Using OpenMP

Portable Shared Memory Parallel Programming

Barbara Chapman, Gabriele Jost, Ruud van der Pas

The MIT Press
Cambridge, Massachusetts
London, England

© 2008 Massachusetts Institute of Technology

All rights reserved. No part of this book may be reproduced in any form by any electronic or mechanical means (including photocopying, recording, or information storage and retrieval) without permission in writing from the publisher.

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

Contents

Series Foreword xiii
Foreword xv
Preface xix

1 Introduction 1

1.1 Why Parallel Computers Are Here to Stay 1
1.2 Shared-Memory Parallel Computers 3
1.2.1 Cache Memory Is Not Shared 4
1.2.2 Implications of Private Cache Memory 6
1.3 Programming SMPs and the Origin of OpenMP 6
1.3.1 What Are the Needs? 7
1.3.2 A Brief History of Saving Time 7
1.4 What Is OpenMP? 8
1.5 Creating an OpenMP Program 9
1.6 The Bigger Picture 11
1.7 Parallel Programming Models 13
1.7.1 Realization of Shared- and Distributed-Memory Models 14
1.8 Ways to Create Parallel Programs 15
1.8.1 A Simple Comparison 16
1.9 A Final Word 21

2 Overview of OpenMP 23

2.1 Introduction 23
2.2 The Idea of OpenMP 23
2.3 The Feature Set 25
2.3.1 Creating Teams of Threads 25
2.3.2 Sharing Work among Threads 26
2.3.3 The OpenMP Memory Model 28
2.3.4 Thread Synchronization 29
2.3.5 Other Features to Note 30
2.4 OpenMP Programming Styles 31

2.5 Correctness Considerations 32
2.6 Performance Considerations 33
2.7 Wrap-Up 34

3 Writing a First OpenMP Program 35
3.1 Introduction 35
3.2 Matrix Times Vector Operation 37
3.2.1 C and Fortran Implementations of the Problem 38
3.2.2 A Sequential Implementation of the Matrix Times Vector Operation 38
3.3 Using OpenMP to Parallelize the Matrix Times Vector Product 41
3.4 Keeping Sequential and Parallel Programs as a Single Source Code 47
3.5 Wrap-Up 50

4 OpenMP Language Features 51
4.1 Introduction 51
4.2 Terminology 52
4.3 Parallel Construct 53
4.4 Sharing the Work among Threads in an OpenMP Program 57
4.4.1 Loop Construct 58
4.4.2 The Sections Construct 60
4.4.3 The Single Construct 64
4.4.4 Workshare Construct 66
4.4.5 Combined Parallel Work-Sharing Constructs 68
4.5 Clauses to Control Parallel and Work-Sharing Constructs 70
4.5.1 Shared Clause 71
4.5.2 Private Clause 72
4.5.3 Lastprivate Clause 73
4.5.4 Firstprivate Clause 75
4.5.5 Default Clause 77
4.5.6 Nowait Clause 78

4.5.7 Schedule Clause 79
4.6 OpenMP Synchronization Constructs 83
4.6.1 Barrier Construct 84
4.6.2 Ordered Construct 86
4.6.3 Critical Construct 87
4.6.4 Atomic Construct 90
4.6.5 Locks 93
4.6.6 Master Construct 94
4.7 Interaction with the Execution Environment 95
4.8 More OpenMP Clauses 100
4.8.1 If Clause 100
4.8.2 Num_threads Clause 102
4.8.3 Ordered Clause 102
4.8.4 Reduction Clause 105
4.8.5 Copyin Clause 110
4.8.6 Copyprivate Clause 110
4.9 Advanced OpenMP Constructs 111
4.9.1 Nested Parallelism 111
4.9.2 Flush Directive 114
4.9.3 Threadprivate Directive 118
4.10 Wrap-Up 123

5 How to Get Good Performance by Using OpenMP 125
5.1 Introduction 125
5.2 Performance Considerations for Sequential Programs 125
5.2.1 Memory Access Patterns and Performance 126
5.2.2 Translation-Lookaside Buffer 128
5.2.3 Loop Optimizations 129
5.2.4 Use of Pointers and Contiguous Memory in C 136
5.2.5 Using Compilers 137
5.3 Measuring OpenMP Performance 138
5.3.1 Understanding the Performance of an OpenMP Program 140

