

Introduction to Parallel Computing

The constantly increasing demand for more computing power can seem impossible to keep up with. However, multicore processors capable of performing computations in parallel allow computers to tackle ever larger problems in a wide variety of applications. This book provides a comprehensive introduction to parallel computing, discussing both theoretical issues such as the fundamentals of concurrent processes, models of parallel and distributed computing, and metrics for evaluating and comparing parallel algorithms, as well as practical issues, such as methods of designing and implementing shared- and distributed-memory programs, and standards for parallel program implementation, in particular MPI and OpenMP interfaces.

Each chapter presents the basics in one place, followed by advanced topics, allowing both novices and experienced practitioners to quickly find what they need. A glossary and more than 80 exercises with selected solutions aid comprehension. The book is recommended as a text for advanced undergraduate or graduate students and as a reference for practitioners.

Zbigniew J. Czech is Professor of Computer Science at Silesian University of Technology, Gliwice, Poland. His research interests include computer programming, design and analysis of algorithms, and parallel computing, on which he has more than 45 years of experience lecturing and conducting research. He has served as a research fellow at the University of York and the University of Canterbury in the United Kingdom, and has lectured at numerous universities in Poland and elsewhere, including the University of California–Santa Barbara, Indiana University–Purdue University, and the University of Queensland.

INTRODUCTION TO PARALLEL COMPUTING

Zbigniew J. Czech
Silesian University of Technology



CAMBRIDGE UNIVERSITY PRESS

University Printing House, Cambridge CB2 8BS, United Kingdom
One Liberty Plaza, 20th Floor, New York, NY 10006, USA
477 Williamstown Road, Port Melbourne, VIC 3207, Australia
4843/24, 2nd Floor, Ansari Road, Daryaganj, Delhi - 110002, India
79 Anson Road, #06-04/06, Singapore 079906

Cambridge University Press is part of the University of Cambridge.

It furthers the University's mission by disseminating knowledge in the pursuit of education, learning, and research at the highest international levels of excellence.

www.cambridge.org
Information on this title: www.cambridge.org/9781107174399

© Zbigniew J. Czech 2016

This publication is in copyright. Subject to statutory exception and to the provisions of relevant collective licensing agreements, no reproduction of any part may take place without the written permission of Cambridge University Press.

First published 2016

Printed in the United States of America by Sheridan Books, Inc.

A catalogue record for this publication is available from the British Library.

Library of Congress Cataloging-in-Publication Data

Names: Czech, Zbigniew J.

Title: Introduction to parallel computing / Zbigniew J. Czech, Silesia University of Technology.

Description: Cambridge, United Kingdom ; New York, NY, USA : Cambridge University Press, 2016. | Includes bibliographical references and index.

Identifiers: LCCN 2016051952 | ISBN 9781107174399 (hardback : alk. paper)

Subjects: LCSH: Parallel processing (Electronic computers)

Classification: LCC QA76.58.C975 2016 | DDC 004/.35 – dc23

LC record available at <https://lcn.loc.gov/2016051952>

ISBN 978-1-107-17439-9 Hardback

Cambridge University Press has no responsibility for the persistence or accuracy of URLs for external or third-party Internet Web sites referred to in this publication and does not guarantee that any content on such Web sites is, or will remain, accurate or appropriate.

Contents

<i>List of Figures</i>	<i>page xi</i>
<i>List of Tables</i>	xix
<i>Preface</i>	xxi
1 Concurrent Processes	1
1.1 Basic Concepts	1
1.1.1 Communication between Processes	2
1.1.2 Concurrent, Parallel, and Distributed Program	3
1.2 Concurrency of Processes in Operating Systems	4
1.2.1 Threads	5
1.3 Correctness of Concurrent Programs	6
1.4 Selected Problems in Concurrent Programming	8
1.4.1 The Critical Section Problem	8
1.4.2 The Producer and Consumer Problem	11
1.4.3 The Dining Philosophers Problem	14
1.4.4 The Readers and Writers Problem	17
1.4.5 Monitors	18
1.5 Notes to the Chapter	24
1.6 Exercises	25
1.7 Bibliographic Notes	34
2 Basic Models of Parallel Computation	35
2.1 The Shared Memory Model	35
2.1.1 The RAM Model	35
2.1.2 The PRAM Model	39
2.2 The Network Model	41
2.2.1 Mesh	43
2.2.2 Mesh of trees	43
2.2.3 Cube	44
2.2.4 Cube Connected Cycles	45
2.2.5 Butterfly	46

vi	Contents	
	2.3 Comparison of Parallel Computation Models	47
	2.4 Notes to the Chapter	56
	2.5 Exercises	58
	2.6 Bibliographic Notes	61
3	Elementary Parallel Algorithms	63
	3.1 Evaluation of Parallel Algorithms	63
	3.1.1 Scalable Parallel Systems	67
	3.1.2 Isoefficiency Function	68
	3.2 Amdahl's Law	69
	3.3 Gustafson–Barsis's Law	70
	3.4 Karp–Flatt Metric	71
	3.5 Algorithms for the Shared Memory Model	72
	3.5.1 Finding the Minimum and Sum of Elements in $O(\log n)$ Time	73
	3.5.2 Brent's Theorem	77
	3.5.3 Prefix Computation	80
	3.5.4 Finding the Minimum in $O(1)$ Time	81
	3.5.5 Sorting in $O(\log n)$ Time	83
	3.5.6 Matrix–Matrix Multiplication	84
	3.5.7 Computations on Lists	87
	3.5.8 The Euler Cycle Method	88
	3.6 Algorithms for the Network Model	92
	3.6.1 Matrix–Vector Multiplication in a One-dimensional Torus Network	92
	3.6.2 Matrix–Matrix Multiplication in a Two-dimensional Torus Network	94
	3.6.3 Reduction Operation in a Cube Network	96
	3.6.4 Broadcast in a Cube Network	97
	3.6.5 Prefix Computation in a Cube Network	98
	3.7 Classes of Problems Solved in Parallel	100
	3.8 Notes to the Chapter	103
	3.8.1 Cole's Parallel Sorting Algorithm	103
	3.8.2 Bitonic Sort—Batcher's Network	112
	3.8.3 The Parallel Computation Thesis	117
	3.9 Exercises	118
	3.10 Bibliographic Notes	123
4	Designing Parallel Algorithms	125
	4.1 Steps of Designing	125
	4.2 Problem Decomposition	125
	4.2.1 Types of Decomposition	125
	4.2.2 Functional Decomposition	130
	4.2.3 Data Decomposition	131
	4.2.4 Recursive Decomposition	133
	4.2.5 Exploratory Decomposition	135
	4.2.6 Speculative Decomposition	136
	4.3 Granularity of Computation	137

