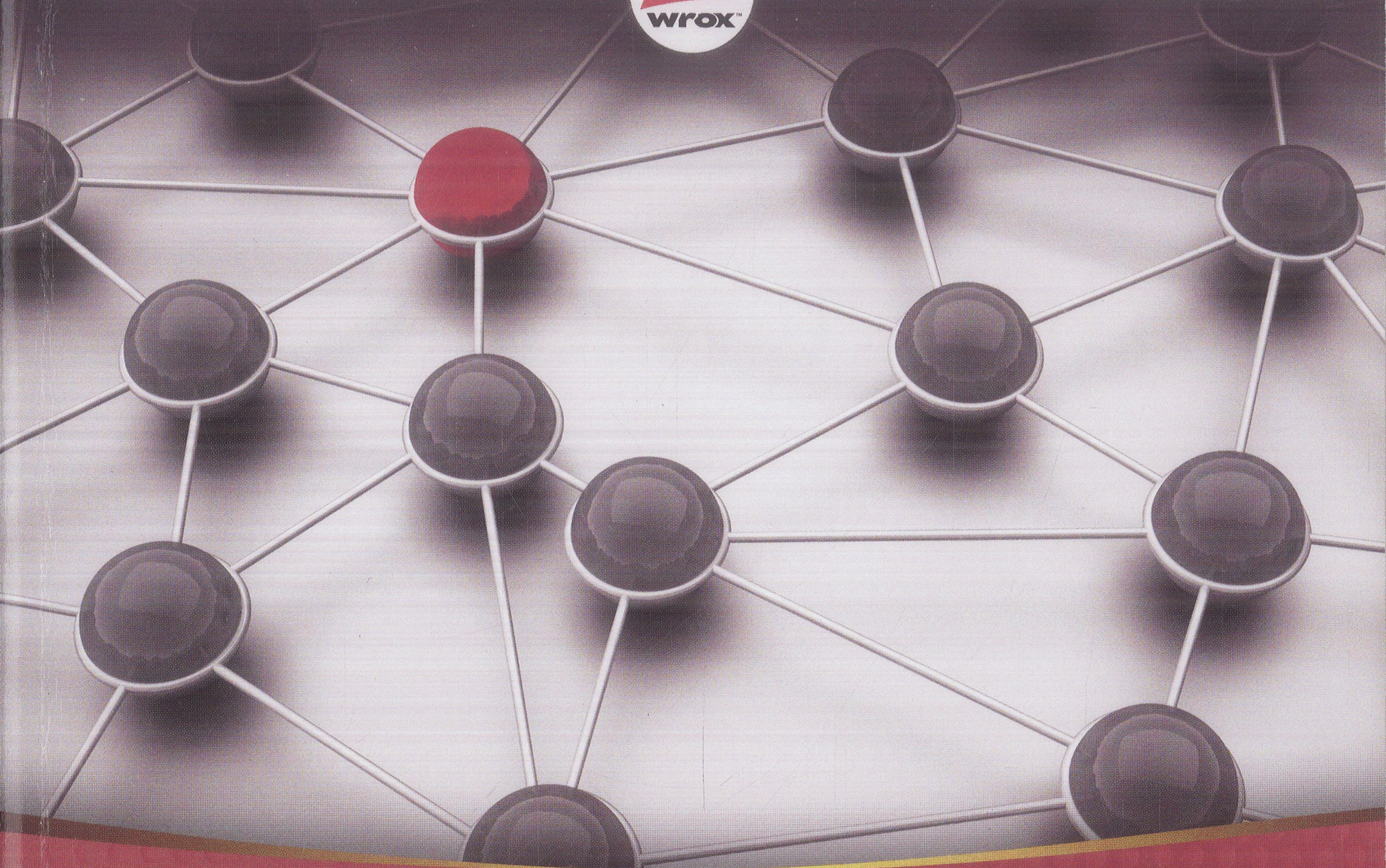
Join the discussion @ p2p.wrox.com

Wrox Programmer to Programmer™



Parallel Programming with Intel® Parallel Studio XE

Foreword by James Reinders, *Director, Parallel Evangelist, Intel*

Stephen Blair-Chappell, Andrew Stokes

CONTENTS

FOREWORD *xxv*

INTRODUCTION *xxvii*

PART I: AN INTRODUCTION TO PARALLELISM

CHAPTER 1: PARALLELISM TODAY 3

The Arrival of Parallelism 3
The Power Density Race 3
The Emergence of Multi-Core and Many-Core Computing 4
The Top Six Challenges 7
Legacy Code 7
Tools 7
Education 8
Fear of Many-Core Computing 8
Maintainability 8
Return on Investment 9
Parallelism and the Programmer 9
Types of Parallelism 9
Intel's Family of Parallel Models 10
Cilk Plus and Threading Building Blocks 10
Domain-Specific Libraries 11
Established Standards 11
Research and Development 11
Choosing the Right Parallel Constructs 11
High-Level vs. Low-Level Constructs 12
Data Parallelism vs. General Parallelism 12
Examples of Mixing and Matching Parallel Constructs 12
Parallel Programming Errors 15
Data Races 15
Determinacy Races 15
Deadlocks 16
Poor Load Balancing 17
Threading/Tasking Overhead 17
Synchronization Overhead 17
Memory Errors 17

Speedup and Scalability 19
Calculating Speedup 19
Predicting Scalability 21
Parallelism and Real-Time Systems 22
Hard and Soft Real-Time 22
A Hard Real-Time Example using RTX 23
Advice for Real-Time Programmers 23
Summary 24

CHAPTER 2: AN OVERVIEW OF PARALLEL STUDIO XE 25

Why Parallel Studio XE? 25
What's in Parallel Studio XE? 26
Intel Parallel Studio XE 26
Intel Parallel Advisor 28
The Advisor Workflow 28
Surveying the Site 28
Annotating Code 29
Checking Suitability 29
Checking Correctness 30
Replacing Annotations 31
Intel Parallel Composer XE 31
