Topics in Parallel and Distributed Computing: Introducing Concurrency in Undergraduate Courses[1,2]

Chapter 1

Editors' Introduction and Roadmap

Sushil K. Prasad

Anshul Gupta

Arnold L. Rosenberg

Alan Sussman

Charles Weems

August 25, 2015

[1]How to cite this book: Prasad, Gupta, Rosenberg, Sussman, and Weems. *Topics in Parallel and Distributed Computing: Introducing Concurrency in Undergraduate Courses*, 1st Edition, Morgan Kaufmann, ISBN : 9780128038994, Pages: 360.

[2]Free preprint version of the CDER book: `http://grid.cs.gsu.edu/~tcpp/curriculum/?q=cedr_book`.

TABLE OF CONTENTS

CHAPTER 1

EDITORS' INTRODUCTION AND ROADMAP

1.1 Why This Book?

Parallel and Distributed Computing (PDC) now permeates most computing activities. It is no longer sufficient for even novice programmers to acquire only traditional sequential programming skills. In recognition of this technological evolution, a working group was organized under the aegis of the US National Science Foundation (NSF) and the IEEE Computer Society Technical Committee on Parallel Processing (TCPP) to draft guidelines for the development of curriculum in PDC for low-level (core) undergraduates courses in computational subjects such as Computer Science (CS), Computer Engineering (CE), and Computational Science. The NSF/TCPP working group proposed a set of core PDC topics, with recommendations for the level of covering each in undergraduate curricula. The resulting curricular guidelines have been organized into four tables, one for each of computer architecture, computer programming, algorithms, and cross-cutting topics[1].

The initial enthusiastic reception of these guidelines led to a commitment within the working group to continue to develop the guidelines and to foster their adoption at a broad range of academic institutions. Toward these ends, the Center for Curriculum Development and Educational Resources (CDER) was founded, with the five editors of this volume comprising the initial Board of Directors. CDER has initiated several activities toward the end of fostering PDC education.

1. A *courseware repository* has been established for pedagogical materials – sample lec-

[1]NSF-supported TCPP/CDER Curriculum: `http://www.cs.gsu.edu/~tcpp/curriculum` – Prelimnary Version released in 2010, Version 1 in 2012.

This material is based upon work partially supported by the National Science Foundation under Grants IIS 1143533, CCF 1135124, CCF 1048711 and CNS 0950432. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

tures, recommended problem sets, experiential anecdotes, evaluations, etc. This is a living repository: CDER invites the community to contribute existing and new material to it[2].

2. An *Early Adopter Program* has been established to foster the adoption and evaluation of the guidelines. This activity has fostered educational work on PDC at more than 100 educational institutions in North and South America, Europe, and Asia. The Program has thereby played a major role in establishing a worldwide community of people interested in developing and implementing PDC curricula.

3. The *EduPar workshop series* has been established. The original instantiation of EduPar was as a satellite of the International Parallel and Distributed Prossing Symposium (IPDPS). EduPar was – and continues to be – the first education-oriented workshop at a major research conference. The success of EduPar led to the development of a sibling workshop, EduHPC, at the Supercomputing Conference (SC). In August, 2015, EduPar and EduHPC will be joined by a third sibling workshop, Euro-EduPar, which will be a satellite of the International Conference on Parallel Computing (EuroPar). CDER has also sponsored panels and BOF sessions at the ACM Conference on Computer Science Education (SIGCSE).

The preceding activities have succeeded not only as individual activities, but even more exciting, by spawning a vibrant international community of educators who are committed to PDC education. It is from this community that the desirability of the *CDER Book Project*, a series of volumes to support both instructors and students of PDC, became evident. This is the first volume in the series.