4.4	Minimizing Cost of Parallel Algorithm	139
4.4.1	The Parallel Overhead	139
4.4.2	Redundant Computations	140
4.4.3	Processor Idling	140
4.4.4	References to Common Data	141
4.4.5	Overlapping Communication and Computation	142
4.5	Assigning Tasks to Processors	143
4.5.1	Load Balancing	143
4.5.2	Static Load Balancing	145
4.5.3	Dynamic Load Balancing	151
4.6	Notes to the Chapter	156
4.6.1	Foster's Method	156
4.6.2	Partitioning	158
4.6.3	Communication	158
4.6.4	Agglomeration	159
4.6.5	Mapping	160
4.7	Exercises	161
4.8	Bibliographic Notes	173
5	Architectures of Parallel Computers	175
5.1	Classification of Architectures	175
5.1.1	Multicore Processors	178
5.2	Processor Arrays	180
5.3	Multiprocessor Computers	181
5.3.1	Shared-memory Multiprocessors	182
5.3.2	Distributed-memory Multiprocessors	183
5.3.3	Distributed Shared Memory	183
5.4	Clusters	184
5.4.1	Symmetric Multiprocessor Clusters	184
5.4.2	Multicore Processor Clusters	185
5.4.3	Computer Clusters	185
5.4.4	Features and Use of Clusters	187
5.5	Computers of Unconventional Architectures	189
5.5.1	Dataflow Computers	189
5.5.2	Systolic Computers	196
5.6	Interconnection Networks	198
5.6.1	Characteristics of Interconnection Networks	198
5.6.2	Network Topologies	199
5.7	Notes to the Chapter	206
5.8	Exercises	209
5.9	Bibliographic Notes	212
6	Message-passing Programming	214
6.1	Introduction	214
6.2	The MPI Model of Computation	215
6.3	Minimum Graph Bisection	216
6.3.1	Program Compilation and Execution	218
6.3.2	Functions MPI_Init and MPI_Finalize	219

viii	Contents	
	6.3.3 Functions MPI_Comm_rank and MPI_Comm_size	220
	6.3.4 Functions MPI_Send and MPI_Recv	220
	6.3.5 Collective Communication—Functions MPI_Bcast and MPI_Reduce	224
6.4	Sorting	228
	6.4.1 Creating New Communicators—Function MPI_Comm_split	228
	6.4.2 Collecting and Spreading Data—Functions MPI_Gather and MPI_Scatter	230
6.5	Finding Prime Numbers	232
	6.5.1 Function MPI_Gatherv	234
	6.5.2 Function MPI_Wtime	235
6.6	Matrix–Vector Multiplication	236
6.7	Exercises	240
6.8	Bibliographic Notes	241
7	Shared-memory Programming	243
7.1	Introduction	243
7.2	The OpenMP Model of Computation	244
7.3	Creating a Parallel Program	246
7.4	Basic Constructs	249
	7.4.1 The Construct and Region Concepts	249
	7.4.2 Parallel Construct	250
	7.4.3 Program Compilation and Execution	252
	7.4.4 Loop Construct	252
	7.4.5 Sections Construct	255
	7.4.6 Single Construct	257
	7.4.7 Task Construct	258
	7.4.8 Taskyield Construct	259
7.5	Clauses	259
	7.5.1 The Purpose of Clauses	259
	7.5.2 Shared Clause	260
	7.5.3 Private Clause	260
	7.5.4 Firstprivate Clause	260
	7.5.5 Lastprivate Clause	261
	7.5.6 Default Clause	261
	7.5.7 Nowait Clause	262
	7.5.8 Schedule Clause	262
	7.5.9 Reduction Clause	264
	7.5.10 If Clause	266
	7.5.11 Num_threads Clause	267
	7.5.12 Copyin Clause	268
	7.5.13 Copyprivate Clause	268
7.6	Master and Synchronization Constructs	269
	7.6.1 Master Construct	269
	7.6.2 Barrier Construct	270
	7.6.3 Taskwait Construct	270
	7.6.4 Critical Construct	271

Contents ix

7.6.5	Ordered Construct	271
7.6.6	Atomic Construct	272
7.6.7	Flush Construct	273
7.7	Threadprivate Directive	274
7.8	Minimum Graph Bisection	275
7.9	Sorting	276
7.10	Finding Prime Numbers	277
7.11	Exercises	281
7.12	Bibliographic Notes	281
	<i>Solutions to Selected Exercises</i>	283
	<i>Glossary</i>	305
	<i>References</i>	323
	<i>Index</i>	343

List of Figures

1.1	Sequential processes \mathcal{P} and \mathcal{P}' that are equivalent with respect to the results of computation; t denotes the time axis.	<i>page 2</i>
1.2	Two possible scenarios of execution of concurrent processes \mathcal{P}_1 and \mathcal{P}_2 .	3
1.3	Parallel execution of operations of processes $\mathcal{P}_i, \mathcal{P}_j$, and \mathcal{P}_k .	4
1.4	Interleaving of operations of processes—one real processor “implements” three virtual processors.	5
1.5	Solving the critical section problem with a binary semaphore.	9
1.6	Solving the critical section problem for n tasks where $n > 2$.	11
1.7	Solving the producer and consumer problem with an unbounded buffer.	12
1.8	Solving the producer and consumer problem with a bounded buffer.	13
1.9	The dining philosophers, $P_0 \dots P_4$ – philosophers, $F_0 \dots F_4$ – forks.	14
1.10	Solving the problem of the dining philosophers where deadlock may occur.	15
1.11	Solving the problem of the dining philosophers with reducing the number of philosophers simultaneously present at the table (operations in lines 7–9 and 11–13 constitute the pre- and post-protocol, respectively).	16
1.12	An asymmetric solution to the dining philosophers problem.	16
1.13	Solving the problem of readers and writers with semaphores.	18
1.14	Solving the producer and consumer problem with a monitor <i>Producer_Consumer</i> .	20
1.15	A monitor for the producer and consumer problem.	21
1.16	A monitor for the dining philosophers problem.	22
1.17	A monitor for the readers and writers problem.	22
1.18	A dependency graph for tasks A, B, C , and D .	26
1.19	Synchronization in two-task barrier.	28
1.20	A structure of task communication in a butterfly barrier.	29
1.21	A structure of task communication in a dissemination barrier.	30
1.22	Solving the problem of readers and writers with semaphores without starvation of writers.	31

