ELSEVIER

Parallel Computing 28 (2002) 1409–1449

PARALLEL COMPUTING

www.elsevier.com/locate/parco

Instrumentation database system for performance analysis of parallel scientific applications ☆

Jeffrey Nesheiwat, Boleslaw K. Szymanski *

Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY 12180, USA

Received 31 October 2000; received in revised form 8 November 2001; accepted 11 June 2002

Abstract

The complexity and computational intensity of scientific computing has fueled research on parallel computing and performance analysis. The purpose of this paper is to present a novel approach to performance analysis of large parallel programs. At the core of this approach is an instrumentation database (IDB) that enables comparative analysis of parallel code performance across architectures and algorithms.

The basis of the IDB approach is scalable collection of performance data so that problem size and run-time environments do not affect the amount of information collected. This is achieved by uncoupling performance data collection from the underlying architecture and associating it with the control flow graph of the program. An important contribution of the IDB approach is the use of database technology to map program structure onto relational schema that represents the control flow hierarchy, its corresponding statistical data, and static information that describes the execution environment.

To demonstrate the benefits of the proposed approach, we have implemented a POSIX compliant probe library, automated instrumentation tool, front-end visualization programs, database schema using an object-relational DBMS (PostgreSQL), and SQL queries. We also developed a methodology, based on these tools, for interactive performance analysis and demonstrated this methodology on several different parallel scientific applications.

Keywords: Performance analysis; Database; Instrumentation; High performance computing

☆ This paper is based on a talk presented at the High Performance Computing (HPC 2000) workshop, 12–15 June 2000, Cetraro, Italy.

* Corresponding author.

E-mail addresses: neshj@cs.rpi.edu (J. Nesheiwat), szymansk@cs.rpi.edu (B.K. Szymanski).

URL: http://www.cs.rpi.edu/~szymansk.

0167-8191/02/$ - see front matter
PII: S0167-8191(02)00149-7

1. Introduction

Since the primary reason for writing parallel codes is speed [1], it comes as no surprise that performance analysis is a vital part of the development process. Analysis tries to determine if a given algorithm is as fast as it can be, where the program can be further optimized, and how efficiently the underlying system is being used. We present the instrumentation database (IDB) system which comprises of the following three primary components:

- automated instrumentation tool,
- probe library and multi-language application programmer's interface (API),
- experiment definition file tool (EDFtool) and visualization tool (Vistool) graphical front-end.

We demonstrate the features of the system and resulting methodology of its use by evaluating performance results of several scientific computing applications.

2. State of the art

In this section, we briefly review some of the parallel performance analysis tools (PATs) focusing on main differences between them and the IDB system.

2.1. Upshot

The MPIch implementation of the message passing interface [2,3] comes bundled with Upshot, a TCL/Tk program used to visualize the performance of MPI programs. Upshot works as a visual front-end for MPE, a profiling interface for MPI. Instrumentation is added when the user links in MPI profiling libraries by specifying command-line options at compile-time. Upshot provides a graphical front-end to the resulting profiler data for MPI specific functions. Instrumentation is transparent to the user because MPI functions are overloaded with calls to instrumented functions in the profiling library. The user can further extend instrumentation by taking the additional step of adding calls in their program that explicitly measure additional code segments.

Gantt charts are used to show processor state and arrows between processors represent message passing. Gantt charts are a powerful Vistool but suffer from poor scalability. Fig. 1 shows an Upshot visualization of an adaptive finite element application with heavy interprocessor communication. Heavy communication, many processing elements, or long running programs quickly make this type of visualization unwieldy for analysis. Upshot's usefulness if bound by the size of the application being analyzed.

Parallel code developers often need to answer different performance questions from the same instrumentation data set. Upshot does not provide the flexibility of multiple visualization modes to meet this need.