Using OpenMP

Portable Shared Memory Parallel Programming

Barbara Chapman, Gabriele Jost, Ruud van der Pas

The MIT Press
Cambridge, Massachusetts
London, England

This book was set in LaTeX by the authors and was printed and bound in the United States of America.

Library of Congress Cataloging-in-Publication Data

Chapman, Barbara, 1954-
Using OpenMP : portable shared memory parallel programming / Barbara Chapman, Gabriele Jost, Ruud van der Pas.
p. cm. – (Scientific and engineering computation)
Includes bibliographical references and index.
ISBN-13: 978-0-262-53302-7 (paperback : alk. paper)
1. Parallel programming (Computer science) 2. Application program interfaces (Computer software) I. Jost, Gabriele. II. Pas, Ruud van der. III. Title.
QA76.642.C49 2007
005.2'75–dc22

2007026656

Preface

At Supercomputing 1997, a major conference on High Performance Computing, Networking, and Storage held in San Jose, California, a group of High Performance Computing experts from industry and research laboratories used an informal "Birds of a Feather" session to unveil a new, portable programming interface for shared-memory parallel computers. They called it OpenMP. The proposers included representatives from several hardware companies and from the software house Kuck and Associates, as well as scientists from the Department of Energy who wanted a way to write programs that could exploit the parallelism in shared memory machines provided by several major hardware manufacturers.

This initiative could not have been more timely. A diversity of programming models for those early shared-memory systems were in use. They were all different enough to inhibit an easy port between them. It was good to end this undesirable situation and propose a unified model.

A company was set up to own and maintain the new informal standard. It was named the OpenMP Architecture Review Board (ARB). Since that time, the number of vendors involved in the specification and maintenance of OpenMP has steadily grown. There has been increased involvement of application developers, compiler experts, and language specialists in the ARB too.

The original proposal provided directives, a user-level library, and several environment variables that could be used to turn Fortran 77 programs into shared-memory parallel programs with minimal effort. Fairly soon after the first release, the specification was further developed to enable its use with C/C++ programs and to take features of Fortran 90 more fully into account. Since then, the bindings for Fortran and C/C++ have been merged, both for simplicity and to ensure that they are as similar as possible. Over time, support for OpenMP has been added to more and more compilers. So we can safely say that today OpenMP provides a compact, yet flexible shared-memory programming model for Fortran, C, and C++ that is widely available to application developers.

Many people collaborated in order to produce the first specification of OpenMP. Since that time, many more have worked hard in the committees set up by the ARB to clarify certain features of the language, to consider extensions, and to make their implementations more compatible with each other. Proposals for a standard means to support interactions between implementations and external tools have been intensively debated. Ideas for new features have been implemented in research prototypes. Other people have put considerable effort into promoting the use of OpenMP and in teaching novices and experts alike how to utilize its features to solve a variety of programming needs. One of the authors founded a not-for-

profit company called cOMPunity to help researchers participate more fully in the evolution of OpenMP and to promote interactions between vendors, researchers, and users. Many volunteers helped cOMPunity achieve its goals.

At the time of writing, hardware companies are poised to introduce a whole new generation of computers. They are designing and building multicore platforms capable of supporting the simultaneous execution of a growing number of threads in a shared-memory system. Even laptops are already small parallel computers. The question is when and how the software will be adapted to take advantage of this trend. For a while, improved throughput is going to be the main benefit of multicore technology. It is quite typical to deploy multiple independent activities on a laptop or PC, but how many cores are needed for this? At some point, users will expect individual applications to take advantage of the additional processing power. To do so, a parallel programming model is required. We think OpenMP is in a perfect position to satisfy this need — not only today, but also in the future.

Why a book on OpenMP? After all, the OpenMP specification can be downloaded from the web. The answer lies in the fact that, although the specification has been written in a relatively informal style and has numerous examples, it is still not a particularly suitable starting point for learning how to write real programs. Moreover, some of the factors that may influence a program's performance are not mentioned anywhere in that document. Despite its apparent simplicity, then, additional information is needed. This book fills in those gaps.

Chapter 1 provides background information and explains where OpenMP is applicable, as well as how it differs from other programming interfaces.

Chapter 2 gives a brief overview of the features of OpenMP. It is intended as a high-level introduction that can be read either before or after trying out OpenMP. Among other topics, it explains how OpenMP deals with problems arising from the complex memory hierarchy present on most modern computers.