xii **List of Figures**

1.23	Solving the problem of resource allocation according to the SJN principle.	32
2.1	The RAM model of sequential computation.	36
2.2	(a) A RAM program to compute the value of polynomial; (b) allocation of variables to memory cells (aux. denotes an auxiliary cell).	37
2.3	(a) A pseudocode of RAM program to compute the value of a polynomial, expressed on the middle level of abstraction; each step of the program, consisting of three phases: fetching the argument, performing the operation and saving the result (some phases may be empty), is executed in unit time (Figure 2.2a); (b) an equivalent program written on the high level of abstraction.	38
2.4	The PRAM model of parallel computation.	39
2.5	The network model of parallel computation.	41
2.6	A completely-connected network.	43
2.7	A one-dimensional mesh (a); and one-dimensional torus (ring) (b).	43
2.8	A two-dimensional mesh 4×4 (a); two-dimensional torus 4×4 (b); and three-dimensional torus $3 \times 3 \times 3$ (c).	44
2.9	A two-dimensional mesh of trees 4×4 .	45
2.10	A zero-dimensional cube (a); one-dimensional (b); two-dimensional (c); three-dimensional (d); four-dimensional (e).	45
2.11	A three-dimensional cube connected cycles.	46
2.12	(a) A three-dimensional butterfly network ($k = 3$); (b) transformation of a butterfly network into a cube by merging vertices in columns and replacing multiple edges with a single edge.	47
2.13	Embeddings of a binary tree structure network (a); into two-dimensional mesh (b); and into three- and two-dimensional cubes (c–d).	51
2.14	Embedding one-dimensional mesh (a) into two-dimensional mesh (b).	52
2.15	Embedding two-dimensional mesh (a) into one-dimensional mesh (b).	53
2.16	Embedding one-dimensional torus (a) into three-dimensional cube (b).	54
2.17	A two-dimensional mesh 4×2 (a); matrix P in which sequences a_i and b_i are separated by a dot for greater clarity (b); and embedding of two-dimensional mesh 4×2 into three-dimensional cube (c).	55
2.18	A logic circuit that for a string of bits $\langle x_1, x_2, x_3 \rangle$ computes a string of bits $\langle y_1, y_2, y_3, y_4 \rangle$, where $y_1 = x_1, y_2 = x_2, y_3 = x_3$, and y_4 is the parity bit defined as $y_4 = \bar{x}_1\bar{x}_2x_3 + \bar{x}_1x_2\bar{x}_3 + x_1\bar{x}_2\bar{x}_3 + x_1x_2x_3$.	57
2.19	(a) A doubly twisted torus; (b) three-dimensional mesh, capital letters denote coordinates of mesh processors: $A = (1, 1, 1), B = (1, 2, 1), C = (1, 3, 1), D = (1, 4, 1), E = (2, 1, 1), F = (2, 2, 1), G = (2, 3, 1), H = (2, 4, 1), I = (1, 1, 2), J = (1, 2, 2), K = (1, 3, 2), L = (1, 4, 2), M = (2, 1, 2), N = (2, 2, 2), O = (2, 3, 2), P = (2, 4, 2)$.	59
2.20	Embeddings of binary trees of heights $h = 1$ (a), $h = 2$ (b), $h = 3$ (c), and $h = 4$ (d), into meshes of appropriate size.	59
2.21	A tree of 2^k vertices with roots p and q ; r and s are roots of complete binary trees of height $k - 2$.	60

2.22	The de Bruijn network of dimension $k = 3$.	61
3.1	Speedup $S(p, n)$ as a function of sequential fraction s a sequential algorithm for fixed n ; p is set to 1000.	69
3.2	Scaled speedup $\Psi(p, n)$ as a function of sequential fraction σ of computation in a parallel algorithm; p is fixed at 1000.	71
3.3	Finding the minimum element, $n = 2^r$.	74
3.4	An illustration of finding the minimum element by algorithm in Figure 3.3 ($n = 8$).	75
3.5	Finding the sum of elements for any n .	76
3.6	An illustration of finding the sum of elements by algorithm in Figure 3.5 ($n = 7$).	76
3.7	Finding the sum of elements using p processors.	76
3.8	(a) Parallel running time of algorithm given in Figure 3.7 for $c_1 = 1$, $c_2 = 10$, $n = 200$; (b) speedup (Equation (3.25)) for different values of n ; (c) efficiency (Equation (3.27)) for different values of n .	78
3.9	Simulating aggregation of $n = 8$ numbers with $p = 3$ processors.	79
3.10	A parallel prefix algorithm.	81
3.11	An illustration of parallel prefix algorithm (\otimes_i^j denotes $x_i \otimes x_{i+1} \otimes \dots \otimes x_j$ for $i < j$).	82
3.12	An algorithm to find the minimum in $O(1)$ time.	82
3.13	A parallel sorting algorithm in $O(\log n)$ time.	83
3.14	A matrix–matrix multiplication algorithm.	85
3.15	A list ranking algorithm; each element i of list L is assigned a processor P_i for $1 \leq i \leq n$.	87
3.16	An illustration of list ranking algorithm on 6-element list L ; values $w[i]$ that are set in the course of computation are highlighted in gray; (a) list L after initialization, see lines 2–8 of algorithm in Figure 3.15; (b)–(d) list L after subsequent executions of for loop in lines 9–16.	88
3.17	(a) A tree representation in the form of modified adjacency lists L ; (b) sample tree T ; (c) graphical presentation of the Euler cycle created for T ; (d) the successor function s specifying the Euler cycle stored in two-dimensional array S .	89
3.18	An algorithm to transform unrooted tree into rooted tree.	91
3.19	(a) Depth-first search of the rooted tree; (b) path \mathcal{D} , prefix sums $\sigma[v, u]$ and arc markers $z[v, u]$ (f and b mark “forward” and “backward” arcs, respectively); (c) arrays $parent[v]$ and $\delta[v]$ denoting, respectively, a parent and a number of descendants of vertex v , where $v \in V$.	92
3.20	A matrix–vector multiplication algorithm in a one-dimensional torus.	93
3.21	A matrix–matrix multiplication algorithm in a two-dimensional torus.	95
3.22	Distributing elements of matrices a and b between processors after the first (a), second (b), and third (c) iteration of for loop in lines 14–24 of algorithm in Figure 3.21.	96
3.23	An algorithm to reduction operation in a cube network.	97
3.24	A reduction operation in a three-dimensional cube; sample values to be summed up (computed in lines 2–12) (a), and the values after the first (b), second (c), and third (d) iteration in lines 13–19 of the algorithm in Figure 3.23.	98