Motivation and Goals of the CDER Book Project: Curricular guidelines such as those promulgated by both the CDER Center and the CS2013 ACM/IEEE Computer Science Curriculum Joint Task Force[3] are an essential first step in propelling the teaching of PDC-

[2]CDER Courseware Repository: `http://www.cs.gsu.edu/~tcpp/curriculum/?q=courseware_management`

related material into the 21st century. But such guidelines are only a first step: both instructors and students will benefit from suitable textual material to effectively translate guidelines into curriculum. Moreover, experience to this point has made it clear that the members of the PDC community have much to share with each other and with aspiring new members, in terms of creativity in forging new directions and experience in evaluating existing ones. The Book Project's goal is to engage the community to address the need for suitable textbooks and related textual material to integrate PDC topics into the lower level core courses (which we affectionately, and hopefully transparently, refer to as CS1, CS2, Systems, Data Structures and Algorithms, Logic Design, etc.). The current edited book series intends, over time, to cover all of these proposed topics. This first volume in the projected series has two parts.

Part I - For instructors: These chapters are aimed at instructors to provide background, scholarly materials, insights into pedagogical strategies, and descriptions of experience with both strategies and materials. The emphasis is on the basic concepts and references on what and how to teach PDC topics in the context of the existing topics in various core courses.

Part 2 - For students: These chapters aim to provide supplemental textual material for core courses which students can rely on for learning and exercises.

Print and Free Web Publication: While a print version through a renowned commercial publisher will foster our dissemination efforts in a professional format, the preprint versions of all the chapters will be freely available on the CDER Book Project website[4].

[3]The ACM/IEEE Computer Science Curricula 2013 document (`http://www.acm.org/education/CS2013-final-report.pdf`) explicitly refers to our proposed curriculum on page 146 as follows:

> As multi-processor computing continues to grow in the coming years, so too will the role of parallel and distributed computing in undergraduate computing curricula. In addition to the guidelines presented here, we also direct the interested reader to the document entitled "NSF/TCPP Curriculum Initiative on Parallel and Distributed Computing - Core Topics for Undergraduates", available from the website: `http://www.cs.gsu.edu/~tcpp/curriculum`.

[4]CDER Book Project - Free Preprint Version: `http://cs.gsu.edu/~tcpp/curriculum/?q=CDER_Book_Project`

Organization: This introductory chapter is organized as follows. Section 1.2 gives brief outlines of each of the nine subsequent chapters. Section 1.3 provides a roadmap for the readers to find suitable chapters and sections within these which are relevant for specific courses or PDC topics. Section 1.4 contains examples of future chapters and invitation for chapter proposals.

1.2 Chapter Introductions

1.2.1 Part I - For instructors

In Chapter 2, titled *Hands-on Parallelism with no Prerequisites and Little Time Using Scratch*, Steven Bogaerts discusses hands-on exploration of parallelism, targeting instructors of students with no or minimal prior programming experience. The hands-on activities are developed in the Scratch programming language. Scratch is in use at various higher education institutions as a first language for both majors and non-majors. One advantage of Scratch is that programming is done by dragging and connecting blocks; therefore, students can create interesting programs extremely quickly. This is a crucial feature for the prerequisite-free approach of this proposed chapter, which, as the chapter shows, is sufficient for introducing key parallelism ideas such as synchronization, race conditions, and the difference between blocking and non-blocking commands. The chapter can be used by instructors of CS0 or CS1, or computer literacy courses for non-majors, or even high-school students.

In Chapter 3, titled *Parallelism in Python for Novices*, Steven Bogaerts and Joshua Stough target instructors of courses for novice programmers, with material on parallelism and concurrency in the Python programming language. Topics covered include tasks and threads, recursion, and various features of the Python multiprocessing module. The chapter includes multiple small examples, demonstration materials, and sample exercises that can be used by instructors to teach parallel programming concepts to students just being introduced to basic programming concepts. More specifically, the chapter addresses Python multiprocessing features such as fork/join threading, message passing, sharing resources between threads,

and using locks. Examples of the utility of parallelism are drawn from application areas such as searching, sorting, simulations and image processing.