Intel C/C++ Optimizing Compiler 31
Profile-Guided Optimization 32
Cilk Plus 33
OpenMP 37
Intel Threading Building Blocks 38
Intel Integrated Performance Primitives 40
An Application Example 41
IPP and Threading 42
Intel Parallel Debugger Extension 43
Intel Debugger 43
Math Kernel Library 44
VTune Amplifier XE 45
Hotspot Analysis 46
Concurrency Analysis 46
Locks and Waits Analysis 46
Dissassembly Source View 48
Parallel Inspector XE 48
Predefined Analysis Types 48
Errors and Warnings 49

Static Security Analysis 51
Different Approaches to Using Parallel Studio XE 52
Summary 52

CHAPTER 3: PARALLEL STUDIO XE FOR THE IMPATIENT 53

The Four-Step Methodology 54
Example 1: Working with Cilk Plus 54
Obtaining a Suitable Serial Program 55
Running the Serial Example Program 57
Creating the Project 57
Running the Serial Version of the Code 58
Step 1: Analyze the Serial Program 60
Using Intel Parallel Amplifier XE for Hotspot Analysis 60
Step 2: Implement Parallelism using Cilk Plus 62
Step 3: Debug and Check for Errors 63
Checking for Errors 64
Narrowing the Scope of the Shared Variables 67
Adding Cilk Plus Reducers 68
Running the Corrected Application 71
Step 4: Tune the Cilk Plus Program 71
Example 2: Working with OpenMP 73
Step 1: Analyze the Serial Program 74
Step 2: Implement Parallelism using OpenMP 74
Step 3: Debug and Check for Errors 75
Making the Shared Variables Private 75
Adding a Reduction Clause 76
Step 4: Tune the OpenMP Program 76
Improving the Load Balancing 80
Summary 84

PART II: USING PARALLEL STUDIO XE

CHAPTER 4: PRODUCING OPTIMIZED CODE 87

Introduction 88
The Example Application 89
Optimizing Code in Seven Steps 90
Using the Compiler's Reporting Features 91
Step 1: Build with Optimizations Disabled 91
Step 2: Use General Optimizations 93

