

Parallel MATLAB

for Multicore and Multinode Computers

SOFTWARE • ENVIRONMENTS • TOOLS

The SIAM series on Software, Environments, and Tools focuses on the practical implementation of computational methods and the high performance aspects of scientific computation by emphasizing in-demand software, computing environments, and tools for computing. Software technology development issues such as current status, applications and algorithms, mathematical software, software tools, languages and compilers, computing environments, and visualization are presented.

Editor-in-Chief

Jack J. Dongarra

University of Tennessee and Oak Ridge National Laboratory

Editorial Board

James W. Demmel, University of California, Berkeley

Dennis Gannon, Indiana University

Eric Grosse, AT&T Bell Laboratories

Jorge J. Moré, Argonne National Laboratory

Software, Environments, and Tools

Jeremy Kepner, *Parallel MATLAB for Multicore and Multinode Computers*

Michael A. Heroux, Padma Raghavan, and Horst D. Simon, editors, *Parallel Processing for Scientific Computing*

Gérard Meurant, *The Lanczos and Conjugate Gradient Algorithms: From Theory to Finite Precision Computations*

Bo Einarsson, editor, *Accuracy and Reliability in Scientific Computing*

Michael W. Berry and Murray Browne, *Understanding Search Engines: Mathematical Modeling and Text Retrieval, Second Edition*

Craig C. Douglas, Gundolf Haase, and Ulrich Langer, *A Tutorial on Elliptic PDE Solvers and Their Parallelization*

Louis Komzsik, *The Lanczos Method: Evolution and Application*

Bard Ermentrout, *Simulating, Analyzing, and Animating Dynamical Systems: A Guide to XPPAUT for Researchers and Students*

V. A. Barker, L. S. Blackford, J. Dongarra, J. Du Croz, S. Hammarling, M. Marinova, J. Waśniewski, and P. Yalamov, *LAPACK95 Users' Guide*

Stefan Goedecker and Adolfo Hoisie, *Performance Optimization of Numerically Intensive Codes*

Zhaojun Bai, James Demmel, Jack Dongarra, Axel Ruhe, and Henk van der Vorst, *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide*

Lloyd N. Trefethen, *Spectral Methods in MATLAB*

E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' Guide, Third Edition*

Michael W. Berry and Murray Browne, *Understanding Search Engines: Mathematical Modeling and Text Retrieval*

Jack J. Dongarra, Iain S. Duff, Danny C. Sorensen, and Henk A. van der Vorst, *Numerical Linear Algebra for High-Performance Computers*

R. B. Lehoucq, D. C. Sorensen, and C. Yang, *ARPACK Users' Guide: Solution of Large-Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods*

Randolph E. Bank, *PLTMG: A Software Package for Solving Elliptic Partial Differential Equations, Users' Guide 8.0*

L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling,

G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley, *ScaLAPACK Users' Guide*

Greg Astfalk, editor, *Applications on Advanced Architecture Computers*

Roger W. Hockney, *The Science of Computer Benchmarking*

Françoise Chaitin-Chatelin and Valérie Frayssé, *Lectures on Finite Precision Computations*

Parallel MATLAB

for Multicore and Multinode Computers

Jeremy Kepner

Massachusetts Institute of Technology
Lincoln Laboratory
Lexington, Massachusetts

siam.

Society for Industrial and Applied Mathematics
Philadelphia

Copyright © 2009 by the Society for Industrial and Applied Mathematics

10 9 8 7 6 5 4 3 2 1

All rights reserved. Printed in the United States of America. No part of this book may be reproduced, stored, or transmitted in any manner without the written permission of the publisher. For information, write to the Society for Industrial and Applied Mathematics, 3600 Market Street, 6th Floor, Philadelphia, PA 19104-2688 USA.

Trademarked names may be used in this book without the inclusion of a trademark symbol. These names are used in an editorial context only; no infringement of trademark is intended.

Intel is a registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

Java is a trademark of Sun Microsystems, Inc. in the United States and other countries.

Linux is a registered trademark of Linus Torvalds.

MATLAB is a registered trademark of The MathWorks, Inc. For MATLAB product information, please contact The MathWorks, Inc., 3 Apple Hill Drive, Natick, MA 01760-2098 USA, 508-647-7000, Fax: 508-647-7001, info@mathworks.com, www.mathworks.com.