Chapter 4, titled *Modules for Introducing Threads*, by David Bunde, is meant to be used by instructors to add coverage of threads to either a data structures class, or to a systems programming class that has data structures as a prerequisite. The chapter presents two parallel examples, prime counting and computing the Mandelbrot set. The prime counting example is coded in three different frameworks, POSIX pthreads, the C++11 std::thread class, and Java's java.lang.thread class. Thus, the material can be easily adopted for use in a wide range of curricula, and can also be the basis for a comparative study of different threading environments in a more advanced class. The Mandlebrot example uses OpenMP, and includes a discussion of accomplishing the same effect using a threaded model. Both examples address the ideas of speedup, race conditions, load balancing, and making variables private to enhance speedup. In addition, the prime counting example considers critical sections and mutual exclusion, while the Mandelbrot section looks at parallel overhead and dynamic scheduling.

In Chapter 5, titled *Introducing Parallel and Distributed Computing Concepts in Digital Logic*, Ramachandran Vaidyanathan, Jerry Trahan, and Suresh Rai provide insights for how instructors of introductory digital logic classes can motivate topics from parallel and distributed computing through common logic structures. Some examples include: How circuit fan-in and fan-out can be seen as analogous to broadcast, multicast, and convergecast. Circuit timing analysis offers an opportunity to explore dependences between tasks, data hazards, and synchronization. The topology of Karnaugh maps provides an opening for introducing multidimensional torus networks. Carry lookahead circuits represent parallel prefix operations. Many other connections and analogies are noted throughout the chapter, providing instructors with opportunities to encourage parallel thinking in students, in a different context than traditional programming courses.

In Chapter 6, titled *Networks and MPI for Cluster Computing*, Ryan Grant and Stephen Olivier address parallel computing on clusters, using high performance message passing and

networking techniques. This material is mainly targeted at instructors of introductory classes in systems or data structures and algorithms, and can complement discussions of shared memory parallelism in those courses. The topics in the chapter introduce the ideas necessary to scale parallel programs to the large configurations needed for high performance supercomputing, in a form that is approachable by students in introductory classes. Through examples using the Message Passing Interface (MPI), the ubiquitous message passing API for high performance computing, the material enables students to write parallel programs that run on multiple nodes of a cluster, using both point-to-point message passing and collective operations. Several more advanced topics are also addressed that can be included as enrichment material in introductory classes, or as the starting point for deeper treatment in an upper-level class. Those topics include effective use of modern high performance networks and hybrid shared memory and message passing programming.

1.2.2 Part 2 - For students

Chapter 7, titled *Fork-Join Parallelism with a Data-Structures Focus*, by Dan Grossman, is intended for use by students in a data structures class. It introduces the basic concepts of parallelism. While it introduces the idea of concurrency control and dependences between parallel tasks, examples of these are deferred to Chapter 8 so that students may focus exclusively on methods for achieving parallel execution. Using the java.lang.Thread library to introduce basic issues of parallelism, it then moves to the java.util.concurrent library to focus on fork-join parallelism. Other models of parallelism are described to contrast with the shared-memory, fork-join model. Divide-and-conquer and map-reduce strategies are covered within the fork-join model before moving into a section on analysis of parallel algorithms in terms of make and span, with an overview of Amdahl's law and its relationship to Moore's law. The chapter concludes with additional examples of parallel prefix, pack, quicksort, and mergesort.

Chapter 8, titled *Shared-Memory Concurrency Control with a Data-Structures Focus*, is a continuation of Chapter 7 for use when a data structures class will be delving more deeply

into structures and algorithms where concurrency control is necessary to ensure correctness. It begins by motivating the need for synchronization, and how it can be achieved using locks, both at a conceptual level and in Java. It then demonstrates some of the subtle kinds of bad interleaving and data race scenarios that can occur in careless use of concurrency control. Guidelines are then introduced for managing shared memory to isolate and minimize the opportunities for concurrency control to go awry, while still retaining efficient parallelism. Deadlock is then introduced with a series of examples, followed by guidelines for establishing deadlock-free lock acquisition orders. The chapter concludes with a survey of some additional synchronization primitives, including reader-writer locks, condition variables, semaphores, barriers, and monitors.

