A Scalable Approach to MPI Application Performance Analysis

Shirley Moore[1] and Felix Wolf[1] and Jack Dongarra[1] and Sameer Shende[2] and Patricia Teller[3] and Bernd Mohr[4]

[1] Innovative Computing Laboratory, University of Tennessee, Knoxville, TN 37996 USA
[2] Univeristy of Oregon
[3] University of Texas-El Paso
[4] Forschungszentrum Jülich, ZAM, 52425 Jülich, Germany

Abstract.

1 Introduction

Parallel computing is playing an increasingly critical role in advanced scientific research as simulation and computaion are becoming widely used to augment and/or replace physical experiments. However, the gap between peak and achieved performance for scientific applications runnning on large parallel systems has grown considerably in recent years. The most common parallel programming paradigm for these applications is to use Fortran or C with MPI message passing to implement parallel algorithms. The complex architectures of large parallel systems present difficult challenges for performance optimzation of such applications. Tools are needed that collect and present relevant informaiton on application performance in a scalable manner so as to enable developers to easily identify and determine the causes of performance bottlenecks.

Performance data encompasses both profile data and trace data. Profiling involves the collect of statistical summaries of various performance metrics broken down by program entities such as routines and nested loops. Performance metrics include time as well as hardware counter metrics such as operation counts and cache and memory event counts. Tracing involves collection of a timestamped sequence of events such as entering and exiting program regions and sending and receiving messages. Profiling can identify regions of a program that are consuming the most resources, while detailed tracing can help identify the causes of performance problems. On large parallel systems, both profiling and tracing present scalability challenges.

Collecting either profile or trace data requires the application program to be instrumented. Instrumentation can be inserted at various stages of the program build process, ranging from source code insertion to compile time to link time to run time options. Although many tools provide an application programmer interface (API), manual insertion of instrumentation library calls into application source code is too tedious to be practical for large-scale applications. Thus our tools support a range of automated instrumentation options that require the user to make only minor changes to the application makefile to invoke a source-to-source preprocessor or compiler options as needed and link in any necessary libraries. Once an instrumented version of the program has been built, only a few environment variables need to be set to control runtime collection of profile and/or trace data.

Collecting profile data for several different metrics on a per-process and per-routine basis, possibly for several runs with different numbers of processors and/or different test cases and/or on different platforms results in a data management problem as well as a presentation and analysis problem. Similar profile data may be collected by different tools but be incompatible because of different data formats. Our solution to the data management problems is a performance data management framework that sits on top of a relational database and provides a common profile data model as well as interfaces to various profile data collection and analysis tools. For presentation of profile data we have developed graphical tools that display the data in 2-dimensional and 3-dimensional graphs. Our tools support multi-experiment analysis of profile data collected from different runs. Because display of a large number of metrics can still be overwhelming, we have also implemented prototype tools that reduce the data by means of multivariate statistical analysis techniques such as principal component analysis and cluster analysis.

Event tracing is a powerful method for analyzing the performance behavior of parallel applications. Because event traces record the temporal and spatial relationships between individual runtime events, they allow application developers to analyze dependences of performance phenomena across concurrent control flow. While event tracing enables the identification of performance problems on a high level of abstraction, it suffers from scalability problems associated with trace file size. In view of present and future architectures consisting of thousands of processors and applications running on at least a major fraction of them, the event tracing approach will become constrained by the potentially enormous size of the resulting event traces. One approach to improving the scalability of event tracing is to reduce the amount of trace data that is collected. One can use filtering to collect trace data for only those routines that are relevant to the analysis to be performed. Our approach uses call-path profiling to determine which routines are relevant and then traces only those routines. Graphical tools such as Vampir [], Intel Trace Analyzer [], and Jumpshot [] are available to view trace files collected for parallel executions. These tools typically show a timeline view of state changes and message passing events. However, analyzing these views for performance bottlenecks can be like searching for a needle in a haystack. Our approach searches the trace file using pattern-matching to automatically identify instances of inefficient behavior. The performance bottlenecks that are found and related to specific program call-paths and node/process/thread locations and can then be focused on using one of the previously mentioned trace file viewing tools.

The remainder of this paper is organized as follows. Section 2 describes our automated approach to insertion of performance instrumentation and collection of performance data. Section 3 describes our performance data management framework. Section 4 describes our scalable approaches to analyzing profile data, including techniques for multi-experiment analysis. Section 5 describes our scalable automated approach to trace file analysis. Section 6 contains conclusions and directions for future research.

2 Automated Performance Instrumentation

3 Performance Data Management Framework

4 Scalable Display and Analysis of Profile Data

5 Automated Analysis of Trace Data

6 Conclusions and Future Work