Python is a trademark or registered trademark of the Python Software Foundation.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Chapters 5 and 7 are based on work previously published as “pMATLAB Parallel MATLAB Library.” The final, definitive version of this paper has been published in the *International Journal of High Performance Computing Applications*, Special Issue on High Productivity Languages and Models, J. Kepner and H. Zima, eds., Volume 21, Issue 3, pages 336–359 (August 2007) by SAGE Publications, Ltd. All rights reserved. © SAGE Publications. The article is available from SAGE Journals Online at <http://hpc.sagepub.com/cgi/content/abstract/21/3/336>.

Chapter 6 is based on work previously published as “Performance Metrics and Software Architecture,” by J. Kepner, T. Meuse, and G. Schrader, in *High Performance Embedded Computing Handbook*, D. R. Martinez, R. A. Bond, and M. M. Vai, eds., CRC Press, 2008. © Taylor & Francis Group LLC.

Figure 6.12 is based on Figure 4.4 on page 109 from *High Performance Computing: Challenges for Future Systems* by David Kuck (1996). It is used by permission of Oxford University Press. Visit <http://www.oup.com>.

This work is sponsored by the Department of the Air Force under Air Force Contract FA8721-05-C-0002. Opinions, interpretations, conclusions, and recommendations are those of the author and are not necessarily endorsed by the United States Government.

Library of Congress Cataloging-in-Publication Data

Kepner, Jeremy.

Parallel MATLAB for multicore and multinode computers / Jeremy Kepner.

p. cm. — (Software, environments, and tools ; 21)

Includes index.


ISBN 978-0-898716-73-3

1. Parallel processing (Electronic computers) 2. MATLAB. 3. Multiprocessors. I. Title.

QA76.58.K46 2009

004'.35—dc22

2009013013

 **siam** is a registered trademark.

For
Jemma
Alix
Clay
Diane
and
Gordon

Contents

List of Figures	xiii
List of Tables	xv
List of Algorithms	xvii
Preface	xix
Acknowledgments	xxiii
I Fundamentals	1
1 Primer: Notation and Interfaces	3
1.1 Algorithm notation	3
1.1.1 Distributed array notation	4
1.1.2 Distributed data access	5
1.2 Parallel function interfaces	6
1.2.1 Map-based programming	8
1.2.2 Parallel execution	9
2 Introduction to pMatlab	11
2.1 Program: Mandelbrot (fine-grained embarrassingly parallel) . . .	12
2.1.1 Getting started	12
2.1.2 Parallel design	13
2.1.3 Code	15
2.1.4 Debug	18
2.1.5 Test	19
2.2 Program: ZoomImage (coarse-grained embarrassingly parallel) . .	21
2.2.1 Getting started	21
2.2.2 Parallel design	22
2.2.3 Code	25
2.2.4 Debug	27
2.2.5 Test	28

2.3	Program: ParallelIO	29
2.3.1	Getting started	29
2.3.2	Parallel design	30
2.3.3	Code	31
2.3.4	Debug	34
2.3.5	Test	35
2.4	Why these worked	35
3	Interacting with Distributed Arrays	37
3.1	Getting started	38
3.2	Parallel design	39
3.3	Code	43
3.4	Interactive debug and test	46
3.4.1	Serial code correct	46
3.4.2	Parallel code correct	47
3.4.3	Local communication correct	47
3.4.4	Remote communication correct	48
3.4.5	Measuring performance	49
3.5	Advanced topic: Introduction to parallel pipelines	49
II	Advanced Techniques	53
4	Parallel Programming Models	55
4.1	Design: Knowing when to “go parallel”	55
4.1.1	Memory and performance profiling	56
4.1.2	Parallel programming patterns	59
4.1.3	Blurimage parallel design	62
4.2	Coding	67
4.2.1	Manager/worker	67
4.2.2	Message passing	69
4.2.3	Distributed arrays	70
4.3	Debug	71
4.4	Testing	72
4.5	Performance summary	74
4.6	Summary	74
5	Advanced Distributed Array Programming	77
5.1	Introduction	77
5.2	Pure versus fragmented distributed arrays	78
5.3	MATLAB distributed array interface and architecture design	80
5.4	Maps and distributions	80
5.5	Parallel support functions	83
5.5.1	Cyclic distributions and parallel load balancing	83
5.6	Concurrency versus locality	87
5.7	Ease of implementation	89