5.3.2 Overheads of the OpenMP Translation 142
5.3.3 Interaction with the Execution Environment 143
5.4 Best Practices 145
5.4.1 Optimize Barrier Use 145
5.4.2 Avoid the Ordered Construct 147
5.4.3 Avoid Large Critical Regions 147
5.4.4 Maximize Parallel Regions 148
5.4.5 Avoid Parallel Regions in Inner Loops 148
5.4.6 Address Poor Load Balance 150
5.5 Additional Performance Considerations 152
5.5.1 The Single Construct Versus the Master Construct 153
5.5.2 Avoid False Sharing 153
5.5.3 Private Versus Shared Data 156
5.6 Case Study: The Matrix Times Vector Product 156
5.6.1 Testing Circumstances and Performance Metrics 157
5.6.2 A Modified OpenMP Implementation 158
5.6.3 Performance Results for the C Version 159
5.6.4 Performance Results for the Fortran Version 164
5.7 Fortran Performance Explored Further 167
5.8 An Alternative Fortran Implementation 180
5.9 Wrap-Up 189

6 Using OpenMP in the Real World 191
6.1 Scalability Challenges for OpenMP 191
6.2 Achieving Scalability on cc-NUMA Architectures 193
6.2.1 Memory Placement and Thread Binding: Why Do We Care? 193
6.2.2 Examples of Vendor-Specific cc-NUMA Support 196
6.2.3 Implications of Data and Thread Placement on cc-NUMA Performance 199
6.3 SPMD Programming 200
Case Study 1: A CFD Flow Solver 201
6.4 Combining OpenMP and Message Passing 207
6.4.1 Case Study 2: The NAS Parallel Benchmark BT 211

6.4.2 Case Study 3: The Multi-Zone NAS Parallel Benchmarks 214
6.5 Nested OpenMP Parallelism 216
6.5.1 Case Study 4: Employing Nested OpenMP for Multi-Zone CFD Benchmarks 221
6.6 Performance Analysis of OpenMP Programs 228
6.6.1 Performance Profiling of OpenMP Programs 228
6.6.2 Interpreting Timing Information 230
6.6.3 Using Hardware Counters 239
6.7 Wrap-Up 241

7 Troubleshooting 243
7.1 Introduction 243
7.2 Common Misunderstandings and Frequent Errors 243
7.2.1 Data Race Conditions 243
7.2.2 Default Data-Sharing Attributes 246
7.2.3 Values of Private Variables 249
7.2.4 Problems with the Master Construct 250
7.2.5 Assumptions about Work Scheduling 252
7.2.6 Invalid Nesting of Directives 252
7.2.7 Subtle Errors in the Use of Directives 255
7.2.8 Hidden Side Effects, or the Need for Thread Safety 255
7.3 Deeper Trouble: More Subtle Problems 259
7.3.1 Memory Consistency Problems 259
7.3.2 Erroneous Assumptions about Memory Consistency 262
7.3.3 Incorrect Use of Flush 264
7.3.4 A Well-Masked Data Race 266
7.3.5 Deadlock Situations 268
7.4 Debugging OpenMP Codes 271
7.4.1 Verification of the Sequential Version 271
7.4.2 Verification of the Parallel Code 272
7.4.3 How Can Tools Help? 272
7.5 Wrap-Up 276

8 Under the Hood: How OpenMP Really Works 277
8.1 Introduction 277
8.2 The Basics of Compilation 278
8.2.1 Optimizing the Code 279
8.2.2 Setting Up Storage for the Program's Data 280
8.3 OpenMP Translation 282
8.3.1 Front-End Extensions 283
8.3.2 Normalization of OpenMP Constructs 284
8.3.3 Translating Array Statements 286
8.3.4 Translating Parallel Regions 286
8.3.5 Implementing Worksharing 291
8.3.6 Implementing Clauses on Worksharing Constructs 294
8.3.7 Dealing with Orphan Directives 297
8.3.8 OpenMP Data Environment 298
8.3.9 Do Idle Threads Sleep? 300
8.3.10 Handling Synchronization Constructs 302
8.4 The OpenMP Runtime System 303
8.5 Impact of OpenMP on Compiler Optimizations 304
8.6 Wrap-Up 304

9 The Future of OpenMP 307
9.1 Introduction 307
9.2 The Architectural Challenge 309
9.3 OpenMP for Distributed-Memory Systems 311
9.4 Increasing the Expressivity of OpenMP 312
9.4.1 Enhancing OpenMP Features 312
9.4.2 New Features and New Kinds of Applications 314
9.5 How Might OpenMP Evolve? 317
9.6 In Conclusion 318

A Glossary 321
References 331
Index 349