Chapter 25
Runtime Adaption Techniques for HPC Applications

Edgar Gabriel
University of Houston, USA

ABSTRACT
This chapter discusses runtime adaption techniques targeting high-performance computing applications. In order to exploit the capabilities of modern high-end computing systems, applications and system software have to be able to adapt their behavior to hardware and application characteristics. Using the Abstract Data and Communication Library (ADCL) as the driving example, the chapter shows the advantage of using adaptive techniques to exploit characteristics of the network and of the application. This allows to reduce the execution time of applications significantly and to avoid having to maintain different architecture dependent versions of the source code.

INTRODUCTION
High Performance Computing (HPC) has reshaped science and industry in many areas. Recent ground-breaking achievements in biology, drug design and medical computing would not have been possible without the usage of massive computational resources. However, software development for HPC systems is currently facing significant challenges, since many of the software technologies applied in the last ten years have reached their limits. The number of applications being capable of efficiently using several thousands of processors or achieving a sustained performance of multiple teraflops is very limited and is usually the result of many person-years of optimizations for a particular platform. These optimizations are however often not portable. As an example, an application optimized for a commodity PC cluster performs (often) poorly on an IBM Blue Gene or the NEC Earth Simulator. Among the problems application developers face are the wide variety of available hardware and software components, such as

DOI: 10.4018/978-1-60566-661-7.ch025
and the influence of each of these components on the performance of their application. Hence, an end-user faces a unique execution environment on each parallel machine he uses. Even experts struggle to fully understand correlations between hardware and software parameters of the execution environment and their effect on the performance of a parallel application.

**Motivating Example**

In the following, we would like to clarify the dilemma of an application developer using a realistic and common example. Consider a regular 3-dimensional finite difference code using an iterative algorithm to solve the resulting system of linear equations. The parallel equation solver consists of three different operations requiring communication: scalar products, vector norms and matrix-vector products. Although the first two operations do have an impact on the scalability of the algorithm, the dominating operation from the communication perspective is the matrix-vector product. The occurring communication pattern for this operation is neighborhood communication, i.e. each process has to exchange data with its six neighboring processes multiple times per iteration of the solver. Depending on the execution environment and some parameters of the application (e.g. problem size), different implementations for the very same communication pattern can lead to optimal performance. We analyze the execution times for 200 iterations of the equation solver applied for a steady problem using 32 processes on the same number of processors on a state-of-the-art PC cluster for two different problem sizes (32×32×32 and 64×32×32 mesh points per process) and two different network interconnects (4×InfiniBand and Gigabit Ethernet). The neighborhood communication has been implemented in four different ways, named here *fcfs, fcfs-pack, ordered, overlap*. While the nodes/processors have been allocated exclusively for these measurements using a batch-scheduler, the network interconnect was shared with other applications using the same PC cluster.

The results indicate, that already for this simple test-case on a single platform three different implementations of the neighborhood communication lead to the best performance of this application: although the differences between the different implementations are not dramatic over this network interconnect, *fcfs* shows the best performance for both problem sizes when using the InfiniBand interconnect. This implementation is initiating all required communications simultaneously using asynchronous communication followed by a Waitall operation on all pending messages. However, for the Gigabit Ethernet interconnect the *fcfs* approach seems to congest the network. Instead, the implementation which is overlapping communication and computation (*overlap*), is showing the best performance for the small problem size (6.2 seconds for *overlap* vs. 6.6 seconds for *fcfs*, 7.5 seconds for *fcfs-pack*, 8.1 seconds for *ordered*) while the *ordered* algorithm, which limits the number of messages concurrently on the fly, is the fastest implementation for the large problem size for this network interconnect (14.7 seconds for *ordered* vs. 26.9 seconds for *fcfs*, 19.9 for *fcfs-pack* and 23.4 seconds for *overlap*). The implementation considered to be the fastest one over the InfiniBand network leads thus to a performance penalty of nearly
Related Content

A New Parallel Data Cube Construction Scheme

Cell Processing for Two Scientific Computing Kernels
[www.igi-global.com/chapter/cell-processing-two-scientific-computing/36414](www.igi-global.com/chapter/cell-processing-two-scientific-computing/36414)

Speculative Scheduling of Parameter Sweep Applications Using Job Behavior Descriptions
[www.igi-global.com/article/speculative-scheduling-parameter-sweep-applications/2166](www.igi-global.com/article/speculative-scheduling-parameter-sweep-applications/2166)

Performance Evaluation of Cloud Data Centers with Batch Task Arrivals