5.8	Implementation specifics	90
5.8.1	Program execution	92
5.9	Array redistribution	93
5.10	Message passing layer	95
5.11	Performance concepts	98
5.11.1	Performance, performance, performance	98
5.11.2	Benchmarks	99
5.11.3	Minimize overhead	99
5.11.4	Embarrassingly parallel implies linear speedup	100
5.11.5	Algorithm and mapping are orthogonal	100
5.11.6	Do no harm	100
5.11.7	Watch the SLOC	100
5.11.8	No free lunch	100
5.11.9	Four easy steps	100
5.12	User results	102
6	Performance Metrics and Software Architecture	107
6.1	Introduction	107
6.2	Characterizing a parallel application	108
6.2.1	Characteristics of the example programs	108
6.2.2	Serial performance metrics	112
6.2.3	Degrees of parallelism	113
6.2.4	Parallel performance metrics (no communication)	115
6.2.5	Parallel performance metrics (with communication)	116
6.2.6	Amdahl's Law	117
6.2.7	Characterizing speedup	119
6.2.8	Spatial and temporal locality	121
6.3	Standard parallel computer	123
6.3.1	Network model	125
6.3.2	Kuck hierarchy	128
6.4	Parallel programming models	129
6.5	System metrics	131
6.5.1	Performance	131
6.5.2	Form factor	132
6.5.3	Efficiency	133
6.5.4	Software cost	135
6.5.5	Software productivity	137
III	Case Studies	141
7	Parallel Application Analysis	143
7.1	Historical overview	143
7.2	Characterizing the application space	145
7.2.1	Physical memory hierarchy	146
7.2.2	Spatial/temporal locality	147
7.2.3	Logical memory hierarchy	148

7.3	HPC Challenge: Spanning the application space	148
7.3.1	Stream	149
7.3.2	FFT	149
7.3.3	RandomAccess	151
7.3.4	HPL	151
7.4	Intrinsic algorithm performance	152
7.4.1	Data structures	152
7.4.2	Computational complexity	152
7.4.3	Degrees of parallelism	154
7.4.4	Communication complexity	155
7.5	Hardware performance	155
7.5.1	Spatial and temporal locality	156
7.5.2	Performance efficiency	156
7.5.3	Estimating performance	157
7.5.4	Analysis results	159
7.5.5	Performance implications	159
7.6	Software performance	160
7.6.1	Stream	161
7.6.2	RandomAccess	161
7.6.3	FFT	162
7.6.4	HPL	163
7.7	Performance versus effort	163
8	Stream	169
8.1	Getting started	169
8.2	Parallel design	171
8.3	Code	173
8.4	Debug	176
8.5	Test	177
9	RandomAccess	181
9.1	Getting started	182
9.2	Parallel design	183
9.2.1	Spray algorithm	184
9.2.2	Tree algorithm	187
9.3	Code	189
9.3.1	Spray code	191
9.3.2	Tree code	193
9.3.3	Coding summary	194
9.4	Debug	195
9.5	Test	197
9.5.1	Multicore performance	197
9.5.2	Multinode performance	198
10	Fast Fourier Transform	201
10.1	Getting started	202
10.2	Parallel design	203

10.3	Code	206
10.4	Debug	208
10.5	Test	209
	10.5.1 Multicore performance	209
	10.5.2 Multinode performance	211
11	High Performance Linpack	215
11.1	Getting started	216
11.2	Parallel design	217
	11.2.1 Parallel LU	219
	11.2.2 Critical path analysis	221
11.3	Code	224
11.4	Debug	227
11.5	Test	229
	11.5.1 Multicore performance	229
	11.5.2 Multinode performance	230
Appendix	Notation for Hierarchical Parallel Multicore Algorithms	233
A.1	Introduction	233
A.2	Data parallelism	234
	A.2.1 Serial algorithm	234
	A.2.2 Parallel algorithm	235
	A.2.3 Block parallel algorithm	238
	A.2.4 Hierarchical parallel algorithm	239
	A.2.5 Hierarchical block parallel algorithm	240
A.3	Pipeline parallelism	242
	A.3.1 Implicit pipeline parallel	242
	A.3.2 Task pipeline parallel	243
	A.3.3 Fine-grained task pipeline parallel	246
	A.3.4 Generic hierarchical block parallel algorithm	247
Index		251

