Contents

Preface xix

Bio xxiii

1 Introduction to Parallel Processing in R 1

1.1 Recurring Theme: The Principle of Pretty Good Parallelism 1

1.1.1 Fast *Enough* 1

1.1.2 "R+X" 2

1.2 A Note on Machines 3

1.3 Recurring Theme: Hedging One's Bets 3

1.4 Extended Example: Mutual Web Outlinks 4

1.4.1 Serial Code 5

1.4.2 Choice of Parallel Tool 7

1.4.3 Meaning of "snow" in This Book 8

1.4.4 Introduction to snow 8

1.4.5 Mutual Outlinks Problem, Solution 1 8

1.4.5.1 Code 9

1.4.5.2 Timings 10

1.4.5.3 Analysis of the Code 11

2 "Why Is My Program So Slow?": Obstacles to Speed 17

2.1 Obstacles to Speed 17

2.2 Performance and Hardware Structures 18

2.3 Memory Basics 20

2.3.1 Caches 20

2.3.2 Virtual Memory 22

2.3.3 Monitoring Cache Misses and Page Faults 23

2.3.4 Locality of Reference 23

2.4 Network Basics 23

2.5 Latency and Bandwidth 24

2.5.1 Two Representative Hardware Platforms: Multicore Machines and Clusters 25

2.5.1.1 Multicore 26

2.5.1.2 Clusters 29

2.5.2 The Principle of "Just Leave It There" 29

2.6 Thread Scheduling 29

2.7 How Many Processes/Threads? 31

2.8 Example: Mutual Outlink Problem 31

2.9 "Big O" Notation 32

2.10 Data Serialization 33

2.11 "Embarrassingly Parallel" Applications 33

2.11.1 What People Mean by "Embarrassingly Parallel" . . 33

2.11.2 Suitable Platforms for Non-Embarrassingly Parallel Applications 34

3 Principles of Parallel Loop Scheduling 35

3.1 General Notions of Loop Scheduling 36

3.2 Chunking in snow 38

3.2.1 Example: Mutual Outlinks Problem 38
3.3 A Note on Code Complexity 40
3.4 Example: All Possible Regressions 41
3.4.1 Parallelization Strategies 41
3.4.2 The Code 42
3.4.3 Sample Run 45
3.4.4 Code Analysis 45
3.4.4.1 Our Task List 46
3.4.4.2 Chunking 47
3.4.4.3 Task Scheduling 47
3.4.4.4 The Actual Dispatching of Work 48
3.4.4.5 Wrapping Up 50
3.4.5 Timing Experiments 50
3.5 The partools Package 52
3.6 Example: All Possible Regressions, Improved Version 52
3.6.1 Code 53
3.6.2 Code Analysis 56
3.6.3 Timings 56
3.7 Introducing Another Tool: multicore 57
3.7.1 Source of the Performance Advantage 58
3.7.2 Example: All Possible Regressions, Using multicore . 59
3.8 Issues with Chunk Size 63
3.9 Example: Parallel Distance Computation 64
3.9.1 The Code 65
3.9.2 Timings 68
3.10 The foreach Package 69
3.10.1 Example: Mutual Outlinks Problem 70

3.10.2 A Caution When Using foreach 72
3.11 Stride 73
3.12 Another Scheduling Approach: Random Task Permutation 74
3.12.1 The Math 74
3.12.2 The Random Method vs. Others, in Practice 76
3.13 Debugging snow and multicore Code 77
3.13.1 Debugging in snow 77
3.13.2 Debugging in multicore 78

4 The Shared-Memory Paradigm: A Gentle Introduction via R 79

4.1 So, What Is Actually Shared? 80
4.1.1 Global Variables 80
4.1.2 Local Variables: Stack Structures 81
4.1.3 Non-Shared Memory Systems 83
4.2 Clarity of Shared-Memory Code 83
4.3 High-Level Introduction to Shared-Memory Programming: Rdsm Package 84
4.3.1 Use of Shared Memory 85
4.4 Example: Matrix Multiplication 85
4.4.1 The Code 86
4.4.2 Analysis 87
4.4.3 The Code 88
4.4.4 A Closer Look at the Shared Nature of Our Data . . 89
4.4.5 Timing Comparison 90
4.4.6 Leveraging R 91
4.5 Shared Memory Can Bring A Performance Advantage . . . 91
4.6 Locks and Barriers 94