Using the General Options on the Example Application 94
Generating Optimization Reports Using /Qopt-report 95
Step 3: Use Processor-Specific Optimizations 96
What Is Auto-Vectorization? 97
Auto-Vectorization Guidelines 98
Turning On Auto-Vectorization 99
Enhancing Auto-Vectorization 99
Building for Non-Intel CPUs 100
Determining That Auto-Vectorization Has Happened 100
When Auto-Vectorization Fails 101
Helping the Compiler to Vectorize 103
Step 4: Add Interprocedural Optimization 108
Adding Interprocedural Optimization to the Example Application 108
The Impact of Interprocedural Optimization on Auto-Vectorization 109
Step 5: Use Profile-Guided Optimization 112
Benefits of Profile-Guided Optimization 112
The Profile-Guided Optimization Steps 113
The Results 116
Step 6: Tune Auto-Vectorization 116
Activating Guided Auto-Parallelization 116
An Example Session 117
More on Auto-Vectorization 118
Building Applications to Run on More Than One Type of CPU 118
Additional Ways to Insert Vectorization 120
Using Cilk Plus Array Notation 121
Manual CPU Dispatch: Rolling Your Own CPU-Specific Code 124
Source Code 125
Summary 130

CHAPTER 5: WRITING SECURE CODE 131

A Simple Security Flaw Example 132
Understanding Static Security Analysis 134
False Positives 135
Static Security Analysis Workflow 136
Conducting a Static Security Analysis 136
Investigating the Results of the Analysis 138
Working with Problem States 140
The Build Specification 145
Creating a Build Specification File by Injection 146
Utility Options 146
The Directory Structure of the Results 147

Using Static Security Analysis in a QA Environment **149**
Regression Testing 149
Metrics Tracking 150
Source Code **152**
Summary **154**

CHAPTER 6: WHERE TO PARALLELIZE **155**

Different Ways of Profiling **156**
The Example Application **157**
Hotspot Analysis Using the Intel Compiler **158**
Profiling Steps 159
An Example Session 160
Overhead Introduced by Profiling 163
Hotspot Analysis Using the Auto-Parallelizer **165**
Profiling Steps 165
An Example Session 166
Programming Guidelines for Auto-Parallelism 168
Additional Options 168
Helping the Compiler to Auto-Parallelize 169
Hotspot Analysis with Amplifier XE **171**
Conducting a Default Analysis 171
Finding the Right Loop to Parallelize 172
Large or Long-Running Applications 174
Reducing the Size of Data Collected 174
Using the Pause and Resume APIs 175
Source Code **177**
Summary **180**

CHAPTER 7: IMPLEMENTING PARALLELISM **181**

C or C++, That Is the Question **182**
Taking a Simple Approach **183**
The Beauty of Lambda Functions **183**
Parallelizing Loops **185**
The for Loop 185
The Cilk Plus cilk_for Loop 185
The OpenMP for Loop 187
The TBB for Loop 188
Nested for Loops 188
The for Loop with Reduction 189
Cilk Plus Reduction 190

OpenMP Reduction 190
TBB Reduction 191
The while Loop 191
Cilk Plus 191
OpenMP 192
TBB 193
Parallelizing Sections and Functions 193
The Serial Version 194
Cilk Plus 195
OpenMP 196
TBB 197
Parallelizing Recursive Functions 198
The Serial Version 198
Cilk Plus 199
OpenMP 200
TBB 200
Parallelizing Pipelined Applications 201
Parallel Pipelined Patterns 202
The Serial Version 203
OpenMP 205
TBB 206
Parallelizing Linked Lists 208
Serial Iteration of the Linked List 209
Parallel Iteration of the Linked List 209
Source Code 211
Summary 215

CHAPTER 8: CHECKING FOR ERRORS 217

Parallel Inspector XE Analysis Types 218
Detecting Threading Errors 219
Types of Threading Problems 219
Thread Information 219
Potential Privacy Infringement 220
Data Races 220
Deadlocks 220
An Example Application Involving Deadlocks 220
Detecting Deadlocks 221
Detecting Data Races 225
Running the Threaded Program 225
First Results of the Analysis 225