List of Figures

1	Performance versus effort.	xx
2	Development of parallel MATLAB.	xxiv
1.1	Parallel maps.	6
1.2	Anatomy of a map.	8
1.3	Parallel execution.	10
2.1	Mandelbrot output.	13
2.2	Mandelbrot parallel design.	16
2.3	ZoomImage output.	22
2.4	ZoomImage parallel design.	25
2.5	ParallelIO design.	32
3.1	Beamformer output.	38
3.2	Beamformer physics.	39
3.3	Beamformer parallel design.	43
3.4	Beamformer parallel pipeline design.	51
4.1	Blurimage output.	56
4.2	Blurimage profiler results.	58
4.3	Standard parallel communication patterns.	61
4.4	Distributed array mappings.	63
4.5	Blurimage parallel designs.	64
4.6	Four-step debug process.	73
5.1	Pure versus fragmented distributed arrays.	79
5.2	Block-cyclic distributions.	82
5.3	Concurrency and locality.	88
5.4	Layered library architecture.	92
5.5	Execution framework.	93
5.6	MatlabMPI file I/O based communication.	96
5.7	Network speedtest results.	97
5.8	MatlabMPI versus MPI.	98
5.9	Array communication overhead.	99

6.1	Types of parallelism.	114
6.2	Amdahl's Law.	118
6.3	Speedup curves.	119
6.4	Speedup ranges.	120
6.5	Weak versus strong scaling.	121
6.6	Spatial/temporal locality model.	122
6.7	Spatial/temporal scores of example programs.	123
6.8	Parallel architecture and memory hierarchy.	124
6.9	Network topologies.	125
6.10	Network performance.	127
6.11	Corner turn.	128
6.12	Two-level hierarchy.	129
6.13	Example Kuck hierarchies.	130
6.14	Example computing rack.	133
6.15	Speedup versus effort.	138
6.16	Parallel productivity versus effort.	139
7.1	Supercomputing evolution.	144
7.2	Memory hierarchy and HPC Challenge.	146
7.3	Spatial versus temporal locality.	147
7.4	Communication across the hierarchy.	148
7.5	Communication patterns of HPC Challenge.	150
7.6	HPC Challenge performance estimates.	160
7.7	Speedup versus relative code size.	166
7.8	HPC Challenge productivity.	167
8.1	Parallel Stream design.	172
8.2	Stream performance versus size.	179
9.1	RandomAccess spray communication pattern.	186
9.2	RandomAccess tree communication pattern.	189
9.3	Parallel RandomAccess relative performance.	200
10.1	Parallel FFT design.	205
10.2	Parallel FFT relative performance.	212
11.1	Parallel LU design.	222
11.2	Parallel LU critical path.	223
11.3	Parallel HPL relative performance.	231
A.1	Single processor Kuck diagram.	236
A.2	Parallel maps.	237
A.3	Parallel processor Kuck diagram.	238
A.4	Hierarchical parallel processor Kuck diagram.	241
A.5	Two tasks connected by a conduit.	244
A.6	Replicated task connected by an unreplicated task.	246

List of Tables

2.1	Mandelbrot parallel mapping.	15
2.2	Debugging steps.	19
2.3	Multicore and multinode Mandelbrot performance.	20
2.4	ZoomImage parallel mapping.	24
2.5	Multicore and multinode ZoomImage performance.	28
2.6	Multicore and multinode ParallelIO performance.	36
3.1	Beamformer parallel mapping.	43
3.2	Multicore and multinode Beamformer performance.	50
3.3	Beamformer parallel pipeline mapping.	50
4.1	Taxonomy of parallel applications.	60
4.2	Multicore and multinode Blurimage performance.	73
5.1	Distributed array design goals.	81
5.2	Distributed array parallel support functions.	84
5.3	Distributed array distributions.	85
5.4	Mandelbrot load imbalance.	87
5.5	Implementation levels.	91
5.6	MPI functions.	95
5.7	Selected user application results.	103
6.1	Parallel algorithm characteristics.	111
6.2	Software implementation efficiency estimates.	135
6.3	Software coding rate estimates.	137
6.4	Approximate relative productivity.	139
7.1	HPC Challenge parallel algorithm characteristics.	153
7.2	Memory hierarchy parameters.	157
7.3	HPC Challenge implementation results.	164
8.1	Multicore and multinode Stream triadd performance.	178
9.1	Multicore RandomAccess performance.	197
9.2	Multinode RandomAccess performance.	199

10.1	Multicore FFT performance.	210
10.2	Multinode FFT performance.	211
11.1	Multicore HPL performance.	229
11.2	Multinode HPL performance.	230