xiv **List of Figures**

3.25	A prefix computation algorithm in a cube network.	99
3.26	Values of variables s and b in the cube composed of $p = 8$ processors before execution of the for loop in lines 6–16 (a), and then after the first (b), second (c) and third (d) iteration of this loop in the algorithm in Figure 3.25.	99
3.27	Reduction of problem A to B .	101
3.28	A logic circuit.	103
3.29	A merge sort tree; elements of a sorted sequence of length n for $n = 2^l$ are initially at leaves of the tree.	104
3.30	A sequence merging algorithm.	105
3.31	(a) Merging sequences J and K with help of a good sampler L ; (b) example for sequences $J = (2, 3, 7, 8, 10, 14, 15, 17)$, $K = (1, 4, 6, 9, 11, 12, 13, 16)$ and $L = (5, 10, 13)$.	105
3.32	Cole's parallel sorting algorithm—operations carried out in incomplete vertex w at stage t .	107
3.33	Cole's parallel sorting algorithm—operations carried out in a complete vertex w at stages $t + 1, t + 2, \dots$	107
3.34	An illustration of Cole's parallel sorting algorithm.	108
3.35	A comparator (a–b) and the comparator networks corresponding to sequential insertion sort (c) and to parallel odd-even transposition sort (d).	112
3.36	A recursive scheme of bitonic sorting network.	114
3.37	(a) Sorting 8-element bitonic sequence; (b) merging two nondecreasing 4-element sequences: $(2, 5, 6, 9)$ and $(1, 3, 5, 7)$.	114
3.38	A recursive scheme of merging network.	115
3.39	A recursive scheme of Batcher's sorting network.	116
3.40	Sorting 8-element sequence by Batcher's network.	117
3.41	Array packing.	119
3.42	A sample forest.	120
3.43	Prefix computation on a list; (a) initial form of the list; (b) the list after computation of prefixes.	120
3.44	A bubble sort algorithm (a); demonstration of odd-even transposition sort of array $a[1..5] = [16, 11, 9, 0, -2]$ (symbol \leftrightarrow denotes the comparison and possible swap of adjacent elements $a[j - 1]$ and $a[j]$) (b).	122
4.1	An illustration of Eratosthenes sieve for interval $[2..25]$.	126
4.2	A task dependency graph.	127
4.3	A task dependency graph in a pipeline.	128
4.4	A task dependency graph in data decomposition. The task S collects data concerning completion of all the tasks.	129
4.5	An enlarged filter mask with assigned weights in each field (a) and image L with the mask placed on pixel (x, y) (b).	130
4.6	Filter weights: (a) averaging filter; (b) l_p2 filter; (c) Gaussian filter.	131
4.7	A task dependency graph.	131
4.8	A decomposition of input data in the summation problem.	132
4.9	An algorithm to find both the minimum and maximum elements.	133
4.10	An example of recursive decomposition of the problem of finding both the minimum and maximum elements in array $a[1..8]$.	134

4.11	A quicksort algorithm.	134
4.12	A recursive decomposition in quicksort. As a pivot (marked in gray) the larger of two different leftmost elements in a given subarray is selected.	135
4.13	A maze (a) and decomposition of the problem of finding the way through the maze into three tasks (b).	135
4.14	A decomposition of a tree into p subtrees.	136
4.15	Parallel execution of tasks A , B_1 , B_2 and B_3 .	137
4.16	Multiplying matrix a by vector x : (a) decomposition based on input data; (b) decomposition based on intermediate data.	138
4.17	A task dependency graph taking into account granularity of computation for an image of size $n \times n = 10 \times 10$.	139
4.18	Processor load balancing: (a) imperfect; (b) perfect.	143
4.19	A decomposition of $n \times n$ matrix on p blocks with respect to rows (a) and columns (b).	146
4.20	A decomposition of $n \times n$ matrix on p blocks with respect to both dimensions, $t = p - \sqrt{p} + 1$, $w = (r - 1)s + 1$: (a) blocks of size $(n/\sqrt{p}) \times (n/\sqrt{p})$; (b) blocks of size $(n/r) \times (n/s)$, where $p = r \times s$.	147
4.21	Two decompositions of array $a[1..19]$ on five segments.	148
4.22	A partition of Poland area into 223 cells and five segments assigned to processors P_1, P_2, \dots, P_5 . Sizes of segments are equal to 42, 42, 46, 46, and 47 cells, respectively.	148
4.23	Two partitions of the vertex set of a graph into subsets assigned to processors P_1, P_2 , and P_3 ($k = 3$); the numbers of edges connecting the subgraphs are 18 (a) and 10 (b).	148
4.24	An assignment of tasks to processors P_0, P_1, P_2 , and P_3 (a) and the structure of two-dimensional cube (b).	149
4.25	Executing tasks: (a) in line with assignment depicted in Figure 4.24; (b) the time-optimal.	150
4.26	A task dependency graph considering the costs of computation and communication, $n \times n = 10 \times 10$.	150
4.27	Executing tasks in the image processing problem (for $a = 1$).	151
4.28	A centralized method of load balancing.	152
4.29	A scheme of decentralized load balancing.	153
4.30	A distributed method of load balancing.	153
4.31	The work pool method.	155
4.32	A parallel programming model. The left side describes a parallel computation represented by a directed graph whose vertices depict tasks, and arcs—channels. The right side illustrates a single task that encapsulates a sequential program, local memory and a set of I/O ports that define an interface of a task to its environment.	156
4.33	Foster's method of designing parallel algorithms.	157
4.34	A sequential algorithm to estimate the value of π .	163
4.35	Computing estimation of π ; (a) $n = 100$, $T = 85$, $\pi \approx 4T/n \approx 3.400$; (b) $n = 500$, $T = 388$, $4T/n \approx 3.104$; (c) $n = 1000$, $T = 790$, $4T/n \approx 3.160$.	163
4.36	An image of the Mandelbrot set.	164
4.37	A sequential program to compute the image of the Mandelbrot set.	165

