Reviews

Practical Parallel Programming, by Gregory V. Wilson. ISBN 0-262-23186-7, 1995, $52.50, 576 pp. Hardcover, Available from The MIT Press, Cambridge, MA, USA.

Ever since I started programming parallel computers, I have been searching for a book on parallel programming that addressed different parallel programming paradigms and provided different options for implementing applications on parallel machines. Most books I read on this topic dealt with specific techniques such as data-parallel or message-passing and rarely dealt with a wide variety of issues that this book addresses. *Practical Parallel Programming* by Greg Wilson is intended for scientists interested in implementing real-world applications on parallel computers, but who are not familiar with different approaches. This book is particularly helpful to those who have programmed parallel machines using a single paradigm, are frustrated with the shortcomings of that paradigm, and who would like to experiment with other techniques in their applications. *Practical Parallel Programming* is also useful to system designers as a survey of various parallel programming paradigms, which their systems will have to support efficiently.

Practical Parallel Programming is split into six chapters. Chapter 1 covers the basic concepts every parallel programmer should know and addresses various issues in parallel programming using the widely used and popular paradigms in addition to some less-known alternatives. As with any book with such breadth, it would be appropriate to take a quick tour of this book.

Chapter 2 deals with the fundamentals in parallel architectures and provides motivation for parallel programming by describing various applications in many disciplines. Important concepts such as pipelining, vectorization, caching, and interconnection networks are explained. Most importantly, unlike many other parallel programming books, this book does not let the focus shift away from parallel programming. The author uses great illustrations to enhance the programmer's understanding of parallel architectures. If the reader has previously been introduced to parallel architectures, these illustrations can be used to recall most concepts without going through the text in Chapter 2.

One of the best features of *Programming* is that it describes various applications of parallel programming before introducing specific parallel programming techniques. Thus, when any new concepts are introduced in later chapters, readers can refer to Chapter 2 and see where these concepts are applicable. For example, after reading about various decomposition techniques in Chapter 2, one can see how these strategies are applicable to specific problems described in the previous sections. This chapter contains some of the most succinct explanations of various terms and measures commonly used in parallel programming. Chapter 2 also introduces the theoretical aspects of parallel programming, such as the parallel random access memory (PRAM) model.

Chapters 3–5 cover the three most important parallel programming paradigms in universal use, namely, data-parallel, shared-variables, and message-passing.

Chapter 3 starts by describing the primitives of data-parallel languages. Wilson has used a language called Fortran K, a subset of Fortran 90, to illustrate the concepts. (Syntax for this language is found in Appendix 1. However, in order to understand the example programs, Fortran or C programmers can simply rely on their intuition about various constructs.) As new concepts and primitives are introduced, this language is extended to include those primitives. Chapter 3 also describes the core of a data-parallel language, which consists of primitives for data movement and data alignment. With examples from the lingua franca of data-parallel paradigm, High-Performance Fortran (HPF). This chapter includes

Received August 1996
Accepted September 1996

ISSN 1058-9244/97/$8
Scientific Programming, Vol. 6, pp. 327–329 (1997)

a section on collective operations, such as reductions and broadcast. My main objection to the organization of Chapter 3 is that the inclusion of this section misleads the reader into believing that collective operations are a part of data-parallel paradigm and are not relevant in any other paradigms. Some of these concepts should appear in Chapter 2. The last section of Chapter 3 introduces automatic parallelization and data dependence in terms familiar even to readers without any background in compilers.

Chapter 4 deals with another widely used paradigm, shared-memory programming. This subject is treated extensively and nowhere does *Programming* suggest that this paradigm is used on shared-memory architectures only. Many other books seem to confuse architectural and programming paradigms. This is carefully avoided throughout this book. However, readers are cautioned that before reading Chapter 4, they should brush-up on low-level programming and concepts about concurrency. Often many application programmers are unaware of the concepts from operating system design, and this chapter uses some of these concepts heavily. For example, while explaining fork-join primitives, the author uses pointers, which may not be familiar to Fortran programmers. Also, while explaining atomic operations, assembly language routines are used, which some application programmers may find difficult to grasp. Casual readers might skip the specifics and concentrate on important concepts in process synchronization such as semaphores, test-and-set mechanism, monitors, and barrier synchronization, which this book explains very lucidly. Once again, the organization of this book disappoints me, because widely applicable concepts such as scheduling and mapping are introduced in the context of shared-memory programming. Chapter 4 also summarizes some aspects of parallel I/O, a very important field of research and development in parallel programming, but the issues involved are not seriously addressed.

Chapter 5 explains message-passing parallel programming, which recently has become very popular among application programmers. This chapter starts with Hoare's Communicating Sequential Processes (CSP) model [1]. Illustrations are provided for concepts such as channels. Most examples given are "toy" examples. At first they do not seem applicable to most traditional practical applications of parallelism, but readers will find it helpful to go back to Chapter 2 and think about the applicability of message-passing in the applications described there. Chapter 5 then discusses shortcomings of the channel model and describes the crystalline model, which is a restricted form of the popular single-program multiple-data (SPMD) model. This chapter introduces widely applicable concepts such as timing and debugging parallel programs in the context of the message-passing model. Chapter 6 also introduces skeletons and application templates, which are (unfortunately) not in widespread use currently. *Programming* lacks a section on practical message-passing systems in popular use, such as PVM and MPI. It would have made comparison possible between the "toy" system described in this book and the standard message-passing libraries.

