

Chapman & Hall/CRC Computational Science Series

Scientific Computing with Multicore and Accelerators

Edited by
Jakub Kurzak
David A. Bader
Jack Dongarra



CRC Press

Taylor & Francis Group

Boca Raton London New York

CRC Press is an imprint of the
Taylor & Francis Group, an **informa** business
A CHAPMAN & HALL BOOK

CRC Press
Taylor & Francis Group
6000 Broken Sound Parkway NW, Suite 300
Boca Raton, FL 33487-2742

© 2011 by Taylor and Francis Group, LLC
CRC Press is an imprint of Taylor & Francis Group, an Informa business

No claim to original U.S. Government works

Printed in the United States of America on acid-free paper
10 9 8 7 6 5 4 3 2 1

International Standard Book Number: 978-1-4398-2536-5 (Hardback)

This book contains information obtained from authentic and highly regarded sources. Reasonable efforts have been made to publish reliable data and information, but the author and publisher cannot assume responsibility for the validity of all materials or the consequences of their use. The authors and publishers have attempted to trace the copyright holders of all material reproduced in this publication and apologize to copyright holders if permission to publish in this form has not been obtained. If any copyright material has not been acknowledged please write and let us know so we may rectify in any future reprint.

Except as permitted under U.S. Copyright Law, no part of this book may be reprinted, reproduced, transmitted, or utilized in any form by any electronic, mechanical, or other means, now known or hereafter invented, including photocopying, micro-filming, and recording, or in any information storage or retrieval system, without written permission from the publishers.

For permission to photocopy or use material electronically from this work, please access www.copyright.com (<http://www.copyright.com/>) or contact the Copyright Clearance Center, Inc. (CCC), 222 Rosewood Drive, Danvers, MA 01923, 978-750-8400. CCC is a not-for-profit organization that provides licenses and registration for a variety of users. For organizations that have been granted a photocopy license by the CCC, a separate system of payment has been arranged.

Trademark Notice: Product or corporate names may be trademarks or registered trademarks, and are used only for identification and explanation without intent to infringe.

Library of Congress Cataloging-in-Publication Data

Scientific computing with multicore and accelerators / edited by Jakub Kurzak, David A. Bader, and Jack Dongarra.

p. cm. -- (Chapman & Hall/CRC computational science ; 10)

Summary: "The current trend in microprocessor architecture is toward powerful multicore designs in which a node contains several full-featured processing cores, private and shared caches, and memory. The IBM Cell Broadband Engine (B.E.) and Graphics Processing Units (GPUs) are two accelerators that are used for a variety of computations, including signal processing and quantum chemistry. This is the first reference on the use of Cell B.E. and GPUs as accelerators for numerical kernels, algorithms, and computational science and engineering applications. With contributions from leading experts, the book covers a broad range of topics on the increased role of these accelerators in scientific computing"-- Provided by publisher.

Includes bibliographical references and index.

ISBN 978-1-4398-2536-5 (hardback)

1. Science--Data processing. 2. Engineering--Data processing. 3. High performance computing. 4. Multiprocessors. I. Kurzak, Jakub. II. Bader, David A. III. Dongarra, J. J. IV. Title. V. Series.

Q183.9.S325 2010

502.85--dc22

2010037123

Visit the Taylor & Francis Web site at
<http://www.taylorandfrancis.com>

and the CRC Press Web site at
<http://www.crcpress.com>

Contents

List of Figures	xvii
List of Tables	xxv
Preface	xxvii
About the Editors	xxix
Contributors	xxxii
I Dense Linear Algebra	1
1 Implementing Matrix Multiplication on the Cell B. E.	3
<i>Wesley Alvaro, Jakub Kurzak, and Jack Dongarra</i>	
1.1 Introduction	3
1.1.1 Performance Considerations	4
1.1.2 Code Size Considerations	4
1.2 Implementation	5
1.2.1 Loop Construction	5
1.2.2 $C = C - A \times B$ trans	6
1.2.3 $C = C - A \times B$	10
1.2.4 Advancing Tile Pointers	12
1.3 Results	16
1.4 Summary	17
1.5 Code	18
Bibliography	19
2 Implementing Matrix Factorizations on the Cell B. E.	21
<i>Jakub Kurzak and Jack Dongarra</i>	
2.1 Introduction	21
2.2 Cholesky Factorization	22
2.3 Tile QR Factorization	23
2.4 SIMD Vectorization	26
2.5 Parallelization—Single Cell B. E.	28
2.6 Parallelization—Dual Cell B. E.	30
2.7 Results	31