Chapter 9, titled *Parallel Computing in Python-Based Computer Science Course*, by Thomas Cormen, is written for use by students in a Python-based data structures or algorithms class. It introduces parallel algorithms in a conceptual manner using sequential Python, with indications of which loops would be executable in parallel. The algorithms are designed using scan (prefix) and reduction operations, with consideration for whether the number of values (n) is less than or equal to the number of processors (p), versus the case where n is greater than p. Exclusive and inclusive scans are differentiated, as are copy scans. After the basic scan operations are covered, melding, permuting, and partitioning are shown and partitioning is analyzed for its parallel execution time. Segmented scans and reductions are then introduced and analyzed in preparation for developing a parallel quicksort algorithm.

Chapter 10, titled *Parallel Programming Illustrated Through Conway's Game of Life*, by Victor Eijkhout, is a student-oriented chapter that presents different means of executing Conway's Game of Life in parallel. After introducing the game, it provides an overview of array processing, short-vector processing, vector pipelining, and GPU computing. After showing how OpenMP can be used, aggregation into a coarser granularity of parallelism is described. Shared memory programming is then contrasted with distributed memory, and MPI is introduced. Blocking and non-blocking sends are then distinguished, leading to the

issue of deadlock. Dataflow as well as master-worker paradigms are shown, which brings up the topic of task scheduling. The chapter concludes with discussions of data partitioning, combining work to reduce communication, and load balancing.

1.3 How to Find a Topic or Material for a Course?

The following table lists the remaining chapters in the book, core undergraduate courses they can be used for (see list below), and their prerequisites, if any. More detailed tables in the Appendix list the topics covered in each chapter.

CORE COURSES:

CS0: Computer Literacy for Non-majors

CS1: Introduction to Computer Programming (First Course)

CS2: Second Programming Course in the Introductory Sequence

Systems: Introductory Systems/Architecture Course

DS/A: Data Structures and Algorithms

CE1: Digital Logic (First Course)

Chapter number	Short title	Primary core course	Other courses	Prerequisites
Part I				
2	Hands-on Parallelism Using Scratch	CS0	CS1	-
3	Parallelism in Python	CS0	CS1, CS2, DS	-
4	Introducing Threads	Systems	CS2	Java/C/C++
5	PDC Concepts in Digital Logic	CE1	CS2, DS/A	Math
6	Networks and MPI	Systems	CS2, DS/A	CS0/CS1
Part II				
7	Fork-Join Data-Structures	DS/A	CS2	CS1, CS2
8	Shared-Memory Concurrency	DS/A	CS2	CS1, CS2
9	Parallel Computing in Python	DS/A		CS1/CS2
10	Parallel Programming in Conway's Game of Life	CS2/DSA		CS1, Python, algorithms analysis

1.4 Invitation to Write for Volume 2

This volume has evolved organically based on contributions received in response to two calls for book chapters, in 2013 and 2014; all contributions have been rigorously reviewed. We would like to invite proposals for chapters on parallel and distributed computing topics, for either instructors or students, for a subsequent volume of this book. More specifically, we are interested in chapters on topics from the current TCPP/CDER curriculum guidelines

at `http://www.cs.gsu.edu/~tcpp/curriculum` for introductory courses that have not been addressed by the chapters in this volume. Examples of such topics include memory hierarchy issues, SIMD architectures (such as accelerators) and programming models for them, parallel versions of common algorithms and their analysis, etc. Future volumes are also planned that will address more advanced, specialized topics in parallel and distributed computing that are targeted at students in upper level classes.

Editor and Author Biograhical Sketches

1.4.1 Editors

Anshul Gupta is a Principal Research Staff Member in Mathematical Sciences department at IBM T.J. Watson Research Center. His research interests include sparse matrix computations and their applications in optimization and computational sciences, parallel algorithms, and graph/combinatorial algorithms for scientific computing. He has coauthored several journal articles and conference papers on these topics and a textbook titled "Introduction to Parallel Computing." He is the primary author of Watson Sparse Matrix Package (WSMP), one of the most robust and scalable parallel direct solvers for large sparse systems of linear equations.