xvi **List of Figures**

4.38	A sequential program to solve the n -body problem.	167
4.39	The worst (a) and the optimal binary search tree (b) for keys in Table 4.1. The expected costs of search in these trees are, respectively, 4.16 and 2.9.	169
4.40	A sequential program to find the optimal binary search tree applying dynamic programming.	172
4.41	Arrays e and R obtained by the program in Figure 4.40 for input data in Table 4.1.	172
5.1	Flynn's taxonomy.	176
5.2	A pipelined execution of instruction stream $i_1, i_2, \dots, i_8, \dots$	176
5.3	The processor and main memory.	177
5.4	A diagram of a dual-core processor.	179
5.5	A typical organization of a processor array (SIMD).	180
5.6	A structure of a typical shared-memory multiprocessor.	182
5.7	A structure of a typical distributed-memory multiprocessor.	183
5.8	A structure of a computer cluster consisting of SMP nodes; M—shared memory; P—processor.	185
5.9	A cluster composed of computers: connected by a single wire (a); connected into a ring (b); a front-end computer provides support for task dispatching and cluster management.	186
5.10	A data flow graph to evaluate triangle's area.	190
5.11	Vertex states before and after execution of (a) dyadic operation $op2$; (b) monadic operation $op1$; (c) copy operation.	191
5.12	A general dataflow computer structure.	191
5.13	Elementary operations related to conditional computation: (a) relation with operator rop ; (b) sink; (c) and (d) gates; (e) merge.	192
5.14	Advanced conditional operations and their implementation: (a) select; (b) switch.	193
5.15	Computing the sum of the first n terms of the harmonic series.	193
5.16	Approximating the value of $\sqrt{2}$; sqr denotes the square operation.	194
5.17	Multiplying matrices $A = \{a_{i,j}\}$ and $B = \{b_{i,j}\}$ in a systolic array; "0" denotes one cycle delay.	197
5.18	Multiplying matrices $A = \{a_{i,j}\}$ and $B = \{b_{i,j}\}$ of size $n \times n$.	197
5.19	Multiplying $n \times n$ matrix $A = \{a_{i,j}\}$ by n -element vector $X = \{x_i\}$.	197
5.20	Multiplying $n \times n$ matrix $A = \{a_{i,j}\}$ by n -element vector $X = \{x_i\}$ in a systolic array; "0" denotes one cycle delay.	198
5.21	Interconnection networks: static (a) and dynamic (b). The network nodes (processors, memory modules, etc.) and switches are marked with circles without and with shading, respectively.	199
5.22	A computer structure with a bus topology (P – processors, M – shared memory modules, C – caches, D – I/O devices).	200
5.23	A structure of a crossbar network.	200
5.24	A multistage network of stages S_1, S_2, \dots, S_k .	201
5.25	An omega network for $p = m = 8$.	201
5.26	States of the omega network switches during transmission of a message between processor P_{001} and memory module M_{101} .	202

5.27	A butterfly network. Points a, b, \dots, p on the right side of the figure are connected by links to the corresponding points on the left side of the figure.	203
5.28	Tree networks: (a) one-dimensional mesh; (b) star; (c) static binary tree; (d) dynamic binary tree.	204
5.29	A fat tree.	205
5.30	A fat tree implemented by means of switches arranged in two butterfly networks connected to each other with the opposite sides; groups of switches surrounded by a dashed line provide gradual multiplication of data routes at higher levels of the tree.	205
5.31	Sequential and parallel versions of SAXPY function.	208
5.32	An illustration of odd-even transposition sort carried out in a one-dimensional systolic array.	210
5.33	Beneš network of dimension $r = 3$ consisting of $2r + 1$ stages. Each stage contains 2^r switches.	211
5.34	A two-dimensional ($r = 2$) Beneš network that rearranges an input permutation according to one-to-one mapping $\pi(i) = (4, 8, 3, 2, 1, 7, 6, 5)$ for $i = 1, 2, \dots, 8$.	211
6.1	An example of minimum graph bisection.	217
6.2	An MPI program to find the minimum graph bisection.	218
6.3	The minimum bisection of the graph given in Figure 4.23.	224
6.4	A broadcast scheme for eight processes.	224
6.5	A minimum graph bisection program—improved version.	226
6.6	An MPI sorting program.	229
6.7	An MPI program to find the prime numbers.	232
6.8	An illustration of MPI_Gatherv.	235
6.9	Speedups S and efficiency E as a function of the number of processes for the MPI program to find prime numbers; the dashed line marks the maximum speedup equal to the number of processes.	237
6.10	An MPI program to matrix–vector multiplication.	238
6.11	The times of computation T_o and communication T_k , and the total running time $T = T_o + T_k$ of the matrix–vector multiplication program (each graph depicts 9 series of measurements).	240
7.1	The fork-join paradigm of parallel execution in OpenMP; ϕ denotes the initial or master thread.	246
7.2	Numbers of iterations executed by threads.	265
7.3	The OpenMP program to find the minimum graph bisection.	275
7.4	The sorting program—sequential version.	277
7.5	An OpenMP sorting program.	278
7.6	The OpenMP program to find prime numbers.	279
7.7	The speedup, S , and efficiency, E , as a function of the number of threads executing the OpenMP program to find the prime numbers; the dashed line represents the maximum achievable speedup equal to the number of threads.	280
S.1	The RAM programs to compute the $GCD(x, y)$ expressed on the middle (a) and low (b) level of abstraction. Before computation the numbers x and y are stored in cells M_1 and M_2 , the result is stored in cell M_1 .	285

xviii **List of Figures**

- S.2 An illustration of embedding of a two-dimensional mesh of size 4×4 into a four-dimensional cube. Capital letters denote coordinates of a mesh processors: $A = (1, 1)$, $B = (1, 2)$, $C = (1, 3)$, $D = (1, 4)$, $E = (2, 1)$, $F = (2, 2)$, $G = (2, 3)$, $H = (2, 4)$, $I = (3, 1)$, $J = (3, 2)$, $K = (3, 3)$, $L = (3, 4)$, $M = (4, 1)$, $N = (4, 2)$, $O = (4, 3)$, $P = (4, 4)$. 286
- S.3 (a) A two-dimensional torus; the wraparound connections marked by the dashed lines correspond to unused links of a cube after embedding into it a mesh of size 4×4 , see Figure S.2; (b) an alternative diagram of a four-dimensional cube. 286
- S.4 A parallel prefix algorithm of cost $O(n)$. 287
- S.5 An illustration of the parallel prefix algorithm for $n = 16$ and $r = \log n = 4$ and $p = n/r = 4$; part (a) shows array s after completion of stage 1 of the algorithm; part (b) depicts subsequent steps of computation in stage 2; part (c) presents the sequential updating of prefixes computed in stage 1. 287
- S.6 A parallel matrix–matrix multiplication algorithm of running time $O(n)$ and optimal cost $O(n^3)$. 288
- S.7 A parallel matrix–matrix multiplication algorithm of running time $O(1)$ and optimal cost $O(n^3)$. 288
- S.8 An algorithm to find the sequence of tree vertices visited in preorder. 289
- S.9 An all-to-all broadcast procedure in a two-dimensional torus. 290
- S.10 An illustration of all-to-all broadcast in a two-dimensional torus of size 3×3 ; processors that communicate with each other during the data broadcast in rows and columns are marked with thick lines; (a) after initialization in line 3; data to be broadcast: a, b, \dots, i ; (b) after broadcast of data in rows; (c) the final state after broadcast of data in columns. 290
- S.11 A matrix–matrix multiplication algorithm in a cube. 291
- S.12 An illustration of the matrix–matrix multiplication algorithm in a cube; (a) initial state; (b) after execution of operation (i) in step 1; (c) after execution of operation (ii) in step 1; (d) after execution of operation (iii) in step 1; (e) after execution of step 2; (f) after execution of step 3. 293
- S.13 Exchanging data between tasks in a virtual ring when computing the resultant forces F_i for $i = 0, 1, \dots, n - 1$; the number of tasks $p = 3$, the number of bodies $n = 9$. 295
- S.14 A parallel program to solve the n -body problem by p tasks. 296
- S.15 Roadrunner architecture; (A) 8 InfiniBand switches at the higher level of the fat tree; (B) 18 InfiniBand switches at the lower level of the fat tree; (C) 18 connected units. 299
- S.16 An MPI program to compute the image of the Mandelbrot set. 300
- S.17 An OpenMP program to compute the approximate value of π . 302
- S.18 An OpenMP program to construct the optimal binary search tree. 303
- S.19 A function to reconstruct the structure of the optimal binary search tree based on array R . 304