2.8	Summary	32
2.9	Code	33
	Bibliography	34
3	Dense Linear Algebra for Hybrid GPU-Based Systems	37
	<i>Stanimire Tomov and Jack Dongarra</i>	
3.1	Introduction	37
3.1.1	Linear Algebra (LA)—Enabling New Architectures	38
3.1.2	MAGMA—LA Libraries for Hybrid Architectures	38
3.2	Hybrid DLA Algorithms	39
3.2.1	How to Code DLA for GPUs?	39
3.2.2	The Approach— <i>Hybridization of DLA Algorithms</i>	41
3.2.3	One-Sided Factorizations	43
3.2.4	Two-Sided Factorizations	46
3.3	Performance Results	50
3.4	Summary	53
	Bibliography	54
4	BLAS for GPUs	57
	<i>Rajib Nath, Stanimire Tomov, and Jack Dongarra</i>	
4.1	Introduction	57
4.2	BLAS Kernels Development	58
4.2.1	Level 1 BLAS	60
4.2.2	Level 2 BLAS	61
4.2.2.1	xGEMV	61
4.2.2.2	xSYMV	63
4.2.3	Level 3 BLAS	64
4.2.3.1	xGEMM	65
4.2.3.2	xSYRK	66
4.2.3.3	xTRSM	67
4.3	Generic Kernel Optimizations	68
4.3.1	Pointer Redirecting	68
4.3.2	Padding	72
4.3.3	Auto-Tuning	72
4.4	Summary	77
	Bibliography	79
II	Sparse Linear Algebra	81
5	Sparse Matrix-Vector Multiplication on Multicore and Accelerators	83
	<i>Samuel Williams, Nathan Bell, Jee Whan Choi, Michael Garland, Leonid Oliker, and Richard Vuduc</i>	
5.1	Introduction	84
5.2	Sparse Matrix-Vector Multiplication: Overview and Intuition	84

5.3	Architectures, Programming Models, and Matrices	86
5.3.1	Hardware Architectures	86
5.3.2	Parallel Programming Models	89
5.3.3	Matrices	90
5.4	Implications of Architecture on SpMV	91
5.4.1	Memory Subsystem	91
5.4.2	Processor Core	92
5.5	Optimization Principles for SpMV	93
5.5.1	Reorganization for Efficient Parallelization	93
5.5.2	Orchestrating Data Movement	95
5.5.3	Reducing Memory Traffic	96
5.5.4	Putting It All Together: Implementations	97
5.6	Results and Analysis	99
5.6.1	Xeon X5550 (Nehalem)	100
5.6.2	QS22 PowerXCell 8i	102
5.6.3	GTX 285	103
5.7	Summary: Cross-Study Comparison	105
	Acknowledgments	107
	Bibliography	108

III Multigrid Methods 111

6	Hardware-Oriented Multigrid Finite Element Solvers on GPU-Accelerated Clusters	113
	<i>Stefan Turek, Dominik Göttsche, Sven H.M. Buijssen, and Hilmar Wobker</i>	
6.1	Introduction and Motivation	113
6.2	FEAST—Finite Element Analysis and Solution Tools	116
6.2.1	Separation of Structured and Unstructured Data	117
6.2.2	Parallel Multigrid Solvers	117
6.2.3	Scalar and Multivariate Problems	118
6.2.4	Co-Processor Acceleration	119
6.3	Two FEAST Applications: FEASTSOLID and FEASTFLOW	120
6.3.1	Computational Solid Mechanics	120
6.3.2	Computational Fluid Dynamics	121
6.3.3	Solving CSM and CFD Problems with FEAST	122
6.4	Performance Assessments	124
6.4.1	GPU-Based Multigrid on a Single Subdomain	124
6.4.2	Scalability	125
6.4.3	Application Speedup	125
6.5	Summary	128
	Acknowledgments	128
	Bibliography	128