Controlling the Right Level of Detail 227
Testing All the Code Paths 227
Avoiding Being Overwhelmed by the Amount of Data 228
Using Suppression Files 228
Fixing Data Races 233
Using Cilk Plus 233
Cilk Plus Reducers 234
Cilk Plus Holders 234
Using OpenMP 236
Using Locks 236
Using Critical Sections 236
Using Atomic Operations 236
Using a reduction Clause 237
Using TBB 237
Detecting Memory Errors 238
Types of Memory Errors 239
An Example Application for Memory Analysis 240
Creating a Custom Analysis 245
The Source Code 247
Summary 249

CHAPTER 9: TUNING PARALLEL APPLICATIONS 251

Introduction 251
Defining a Baseline 252
Ensuring Consistency 252
Measuring the Performance Improvements 253
Measuring the Baseline Using the Amplifier XE Command Line 253
Identifying Concurrency Hotspots 255
Thread Concurrency and CPU Usage 255
Identifying Hotspots in the Code 256
Analyzing the Timeline 258
Questions to Answer 259
Fixing the Critical Section Hotspot 260
Analyzing an Algorithm 261
Conducting Further Analysis and Tuning 264
Using Other Viewpoints 268
Using Locks and Waits Analysis 268
Other Analysis Types 269
Using the Intel Software Autotuning Tool 271
Source Code 272
Summary 275

CHAPTER 10: PARALLEL ADVISOR–DRIVEN DESIGN 277

Using Parallel Advisor 277
Understanding the Advisor Workflow 279
Finding Documentation 280
Getting Started with the NQueens Example Program 280
Surveying the Site 282
Running a Survey Analysis 282
The Survey Report 282
Finding Candidate Parallel Regions 283
The Survey Source Window 284
How Survey Analysis Works 285
Annotating Your Code 286
Site Annotations 286
Lock Annotations 287
Adding Annotations 288
Checking Suitability 290
Running a Suitability Analysis 290
The Suitability Report 291
Parallel Choices 292
Using the Suitability Report 293
How Suitability Analysis Works 294
Checking for Correctness 295
Running a Correctness Analysis 296
The Correctness Report 296
The Correctness Source Window 297
Understanding Common Problems 298
Using the Correctness Report 301
Correctness Analysis Limitation 301
How Correctness Analysis Works 302
Replacing Annotations 304
The Summary Report 304
Common Mappings 305
Summary 308

CHAPTER 11: DEBUGGING PARALLEL APPLICATIONS 309

Introduction to the Intel Debugger 309
The Parallel Debugger Workflow 310
Using the Intel Debugger to Detect Data Races 311
Building the Serial Program 312
Adding Parallelism 313

Observing the Results 315
Serializing the Parallel Code 315
Detecting Data Races 317
Using Filters 319
Using Suppression Filters to Discard Unwanted Events 319
Creating the Filters 320
Fixing the Data Races 323
Using Focus Filters to Examine a Selected Portion of Code 325
Creating the Filters 327
Correcting the mbox Data Race 329
More About Filters 332
Runtime Investigation: Viewing the State of Your Application 333
Using the OpenMP Tasks Window to Investigate Variables Within a Parallel Region 334
Using the OpenMP Spawn Tree Window to View the Behavior of Parallel Code 336
Summary 339

CHAPTER 12: EVENT-BASED ANALYSIS WITH VTUNE AMPLIFIER XE 341

Testing the Health of an Application 342
What Causes a High CPI? 342
Is CPI on Its Own a Good Enough Measure of Health? 343
Conducting a System-Wide Analysis 343
Conducting a Hotspot Analysis 345
Hotspot Analysis Types 346
User Mode Hotspots Versus Lightweight Hotspots 346
Finding Hotspots in Code 350
Conducting a General Exploration Analysis 352
A Quick Anatomy Class 355
CPU Internals 355
Categories of Execution Behavior 356
Fixing Hardware Issues 358
Reducing Cache Misses 359
Using More Efficient Instructions 360
Using the Intel Compiler 361
Using Amplifiers XE's Other Tools 364
Using Predefined Analysis Types 364
Using Viewpoints 364
Using APIs 366
The Pause and Resume API 366
The Frame API 368