List of Tables

2.1	A sample list of processor instructions	<i>page 37</i>
2.2	Selected parameters of interconnection network topologies (p is the number of vertices of a network)	48
3.1	Sequential fractions f of programs to find prime numbers calculated on the basis of speedups of Figure 6.9 (p. 237) and Figure 7.7 (p. 280) (N denotes not calculated values due to limitation in a number of cores in a computer in which experiments were conducted, see footnote on p. 236)	73
3.2	An illustration of algorithm in Figure 3.12	82
3.3	Relations checked by processors and values of array w	84
4.1	Probabilities p_i and q_i for a sample sequence of $n = 6$ keys	169
5.1	Selected metrics describing dynamic interconnection networks	206
6.1	Basic datatypes defined in MPI with corresponding C types.	221
6.2	MPI's predefined reduce operations.	227
6.3	MPI's datatypes for MPI_MAXLOC and MPI_MINLOC operations.	227
6.4	Numbers of primes π_i found by processes in their subintervals, $i = 0, 1, \dots, p - 1$; p is the number of processes; $n = 10^7$	234
6.5	Results of measurements for the MPI program to find prime numbers in range $[2..n]$, p – number of processes running the computation (seq. denotes the sequential version), t – computation time in seconds, S – speedup, E – efficiency	237
7.1	Numbers of iterations assigned to threads for different types of the schedule clause	264
7.2	Valid operators and their initialization values in the reduction clause	266
7.3	Results of measurements for the OpenMP program to find the prime numbers in the interval $[2..n]$, w – number of threads used (seq. denotes the sequential version), t – running time in seconds, S – speedup, E – efficiency	280
S.1	The Euler path \mathcal{D} , the tags $z[v, u]$ of arcs (f denotes “forward” and b “backward”), the weights of arcs and prefix sums computed in the course of preorder traversal of the tree in Figure 3.19a (a); the resulting array pre [1..7] (b)	289

Preface

Solving contemporary scientific and technology problems requires the use of computers with a high speed of computation. Over the last 60 years the rate of this speed has increased 16 trillion (10^{12}) times. In the 1950s the speed of computation of a Univac 1 computer was about 1 kflop/s (flop denotes a floating-point operation), and in 2015 China's supercomputer Tianhe-2 (Milky Way-2), which contained 3 120 000 cores working in parallel, achieved the computation speed of more than 33 Pflop/s (petaflop stands for one quadrillion, 10^{15} , floating-point operations). Despite a significant increase in computational capabilities, researchers simplify models of considered problems because their numerical simulation takes too long. The demand for more and more computing power has increased and it is believed that this trend will continue in the future. There are several reasons behind this trend. Models of investigated phenomena and processes have become more complex and larger amounts of data are being processed. The requirements regarding accuracy of results also grow, which entails a higher resolution of models being developed. The fields in which, through large computing power, significant results have been achieved include: aeronautics, astrophysics, bioinformatics, chemistry, economics and trade, energy, geology and geophysics, materials science, climatology, cosmology, medicine, meteorology, nanotechnology, defense, and advanced engineering.

For example, in the U.S. National Aeronautics and Space Agency (NASA), simulation problems related to research missions conducted by space shuttles have been investigated [50]. A parallel computer SGI Altix with 10 240 processors, consisting of 20 nodes each holding 512 processors, installed in the J. Ames Center allowed for simulation of a pressure distribution around a space shuttle during its flight. A package of computational fluid dynamics used for this goal was a tool for designing geometry of the parts of a space shuttle, that is, launchers and orbital units. Another group of issues resolved in the NASA research centers concerned jet drive units. One of the tasks was to simulate a flow of liquid fuel supplied to a space shuttle main engine by a turbine pump ([31], sect. 2.4).

In order to improve aircraft performance and safety, NASA conducts research in new aircraft technologies. One of the objectives is the accurate prediction of aerodynamic and structural performance for rotorcraft designed for civil and military applications. New physics-based computational tools to predict rotorcraft flowfields by

xxii Preface

using the non-linear, three-dimensional, Navier-Stokes equations have been developed. The tools have enabled high-fidelity simulations of a UH-60 Blackhawk helicopter rotor in high-speed forward flight [68, 69]. All simulations were run on the Pleiades petascale supercomputer [51] installed in the NASA Advanced Supercomputing Division at Ames Research Center. Each run of simulation used 1536–4608 cores included in Pleiades Westmere nodes.

After the final shuttle mission in 2011, NASA began to concentrate its efforts on human space exploration in and beyond low-Earth orbit. The efforts include design of the Space Launch System (SLS), the next generation heavy lift launch vehicle with its first developmental flight planned in late 2017. Pleiades plays an important role in producing comprehensive computational fluid dynamics (CFD) simulations for design analyses of SLS vehicle. The analyses are used to predict the aerodynamic performance, load, and pressure signatures for design variations of both crew and cargo vehicles. Up to 2012, more than 3300 cases for seven different SLS designs have been simulated on Pleiades using three independent CFD flow solvers: OVERFLOW, USM3D, and NASA's Cart3D. Best practices for simulating launch vehicle ascent using those solvers were established during the Constellation Program. Simulations were run on either the Columbia or Pleiades supercomputer using 64 to 96 processors [218].

A significant impact on human development has been research in climatology. This will help to answer such fundamental questions as: Is the observed recently average Earth temperature rise a sign of ongoing global warming and a looming climate catastrophe? and also: Is the temperature growth due to natural reasons or is it the result of the greenhouse effect¹ caused by increased emissions of carbon dioxide (CO₂) and particulate matter into the atmosphere, resulting from human activities including burning coal and other fossil fuels? Answers to these questions are sought by numerical simulation of climate. For this study a program based on the Community Climate System Model (CCSM3) [80], maintained by the National Center for Atmospheric Research (NCAR), was used. It contained four components that described the state of the atmosphere, land, oceans, and ice caps. The components of a program were executed in parallel on disjoint sets of various number of processors, which exchanged data describing, among other parameters, flows of masses and energy. The CCSM3 version of a program allowed for simulation of the globe climate with 75 km resolution (distance of points in a numeric grid²) over several hundred years [395]. Simulations were capable of analyzing “what if?” cases; for example, one can predict to what extent sea levels will rise as a result of melting ice, if the amount of CO₂ emitted into the atmosphere doubles.