List of Algorithms

Algorithm 1.1	Add one serial	4
Algorithm 1.2	Add one parallel	5
Algorithm 2.1	Mandelbrot serial	14
Algorithm 2.2	Mandelbrot parallel	15
Algorithm 2.3	ZoomImage serial	23
Algorithm 2.4	ZoomImage parallel	24
Algorithm 2.5	Serial IO	30
Algorithm 2.6	ParallelIO	31
Algorithm 3.1	Beamformer serial	41
Algorithm 3.2	Beamformer parallel	42
Algorithm 3.3	Beamformer pipeline	51
Algorithm 4.1	Blurimage serial	57
Algorithm 4.2	Blurimage manager/worker	65
Algorithm 4.3	Blurimage message passing	66
Algorithm 4.4	Blurimage distributed array	66
Algorithm 5.1	Network speedtest	96
Algorithm 8.1	Serial Stream	171
Algorithm 8.2	Parallel Stream	173
Algorithm 9.1	Serial RandomAccess	183
Algorithm 9.2	Parallel RandomAccess spray	185
Algorithm 9.3	Parallel RandomAccess tree	188
Algorithm 10.1	Parallel FFT	204
Algorithm 11.1	Serial LU	218
Algorithm 11.2	Parallel LU	220
Algorithm A.1	Serial 2D filter	235
Algorithm A.2	Parallel 2D filter	237
Algorithm A.3	Block parallel 2D filter	239
Algorithm A.4	Hierarchical parallel 2D filter	240
Algorithm A.5	Hierarchical block parallel 2D filter	242
Algorithm A.6	Implicit pipeline 2D filter	243
Algorithm A.7	Task parallel 2D filter	245
Algorithm A.8	Fine-grained task parallel 2D filter	247
Algorithm A.9	Generic hierarchical block parallel 2D filter	248
Algorithm A.10	Hierarchical helper functions	249

Preface

MATLAB[®] is currently the dominant language of technical computing with approximately one million users worldwide, many of whom can benefit from the increased power offered by widely available multicore processors and multinode computing clusters. MATLAB is also an ideal environment for learning about parallel computing, allowing the user to focus on parallel algorithms instead of the details of the implementation. The succinctness of MATLAB allows many specific examples to be presented to illustrate parallel programming concepts.

The subject of this book is parallel programming in MATLAB and is hopefully the first of many books on this topic, as there are now a wide variety of parallel MATLAB libraries [Choy 2004] for users to choose from. I am fortunate to have been involved in the development of two of these libraries [Kepner 2004, Kepner 2003]. The most widely used of these libraries include pMatlab (developed by MIT Lincoln Laboratory), the Parallel Computing Toolbox (developed by The MathWorks, Inc.), and StarP (developed at MIT, UCSB, and Interactive Supercomputing, Inc.). All of these libraries provide direct support for parallel computing using distributed arrays and other parallel programming models. The specific examples in this book are written using the freely available pMatlab library. pMatlab has the advantage of running either standalone or on top of the aforementioned parallel MATLAB libraries. So all the examples in the book can be run in any of the parallel MATLAB environments. Fortunately, the concepts illustrated in the book are independent of the underlying implementation and valid for any particular parallel programming syntax the user may prefer to use. The software along with installation instructions can be found at the book website:

<http://www.siam.org/KepnerBook>

The expressive power of MATLAB allows us to concentrate on the techniques for creating parallel programs that run well (as opposed to the syntactic mechanics of writing parallel programs). Our primary focus is on the designing, coding, debugging, and testing techniques required to quickly produce well-performing parallel programs in a matter of hours instead of weeks or months (see Figure 1). These techniques have been developed over the years through thousands of one-on-one interactions with hundreds of parallel MATLAB users.

A general familiarity with MATLAB is assumed (see [Higham & Higham 2005] and [Moler 2004] for an introduction to MATLAB). The target audience of the book

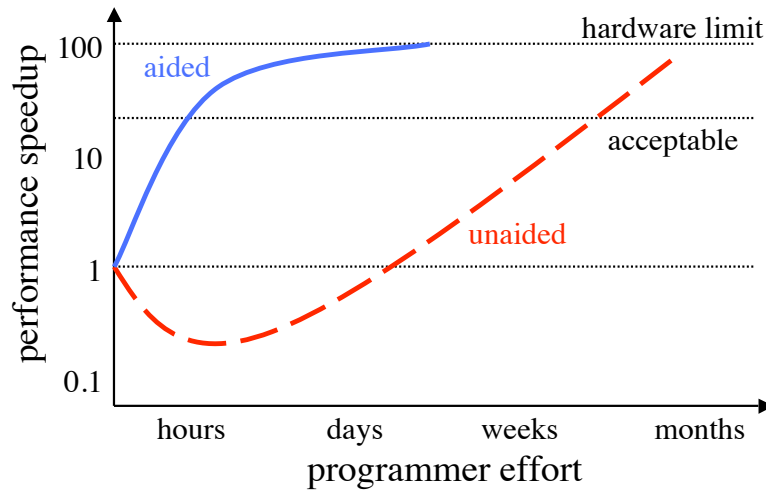


Figure 1. Performance versus effort.