7	Mixed-Precision GPU-Multigrid Solvers with Strong Smoothers	131
	<i>Dominik Göddeke and Robert Strzodka</i>	
7.1	Introduction	131
7.1.1	Numerical Solution of Partial Differential Equations	132
7.1.2	Hardware-Oriented Discretization of Large Domains	132
7.1.3	Mixed-Precision Iterative Refinement Multigrid	133
7.2	Fine-Grained Parallelization of Multigrid Solvers	134
7.2.1	Smoothers on the CPU	134
7.2.2	Exact Parallelization: Jacobi and Tridiagonal Solvers	136
7.2.3	Multicolor Parallelization: Gauß-Seidel Solvers	138
7.2.4	Combination of Tridiagonal and Gauß-Seidel Smoothers	139
7.2.5	Alternating Direction Implicit Method	140
7.3	Numerical Evaluation and Performance Results	141
7.3.1	Test Procedure	141
7.3.2	Solver Configuration and Hardware Details	142
7.3.3	Numerical Evaluation	142
7.3.4	Runtime Efficiency	143
7.3.5	Smoother Selection	145
7.4	Summary and Conclusions	145
	Acknowledgments	145
	Bibliography	146
IV	Fast Fourier Transforms	149
8	Designing Fast Fourier Transform for the IBM Cell Broadband Engine	151
	<i>Virat Agarwal and David A. Bader</i>	
8.1	Introduction	151
8.2	Related Work	152
8.3	Fast Fourier Transform	154
8.4	Cell Broadband Engine Architecture	155
8.5	FFTC: Our FFT Algorithm for the Cell/B.E. Processor	158
8.5.1	Parallelizing FFTC for the Cell	158
8.5.2	Optimizing FFTC for the SPEs	159
8.6	Performance Analysis of FFTC	163
8.7	Conclusions	166
	Acknowledgments	167
	Bibliography	167
9	Implementing FFTs on Multicore Architectures	171
	<i>Alex Chunghen Chow, Gordon C. Fossum, and Daniel A. Brokenshire</i>	
9.1	Introduction	172
9.2	Computational Aspects of FFT Algorithms	173

9.2.1	An Upper Bound on FFT Performance	174
9.3	Data Movement and Preparation of FFT Algorithms	175
9.4	Multicore FFT Performance Optimization	177
9.5	Memory Hierarchy	178
9.5.1	Registers and Load and Store Operations	180
9.5.1.1	Applying SIMD Operations	180
9.5.1.2	Instruction Pipeline	181
9.5.1.3	Multi-Issue Instructions	182
9.5.2	Private and Shared Core Memory, and Their Data Movement	183
9.5.2.1	Factorization	183
9.5.2.2	Parallel Computation on Shared Core Mem- ory	183
9.5.3	System Memory	184
9.5.3.1	Index-Bit Reversal of Data Block Addresses	184
9.5.3.2	Transposition of the Elements	184
9.5.3.3	Load Balancing	185
9.6	Generic FFT Generators and Tooling	185
9.6.1	A Platform-Independent Expression of Performance Planning	185
9.6.2	Reducing the Mapping Space	186
9.6.3	Code Generation	187
9.7	Case Study: Large, Multi-Dimensional FFT on a Network Clustered System	187
9.8	Conclusion	190
	Bibliography	190

V Combinatorial Algorithms 193

10 Combinatorial Algorithm Design on the Cell/B.E. Processor 195

	<i>David A. Bader, Virat Agarwal, Kamesh Madduri, and Fabrizio Petrini</i>	
10.1	Introduction	195
10.2	Algorithm Design and Analysis on the Cell/B.E.	198
10.2.1	A Complexity Model	198
10.2.2	Analyzing Algorithms	199
10.3	List Ranking	201
10.3.1	A Parallelization Strategy	201
10.3.2	Complexity Analysis	202
10.3.3	A Novel Latency-Hiding Technique for Irregular Appli- cations	203
10.3.4	Cell/B.E. Implementation	205
10.3.5	Performance Results	206
10.4	Conclusions	213
	Acknowledgments	214
	Bibliography	214