Sushil K. Prasad (BTech85 IIT Kharagpur, MS86 Washington State, Pullman; PhD90 Central Florida, Orlando - all in Computer Science/Engineering) is a Professor of Computer Science at Georgia State University and Director of Distributed and Mobile Systems (DiMoS) Lab. He has carried out theoretical as well as experimental research in parallel and distributed computing, resulting in 140+ refereed publications, several patent applications, and about $3M in external research funds as principal investigator and over $6M overall (NSF/NIH/GRA/Industry).

Sushil has been honored as an ACM Distinguished Scientist in Fall 2013 for his research on parallel data structures and applications. He was the elected chair of IEEE Technical Committee on Parallel Processing for two terms (2007-11), and received its highest honors in 2012 - IEEE TCPP Outstanding Service Award. Currently, he is leading the NSF-supported IEEE-TCPP curriculum initiative on parallel and distributed computing with a vision to ensure that all computer science and engineering graduates are well-prepared in parallelism through their core courses in this era of multi- and many-cores desktops and handhelds. His current research interests are in Parallel Data Structures and Algorithms, and Computation over Geo-Spatiotemporal Datasets over

Cloud, GPU and Multicore Platforms. His homepage is `www.cs.gsu.edu/prasad`.

Arnold L. Rosenberg is a Research Professor in the Computer Science Department at Northeastern University; he also holds the rank of Distinguished University Professor Emeritus in the Computer Science Department at the University of Massachusetts Amherst. Prior to joining UMass, Rosenberg was a Professor of Computer Science at Duke University from 1981 to 1986, and a Research Staff Member at the IBM Watson Research Center from 1965 to 1981. He has held visiting positions at Yale University and the University of Toronto. He was a Lady Davis Visiting Professor at the Technion (Israel Institute of Technology) in 1994, and a Fulbright Senior Research Scholar at the University of Paris-South in 2000. Rosenberg's research focuses on developing algorithmic models and techniques to exploit the new modalities of "collaborative computing" (wherein multiple computers cooperate to solve a computational problem) that result from emerging computing technologies. Rosenberg is the author or coauthor of more than 170 technical papers on these and other topics in theoretical computer science and discrete mathematics. He is the coauthor of the research book "Graph Separators, with Applications" and the author of the textbook "The Pillars of Computation Theory: State, Encoding, Nondeterminism"; additionally, he has served as coeditor of several books. Dr. Rosenberg is a Fellow of the ACM, a Fellow of the IEEE, and a Golden Core member of the IEEE Computer Society. Rosenberg received an A.B. in mathematics at Harvard College and an A.M. and Ph.D. in applied mathematics at Harvard University. More details are available at `http://www.cs.umass.edu/~rsnbrg/`.

Alan Sussman is a Professor in the Department of Computer Science and Institute for Advanced Computer Studies at the University of Maryland. Working with students and other researchers at Maryland and other institutions he has published over 100 conference and journal papers and received several best paper awards in various topics related to software tools for high performance parallel and distributed computing, and has contributed chapters to 6 books. His research interests include peer-to-peer

distributed systems, software engineering for high performance computing, and large scale data intensive computing. He is an associate editor for the Journal of Parallel and Distributed Computing, a subject area editor for the Parallel Computing journal, and an associate editor for IEEE Transactions on Services Computing. Software tools he has built have been widely distributed and used in many computational science applications, in areas such as earth science, space science, and medical informatics. He received his Ph.D. in computer science from Carnegie Mellon University.

Charles Weems is co-director of the Architecture and Language Implementation lab at the University of Massachusetts. His current research interests include architectures for media and embedded applications, GPU computing, and high precision arithmetic. Previously he led development of two generations of a heterogeneous parallel processor for machine vision, called the Image Understanding Architecture, and co-directed initial work on the Scale compiler that was eventually used for the TRIPS architecture. He is the author of numerous articles, has served on many program committees, chaired the 1997 IEEE CAMP Workshop, the 1999 IEEE Frontiers Symposium, co-chaired IEEE IPDPS in 1999, 2000, and 2013, was general vice-chair for IPDPS from 2001 through 2005, and co-chairs the LSPP workshop. He has co-authored twenty-six introductory CS texts, and co-edited the book Associative Processing and Processors. He is a member of ACM, Senior Member of IEEE, a member of the Executive Committee of the IEEE TC on Parallel Processing, has been an editor for IEEE TPDS, Elsevier JPDC, and is an editor with Parallel Computing.