The results presented in [112] attained on the Cray X1E and XT4 computers and IBM p575 and p690 Cluster indicated that the Community Atmosphere Model component could simulate a state of the atmosphere over a span up to several tens of years, in a single day of computation by making use of several hundred processors. Scalability of the component to simulate a state of oceans (Parallel Ocean Program) was better. By making use of 500–1000 processors, results of

¹ This issue of how hot the world will be due to the greenhouse effect was found in *Science* journal as one of the 25 great puzzles of science in 2005 [117] (see also [340]).

² Simulation results for shorter periods of time with 10–20 km resolution were obtained using the Japanese Earth Simulator parallel computer, see for example [271].

simulation for a period of several hundred years could be obtained in a single day of calculation.

Since CCSM3's release in 2007, the work to improve and adapt its core to be implemented on petaflop computers with a number of processors from 100 to 200 thousands has been continued [112]. In 2010 NCAR released the fourth version of CCSM (CCSM4) [150], www.cesm.ucar.edu/model/cssm4.0/, and in the same year the successor to CCSM, called the Community Earth System Model version 1 (CESM1), [www.cesm.ucar.edu/models/cesm1.0.](http://www.cesm.ucar.edu/models/cesm1.0/) [254], was published as a unified code release that included CCSM4 code as a subset.

Based on the CCSM4_alpha code version, high-resolution, century-scale simulations of the Earth's climate were run on the teraflop Cray XT4 and XT5 supercomputers [98]. The resolutions adopted for experiments were 50 km for the atmosphere and land surface, and 20 km for ocean/sea-ice. The tests revealed that the development version of CCSM4 was capable to achieve 2.3 simulated years per day on the Cray XT5 utilizing 5844 computing cores. One of the enhancements in the CESM1 model relative to the CCSM4 was the inclusion of an atmospheric component that extended in altitude to the lower thermosphere. This atmospheric model, known as the Whole Atmosphere Community Climate Model, allowed simulation of the climate change from 1850 to 2005 [261]. The computations were performed on the IBM Bluefire supercomputer. Using 192 POWER6 processors, the model was capable of generating approximately 4.5 simulated years per day.

Over the last decade, several other models connected with the Earth's climate have emerged. One of them is the Non-hydrostatic Icosahedral Atmospheric Model (NICAM) developed mainly at the Japan Agency for Marine-Earth Science and Technology, the University of Tokyo, and the RIKEN research institute. Recently, NICAM has produced the global atmosphere simulation with an unprecedented horizontal resolution of 870 m. The simulation has been executed with a performance of 230 Tflop/s for 68 billion grid cells while using 20 480 nodes (163 840 cores) of the K computer [334, 223, 405].

Powerful parallel computers have played a significant role in human genome reading. In 1990, the U.S. and UK initiated the Human Genome Project (HGP), which aimed to decipher the information contained in the human DNA (Deoxyribonucleic Acid). The project was joined by China, France, Japan, and Germany. Parallel work on the description of the human genome has been undertaken by American Craig Venter and his employees in his biotechnology enterprise The Institute for Genetic Research, transformed later into Celera Genomics. The research teams initially competed against each other, but over time the race transformed into cooperation, so that studies were not duplicated, which made it possible to control faithfulness of genome reading. In May 2001, the teams published results of their research independently [203, 390] and in April 2003 they announced completion of the projects. Studies have shown that the human genome containing more than 3 billion base pairs has 30 000 genes coding proteins and RNA molecules. They represent approximately 1.5% of the total DNA. The rest is known as noncoding DNA ("junk DNA") accumulated during man's evolution [400].

To decipher mysteries of DNA Celera Genomics used a parallel computer of cluster architecture consisting of ten 4-processor SMP (Symmetric Multiprocessor) nodes, each with 4 GB of memory (Compaq ES40), and a 16-processor NUMA (Nonuniform Memory Access) computer equipped with memory of capacity of

xxiv Preface

64 GB (Compaq GS160). Execution of a DNA sequencing program required approximately 20 000 hours [390]. The computational infrastructure of the HGP consortium comprised a computing server Compaq ES40 consisting of 27 nodes, each with 108 processors, and a file server with external memory of capacity of 1 TB [203].

Description of human genome greatly accelerated development of bioinformatics, a separate discipline of research and applications. Contemporary bioinformatics represents a convergence of various areas, including modeling of biological phenomena, genomics, biotechnology. In the last few decades an enormous amount of biological data has been collected, which generated demand for novel algorithms and tools to analyze and decipher the complexity of such large data. Attaining these objectives requires high-performance computing and advanced storage capabilities. A review of applications of supercomputers in sequence analysis and genome annotation, two of the emerging and most important branches of bioinformatics, is given in [115].

Despite gathering a vast amount of knowledge about the human genome, a number of questions have not been answered yet. The role of pseudo-genes that have a form of genes but do not encode proteins is not known. What is the meaning of “discontinuity” of genes associated with presence of the so-called introns? And does, however, the noncoding DNA contain useful information? The further studies of the genome will be continued, because its complete understanding is important for many fields. For example, in medicine it will give us an opportunity to diagnose gene-based hereditary diseases and conduct gene therapy. Genetic tests already allow for determining paternity and are used in criminology. In the future, genetic knowledge will facilitate individual adjustment of drugs to the needs of patients, which will extend people’s life expectancy and improve their quality of life.

It is likewise important to know the genomes of plants. By their modifications one can breed new varieties of plants characterized by higher fertility and resistance to drought and pests. For example, the results of decoding a maize genome employing a Blue Gene/L computer [142, 143] with 1024 processors were reported in [211].

In recent decades an increasing role of computers in research has been observed. Some scientific discoveries, such as the aforementioned reading of the human genome, have been made due to substantial power of computation. The use of advanced methods and means of computing to solve complex problems is a domain of computational science. The third pillar of this field of science, in addition to theory and experiment, is computer modeling and simulation of large-scale phenomena. The importance of computational science in the context of competitiveness and prosperity of the society was a subject of the U.S. President’s Information Technology Advisory Committee report [312] (see also [291]).

Parallel computers are expensive installations with a cost of tens of millions of dollars. Therefore, for a long time only government research institutions were equipped with them. But recently they have begun to appear in industrial sectors of high income. Oil companies benefit from high-end computers by using them to manage effectively the existing oil and gas deposits [220] (access.ncsa.uiuc.edu/Stories/oil) and to search for new ones. Car manufacturers use sophisticated software to simulate a vehicle collision with obstacles [338] and to simulate flow of air around a car body. This helps to increase safety and efficiency of the proposed designs and to reduce the time of introducing new car models into sale [275]. Pharmaceutical companies use high-speed computers for designing new drugs [28, 64, 119]. The faster a drug is discovered the sooner it can be patented and brought to the market.