VI Stencil Algorithms

217

11 Auto-Tuning Stencil Computations on Multicore and Accelerators	219
<i>Kaushik Datta, Samuel Williams, Vasily Volkov, Jonathan Carter, Leonid Oliker, John Shalf, and Katherine Yelick</i>	
11.1 Introduction	220
11.2 Stencil Overview	221
11.3 Experimental Testbed	222
11.4 Performance Expectation	223
11.4.1 Stencil Characteristics	225
11.4.2 A Brief Introduction to the Roofline Model	225
11.4.3 Roofline Model-Based Performance Expectations	227
11.5 Stencil Optimizations	231
11.5.1 Parallelization and Problem Decomposition	232
11.5.2 Data Allocation	233
11.5.3 Bandwidth Optimizations	233
11.5.4 In-Core Optimizations	234
11.5.5 Algorithmic Transformations	235
11.6 Auto-tuning Methodology	236
11.6.1 Architecture-Specific Exceptions	238
11.7 Results and Analysis	239
11.7.1 Nehalem Performance	240
11.7.2 Barcelona Performance	242
11.7.3 Clovertown Performance	243
11.7.4 Blue Gene/P Performance	244
11.7.5 Victoria Falls Performance	244
11.7.6 Cell Performance	245
11.7.7 GTX280 Performance	246
11.7.8 Cross-Platform Performance and Power Comparison	247
11.8 Conclusions	249
Acknowledgments	251
Bibliography	251
12 Manycore Stencil Computations in Hyperthermia Applications	255
<i>Matthias Christen, Olaf Schenk, Esra Neufeld, Maarten Paulides, and Helmar Burkhart</i>	
12.1 Introduction	255
12.2 Hyperthermia Applications	256
12.3 Bandwidth-Saving Stencil Computations	259
12.3.1 Spatial Blocking and Parallelization	259
12.3.2 Temporal Blocking	261
12.3.2.1 Temporally Blocking the Hyperthermia Stencil	263

12.3.2.2	Speedup for the Hyperthermia Stencil . . .	264
12.4	Experimental Performance Results	266
12.4.1	Kernel Benchmarks	268
12.4.2	Application Benchmarks	271
12.5	Related Work	273
12.6	Conclusion	273
	Acknowledgments	274
	Bibliography	274
VII Bioinformatics		279
13 Enabling Bioinformatics Algorithms on the Cell/B.E. Processor		281
	<i>Vipin Sachdeva, Michael Kistler, and Tzy-Hwa Kathy Tzeng</i>	
13.1	Computational Biology and High-Performance Computing	281
13.2	The Cell/B.E. Processor	283
13.2.1	Cache Implementation on Cell/B.E.	283
13.3	Sequence Analysis and Its Applications	284
13.4	Sequence Analysis on the Cell/B.E. Processor	286
13.4.1	ClustalW	286
13.4.2	FASTA	288
13.5	Results	289
13.5.1	Experimental Setup	289
13.5.2	ClustalW Results and Analysis	289
13.5.3	FASTA Results and Analysis	293
13.6	Conclusions and Future Work	294
	Bibliography	295
14 Pairwise Computations on the Cell Processor		297
	<i>Abhinav Sarje, Jaroslaw Zola, and Srinivas Aluru</i>	
14.1	Introduction	298
14.2	Scheduling Pairwise Computations	299
14.2.1	Tiling	300
14.2.2	Tile Ordering	301
14.2.3	Tile Size	302
14.2.3.1	Fetching Input Vectors	303
14.2.3.2	Shifting Column Vectors	303
14.2.3.3	Transferring Output Data	303
14.2.3.4	Minimizing Number of DMA Transfers	304
14.2.4	Extending Tiling across Multiple Cell Processors	305
14.2.5	Extending Tiling to Large Number of Dimensions	306
14.3	Reconstructing Gene Regulatory Networks	308
14.3.1	Computing Pairwise Mutual Information on the Cell	309
14.3.2	Performance of Pairwise MI Computations on One Cell Blade	310

14.3.3	Performance of MI Computations on Multiple Cell Blades	311
14.4	Pairwise Genomic Alignments	312
14.4.1	Computing Alignments	312
14.4.1.1	Global/Local Alignment	313
14.4.1.2	Spliced Alignment	314
14.4.1.3	Syntenic Alignment	315
14.4.2	A Parallel Alignment Algorithm for the Cell BE . . .	316
14.4.2.1	Parallel Alignment using Prefix Computations	316
14.4.2.2	Wavefont Communication Scheme	317
14.4.2.3	A Hybrid Parallel Algorithm	318
14.4.2.4	Hirschberg's Technique for Linear Space . .	320
14.4.2.5	Algorithms for Specialized Alignments . . .	321
14.4.2.6	Memory Usage	321
14.4.3	Performance of the Hybrid Alignment Algorithms . .	321
14.5	Ending Notes	323
	Acknowledgment	324
	Bibliography	324
VIII	Molecular Modeling	329
15	Drug Design on the Cell BE	331
	<i>Cecilia González-Álvarez, Harald Servat, Daniel Cabrera-Benítez, Xavier Aguilar, Carles Pons, Juan Fernández-Recio, and Daniel Jiménez-González</i>	
15.1	Introduction	332
15.2	Bioinformatics and Drug Design	333
15.2.1	Protein-Ligand Docking	335
15.2.2	Protein-Protein Docking	336
15.2.3	Molecular Mechanics	337
15.3	Cell BE Porting Analysis	337
15.4	Experimental Setup	339
15.5	Case Study: Docking with FTDock	339
15.5.1	Algorithm Description	339
15.5.2	Profiling and Implementation	340
15.5.3	Performance Evaluation	341
15.6	Case Study: Molecular Dynamics with Moldy	343
15.6.1	Algorithm Description	343
15.6.2	Profiling and Implementation	343
15.6.3	Performance Evaluation	344
15.7	Conclusions	345
	Acknowledgments	346
	Bibliography	346