1.4.2 Authors

Steven Bogaerts holds a B.S. from Rose-Hulman Institute of Technology and an M.S. and Ph.D. from Indiana University. He is an assistant professor of computer science at DePauw University in Greencastle, IN, and previously held a faculty position at Wittenberg University. Research interests include artificial intelligence and machine learning, as well as parallelism in computer science education, with a particular interest in

including students in research. He is married with two children and a third on the way.

David Bunde is an Associate Professor of Computer Science at Knox College, a liberal arts college in Galesburg Illinois. His Ph.D. is in Computer Science from the University of Illinois at Urbana-Champaign. Prior to that, he earned a B.S. in Mathematics and Computer Science from Harvey Mudd College. He is active in parallel computing education and was selected as an Early Adopter of the NSF/IEEE-TCPP Curriculum on Parallel and Distributed Computing in Spring 2011. His other research focuses on resource management for high-performance computers, particularly the allocation of compute nodes to jobs and the assignment of job tasks to allocated nodes. He is particularly passionate about undergraduate research and includes undergraduate students in most of his work.

Thomas H. Cormen is a Professor in the Dartmouth College Department of Computer Science, where he has been since 1992. He served as the department chair from 2009 to 2015, and he directed the Dartmouth Institute for Writing and Rhetoric from 2004 to 2008. Professor Cormen received the B.S.E. degree in Electrical Engineering and Computer Science from Princeton University in 1978 and the S.M. and Ph.D. degrees in Electrical Engineering and Computer Science from the Massachusetts Institute of Technology in 1986 and 1992, respectively. An ACM Distinguished Educator, he is coauthor of the leading textbook on computer algorithms, Introduction to Algorithms, which he wrote with Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. He is also the author of Algorithms Unlocked. Professor Cormen's primary research interests are in algorithm engineering and parallel computing. He focuses on algorithms and software infrastructure to mitigate the high latency inherent in accessing the outer levels of the memory hierarchy and in interprocessor communication.

Victor Eijkhout's scientific computing career started with a degree in numerical analysis, gradually moving into computational science. He has published numerous papers, as well as the 'Templates' book for iterative linear system solving, and the textbook

'Introduction to High-performance Scientific Computing'. He currently holds a position as research scientist at the Texas Advanced Computing Center.

Ryan E. Grant is a post-doctoral appointee and the scalable interconnects team lead in the Center for Computing Research at Sandia National Laboratories in Albuquerque, New Mexico, USA. His research focuses on high performance networking for Extreme scale systems. He is an active member of the Portals Networking Interface specification team, working on next generation high performance interconnects. In addition, he serves on the MPI specifications body (MPI Forum), as well as being a developer for the Open MPI project. He graduated with a PhD in Computer Engineering in 2012 from Queens University, Canada, where he was an Alexander Graham Bell Canada Research Scholar.

Dan Grossman is an Professor in the Department of Computer Science & Engineering at the University of Washington where he has been a faculty member since 2003. He holds the J. Ray Bowen Professorship for Innovation in Engineering Education.

Dan completed his Ph.D. at Cornell University and his undergraduate studies at Rice University. His research interests lie in the area of programming languages, ranging from theory to design to implementation, with a focus on improving software quality. In recent years, he has focused on better techniques for expressing, analyzing, and executing multithreaded programs. He has collaborated actively with researchers in several other disciplines of computer science, particularly computer architecture.

Dan has served on roughly thirty conference and workshop program committees. He has served on the ACM SIGPLAN Executive Committee and the Steering Committee for the ACM / IEEE-CS 2013 Computer Science Curriculum. He currently serves on the CRA Board and the ACM Education Board.

Dan teaches a popular MOOC on undergraduate topics in programming languages and functional programming.

