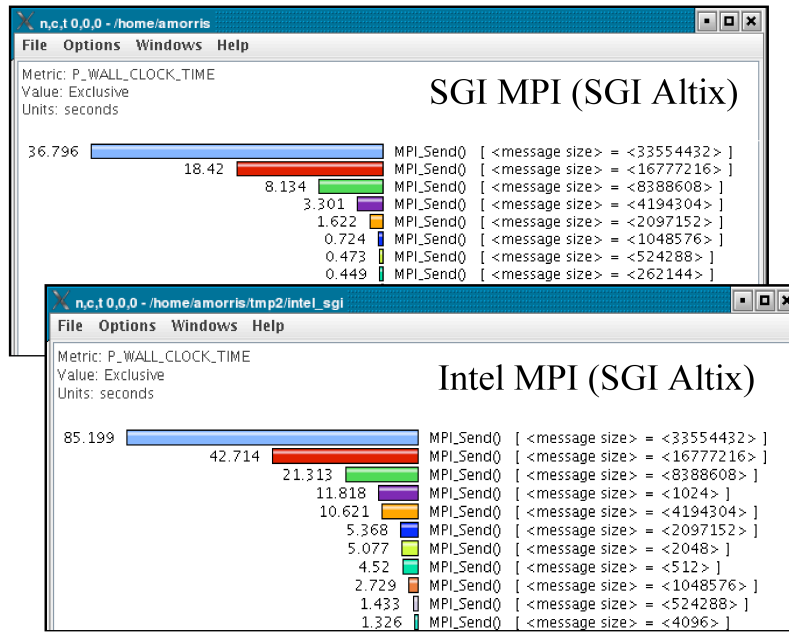


Fig. 2. MPI (SGI vs. Intel) Message Characterization

analysis. Instrumentation calls can be inserted in TAU at the source level, the library level, the binary code level, and even in a virtual machine. Unique in the TAU performance system is an instrumentation API for performance mapping. It uses the SEAA model [6] 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 collected for interval events or atomic events that are triggered at a certain place in the program code. Performance mapping is a powerful concept and technology. It has been used in TAU for callpath profiling [1] and phase profiling [7]. Context events that map atomic events to the currently executing application callstack, are also implemented using TAU's mapping capabilities. Here we apply performance mapping to MPI communication characterization.

TAU's MPI wrapper interposition library helps us track the time spent in each MPI call. It defines a separate name-shifted MPI interface for each MPI routine that can be used to invoke timer calls at routine entry and exit. This mechanism can also be used to access arguments that flow through the MPI routines. Hence, measurement code could be created to track the sizes of mes-

sages for each MPI call. We have followed this approach using TAU's mapping technology to implement a two-level map of the MPI call ID and the size of the message buffer used in the call. With this data, we can determine if a given message buffer size and call have occurred in the past. If not, a new performance structure is created with a name that embeds the MPI call ID and buffer size. At the end of the application, we obtain the performance in each invoked MPI call for each message size used.

In general, TAU can take any routine parameter and create a performance mapping. The measurement library implements routines for different parameter types, such as `TAU_PROFILE_PARAM1L(value, "name")`. The following code segment shows how this is used:

```
void foo(int input) {
    TAU_PROFILE("foo", "", TAU_DEFAULT);
    TAU_PROFILE_PARAM1L(input, "input");
    ...
}
```

When the measurement library is configured with `-PROFILEPARAM`, the parameter mapping API is enabled.

Figure 1 shows a simple program for message communication of different sizes. Figure 2 shows profile output characterizing communication performance for different MPI libraries, SGI and Intel. With such information, we can obtain a better understanding of workload effects. Also shown is the experiment compilation and run commands.

4 Performance Experimentation

Performance experimentation and results management are important components for any workload characterization system. 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 [8] which is then queried by the ParaProf profile browser and other analysis tools such as PerfExplorer [9] for performance data mining operations. The performance data stored in PerfDMF is multi-variate and multi-dimensional, both within single trials and experiments as

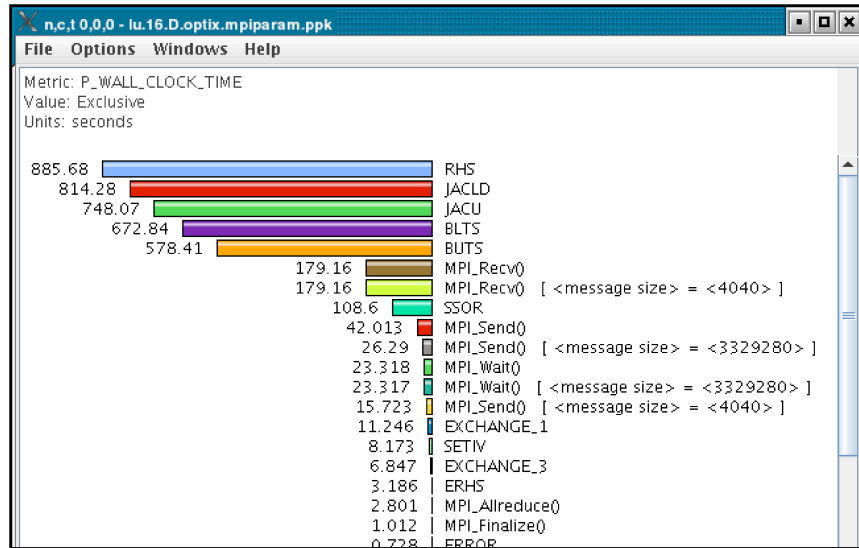


Fig. 3. Profile of LU Benchmark on SGI Altix

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 Workload Characterization Experiments