4.6.1 Race Conditions and Critical Sections 94
4.6.2 Locks 95
4.6.3 Barriers 97
4.7 Example: Maximal Burst in a Time Series 97
4.7.1 The Code 97
4.8 Example: Transforming an Adjacency Matrix 99
4.8.1 The Code 100
4.8.2 Overallocation of Memory 104
4.8.3 Timing Experiment 104
4.9 Example: k-Means Clustering 106
4.9.1 The Code 106
4.9.2 Timing Experiment 113

5 The Shared-Memory Paradigm: C Level 115

5.1 OpenMP 116
5.2 Example: Finding the Maximal Burst in a Time Series . . . 116
5.2.1 The Code 116
5.2.2 Compiling and Running 119
5.2.3 Analysis 120
5.2.4 A Cautionary Note About Thread Scheduling 123
5.2.5 Setting the Number of Threads 123
5.2.6 Timings 124
5.3 OpenMP Loop Scheduling Options 124
5.3.1 OpenMP Scheduling Options 124
5.3.2 Scheduling through Work Stealing 125
5.4 Example: Transforming an Adjacency Matrix 127
5.4.1 The Code 127

5.4.2 Analysis of the Code 129
5.5 Example: Adjacency Matrix, R-Callable Code 132
5.5.1 The Code, for .C() 132
5.5.2 Compiling and Running 134
5.5.3 Analysis 137
5.5.4 The Code, for Rcpp 137
5.5.5 Compiling and Running 140
5.5.6 Code Analysis 141
5.5.7 Advanced Rcpp 143
5.6 Speedup in C 143
5.7 Run Time vs. Development Time 144
5.8 Further Cache/Virtual Memory Issues 144
5.9 Reduction Operations in OpenMP 149
5.9.1 Example: Mutual In-Links 149
5.9.1.1 The Code 150
5.9.1.2 Sample Run 151
5.9.1.3 Analysis 151
5.9.2 Cache Issues 152
5.9.3 Rows vs. Columns 152
5.9.4 Processor Affinity 152
5.10 Debugging 153
5.10.1 Threads Commands in GDB 153
5.10.2 Using GDB on C/C++ Code Called from R 153
5.11 Intel Thread Building Blocks (TBB) 154
5.12 Lockfree Synchronization 155

