Multicore Application Programming
For Windows, Linux, and Oracle® Solaris

Darryl Gove

Addison-Wesley
Contents

Preface xv

Acknowledgments xlix

About the Author xxi

1 Hardware, Processes, and Threads 1
Examining the Insides of a Computer 1
The Motivation for Multicore Processors 3
Supporting Multiple Threads on a Single Chip 4
Increasing Instruction Issue Rate with Pipelined Processor Cores 9
Using Caches to Hold Recently Used Data 12
Using Virtual Memory to Store Data 15
Translating from Virtual Addresses to Physical Addresses 16
The Characteristics of Multiprocessor Systems 18
How Latency and Bandwidth Impact Performance 20
The Translation of Source Code to Assembly Language 21
The Performance of 32-Bit versus 64-Bit Code 23
Ensuring the Correct Order of Memory Operations 24
The Differences Between Processes and Threads 26
Summary 29

2 Coding for Performance 31
Defining Performance 31
Understanding Algorithmic Complexity 33
Examples of Algorithmic Complexity 33
Why Algorithmic Complexity Is Important 37
Using Algorithmic Complexity with Care 38
How Structure Impacts Performance 39
Performance and Convenience Trade-Offs in Source Code and Build Structures 39
Using Libraries to Structure Applications 42
The Impact of Data Structures on Performance 53
Contents

Slim Reader/Writer Locks 214
Semaphores 216
Condition Variables 218
Signaling Event Completion to Other Threads or Processes 219
Wide String Handling in Windows 221
Creating Processes 222
Sharing Memory Between Processes 225
Inheriting Handles in Child Processes 228
Naming Mutexes and Sharing Them Between Processes 229
Communicating with Pipes 231
Communicating Using Sockets 234
Atomic Updates of Variables 238
Allocating Thread-Local Storage 240
Setting Thread Priority 242
Summary 244

7 Using Automatic Parallelization and OpenMP 245
Using Automatic Parallelization to Produce a Parallel Application 245
Identifying and Parallelizing Reductions 250
Automatic Parallelization of Codes Containing Calls 251
Assisting Compiler in Automatically Parallelizing Code 254
Using OpenMP to Produce a Parallel Application 256
Using OpenMP to Parallelize Loops 258
Runtime Behavior of an OpenMP Application 258
Variable Scoping Inside OpenMP Parallel Regions 259
Parallelizing Reductions Using OpenMP 260
Accessing Private Data Outside the Parallel Region 261
Improving Work Distribution Using Scheduling 263
Using Parallel Sections to Perform Independent Work 267
Nested Parallelism 268
Using OpenMP for Dynamically Defined Parallel Tasks 269
Keeping Data Private to Threads 274
Controlling the OpenMP Runtime Environment 276
Waiting for Work to Complete 278
Restricting the Threads That Execute a Region of Code 281
Ensuring That Code in a Parallel Region Is Executed in Order 285
Collapsing Loops to Improve Workload Balance 286
Enforcing Memory Consistency 287
An Example of Parallelization 288
Summary 293

8 Hand-Coded Synchronization and Sharing 295
Atomic Operations 295
Using Compare and Swap Instructions to Form More Complex Atomic Operations 297
Enforcing Memory Ordering to Ensure Correct Operation 301
Compiler Support of Memory-Ordering Directives 303
Reordering of Operations by the Compiler 304
Volatile Variables 308
Operating System–Provided Atomics 309
Lockless Algorithms 312
Dekker’s Algorithm 312
Producer-Consumer with a Circular Buffer 315
Scaling to Multiple Consumers or Producers 318
Scaling the Producer-Consumer to Multiple Threads 319
Modifying the Producer-Consumer Code to Use Atomics 326
The ABA Problem 329
Summary 332

9 Scaling with Multicore Processors 333
Constraints to Application Scaling 333
Performance Limited by Serial Code 334