A Tutorial on Parallel Systems Development

Alex Vrenios

Parallel Programming in C with MPI and OpenMP

By Michael J. Quinn

544 pages

US$92.30


Parallel Programming in C with MPI and OpenMP is intended as a text for graduate and advanced undergraduate computer science and engineering students. It's also aimed at computer professionals interested in parallel systems development. It assumes you're competent in C (or Fortran) and exposed to algorithm analysis. Michael Quinn presents the material clearly, thoroughly, and with the confidence of someone who's taught the subject at the college level for over 20 years.

Quinn begins with the motivation for exploiting parallelism. He presents parallel processing's evolution from the early stages through state-of-the-art supercomputing architectures and discusses approaches to parallel software development. His argument for developing systems with packages such as MPI (Message Passing Interface) and OpenMP is that they'll be easier to
port when that newer, faster parallel computer becomes available. (Although he doesn't mention it explicitly, this approach should also be beneficial when porting to an embedded multiprocessor platform—a popular recent trend.) Following the introduction are two chapters that review parallel hardware and software architectures, respectively. The software chapter also presents Ian Foster's parallel-algorithm design methodology, which Quinn uses throughout the book.

According to Quinn's chapter dependence diagram (in the Preface), we can focus on either the MPI or the OpenMP tutorials at this point. He suggests that a hybrid system, developed using both MPI and OpenMP, might outperform an MPI-only program, so you'll probably want to read both tutorials; he presents and discusses a sample hybrid system design in the last chapter.

THE MPI TUTORIAL

PVM (Parallel Virtual Machine) and MPI are both popular and freely available parallel software development aids. The Oak Ridge National Laboratory developed PVM for internal use and released it to the public after some refinement. A committee known as the MPI Forum developed MPI as a standard. MPI has become more popular to use than PVM, owing mostly to its improved portability. Vendors develop proprietary versions of PVM that are optimized for their platforms, but systems developed using one vendor's PVM might not compile on another's platform. Vendor support of MPI, on the other hand, was required to meet strict interface standards. Quinn chose MPI as the development tool for message passing systems rather than PVM for just that reason.

Chapter 4 begins the MPI Tutorial, which presents 27 MPI functions. This is a robust subset of the MPI suite, which is sufficient "to implement parallel programs for a wide variety of applications." This is a powerful section of the book: Quinn uses Ian Foster's design methodology along with MPI functions to present solutions to increasingly difficult problems. The reader is learning a design methodology, a parallel-system development tool, and principles of parallelism simultaneously!

Quinn developed, tested, and benchmarked each of these programs on a modest PC cluster, and he presents their performance results throughout the text. These results are intended to illustrate the role of network bandwidth and latency in a parallel program's overall performance and to subtly introduce some important performance metrics of a distributed system.
THE OPENMP TUTORIAL

A true multiprocessor offers multiple CPUs, each with equal access to a shared physical-memory area. Twenty years ago, Kai Li, a Yale doctoral student, proposed a shared-memory model for a multicomputer, which used custom software on a network of workstations. The software provided a layer of support for shared virtual memory that spanned his network of uniprocessors. Programs written for a multiprocessor would run more slowly on his system, but his system was a small fraction of a true MP's cost. His dissertation and subsequent publications influenced parallel architecture research for over a decade. OpenMP is a programming language based on this kind of model.

Chapter 17 begins the OpenMP Tutorial, which presents seven compiler directives and uses four important functions. Quinn introduces increasingly complex blocks of sequential code and demonstrates how to transform them into parallel sections. You'll learn enough about OpenMP to convert a complex application to parallel code and to follow the hybrid-approach example to parallel development in chapter 18.

One key difference between MPI and OpenMP is the approach to exploiting parallelism in an application. MPI requires the developer to convert the entire application immediately. OpenMP allows an incremental conversion; the developer can convert, profile, and tune a large application in a stepwise fashion, simplifying the debugging and development process. If you decide to use OpenMP to develop a parallel system, you must either have access to a multiprocessor or run a custom software support layer that emulates these properties on a multicomputer.

CONCLUSION

This book offers solid background on the PVM and MPI development era and detailed knowledge of how to develop parallel software systems on parallel hardware architectures. It meets the needs of those who have access to a true multiprocessor by providing background on OpenMP and some good examples. Finally, it presents a hybrid-system development example that combines the MPI and OpenMP techniques.

Alex Vrenios is the founder and director of research at the Distributed Systems Research Lab. Contact him at DSRL@att.net.