Depiction of the performance achieved versus the user effort invested for a typical parallel MATLAB program. The bottom curve shows the effort required for an unaided user. The top curve shows the effort required for a user aided with expert assistance.

is anyone who needs to adapt their serial MATLAB program to a parallel environment. In addition, this book is suitable as either the primary book in a parallel computing class or as a supplementary text in a numerical computing class or a computer science algorithms class. Ideas are presented using a “hands-on” approach with numerous example programs. Wherever possible, the examples are drawn from widely known and well-documented parallel benchmark codes that have already been identified as representing many applications (although the connection to any particular application may require examining the references). For historical reasons, most of the examples are drawn from the technical computing domain, but an understanding of numerical methods is *not* required in order to use the examples. They are simply convenient and well-documented examples of different types of parallel programming.

The book is organized around two central concepts: the core programming process (i.e., design, code, debug, and test) and the core parallel programming models (i.e., distributed arrays, manager/worker, and message passing) [Lusk 2004]. The distributed array programming model will be the baseline programming model used throughout the book. Distributed arrays are easiest to understand, require the least amount of code to use, and are well-matched to the array nature of MATLAB. Distributed arrays allow very complicated communication patterns to be illustrated far more simply than with a message passing approach and perform well on both multicore and multinode parallel computers. In addition, distributed arrays naturally illustrate the core parallel design concepts of concurrency and locality. Distributed

arrays are sufficient for 90% of parallel applications, but there are instances where other programming models are optimal and these are discussed where appropriate. The ultimate goal of this book is to teach the reader to “think [distributed] matrices, not messages” [Moler 2005].

Throughout the book the approach is to first present concrete examples and then discuss in detail the more general parallel programming concepts these examples illustrate. The book aims to bring these ideas to bear on the specific challenge of writing parallel programs in MATLAB.

To accommodate the different types of readers, the book is organized into three parts. Part I (Chapters 1–3) provides a general conceptual overview of the key programming concepts illustrated by specific examples. Part II (Chapters 4–6) focuses on the analysis techniques of effective parallel programming. Part III (Chapters 7–11) consists of specific case studies. Parts I, II, and III can be treated independently and may be used as modules in a larger course. Finally, in recognition of the severe time constraints of professional users, each chapter is mostly self-contained and key terms are redefined as needed. Each chapter has a short summary and references within that chapter are listed at the end of the chapter. This arrangement allows the professional user to pick up and use any particular chapter as needed.

Chapter 1 begins by introducing some notation and the basic parallel programming interfaces used throughout the rest of the book. Chapter 2 provides a rapid introduction to creating and running simple parallel programs. Chapter 2 is meant to give readers a taste of the ease of use that parallel MATLAB provides. Chapter 3 focuses on a more complex example and highlights interacting with distributed arrays. Chapter 4 is a broad overview of the field of parallel programming, exposing the key principles that will be covered in more detail in the rest of the book. Chapter 4 uses a more sophisticated example than that used in Chapter 3, exposing the boundaries of the different parallel programming models. In addition, Chapter 4 introduces several key questions that must be dealt with in parallel programming:

Design: when to use parallel processing, concurrency versus locality, how to predict parallel performance, which parallel programming model to use: distributed arrays, client/server, or message passing.

Code: how and where to use parallel code, how to write scalable code (e.g., scalable file I/O), good coding style.

Debug: what the techniques are for going from a serial to a fully parallel execution, what kind of errors to check for at each stage along the way.

Test: how to measure performance achieved and compare with what is predicted.

Chapter 5 introduces the theory, algorithmic notation, and an “under the hood” view of distributed array programming. Chapter 6 discusses metrics for evaluating performance and coding of a parallel program. Chapter 7 is a selected survey of parallel application analysis techniques, with a particular emphasis on how the examples used in the book relate to many wider application domains. Chapters 8–11 use a set of well-studied examples drawn from the HPC Challenge benchmark

suite (see <http://www.hpcchallenge.org>) to show detailed solutions to the challenges raised by parallel design, coding, debugging, and testing.

