

Contents

Foreword — V

Preface — IX

List of figures — XVII

Listings — XIX

List of tables — XXIII

Glossary — XXV

1 Setting the stage — 1

- 1.1 Structure of the book — 1
- 1.2 Design space exploration — 3
 - 1.2.1 Parallelism as a library — 3
 - 1.2.2 Parallelism as a language — 6
- 1.3 Code samples — 7
- 1.4 Machine configurations — 8

2 Parallel programming models and concepts — 11

- 2.1 Multi-processing and multi-threading — 11
 - 2.1.1 Threading basics — 13
 - 2.1.2 Thread affinity — 17
 - 2.1.3 The OpenMP API for thread-based programming — 18
 - 2.1.4 Worksharing — 19
 - 2.1.5 OpenMP thread affinity — 23
- 2.2 Task-based parallel programming — 24
- 2.3 Synchronization constructs — 28
 - 2.3.1 Mutual exclusion with locks — 28
 - 2.3.2 Barriers, reductions, and latches — 30
 - 2.3.3 Task barriers — 33
 - 2.3.4 Task dependences — 36
- 2.4 Amdahl's law — 39
 - 2.4.1 Presenting performance results — 40
 - 2.4.2 Effect on performance — 43
 - 2.4.3 Mapping overheads to Amdahl — 43
 - 2.4.4 Other variants of Amdahl's law — 44
- 2.5 Conclusions — 46

3	Many-core and multi-core computer architectures — 48
3.1	Execution mechanisms — 48
3.1.1	Von Neumann architecture and in-order execution — 48
3.1.2	In-order pipelined execution — 52
3.1.3	Out-of-order execution — 54
3.1.4	Branch prediction — 58
3.1.5	Superscalar execution — 59
3.1.6	Simultaneous multi-threading — 60
3.1.7	Single-instruction multiple-data — 64
3.2	The modern memory subsystem — 66
3.2.1	Memory hierarchy — 66
3.2.2	Memory models and memory consistency — 72
3.2.3	Caches — 77
3.2.4	Cache coherence: overview — 77
3.2.5	Cache coherence: the MESI protocol — 78
3.2.6	Performance implications — 83
3.2.7	Non-uniform memory architectures — 87
3.3	Conclusions — 89
4	Compiler and runtime interaction — 90
4.1	Compiler basics — 90
4.2	Implementing a task-based parallel model — 94
4.2.1	Lambda functions and closures — 95
4.2.2	Enqueuing tasks in TBB — 96
4.3	Compilers for parallel programming languages — 99
4.4	Parallel code-generation patterns — 103
4.4.1	Code generation for parallel regions — 103
4.4.2	Code generation for thread-parallel loops — 105
4.4.3	Code generation for SIMD-parallel loops — 109
4.4.4	Code generation for sequential constructs — 112
4.4.5	Code generation for static tasking — 115
4.4.6	Code generation for dynamic tasking — 115
4.5	Example OpenMP implementations — 120
4.5.1	GNU compiler collection — 120
4.5.2	Intel compilers and the LLVM compiler — 123
4.6	Conclusions — 126
5	Fundamental parallel runtime mechanisms — 127
5.1	Managing parallelism — 127
5.1.1	Spawning parallelism — 127
5.1.2	Waiting — 128
5.2	Management of parallelism and hardware structure — 130