16 GPU Algorithms for Molecular Modeling	351
<i>John E. Stone, David J. Hardy, Barry Isralewitz, and Klaus Schulten</i>	
16.1 Introduction	352
16.2 Computational Challenges of Molecular Modeling	352
16.3 GPU Overview	353
16.3.1 GPU Hardware Organization	354
16.3.2 GPU Programming Model	355
16.4 GPU Particle-Grid Algorithms	357
16.4.1 Electrostatic Potential	357
16.4.2 Direct Summation on GPUs	358
16.4.3 Cutoff Summation on GPUs	360
16.4.4 Floating-Point Precision Effects	361
16.5 GPU N-Body Algorithms	361
16.5.1 N-Body Forces	361
16.5.2 N-Body Forces on GPUs	362
16.5.3 Long-Range Electrostatic Forces	364
16.6 Adapting Software for GPU Acceleration	365
16.6.1 Case Study: NAMD Parallel Molecular Dynamics	365
16.6.2 Case Study: VMD Molecular Graphics and Analysis	367
16.7 Concluding Remarks	368
Acknowledgments	369
Bibliography	369
IX Complementary Topics	373
17 Dataflow Frameworks for Emerging Heterogeneous Architectures and Their Application to Biomedicine	375
<i>Umit V. Catalyurek, Renato Ferreira, Timothy D. R. Hartley, George Teodoro, and Rafael Sachetto</i>	
17.1 Motivation	375
17.2 Dataflow Computing Model and Runtime Support	377
17.3 Use Case Application: Neuroblastoma Image Analysis System	378
17.4 Middleware for Multi-Granularity Dataflow	380
17.4.1 Coarse-grained on Distributed GPU Clusters	381
17.4.1.1 Supporting Heterogeneous Resources	381
17.4.1.2 Experimental Evaluation	383
17.4.2 Fine-Grained on Cell	386
17.4.2.1 DCL for Cell—Design and Architecture	386
17.4.2.2 NBIA with DCL	386
17.5 Conclusions and Future Work	388
Acknowledgments	389
Bibliography	389

18 Accelerator Support in the Charm++ Parallel Programming Model	393
<i>Łazmikan V. Kalé, David M. Kunzman, and Lukasz Wesolowski</i>	
18.1 Introduction	393
18.2 Motivations and Goals of Our Work	394
18.3 The Charm++ Parallel Programming Model	396
18.3.1 General Description of Charm++	396
18.3.2 Suitability of Charm++ for Exploiting Accelerators .	397
18.4 Support for Cell and Larrabee in Charm++	398
18.4.1 SIMD Instruction Abstraction	400
18.4.2 Accelerated Entry Methods	401
18.4.3 Support for Heterogeneous Systems	403
18.4.4 Performance	404
18.5 Support for CUDA-Based GPUs	405
18.6 Related Work	407
18.7 Concluding Remarks	408
Bibliography	408
19 Efficient Parallel Scan Algorithms for Manycore GPUs	413
<i>Shubhabrata Sengupta, Mark Harris, Michael Garland, and John D. Owens</i>	
19.1 Introduction	414
19.2 CUDA—A General-Purpose Parallel Computing Architecture for Graphics Processors	416
19.3 Scan: An Algorithmic Primitive for Efficient Data-Parallel Computation	417
19.3.1 Scan	417
19.3.1.1 A Serial Implementation	418
19.3.1.2 A Basic Parallel Implementation	418
19.3.2 Segmented Scan	420
19.4 Design of an Efficient Scan Algorithm	421
19.4.1 Hierarchy of the Scan Algorithm	421
19.4.2 Intra-Warp Scan Algorithm	422
19.4.3 Intra-Block Scan Algorithm	423
19.4.4 Global Scan Algorithm	423
19.5 Design of an Efficient Segmented Scan Algorithm	425
19.5.1 Operator Transformation	425
19.5.2 Direct Intra-Warp Segmented Scan	426
19.5.3 Block and Global Segmented Scan Algorithms	430
19.6 Algorithmic Complexity	433
19.7 Some Alternative Designs for Scan Algorithms	434
19.7.1 Saving Bandwidth by Performing a Reduction	434
19.7.2 Eliminating Recursion by Performing More Work per Block	435
19.8 Optimizations in CUDPP	437