Chapter 6 describes some of the less popular parallel programming models such as tuple spaces and message-driven programming under the category of generative communication. Most of the stress is on tuple spaces with many examples from Linda [2]. (Main features of Linda have been incorporated into Fortran K.) This chapter deals with the basics of generative communication and some solutions to difficult problems such as creating complicated data structures in languages that support generative communication. The examples in Chapter 6, which are more practical than examples in earlier chapters, will help readers get a quick grasp of advanced and less-known concepts. Language developers will find the section on implementation of generative communication very illuminating. Chapter 6 also describes some other higher-level alternative paradigms such as Charm [3] and Orca [4]. However, most of the discussion about Charm focuses on specific shared variables instead of on message-driven execution, which is the main feature of Charm. This chapter lacks discussion of other higher-level programming paradigms such as object-parallel and parallel-functional languages [5–8].

Appendices are provided for non-mainstream topics such as the syntax of the parallel language Fortran K and the history of parallel computing. Appendix 2 deals mainly with the history of parallel architectures. However, one would expect this appendix to include a short history of various parallel programming languages and techniques, which it doesn't. Appendix 3 lists recommended reading material, and has an excellent selection of books, surveys, and papers on parallel programming and architectures. *Programming* also includes a comprehensive glossary. I was pleasantly surprised to read the last appendix; a humorous article about how not to build parallel programming systems. Readers can easily find many parallel programming systems that contain "features" mentioned in this appendix.

Practical Parallel Programming does not address some important issues for parallel programmers. The most important of these omissions is benchmarking. Since performance is the foremost objective in most parallel programming projects, almost every parallel programmer should know about benchmarking parallel machines, what the benchmarks suggest, and the importance of benchmarks in choosing systems to build real-world applications.

It would definitely have helped if the text included code snippets from real-world programming systems,

perhaps for some applications described in Chapter 2. Also, this book takes a way out of a dilemma facing authors of parallel programming books; to describe real systems for different paradigms, or to unify them in a single base language. It would have been better to summarize real-world programming environments, just to contrast with the toy system based on Fortran K.

Also, *Programming* misses some aspects of one of the most popular parallel architectures and associated programming techniques: the network-of-workstations (NOW). Many parallel programmers use this platform and are interested in problems arising in scheduling and mapping in NOW. This book does not discuss heterogeneity and the extensions provided in real-world systems that deal with it.

Practical Parallel Programming comes with a distribution of the Fortran K programming environment. It can be obtained by anonymous FTP from mitpress.mit.edu. This programming environment consists of a Fortran K compiler and run-time system, and it runs only on some single-processor workstations (RS6000, Sun Sparc with SunOS4, Linux 486). When I tried to install Fortran K on my Solaris 2.4 machine, it aborted with errors owing to the differences in some system "include" files. Another problem in providing such an environment is that the focus shifts away from performance, since this environment is only applicable to a single processor. However, this environment does a very good job of allowing the user to experiment with different paradigms in application kernels or toy programs on the machines mentioned earlier.

I found *Practical Parallel Programming* to be a nice book for anyone who needs an introduction to various parallel programming paradigms and techniques and/or who wants to choose between these techniques to implement applications. But some additional documentation about the real-world systems is necessary before one actually attempts to write a parallel program.

REFERENCES

[1] C. A. R. Hoare, *Communicating Sequential Processes*. Englewood Cliffs, NO: Prentice Hall International Series in Computer Science, 1985.

[2] N. Carriero and D. Gelernter, *How to Write Parallel Programs: A First Course*. Cambridge, MA: Mit Press, 1990.

[3] L. V. Kale, "The Chare kernel parallel programming language and system," in *Proc. Int. Conf. Parallel Processing*, 1990, Vol. II, pp. 17–25.

[4] H. E. Bal, M. F. Kaashoek, and A. S. Tanenbaum, "Orca: A language for parallel programming of distributed systems," *IEEE Trans. Software Eng.*, vol. 18, pp. 190–205, March 1992.

[5] L. V. Kale and S. Krishnan, "CHARM++: A portable concurrent object oriented system based on C++," in *Proc. Conf. Object Oriented Programming Systems, Languages and Applications*, ACM SIGPLAN Notices, vol. 28, pp. 91–108, 1993.

[6] A. Yonezawa, J.-P. Briot, and E. Shibayama, "Object-oriented concurrent programming in ABCL/1," in *ACM SIGPLAN Notices, Proc. OOPSLA '86*, 1986.

[7] J. T. Feo, D. C. Cann, and R. R. Oldehoeft, "A report on the Sisal language project," *J. Parallel Distrib. Comput.*, vol. 10, pp. 349–366, 1990.

[8] K. M. Chandy, and C. Kesselman, "CC++: A declarative concurrent object-oriented programming notation," in *Research Directions in Concurrent Object-Oriented Programming*, G. Agha, P. Wegner, and A. Yonezawa, Eds. Cambridge, MA: MIT Press, 1993, pp. 281–313.

Milind A. Bhandarkar
Parallel Programming Laboratory
University of Illinois at Urbana-Champaign
E-mail: milind@cs.uiuc.edu