Stephen L. Olivier is a Senior Member of the Technical Staff in the Center for Computing Research (CCR) at Sandia National Laboratories in New Mexico. His research focuses on run time systems and programming models for high performance computing, including issues in productivity, scalability, and power. He is co-lead for the Qthreads multithreading library and a contributor to the Power API for energy-efficient measurement and control in high performance computing systems. He represents Sandia on the OpenMP Architecture Review Board and Language Committee, chairing its subcommittee on task parallelism. He received his PhD in Computer Science from the University of North Carolina at Chapel Hill, where he was a US Department of Defense National Defense Science and Engineering (NDSEG) Fellow and received the Best Student Paper Award at the ACM/IEEE Supercomputing (SC) Conference.

Joshua Stough (PhD, UNC-Chapel Hill, 2008) is a liberal arts college educator with years experience teaching CS1 and CS2/Data Structures in the Python programming language. Stough teaches computer science around three core concepts: a hybrid lecture/lab class format, the extensive use of props, toys, games, and physical exercises, and guided peer tutoring through a policy of unfettered student collaboration on most programming assignments. These methods allow students to interact with concepts from multiple perspectives, improving fluency and retention, and permits the instructor's guiding presence during the difficult analysis and synthesis stages of learning.

Suresh Rai is Michel B. Voorhies Distinguished Professor in the Division of Electrical and Computer Engineering and Emmet and Toni Stephenson College of Engineering Distinguished Professor at LSU, Baton Rouge, Louisiana. His teaching and research interests include wavelet applications in steganography and network traffic, logic testing, and parallel and distributed processing. He is a co- author of the book Wave Shaping and Digital Circuits, and tutorial texts Distributed Computing Network Reliability and Advances in Distributed System Reliability; last two published from IEEE Computer Society Press. Dr. has guest edited a special issue of IEEE Transactions on Reliability

on the topic Reliability of Parallel and Distributed Computing Networks and IJPE special issue on Dependability of Wireless Systems (2010/2011). He was an Associate Editor for IEEE Transactions on Reliability from 1990 to 2004. Currently, he is on the editorial board of International Journal of Performability Engineering.

Dr. Rai has published about 150 technical papers in the refereed journals and conference proceedings. His paper entitled Analyzing packetized voice and video traffic in an ATM multiplexer, received the best paper award at the 1998 IEEE IPCC Conference (Feb. 16-18, Tempe, Arizona). Dr. Rais research has been funded by AFOSR, NSF, and ARO.

Dr. Rai is a senior member of the IEEE.

Jerry L. Trahan received his B.S. from Louisiana State University in 1983 and his M.S. and Ph.D. from the University of Illinois at Urbana-Champaign in 1986 and 1988, respectively. Since 1988, he has been a faculty member in the Division of Electrical and Computer Engineering, Louisiana State University, where he is currently chair and Chevron Associate Professor of Electrical Engineering. His research interests include algorithms, models of parallel computation, theory of computation, and reconfigurable computing.

Ramachandran Vaidyanathan received his B-Tech and M-Tech from the Indian Institute of Technology, Kharagpur in 1983 and 1985, respectively, and a Ph.D. from Syracuse University in 1990. Since then he has been a faculty member in the Division of Electrical and Computer Engineering at Louisiana State University, Baton Rouge, where he is currently the Elaine T. and Donald C. Delaune Distinguished Associate Professor. His research interests include parallel and distributed computing, algorithms, reconfigurable systems and interconnection networks.

Appendix: Chapters and Topics

The following tables list the topics covered in each chapter. The depth of coverage of each topic is indicated by the intended outcome of teaching that topic, expressed using Bloom's taxonomy of educational objectives:

K = Know the term

C = Comprehend so as to paraphrase/illustrate

A = Apply it in some way

Chapter 2: HANDS-ON PARALLELISM

PDC Concept	Chapter Section			
	2.3.1	2.3.2	2.3.3	2.3.4
Concurrency	A	A	A	A
Why and what is PDC	A			
Time	C			
Communication	A		A	
Broadcast	A		A	
Non-Determinism		C		
Data races		A		
Synchronization		C		
Shared memory			K	A
Distributed memory				A