To demonstrate TAU’s mapping support for workload characterization, the NAS parallel benchmark LU is used as a testcase. Specifically, we are interested in understanding how this MPI benchmark behaves respective of its message communication. TAU’s message size mapping was enabled and experiments were run on a SGI Altix platform. We also can capture memory usage statistics using mapping technologies.

5.1 MPI Message Size Characterization

Each performance experiment ran captured execution time performance for the LU routines. For the MPI routines, execution time performance was broken down based on message size. Figure 3 shows an example (flat) parallel profile for one process of a 16-process LU execution. Seen are the times spent in routines in decreasing order. Most of the time is spent in computation, but message communication is also significant. The communication event IDs encode the size of the message in the names. The majority of the MPI_Recv time was spent receiving messages with 4040 bytes.

Name	Inclusive ...	Calls	Child ...
MPI_Comm_free()	0	1	0
MPI_Comm_rank()	0	1	0
MPI_Comm_size()	0	2	0
MPI_Finalize()	1.012	1	0
MPI_Init()	0.004	1	0
MPI_Irecv()	0.047	612	0
MPI_Recv()	179.165	244,412	0
MPI_Recv() [<message size> = <4040>]	179.165	244,412	0
MPI_Send()	42.013	245,020	0
MPI_Send() [<message size> = <3329280>]	26.29	608	0
MPI_Send() [<message size> = <4040>]	15.723	244,412	0
MPI_Wait()	23.318	612	0
MPI_Wait() [<message size> = <1632>]	0	1	0
MPI_Wait() [<message size> = <1664>]	0	1	0
MPI_Wait() [<message size> = <3264>]	0.001	2	0
MPI_Wait() [<message size> = <3329280>]	23.317	608	0
NEIGHBORS	0	1	0
NODEDIM	0	1	0
PINTGR	0.008	1	6
PRINT_RESULTS	0	1	0

Fig. 4. Message Size Characterization for LU Benchmark

Further analysis of the message characterization shows the distribution of each MPI operation across the message size used for that operation. Figure 4 highlights the inclusive time of MPI_Send and the number of calls for one LU process. Here it is seen that a relatively large number of small messages were sent, accounting for approximately 37% of MPI_Send's total time.

5.2 LU Memory Usage Characterization

TAU performance mapping can also be used to characterize memory usage. This can show how memory is allocated, in what size chunks, and the amount of free space available. Figure 5 displays the heap memory utilization for LU on four processes.

6 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. A novel feature of TAU is its performance mapping technology. The presentation above demonstrates how mapping can be used to characterize message communication and memory usage.

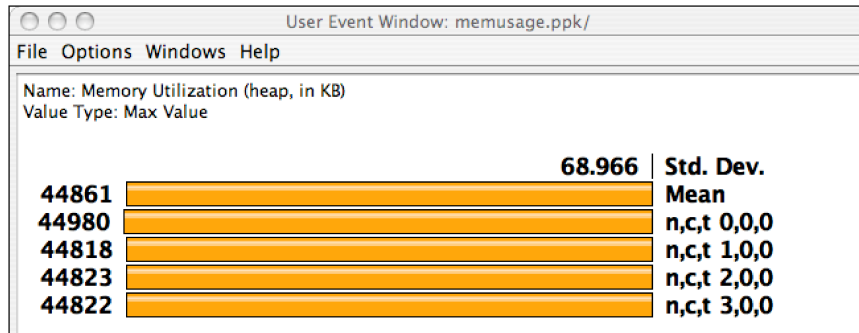


Fig. 5. Memory Consumption Tracking for LU Benchmark

Our objectives in the future include better support for experiment automation and knowledge discovery for workload characterization. We are also working to integrate our tools with IPM.

7 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. S. Shende and A. D. Malony, "The TAU Parallel Performance System," *International Journal of High Performance Computing Applications*, SAGE Publications, 20(2), pp. 287–331, Summer 2006.
2. H. Ong, R. Subramaniyan, C. Leangsuksun, and S. Studham, "OpenWLC: A Scalable Workload Characterization System," *High Availability and Performance Workshop*, in conjunction with *Sixth LACSI Symposium*, Oct. 11-13, 2005. <http://xcr.cenit.latech.edu/wlc/index.php?title=PUBLICATIONS>
3. 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.
4. R. Kufirin, "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.
5. 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.
6. S. Shende, "The Role of Instrumentation and Mapping in Performance Measurement," Ph.D. Dissertation, University of Oregon, August 2001.

7. A. D. Malony, S. Shende, and A. Morris, "Phase-Based Parallel Performance Profiling," In Proceedings of the PARCO 2005 conference, 2005.
8. 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.
9. 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.