5.2.1	Detecting the hardware structure — 130
5.2.2	Thread pinning — 132
5.3	Memory management in parallel runtime systems — 134
5.3.1	Memory efficiency and cache usage — 134
5.3.2	Single-threaded memory allocators — 135
5.3.3	Multi-threaded memory allocators — 138
5.3.4	Specialized memory allocators for parallel runtime systems — 140
5.3.5	Thread-local storage — 141
5.3.6	Data layout of the thread objects — 143
5.4	Conclusions — 145
6	Mutual exclusion and atomicity — 146
6.1	The mutual exclusion problem — 146
6.1.1	Hardware support for locks: atomic instructions — 148
6.1.2	The ABA problem — 151
6.2	Should we be writing locking code? — 152
6.3	Classes of locks — 153
6.4	Properties of lock algorithms — 153
6.4.1	Lock-performance metrics — 154
6.5	Lock algorithms — 157
6.5.1	Test-and-Set locks — 158
6.5.2	Test and Test-and-Set locks — 160
6.5.3	Ticket lock — 161
6.5.4	Queuing locks — 162
6.6	Actual code performance — 163
6.6.1	Uncontended lock overhead — 166
6.6.2	Contended lock throughput — 166
6.6.3	Performance conclusions — 169
6.7	How to wait — 171
6.7.1	Backoff strategies — 172
6.8	Transactional synchronization — 177
6.8.1	Transactional semantics — 178
6.8.2	Implementation in the MESI protocol — 179
6.8.3	Transactional instructions — 180
6.8.4	Transactional locks — 181
6.8.5	Comparison of mutual exclusion and speculation — 184
6.9	Other serializing operations — 185
6.9.1	The master and masked constructs — 185
6.9.2	The single construct — 186
6.10	Atomic operations — 187
6.10.1	Mapping of atomic instructions — 188
6.10.2	Atomic implementation of minimum and maximum — 190

6.11	Conclusions — 192
6.11.1	Conclusions: locks — 192
6.11.2	Conclusions: atomic operations — 192
7	Barriers and reductions — 194
7.1	Barrier fundamentals — 194
7.2	Barrier performance measurement — 196
7.2.1	Barrier micro-benchmark — 197
7.2.2	Barrier performance modeling — 200
7.3	Barrier components — 200
7.3.1	Counters and flags — 200
7.3.2	Broadcast — 203
7.4	Categorization of barrier algorithms — 205
7.5	Barrier algorithms — 206
7.5.1	Counting barrier — 206
7.5.2	All-to-All barrier — 210
7.5.3	Butterfly/hypercube barrier — 212
7.5.4	Dissemination barrier — 213
7.5.5	Tree check-in barriers — 215
7.6	Reductions — 220
7.6.1	Piggy-backing reductions — 224
7.7	Additional optimizations — 225
7.7.1	Hierarchical barriers — 225
7.8	Conclusions — 226
8	Scheduling parallel loops — 228
8.1	Aims of scheduling — 228
8.2	Theoretical limits on scheduling efficiency — 229
8.3	Fundamental scheduling approaches — 231
8.3.1	Static loop scheduling — 231
8.3.2	Dynamic loop scheduling — 233
8.4	Mapping to canonical form — 234
8.5	Compiler loop transformations — 236
8.6	Monotonicity of loop scheduling — 239
8.7	Static-schedule implementation — 240
8.7.1	Blocked loop schedules — 240
8.7.2	Block-cyclic loop scheduling — 242
8.8	Dynamic loop schedule implementation — 242
8.8.1	Guided scheduling — 244
8.8.2	monotonic:dynamic — 245
8.8.3	nonmonotonic:dynamic — 247
8.9	Evaluation of loop schedules — 252

8.10	Other loop-scheduling approaches —	255
8.10.1	Using history —	256
8.10.2	Exposing scheduling to the user —	257
8.11	Conclusions —	257
9	Runtime support for task-parallel models —	259
9.1	Task descriptors —	259
9.2	Task pool implementation —	260
9.2.1	Single task pool —	261
9.2.2	Multi task pool —	268
9.3	Task synchronization —	276
9.3.1	Waiting for a subset of tasks —	277
9.3.2	Waiting for immediate child tasks —	281
9.3.3	Task dependences —	286
9.4	Task scheduling concepts —	289
9.4.1	Task-scheduling points —	290
9.4.2	Breadth-first scheduling and depth-first scheduling —	291
9.4.3	Task stealing —	292
9.5	Task scheduling constraints —	296
9.5.1	Stack scheduling —	296
9.5.2	Cyclic scheduling —	297
9.6	Miscellaneous task topics —	298
9.6.1	Task priorities —	298
9.6.2	Task affinity —	300
9.7	Conclusions —	303
10	Summary and final thoughts —	304
	Bibliography —	307
	Index —	315
	List of acronyms —	327