In the 1970s aircraft manufacturers could simulate a pressure distribution around a single wing of an aircraft. Currently such a simulation is possible for the entire structure of an aircraft [393, 273]. As a consequence, the use of expensive wind tunnels is increasingly rare. Application of high performance computing equipment enables manufacturers not only to increase profits and reduce production costs, but also to gain an advantage over competitors.

Other applications of parallel computers include numerical weather prediction [345, 254], www.science.gov/topicpages/w/wrf+weather+research.html, and forecasting natural disasters [201], such as earthquakes [104], earthquake.usgs.gov/research/, volcanic eruptions, tsunami waves [32, 262], nctr.pmel.noaa.gov, hurricanes [73], www.nhc.noaa.gov, tornados [404], and tropical cyclones [326].

As a result of ongoing efforts to boost performance of processors, complexity of integrated circuits (IC) and their degree of integration increase. Enhancement of this performance by improving the technology used so far has recently faced insurmountable obstacles. Packing more and more transistors into smaller and smaller volumes makes the width of paths inside IC components approach the size of atoms. The rise of total intensity of currents flowing between a large number of transistors causes an increase in the amount of dissipated heat. Collection of this heat from very small volumes becomes very difficult. Boosting the speed of computation by raising the clock frequency has its limits due to delays of signals transmitted along the paths in a chip. All these obstacles have given rise to construction of multicore processors consisting of a number of cores contained in a single chip of a slightly larger size compared with a conventional one. Since each core executes an independent instruction stream, computations in a multicore processor are parallel in nature. Multicore processors are becoming more and more popular. They are used in general purpose and personal computers, computing servers, embedded systems, gaming consoles, etc. Along with the popularization of multicore processors, the importance of issues related to the design and implementation of parallel programs for these types of processors—which are discussed in this book—will grow.

The speed of modern supercomputers due to parallel operation of more than 3 million interconnected processors³ is now of the order of quadrillion operations per second. As a result of technological advances this power systematically increases. Breaking the next barrier of computation speed of 1 Eflop/s (exaflop denotes one quintillion, 10^{18} , operations) will facilitate the solution of key issues in the study of health, prosperity, security, and the development of mankind. In medicine and biological sciences it will be possible to simulate molecular phenomena and to explain complex processes of protein folding [102]. Simulation of electromagnetic, thermal, and nuclear interactions between particles in a variable magnetic field assists in the development of devices in which one can conduct controlled thermonuclear fusion (www.iter.org). Production of such devices on an industrial scale would be a breakthrough in solving the energy problems of the world, as well as in spacecraft propulsion technology. Fast parallel computers with memories of large capacities give opportunity to explore huge databases in world trade and economy. This allows for better understanding of phenomena and economic trends for the benefit of people welfare. In the field of defense—following adoption by a group of states of the Comprehensive Nuclear-Test-Ban Treaty in 1996—performing time-consuming simulation has

³ See the description of Tianhe-2 supercomputer on p. 296.

xxvi **Preface**

become essential to maintain readiness of strategic weapons stockpiles. Fast computations may enhance safety through the use of advanced cryptographic and cryptanalytic methods for encryption and decryption of messages in real-time employing increasingly complex codes.

This book is devoted to the issues concerning implementation of parallel computing. In particular, it discusses the stages of analysis, design, and implementation of parallel programs. The book is recommended as a text for advanced undergraduate or graduate students. It can also be helpful for practitioners who are involved in parallel computing, such as programmers, system designers, and for all those interested in the subject. The reader should be familiar with programming in at least one of the high-level languages, for example C/C++, as well as with the basics of algorithms.

The book is a result of the experience I have gathered over the past dozen years conducting research and giving lectures on parallel computing for students at the Silesian University of Technology, Gliwice, and the University of Silesia, Sosnowiec, in Poland. A large number of valuable comments and suggestions on the first edition of the book were conveyed to me by my colleagues: Agnieszka Debudaj-Grabysz, Sebastian Deorowicz, Wojciech Mikanik, Rafał Skinderowicz, Jacek Widuch, and Wojciech Wieczorek. The doctoral students: Mirosław Błocho, Sergiusz Michalski, and Jakub Nalepa gave me useful remarks to the second edition of the book. Jakub Nalepa prepared implementations of several parallel programs that are enclosed in the exercise solutions, and Jakub Rosner helped in updating the GPUs description. I would like to express my deep appreciation to all people mentioned above for their time and commitment in helping to improve the content and form of the two Polish editions, and the present English edition of the book.

I also thank the staff of the following Polish computing centers where computations of the project were carried out: Academic Computer Center, Gdańsk (TASK); Wrocław Center for Networking and Supercomputing (WCSS); Interdisciplinary Centre for Mathematical and Computational Modelling, University of Warsaw (ICM); Academic Computer Center CYFRONET AGH, Kraków; Poznań Supercomputing and Networking Center (PCSS).

The book consists of seven chapters. In Chapter 1 the concepts of a concurrent process and thread as units executed under supervision of the operating system are introduced. The ways processes communicate with each other, the issue of proving correctness of concurrent programs and selected problems in concurrent programming are also presented. Chapter 2 is devoted to basic models of parallel computation. The details of the PRAM (Parallel Random Access Machine) and of the network models are discussed. Chapter 3 focuses on the elementary parallel algorithms and methods of their evaluation using selected metrics, such as parallel running time, speedup, cost, and efficiency. The problem of scalability of parallel algorithms is formulated, and related to it the Amdahl's law, Gustafson–Barsis's law, and Karp–Flatt metric are described. Chapter 4 is devoted to the methods of parallel algorithms design. The basic steps of design are considered, in particular decomposition of a computational problem into tasks, analysis of computation granularity, minimizing the parallel algorithm cost, and assigning tasks to processors. Chapter 5 deals with parallel computer architectures. It provides a short description of structures of processor arrays, multiprocessors with shared and distributed memory, computing clusters, and computers with unconventional architectures. An overview of interconnection networks is also given. Chapters 6 and 7 focus on principles of parallel program

design for message passing and shared memory models. These principles are illustrated with examples of programs created by employing the MPI (Message Passing Interface) library and OpenMP (Open Multiprocessing) interface. Each chapter of the book is supplemented with exercises that permit the reader better understanding and assimilation of the content presented in a chapter. Solutions to selected exercises, and a glossary of parallel computing terms appear at the end of the book.

Gliwice, Poland, May 2016

Zbigniew J. Czech