19.9	Performance Analysis	437
19.10	Conclusions	440
	Acknowledgments	440
	Bibliography	441
20	High Performance Topology-Aware Communication in Multicore Processors	443
	<i>Hari Subramoni, Fabrizio Petrini, Virat Agarwal, and Davide Pasetto</i>	
20.1	Introduction	444
20.2	Background	445
20.2.1	Intel Nehalem	445
20.2.2	Sun Niagara	446
20.2.3	AMD Opteron	447
20.2.4	MPI	447
20.3	Methodology	448
20.3.1	Basic Memory-Based Copy	448
20.3.2	Vector Instructions	448
20.3.3	Streaming Instructions	449
20.3.4	Kernel-Based Direct Copy	449
20.4	Experimental Results	449
20.4.1	Intra-Socket Performance Results	450
20.4.2	Inter-Socket Performance Results	454
20.4.3	Comparison with MPI	455
20.4.4	Performance Comparison of Different Multicore Architectures	457
20.5	Related Work	458
20.6	Conclusion and Future Work	458
	Bibliography	459

List of Figures

1.1	Basic steps of <code>_GEMM</code> loop optimization.	6
1.2	Basic operation of the $C = C - A \times B^T$ matrix multiplication micro-kernel.	7
1.3	Basic operation of the $C = C - A \times B$ matrix multiplication micro-kernel.	10
1.4	Sample C language implementation of pointer arithmetic for the kernel $C = C - A \times B^T$ with unrolling corresponding to the triplet $\{4, 4, 64\}$	13
1.5	The result of compiling the code from Figure 1.4 to assembly language, with even pipeline instructions in bold.	14
1.6	Organization of the tile offset lookup table. N is the number of tiles.	15
2.1	Tile operations in the Cholesky factorization.	22
2.2	Pseudocode of the (left-looking) Cholesky factorization.	23
2.3	<i>Left:</i> Tile operations in the tile QR factorization. The sequence is left-to-right and top-down. Hatching indicates input data, shade of gray indicates in/out data. <i>Right:</i> Inner blocking in the tile QR factorization.	24
2.4	Pseudocode of the tile QR factorization.	25
2.5	Assignment of work to SPEs for the Cholesky factorization (left) and the tile QR factorization (right).	28
2.6	Direct Acyclic Graphs of the Cholesky factorization (left) and the tile QR factorization (right) for a matrix of size 5×5 tiles.	29
2.7	Execution trace of the tile QR factorization of a 512×512 matrix. (total time: $1645 \mu\text{s}$, execution rate: 109 Gflop/s).	30
2.8	Performance of the Cholesky factorization (left) and the tile QR factorization (right) in single precision on an IBM QS20 blade using two Cell B. E. chips (16 SPEs).	32
3.1	BLAS on GPU (GTX 280) vs CPU ($8 \times$ Intel Xeon 2.33GHz).	40
3.2	Algorithms as a collection of BLAS-based tasks and dependencies among them (DAGs) for hybrid GPU-based computing.	42
3.3	A typical hybrid pattern of computation and communication for the one-sided matrix factorizations in MAGMA's GPU interface.	44