6 The Shared-Memory Paradigm: GPUs 157

6.1 Overview 157

6.2 Another Note on Code Complexity 158

6.3 Goal of This Chapter 159

6.4 Introduction to NVIDIA GPUs and CUDA 159

6.4.1 Example: Calculate Row Sums 160

6.4.2 NVIDIA GPU Hardware Structure 164

6.4.2.1 Cores 164

6.4.2.2 Threads 165

6.4.2.3 The Problem of Thread Divergence 166

6.4.2.4 "OS in Hardware" 166

6.4.2.5 Grid Configuration Choices 167

6.4.2.6 Latency Hiding in GPUs 168

6.4.2.7 Shared Memory 168

6.4.2.8 More Hardware Details 169

6.4.2.9 Resource Limitations 169

6.5 Example: Mutual Inlinks Problem 170

6.5.1 The Code 170

6.5.2 Timing Experiments 173

6.6 Synchronization on GPUs 173

6.6.1 Data in Global Memory Is Persistent 174

6.7 R and GPUs 175

6.7.1 Example: Parallel Distance Computation 175

6.8 The Intel Xeon Phi Chip 176

7 Thrust and Rth 179

7.1 Hedging One's Bets 179

7.2 Thrust Overview 180

7.3 Rth 180

7.4 Skipping the C++ 181

7.5 Example: Finding Quantiles 181

7.5.1 The Code 181

7.5.2 Compilation and Timings 183

7.5.3 Code Analysis 183

7.6 Introduction to Rth 186

8 The Message Passing Paradigm 189

8.1 Message Passing Overview 189

8.2 The Cluster Model 190

8.3 Performance Issues 191

8.4 Rmpi 191

8.4.1 Installation and Execution 192

8.5 Example: Pipelined Method for Finding Primes 193

8.5.1 Algorithm 194

8.5.2 The Code 195

8.5.3 Timing Example 198

8.5.4 Latency, Bandwdith and Parallelism 199

8.5.5 Possible Improvements 199

8.5.6 Analysis of the Code 200

8.6 Memory Allocation Issues 203

8.7 Message-Passing Performance Subtleties 204

8.7.1 Blocking vs. Nonblocking I/O 204

8.7.2 The Dreaded Deadlock Problem 205

9 MapReduce Computation **207**

9.1 Apache Hadoop 208

9.1.1 Hadoop Streaming 208

9.1.2 Example: Word Count 208

9.1.3 Running the Code 209

9.1.4 Analysis of the Code 211

9.1.5 Role of Disk Files 212

9.2 Other MapReduce Systems 212

9.3 R Interfaces to MapReduce Systems 213

9.4 An Alternative: "Snowdoop" 213

9.4.1 Example: Snowdoop Word Count 214

9.4.2 Example: Snowdoop k-Means Clustering 215

10 Parallel Sorting and Merging **219**

10.1 The Elusive Goal of Optimality 219

10.2 Sorting Algorithms 220

10.2.1 Compare-and-Exchange Operations 220

10.2.2 Some "Representative" Sorting Algorithms 220

10.3 Example: Bucket Sort in R 223

10.4 Example: Quicksort in OpenMP 224

10.5 Sorting in Rth 226

10.6 Some Timing Comparisons 229

10.7 Sorting on Distributed Data 230

10.7.1 Hyperquicksort 230

11 Parallel Prefix Scan **233**

11.1 General Formulation 233

11.2 Applications 234

11.3 General Strategies 235

11.3.1 A Log-Based Method 235

11.3.2 Another Way 237

11.4 Implementations of Parallel Prefix Scan 238

11.5 Parallel cumsum() with OpenMP 239

11.5.1 Stack Size Limitations 241

11.5.2 Let's Try It Out 241

11.6 Example: Moving Average 242

11.6.1 Rth Code 243

11.6.2 Algorithm 244

11.6.3 Performance 245

11.6.4 Use of Lambda Functions 247

12 Parallel Matrix Operations **251**

12.1 Tiled Matrices 252

12.2 Example: Snowdoop Approach 253

12.3 Parallel Matrix Multiplication 254

12.3.1 Multiplication on Message-Passing Systems 255

12.3.1.1 Distributed Storage 255

12.3.1.2 Fox's Algorithm 255

12.3.1.3 Overhead Issues 256

12.3.2 Multiplication on Multicore Machines 257

12.3.2.1 Overhead Issues 257

12.3.3 Matrix Multiplication on GPUs 258

12.3.3.1 Overhead Issues 259

12.4 BLAS Libraries 260

12.4.1 Overview 260

12.5 Example: Performance of OpenBLAS 261

12.6 Example: Graph Connectedness 264
12.6.1 Analysis 264
12.6.2 The "Log Trick" 266
12.6.3 Parallel Computation 266
12.6.4 The matpow Package 267
12.6.4.1 Features 267
12.7 Solving Systems of Linear Equations 267
12.7.1 The Classical Approach: Gaussian Elimination and the LU Decomposition 268
12.7.2 The Jacobi Algorithm 270
12.7.2.1 Parallelization 270
12.7.3 Example: R/gputools Implementation of Jacobi . . . 271
12.7.4 QR Decomposition 271
12.7.5 Some Timing Results 272
12.8 Sparse Matrices 272

13 Inherently Statistical Approaches: Subset Methods 275
13.1 Chunk Averaging 275
13.1.1 Asymptotic Equivalence 276
13.1.2 O(·) Analysis 277
13.1.3 Code 278
13.1.4 Timing Experiments 278
13.1.4.1 Example: Quantile Regression 278
13.1.4.2 Example: Logistic Model 278
13.1.4.3 Example: Estimating Hazard Functions . . 281
13.1.5 Non-i.i.d. Settings 282
13.2 Bag of Little Bootstraps 283
13.3 Subsetting Variables 283

A Review of Matrix Algebra **285**

A.1 Terminology and Notation 285

A.1.1 Matrix Addition and Multiplication 286

A.2 Matrix Transpose 287

A.3 Linear Independence 288

A.4 Determinants 288

A.5 Matrix Inverse 288

A.6 Eigenvalues and Eigenvectors 289

A.7 Matrix Algebra in R 290

B R Quick Start **293**

B.1 Correspondences 293

B.2 Starting R 294

B.3 First Sample Programming Session 294

B.4 Second Sample Programming Session 298

B.5 Third Sample Programming Session 300

B.6 The R List Type 301

B.6.1 The Basics 301

B.6.2 The Reduce() Function 302

B.6.3 S3 Classes 302

B.6.4 Handy Utilities 304

B.7 Debugging in R 305

C Introduction to C for R Programmers **307**

C.0.1 Sample Program 307

C.0.2 Analysis 308

C.1 C++ 310

Index **311**