Empirical Optimization for a Sparse Linear Solver: A Case Study

Yoon-Ju Lee, Pedro C. Diniz, Mary W. Hall, and Robert Lucas

This paper describes initial experiences with semi-automated performance tuning of a sparse linear solver in LS-DYNA, a large, widely used engineering application. Through a collection of tools supporting empirical optimization, we alleviate the burden of performance tuning for mapping today's sophisticated engineering software to increasingly complex hardware platforms. We describe a tool that automatically isolates code segments to create benchmark subsets for the purposes of performance tuning. We present a collection of automatically generated empirical results that demonstrate the sensitivity of the application's performance to optimization parameters. Through this case study, we demonstrate the importance of developing automatic performance tuning support for performance-sensitive applications.

KEY WORDS: Memory hierarchy optimization; performance tuning.

1. INTRODUCTION

Developers of today's scientific and engineering applications for high-end computing platforms spend an inordinate amount of their time tuning the performance of their application, often far more time than is required to achieve an initial, working implementation. Further, performance tuning must be repeated each time the code is ported to a new architecture. The process of manual performance tuning involves focusing in on a few key optimization parameters and applying empirical techniques to improve the performance of the application. Unfortunately, the performance of many applications is sensitive to the choice of these optimization parameters, and the process of performance tuning is often time-consuming.

1 Information Sciences Institute, University of Southern California, 4676 Admiralty Way, Suite 1001, Marina del Rey, CA 90292-6695, USA. E-mail: \{yoonju, pedro, mhall, rflucas\}@isi.edu

0885-7458/05/0600-0165 © 2005 Springer Science+Business Media, Inc.
computational components of an application. For each component, the programmer derives a sequence of different implementations for performing the same computation. Each variant is first debugged, and then its performance characteristics are evaluated. This lengthy process continues until the programmer either arrives at a suitable variant or, as is often the case, decides to give up on further performance tuning. New variants may be required for different input data sets, or as the application is ported to other platforms.

We illustrate the complexity of this process with an example taken from LS-DYNA. The LS-DYNA application is a general-purpose, non-linear finite element program capable of solving a vast array of engineering and design problems ranging from bioprosthetic heart valve operation to automotive crash and earthquake engineering.

The LS-DYNA application is a commercial derivative of DYNA originally developed at Lawrence Livermore National Laboratory. It was originally designed to take explicit time steps, allowing it to model strong shocks and waves. Recently it has been enhanced to take implicit time steps as well. This improves both accuracy and time to solution in problems like springback. The computational bottleneck of implicit LS-DYNA is the solution of a large sparse system of symmetric indefinite linear equations. The default multifrontal sparse solver used in LS-DYNA was developed by one of the authors and is the computation that is the focus of this paper.

The current implementation has been ported to four different parallel platforms, requiring substantial performance tuning at each port. The code is primarily written in FORTRAN, but different parallel versions use OpenMP, MPI and architecture-specific language extensions. Certain parameters of the algorithm are known by the author to be important to the application’s sequential and parallel performance, and certain algorithms are more appropriate, depending on execution context. For this reason, implementation variants of the solver’s sub-computations have been developed to improve performance for particular problem sizes and architectures, corresponding to different algorithms and different parameter values.

The goal of this paper is to examine in detail the performance tuning process of the application developer for this solver and use this as a guide towards developing effective tool support to enhance programmer productivity and improve the quality of the result. A key observation is that much of what the developer did could be systematized and automated.