Concurrent programming constructs for parallel MPI applications
The MPI threads library

Tobias Berka · Giorgos Kollias · Helge Hagenauer · Marian Vajteršic · Ananth Grama

Published online: 17 January 2012
© Springer Science+Business Media, LLC 2012

Abstract Concurrency and parallelism have long been viewed as important, but somewhat distinct concepts. While concurrency is extensively used to amortize latency (for example, in web- and database-servers, user interfaces, etc.), parallelism is traditionally used to enhance performance through execution on multiple functional units. Motivated by an evolving application mix and trends in hardware architecture, there has been a push toward integrating traditional programming models for concurrency and parallelism. Use of conventional threads APIs (POSIX, OpenMP) with messaging libraries (MPI), however, leads to significant programmability concerns, owing primarily to their disparate programming models. In this paper, we describe a novel API and associated runtime for concurrent programming, called MPI Threads (MPIT), which provides a portable and reliable abstraction of low-level threading facilities. We describe various design decisions in MPIT, their underlying motivation, and associated semantics. We provide performance measurements for our prototype implementation to quantify overheads associated with various operations. Finally, we discuss two real-world use cases: an asynchronous message queue and a parallel information retrieval system. We demonstrate that MPIT provides a versatile, low overhead programming model that can be leveraged to program large parallel ensembles.

T. Berka (✉) · H. Hagenauer · M. Vajteršic
Department of Computer Sciences, University of Salzburg, Jakob-Haringer-Str. 2, 5020 Salzburg, Austria
e-mail: tberka@cosy.sbg.ac.at

T. Berka · G. Kollias · A. Grama
Department of Computer Science, Purdue University, 305 N. University Street, West Lafayette, IN 47907, USA

M. Vajteršic
Department of Informatics, Mathematical Institute, Slovak Academy of Sciences, Stefaniškova 49, 814 73 Bratislava, Slovakia
1 Introduction

Parallel platforms, comprising of thousands of multicore processors, are commonly used to solve complex problems in simulation and modeling. More recently, these platforms have also been effectively utilized for solving problems in large-scale information retrieval, data analysis, and mining. In fact, some of the largest data centers, with thousands of processing nodes, are devoted to information retrieval from unstructured data (HTML repositories) or more traditional databases (parallel database servers). Beyond parallel execution, concurrency is commonly utilized to improve performance of high-latency operations. These operations include network transfers, I/O operations, and user interaction. The motivation for concurrency in these environments is that by threading high-latency operations, we can utilize available resources while the operations execute in the background.

The evolving hardware and application mix for large-scale parallel platforms motivates integration of traditional APIs for expressing concurrency (threads) and scalable parallelism (messaging). Nodes in conventional machines typically consist of multicore processors. The shared address space supported by these processors lends itself naturally to a threaded API. Platforms with tens of thousands of nodes typically have significant communication latencies. It is desirable to hide the latency of these communication operations through effective use of concurrency.

With respect to data-intensive applications, parallel formulations of applications in online analytical processing (OLAP), information retrieval (IR), and other business processes, have been well-studied. These formulations typically rely on large-scale numerical, statistical, geometric or probabilistic analysis. However, these applications often involve strong user interaction and dynamic data updates, which must happen concurrently. Parallel execution of these operations often requires properties such as atomicity, consistency isolation, and durability, in addition to load balancing and minimizing communication.

Motivated by these considerations, many software systems rely on a mix of traditional threading APIs (typically POSIX or OpenMP) with messaging APIs (typically MPI). The two APIs have been developed independently and have widely differing design philosophies. This reflects in the complexities associated with developing robust high performance hybrid (threaded-messaging) codes. The nominal support for hybrid programming is often restricted to making communication operations thread-safe (reentrant and isolated). The formal behavior of even simple constructs, for example, what happens when two POSIX threads within a process execute a collective communication operation in MPI on the same communicator, is often not well defined or ideally-suited in terms of associated semantics.

Recognizing this disconnect, we propose a new threading API called MPI Threads (MPIT), which provides necessary mechanisms for efficient specification of concurrency within MPI programs. A key feature of MPIT is the novel way in which it allows each MPI process to create an identical set of threads, and how it supports MPI