Chapter 3 is an essential chapter for novice parallel programmers. It discusses a complete OpenMP program (in both Fortran and C versions) that exploits a couple of the most widely used features, and it explains the basics of the OpenMP syntax.

Chapter 4 provides an extensive overview of the OpenMP programming model, with many examples. First, the most widely used features are introduced, with a focus on those that enable work to be shared among multiple threads. Then, some important additional elements of the API are presented. Finally, we describe some of OpenMP's lesser-used parts. In the early sections, our examples are straightforward. Later, we give solutions to some more challenging programming problems.

Chapters 5 and 6 discuss how to get good performance with OpenMP. We include a number of programming tips, along with an extended example that gives insight into the process of investigating performance problems. With the growing number of threads available on new platforms, the strategies given in Chapter 6 for achieving higher levels of scalability are likely to be important for many application developers.

Chapter 7 discusses problems of program correctness. Troubleshooting any application can be hard, but shared-memory parallel programming adds another dimension to this effort. In particular, certain kinds of bugs are nondeterministic. Whether they manifest themselves may depend on one or more external factors, such as the number of threads used, the load on the system, the compiler, and the OpenMP library implementation.

Chapter 8 shows how the compiler translates an OpenMP program to turn it into an application capable of parallel execution. Since OpenMP provides a fairly high level programming model, knowledge of what happens behind the scenes may help the reader understand the impact of its translation and the workings of OpenMP-aware compilers, performance tools, and debuggers. It may also give deeper insight into techniques and strategies for obtaining high levels of performance.

Chapter 9 describes some of the trends that are likely to influence extensions to the OpenMP specification. Included are comments on language features we expect to be included in the reasonably near future.

Acknowledgments

A number of people have worked very hard to help maintain OpenMP, provide feedback to users, debate and develop syntax for new language features, implement those features, and teach others how to use them. It is their work that we present here. We also acknowledge here the continuous efforts of many colleagues on the various committees of the OpenMP Architecture Review Board. We particularly mention Mark Bull, from the University of Edinburgh, without whom progress on the language front is difficult to conceive.

We thank our colleagues who have contributed to the activities of cOMPunity, which enables the participation of researchers and application developers in the work of the ARB. These include Eduard Ayguade, Rudi Eigenmann, Dieter an Mey, Mark Bull, Guy Robinson, and Mitsuhisa Sato.

We thank Michael Resch and colleagues at the High Performance Computing Center (HLRS) of the University of Stuttgart, Germany, for providing logisitical support for the creation of this manuscript and for offering a pleasant working

environment and good company for one of us during a part of the writing phase. We particularly thank Matthias Müller, originally from HLRS, but now at the Dresden University of Technology, for his comments, encouragement, and support and for getting us started with the publisher's software.

Our sincere gratitude goes to the following organizations and individuals that have helped us throughout the writing of this book: Lei Huang, Chunhua Liao, and students in the HPC Tools lab at the University of Houston provided material for some examples and criticized our efforts. We benefited from many helpful discussions on OpenMP scalability issues with the staff of NASA Ames Research Center. In particular, we thank Michael Aftosmis and Marsha Berger for the flowCart example and Henry Jin for many interesting discussions of the NAS Parallel Benchmarks and OpenMP in general. Our thanks go to colleagues at CEPBA (European Center for Parallelism of Barcelona) and UPC (Universitat Politecnica de Catalunya), especially Judit Gimenez and Jesus Labarta for fruitful collaborations in performance analysis of large-scale OpenMP applications, and Eduard Ayguade, Marc Gonzalez, and Xavier Martorell for sharing their experience in OpenMP compiler technology.

Nawal Copty, Eric Duncan, and Yuan Lin at Sun Microsystems gave their help in answering a variety of questions on OpenMP in general and also on compiler and library implementation issues.

We gratefully acknowledge copious feedback on draft versions of this book from Tim Mattson (Intel Corporation) and Nawal Copty and Richard Friedman (both at Sun Microsystems). They helped us find a number of mistakes and made many suggestions for modifications and improvements. Remaining errors are, of course, entirely our responsibility.

Last but not least, our gratitude goes to our families for their continued patience and encouragement. Special thanks go to Dave Barker (a husband) for tolerating awakening to the sound of a popcorn popper (the keyboard) in the wee hours and for providing helpful feedback throughout the project; to Carola and Jonathan (two children) for cheerfully putting up with drafts of book chapters lying around in many likely, and some unlikely, places; and to Marion, Vincent, Stéphanie, and Juliette, who never complained and who provided loving support throughout this journey.