Throughout this book we draw upon numerous classical parallel programming examples that have been well studied and characterized in the literature. It is my hope that in seeing these examples worked out the reader will come to appreciate the elegance of parallel programming in MATLAB as much as I have.

References

- [Choy 2004] Ron Choy, Parallel Matlab survey, 2004, <http://supertech.lcs.mit.edu/~cly/survey.html>
- [Higham & Higham 2005] Desmond J. Higham and Nicholas J. Higham, MATLAB Guide, Second Edition, SIAM, Philadelphia, 2005.
- [Kepner 2004] Jeremy Kepner and Stan Ahalt, MatlabMPI, Journal of Parallel and Distributed Computing, Vol. 64, No. 8, pp. 997–1005, 2004.
- [Kepner 2003] Jeremy Kepner and Nadya Travinin, Parallel Matlab: The Next Generation, Seventh Annual High Performance Embedded Computing Workshop (HPEC 2003), September 23–25, 2003, MIT Lincoln Laboratory, Lexington, MA, <http://www.ll.mit.edu/HPEC/agenda03.htm>
- [Lusk 2004] Ewing Lusk and Marc Snir, Parallel Programming Models, DARPA HPCS Productivity Team Workshop, January 13–14, 2004, Marina Del Ray, CA.
- [Moler 2004] Cleve Moler, Numerical Computing with MATLAB, SIAM, Philadelphia, 2004.
- [Moler 2005] Cleve Moler, Householder Meeting on Numerical Linear Algebra, Champion, PA, May 23–27, 2005.

Acknowledgments

There are many individuals to whom I am indebted for making parallel MATLAB and this book a reality. It is not possible to mention them all, and I would like to apologize in advance to those I may not have mentioned here due to accidental oversight on my part.

The development of parallel MATLAB has been a journey that has involved many colleagues who have made important contributions along the way. This book marks an important milestone in that journey: the broad availability and acceptance of parallel MATLAB. At a more technical level, this milestone signifies the broad availability and acceptance of the merger of the distributed array parallel programming model with high level languages such as MATLAB.

My own part in this journey has been aided by numerous individuals along the way who have directly influenced the content of this book. My introduction to numerical computing began at Pomona College in a freshman seminar taught by Prof. Wayne Steinmetz and was further encouraged by my undergraduate advisors Profs. Catalin Mitescu and Robert Chambers.

In the summer of 1990 I worked at the Aerospace Corporation, where I was introduced to high-level array languages by Dr. John Hackwell. This was followed by my first experience with parallel computation in the fall of 1990 provided by Prof. Alexandre Chorin at UC Berkeley [Kepner 1990]. Since that time I have worked to bring these two capabilities together. My first opportunity to do so occurred in 1997 when I worked with Dr. Maya Gokhale and Mr. Ron Minnich [Kepner et al. 1998]. This work was encouraged at Princeton University by my Ph.D. advisor Prof. David Spergel.

In 1998 I joined MIT Lincoln Laboratory, where Mr. Robert Bond was leading a team developing one of the first distributed array libraries in C++ [Rutledge & Kepner 1999]. In the spring of 2001, at the encouragement of Prof. Stan Ahalt, I wrote MatlabMPI in one week [Kepner 2001]. It's first user was Dr. Gil Raz. MatlabMPI, although difficult to use, developed a following at Lincoln and Ohio State (thanks to the efforts of Prof. Ashok Krishnamurthy and Dr. John Nehrbass).

In the summer of 2001 I was first exposed to the StarP work of Profs. Alan Edelman and John Gilbert and their students Dr. Perry Husbands, Mr. Ron Choy, and Dr. Viral Shah [Husbands 1999]. At this point I realized the distributed array concepts developed in C++ libraries could be brought to MATLAB with MatlabMPI underneath. From this concept Ms. Nadya Bliss created the pMatlab