Using Amplifier XE from the Command Line 369
Finding More Information 370
The Example Application 371
Summary 374

PART III: CASE STUDIES

CHAPTER 13: THE WORLD'S FIRST SUDOKU "THIRTY-NINER" 377

The Sudoku Optimization Challenge 377
The Nature of the Challenge 378
The High-Level Design 379
Optimizing the Solver Using SSE Intrinsics 380
Adding Parallelism to the Generator 382
The Results 383
Hands-On Example: Optimizing the Sudoku Generator 384
About the Code 385
The Solver 386
Finding Hotspots in the Solver 388
Optimizing the Code Using SSE Intrinsics 390
The Generator 390
Finding the Hotspots in the Generator 391
Adding Parallelism to the Generator Using OpenMP 391
Checking Correctness in the Generator 392
Fixing Correctness in the Generator 393
Tuning Performance 394
Summary 396

CHAPTER 14: NINE TIPS TO PARALLEL-PROGRAMMING HEAVEN 397

The Challenge: Simulating Star Formation 397
The Formation of Stars 398
The Hands-On Activities 399
Performance Tuning 400
Application Heuristics 400
Finding the Hotspots 400
Using a Tree-Based N-Bodies Simulation 403
Using a Hashed Octree 405
Architectural Tuning 407
Adding Parallelism 410
Identifying the Hotspot and Discovering the Calling Sequence 410

Implementing Parallelism 410
Detecting Data Races and Other Potential Errors 412
Correcting the Data Race 413
Load Balancing 414
The Results 415
Summary 416

CHAPTER 15: PARALLEL TRACK FITTING IN THE CERN COLLIDER 419

The Case Study 419
The Stages of a High-Energy Physics Experiment 420
The Track Reconstruction Stages 421
Track Finding 423
Track Fitting 425
What Is Array Building Blocks? 427
Parallelizing the Track-Fitting Code 430
Adding Array Building Blocks to Existing Code 430
Code Refactoring 431
An Example of Class Change 431
An Example of Kernel Code Change 432
Changing to Structure of Arrays 433
The Results 434
Correctness 435
Speedup and Scalability 435
Parallelism and Concurrency 438
The Hands-On Project 440
The Activities 440
The Projects 441
Building and Running the Serial Version 441
The Serial Track-Fitting Code 441
The Application Output 443
Parallelizing the Track-Fitting Code 444
Configuring the Array Building Blocks Build Environment 444
Writing the Parallel Driver 448
Identifying the Kernel in the Driver 449
Allocating and Binding 450
Invoking the Kernel 452
Implementing the Kernel 454
Summary 460

CHAPTER 16: PARALLELIZING LEGACY CODE 463

Introducing the Dhrystone Benchmark 464
The Structure of the Code 464
Global and Shared Variables 464
The Hands-On Projects 466
Building the Projects 466
Project Targets 466
An Example Build 467
Adding Amplifier XE APIs to Timestamp the Dhrystone Loop 468
Viewing the Results 469
Parallelizing the C Version 472
Attempt One: Synchronizing Shared Variable Access 472
The Results 473
Is It Successful? 475
Attempt Two: Duplicating Global Variables 476
Initializing and Accessing the Global Variables 477
The Results 477
Is It Successful? 478
Parallelizing the C++ Version 478
Attempt Three: Wrapping the Application in a C++ Class 479
Scheduling the Parallel Runs 480
Silencing the Output 480
The Results 481
Is It Successful? 482
Attempt Four: Using Cilk Plus Holders 482
Developing the Wrappers 483
The Results 486
Is It Successful? 486
Overview of the Results 487
Performance 487
Editing Effort 488
Summary 488

INDEX ***489***