Chapter 3: PARALLELISM IN PYTHON

PDC Concept	Chapter Section			
	3.6	3.7	3.8	3.9
Concurrency	C			
Tasks and threads	A			
Decomposition into atomic tasks	A			
Sorting		A		
Message passing			A	
Synchronization			C	
Performance metrics			C	
Divide & conquer (parallel aspects)				A
Recursion (parallel aspects)				A

Chapter 4: MODULES FOR INTRODUCING THREADS

PDC Concept	Chapter Section		
	4.1	4.2	4.3
Shared memory: Compiler directives/pragmas			A
Shared memory: libraries		A	A
Task/thread spawning		A	A
Data parallel: parallel loops for shared memory			A
Synchronization: critical regions		A	
Concurrency defects: data races		C	C
Load balancing		C	C
Scheduling and mapping			K
Speedup		C	C

Chapter 5: PDC CONCEPTS IN DIGITAL LOGIC

PDC Concept	Chapter Section					
	5.1	5.2	5.3	5.4	5.5	5.6
Concurrency, sequential/parallel		C	C	C	C	A
Interconnects, topologies	K	C	C			A
Performance measures (latency, scalability, efficiency, trade-offs)			K	C	C	C
Recursive decomposition, divide-and-conquer	K	C		A		
Prefix computation				C	A	
(A)synchrony			K		C	A
Pipelining					C	
Data hazards			K			
Buses, shared resources		K				K
Complexity, asymptotics	K					
Dependencies, task graphs			K			
Broadcast, multicast		K				
Reduction, convergecast		K			C	

Chapter 6: NETWORKS AND MPI

PDC Concept	Chapter Section			
	6.1	6.2	6.3	6.4
Programming SPMD	C			
Performance Issues, Computation	K		K	K
Cluster	K	C	K	K
Grid/Cloud	K			
Message Passing	K	C		C
Why/What is Par/Dist Computing	K	K	K	K
Broadcast/Multicast		K		

Chapter 7: FORK-JOIN PARALLELISM

PDC Concept	Chapter Section				
	7.1	7.2	7.3	7.4	7.5
Shared memory		C	A		
Language extensions		C	A		
Libraries			A		
Task/thread spawning		C	A		
Load balancing			K	C	
Performance metrics			K		
Speedup				C	A
Amdahl's Law				A	
Asymptotics				A	
Time				A	A
BSP/CILK			A	A	A
Dependencies				C	
Task graphs				K	
Work				A	A
(Make)span				A	A
Divide & conquer (parallel aspects)			A	A	A
Recursion (parallel aspects)			A		A
Scan (parallel-prefix)					C
Reduction (map-reduce)			A	A	A
Sorting					C
Why and what is parallel/distributed computing		A			
Concurrency		K			

Chapter 8: SHARED-MEMORY CONCURRENCY CONTROL

PDC Concept	Chapter Section						
	8.1	8.2	8.3	8.4	8.5	8.6	8.7
Shared memory			A	A	A		C
Language extensions			C				C
Libraries							K
Synchronization			A	A	A		A
Critical regions			C	C	C		
Concurrency defects			A	A		A	
Memory models				C			
Non-determinism		K	A	A			

Chapter 9: PARALLEL COMPUTING IN A PYTHON-BASED COURSE

PDC Concept	Chapter Section						
	9.1	9.2	9.3	9.4	9.6	9.7	9.8
Shared memory	C						
Parallel loops for shared memory	C	A	A				
Owner computes rule	C	A					
Barriers (Series-parallel composition)	C	A	A				
Recursion		A	A				
Reduction		A				A	
Scan			A	A		A	
Sorting					A		
Parallel Time							A

Chapter 10: CONWAY'S GAME OF LIFE

PDC Concept	Chapter Section										
	1	2	3	4	5	6	7	8	9	10	
Data parallelism, SIMD	C										
SPMD	C			C			C	C			
CUDA, SIMT				C							
Vector extensions		C									
Pipelining			C								
Dataflow									C		
Task graphs									C		
Multicore						C					
Shared vs distributed memory						C	C	C		C	
Message passing								C		C	
Loop parallelism					C						
Data distribution										C	