Scientific Programming 16 (2008) 341–342
DOI 10.3233/SPR-2008-0263
IOS Press

Book Review

Using OpenMP – Portable Shared Memory Parallel Programming by Barbara Chapman, Gabrielle Jost and Ruud Van Der Paas, 2007, ISBN: 9780262533027

As multi-core processors become the mainstream computing engines, considerable burden is on the software designers to take advantage of the new hardware. Until now, software developers take advantage of the increase in processor performance (which in turn implies that your single-thread program will run faster as you buy the latest processor). However, with the advent of throughput-oriented computing as envisaged by multi-core/Chip Multithreading (CMP) models on newer processors, your single-thread program may not run faster, even if you go out and buy the latest chip. In fact it can end up running slower. To us software developers this means that we need to rethink the way we write applications and focus towards concurrent programming, in order to take advantage of the parallel hardware we are getting.

I can hear your cries already – "Oh, but we already write concurrent programs!". Well, you may do, but the majority of applications today are single-threaded, and for a good reason. It is much easier to reason about and write bug-free single-thread programs, compared to multithread programs. After all, if you need parallelism, you can always run multiple instances of your application to improve the throughput. That story is changing now, with the advent of multi-core and chip multithreading processors. A highly concurrent application can take much better advantage of the latest hardware. Hence, parallel programming is becoming mainstream and developers are looking for parallel programming models using which they can easily parallelize their applications. This is where OpenMP enters the picture. OpenMP is a shared-memory Application Programming Interface (API) using which, developers can write shared memory parallel programming applications.

OpenMP simplifies the process of parallelizing an existing serial application. It allows the developer to do this complex task in several small incremental steps by means of inserting OpenMP directives as pragmas in C/C++ applications or as comments in Fortran application. Though OpenMP is rich in features, it is easy for the developers to come up with a correct OpenMP parallel version of their application by learning only a small set of OpenMP directives and inserting them in the source code appropriately. Most of the complexity is hidden inside an OpenMP compliant compiler implementation and OpenMP runtime libraries, which realize the parallelized version of the application.

The book "Using OpenMP – Portable Shared Memory Parallel Programming" by Barbara Chapman, Gabrielle Jost and Ruud Van Der Paas, comes at the right time when most of the developer community is scrambling onto the parallel programming bandwagon. It is an excellent introduction for folks new to OpenMP programming. It also contains in depth insights on subtle issues in OpenMP programming, which will make it interesting to experienced OpenMP developers as well.

The book is organized into 9 chapters, with the first four chapters explaining the basics of OpenMP programming model and the remaining 5 chapters deal with the advanced concepts.

Chapter 1 provides the pertinent historical background of parallel programming hardware and software. It provides details on the evolution of OpenMP as an industry accepted shared memory parallel programming model. It does a brief comparison of the three widely used parallel programming models, namely OpenMP, Message Passing Interface (MPI) and pthreads, using a small program to compute the dot product. In my opinion, it would have been good if the authors had clearly differentiated between memory consistency and memory coherence issues while talking about the OpenMP memory model in Chapter 1. Also, a brief description of NUMA vs. UMA would not have been amiss in Chapter 1.

Chapter 2 provides an overview of the OpenMP constructs, and is followed by a description of how to write a simple OpenMP program for matrix–vector application in Chapter 3. Chapter 4 discusses the major OpenMP language features by illustrating all the important constructs with small code snippets in C/Fortran. After reading Chapters 1–4, anyone new to OpenMP should be all set to dive right into writing OpenMP applications.

Chapter 5 details how to measure and improve the performance of OpenMP applications. In this chapter, the authors first discuss the various factors which affect OpenMP performance such as memory hierarchy, application's memory access patterns, compiler optimizations, and the runtime environment etc., followed by a brief discussion on how to measure OpenMP performance. They also discuss best practices for improving OpenMP performance. Chapter 5 concludes with a case study of matrix–vector multiplication showing how the performance of the OpenMP versions in C and Fortran can be improved, and the impact of the various factors on performance.

Chapter 6 discusses some of the issues in real-life OpenMP applications such as scalability, and data and thread placement in CC-NUMA machines. It also discusses combining OpenMP and message-passing programming, clearly explaining the pros and cons of such a hybrid approach. Chapter 6 contains a case study of real life applications such as CFD flow solver and NAS parallel benchmarks. It concludes with details on how to do performance analysis of OpenMP applications using latest profiling tools.

Chapter 7 discusses the common programming errors developers make in OpenMP applications, and details of how to debug OpenMP applications. Complex issues such as data races, memory consistency problems, and deadlocks are explained in detail with examples and instructions on how they can be avoided. This chapter is a must-read for all OpenMP developers.

Chapter 8 covers the internal details of OpenMP implementation in OpenMP compliant compilers. It describes details on how OpenMP directives are translated, and how runtime support functions are implemented. Chapter 8 concludes with a brief discussion on the impact of OpenMP on compiler optimizations.

Chapter 9 discusses future OpenMP extensions. Since the book is based on OpenMP 2.5, it discusses the extensions planned for OpenMP 3.0. As the OpenMP 3.0 specification has recently been released, the reader may want to look at the new features OpenMP 3.0 specification from www.OpenMP.org while reading this chapter. The book also has an exhaustive list of references for the reader to pursue details on specific topics of interest.

Overall, this book is a good tutorial on OpenMP for developers new to OpenMP. The book discusses the latest state of the art OpenMP implementations. It provides practical perspectives on performance issues one can run into while developing OpenMP applications. If you are a programmer planning to develop OpenMP applications, this is a good book to have on hand. Most of the features are discussed with examples in both C and Fortran, which makes this book relevant for both C/C++ and Fortran OpenMP developers. The book is well organized, therefore, based on your level of OpenMP expertise, you can pick and read relevant chapters.

You can also use this book to teach a short course on OpenMP programming to undergraduates in computer science. A beginner's course on OpenMP programming can cover the first 4 chapters of the book. An intermediate course on OpenMP programming can cover the material from Chapters 5–10. However there are no programming exercises at the end of each chapter. So if you want to use this book as a textbook, you need to create your own set of programming exercises.

This book is a good starting point to understand the OpenMP basics and start writing OpenMP programs. If you are an experienced OpenMP developer, you can use the advanced material in the later chapters of the book, to tune the performance of your OpenMP applications. I recommend this book as a good buy if you are planning to develop parallel applications using OpenMP.

Sandya S. Mannarswamy
Hewlett Packard India Private Ltd.
Bangalore, India
E-mail: sandya.s.mannarswamy@hp.com