3.4	Pseudo-code implementation of the hybrid Cholesky. hA and dB are pointers to the matrix to be factored correspondingly on the host (CPU) and the device (GPU).	45
3.5	Time breakdown for the hybrid QR from MAGMA in single precision arithmetic on GTX280 GPU and Intel Xeon 2.33GHz CPU.	45
3.6	HR computational bottleneck: Level 2 BLAS $y_j = A_j v_j$	48
3.7	Main tasks and their scheduling.	49
3.8	CPU/GPU communications for inner/outer iteration j/i	50
3.9	Performance of MAGMA's hybrid Cholesky in single (left) and double precision (right) arithmetic on GTX 280 <i>vs</i> MKL 10.1 and LAPACK (with multi-threaded BLAS) on Intel Xeon dual socket quad-core 2.33GHz.	51
3.10	Performance of MAGMA's hybrid QR in single (left) and double precision (right) arithmetic on GTX 280 <i>vs</i> MKL 10.1 and LAPACK (with multi-threaded BLAS) on Intel Xeon dual socket quad-core 2.33GHz.	51
3.11	Performance of MAGMA's hybrid LU in single (left) and double precision (right) arithmetic on GTX 280 <i>vs</i> MKL 10.1 and LAPACK (with multi-threaded BLAS) on Intel Xeon dual socket quad-core 2.33GHz.	52
3.12	Performance of the hybrid HR in double precision (bottom) arithmetic on GTX 280 <i>vs</i> MKL 10.1 on Intel Xeon dual socket quad-core 2.33GHz.	52
4.1	Algorithmic view of Level 1 and Level 2 BLAS.	60
4.2	Performance of xGEMV (non-transpose) on a GTX 280.	62
4.3	Two memory access implementations of xGEMV (transpose).	62
4.4	Performance of xGEMV (transpose) on a GTX 280.	63
4.5	Three cases of TB computations in xSYMV.	63
4.6	Performance of xSYMV on a GTX 280.	64
4.7	The GPU GEMM ($C = AB$) of a single TB.	65
4.8	Performance of GEMM ($C = \alpha AB^T + \beta C$) on a GTX 280.	66
4.9	Performance of xSYRK on a GTX 280.	67
4.10	Performance of xTRSM on a GTX 280.	68
4.11	Performance of GEMM on square matrices on a GTX 280.	69
4.12	Performance of GEMM with conditional statements.	70
4.13	Possible illegal memory references in GEMM.	70
4.14	GPU GEMM ($C = \alpha AB + \beta C$) with pointer redirecting.	70
4.15	Performance of DGEMM on a GTX 280.	71
4.16	Performance of SGEMM on a GTX 280.	71
4.17	Performance of xGEMM with padding on a GTX 280.	72
4.18	Performance of auto-tuned DGEMM kernel ($Op(A) = A^T$, $Op(B) = B$) on a GTX 280.	76

4.19	Performance of the auto-tuned SGEMM ($Op(A) = A$, $Op(B) = B^T$) kernel for square matrices on a GTX 280.	76
4.20	Performance comparison of the auto-tuned (solid line) vs CUBLAS 2.3 DGEMMs occurring in the block LU factorization (for block sizes $BS = 64$ on the left and 128 on the right) of a matrix of size 6144×6144	78
5.1	CSR visualization	85
5.2	Benchmark matrices used	91
5.3	Nehalem performance	100
5.4	Auto-tuned Nehalem performance	101
5.5	Cell performance	103
5.6	GPU performance	104
5.7	Architectural performance comparison	106
6.1	FEAST example geometry and mesh.	117
6.2	FEAST accelerator design.	119
6.3	Weak scalability.	125
6.4	Application benchmarks.	126
7.1	Tensor product property of one subdomain, unstructured coarse mesh, system matrix structure and strongly deformed subdomain which still exhibits the tensor product property.	133
7.2	Numbering of the unknowns (left) and illustration of the sparsity structure of the system matrix (right, only the first half of the unknowns shown).	135
7.3	Multicolored <i>MC-GSROW</i> preconditioner.	139
7.4	Convergence rates of the outer solver, on the CPU (left) and the GPU (right), logarithmic scale.	143
7.5	Efficiency of the solvers, on the CPU (left) and the GPU (right), logarithmic scale.	143
7.6	Speedup of the GPU over the CPU, logarithmic scale.	144
8.1	Butterflies of the ordered DIF FFT algorithm.	156
8.2	Cell Broadband Engine Architecture.	156
8.3	Partition of the input array among the SPEs (e.g. 8 SPEs in this illustration).	158
8.4	Vectorization of the first two stages of the FFT algorithm. These stages require a shuffle operation over the output vector to generate the desired output.	159
8.5	Stages of the synchronization barrier using inter-SPE communication.	160
8.6	Running time of our FFTC code on 1K and 4K inputs as we increase the number of SPEs.	163