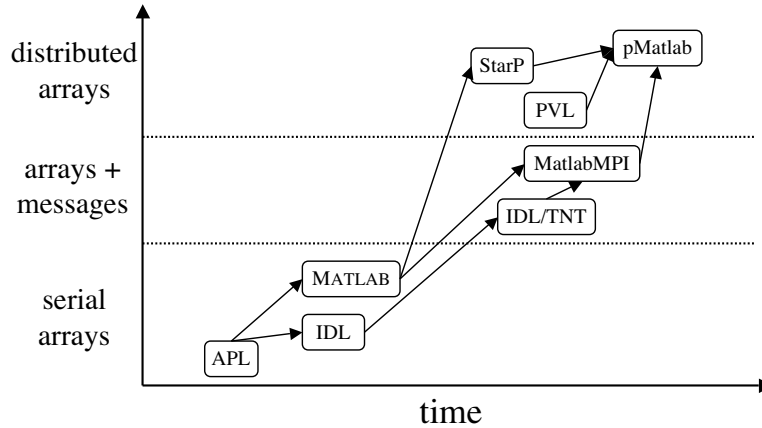


Figure 2. Development of parallel MATLAB.

A selected view of the evolution of some of the technologies that led to the development of pMatlab beginning with the first interactive array based language APL and ending with the current pMatlab library.

library [Kepner & Bliss 2003], which when coupled with the interactive LLGrid concept developed with Dr. Albert Reuther and Mr. Andrew McCabe became very popular at Lincoln [Reuther et al. 2004]. This early parallel MATLAB work was supported internally by Mr. David Martinez and externally by Mr. John Grosh.

Over the next few years I had the opportunity to interact closely with Dr. Cleve Moler and many others at The MathWorks as they brought parallel MATLAB out of the laboratory and made it into a product.

At the same time, I was also involved in the development of a suite of new parallel benchmarks designed to test new parallel architectures and parallel languages. The HPC Challenge benchmark suite was developed with Prof. Jack Dongarra, Dr. Piotr Luszczek, and Dr. Bob Lucas [Luszczek et al. 2006]. Mr. Andrew Funk, Mr. Ryan Haney, Mr. Hahn Kim, Dr. Julia Mullin, Mr. Charlie Rader, as well as many of the aforementioned individuals contributed to the parallel MATLAB implementations of these benchmarks. This later parallel MATLAB work was supported internally by Dr. Ken Senne and externally by Mr. Robert Graybil.

A visual depiction of this particular technology thread leading to pMatlab is shown in Figure 2.

In addition to those folks who have helped with the development of the technology, many additional folks have helped with the development of this book. Among these are my editor at SIAM, Ms. Elizabeth Greenspan, my copy editor at Lincoln, Ms. Dorothy Ryan, and the students in the spring 2008 MIT Applied Parallel Computing course. Finally, I would like to thank several anonymous reviewers whose comments enhanced this book.

References

- [Husbands 1999] Parry Husbands, *Interactive Supercomputing*, 1999, Ph.D. Thesis, MIT, Cambridge, MA.
- [Kepner 1990] Jeremy Kepner, *Comparison of Canonical and Micro-Canonical Sampling Methods in a 2D Ising Model*, 1990, Lawrence Berkeley Lab report LBL-30056.
- [Kepner et al. 1998] J. Kepner, M. Gokhale, R. Minnich, A. Marks & J. DeGood, *Interfacing Interpreted and Compiled Languages for Computing on a Massively Parallel Network of Workstations (MP-NOW)*, HPEC98, September 23–24, 1998, MIT Lincoln Laboratory, Lexington, MA.
- [Kepner 2001] J. Kepner, *Parallel Programming with MatlabMPI*, HPEC 2001, September 25–27, 2001, MIT Lincoln Laboratory, Lexington, MA.
- [Kepner & Bliss 2003] Jeremy Kepner & Nadya Travinin Bliss, *Parallel Matlab: The Next Generation*, 7th High Performance Embedded Computing Workshop (HPEC 2003), September 23–25, 2003, MIT Lincoln Laboratory, Lexington, MA.
- [Luszczek et al. 2006] P. Luszczek, J. Dongarra & J. Kepner, *Design and Implementation of the HPC Challenge Benchmark Suite*, CT Watch, Vol. 2, Number 4A, November 2006.
- [Reuther et al. 2004] A. Reuther, T. Currie, J. Kepner, H. Kim, A. McCabe, M. Moore & N. Travinin Bliss, *On Demand Grid Computing with Grid Matlab and pMatlab*, DOD HPCMP User's Group Conference 2004, June 8, Williamsburg, VA.
- [Rutledge & Kepner 1999] Eddie Rutledge & Jeremy Kepner, *PVL: An Object Oriented Software Library for Parallel Signal Processing*, Cluster 2001, October 9, 2001, Newport Beach, CA.