# BIG CPU, BIG DATA

## Solving the World's Toughest Computational Problems with Parallel Computing

## Second Edition

**Alan Kaminsky**

Department of Computer Science
B. Thomas Golisano College of Computing and Information Sciences
Rochester Institute of Technology

Second edition, January 2019
Published by Barnes & Noble Press
ISBN 9781987019742



Front cover image: The U.S. Department of Energy's Sequoia supercomputer located at the Lawrence Livermore National Laboratory, in Livermore, California, USA. Photograph courtesy of Lawrence Livermore National Laboratory.

Alan Kaminsky
Professor
Department of Computer Science
B. Thomas Golisano College of Computing and Information Sciences
Rochester Institute of Technology
ark@cs.rit.edu
https://www.cs.rit.edu/~ark/

# Preface

With the book *BIG CPU, BIG DATA*, my goal is to teach you how to write parallel programs that take full advantage of the vast processing power of modern multicore computers, compute clusters, and graphics processing unit (GPU) accelerators. Everyone's computer nowadays is a parallel computer, and so every programmer needs to know how to write parallel programs.

I'm not going to teach you parallel programming using popular parallel libraries like MPI, OpenMP, and OpenCL. (If you're interested in learning those, plenty of other books are available.) Why? Two reasons:

- I prefer to program in Java. The aforementioned libraries do not, and in my belief never will, support Java.
- In my experience, teaching and learning parallel programming with the aforementioned libraries is more difficult than with Java.

Instead, I'm going to use my *Parallel Java 2 Library* (PJ2) in this book. PJ2 is free, GNU GPL licensed software available from my web site (https://www.cs.rit.edu/~ark/pj2.shtml). You can download the complete source files, compiled class files, and Javadoc documentation. PJ2 requires Java Development Kit (JDK) 1.7 or higher. Installation instructions are included in the Javadoc.

PJ2 is suitable both for teaching and learning parallel programming and for real-world parallel program development. I use PJ2 and its predecessor, the Parallel Java Library (PJ), in my cryptography research and in my research on combinatorial optimization problems. Others have used PJ to do page rank calculations, ocean ecosystem modeling, salmon population modeling and analysis, medication scheduling for patients in long term care facilities, three-dimensional complex-valued fast Fourier transforms for electronic structure analysis and X-ray crystallography, and Monte Carlo simulation of electricity and gas markets. PJ was also incorporated into the IQM open source Java image processing application.

I am happy to answer general questions about PJ2, receive bug reports, and entertain requests for additional features. Please contact me by email at ark@cs.rit.edu. I regret that I am unable to provide technical support, specific

installation instructions for your system, or advice about configuring your parallel computer hardware.

More fundamental than the language or library, however, are parallel programming concepts and patterns, such as work sharing parallel loops, parallel reduction, and communication and coordination. Whether you use OpenMP's compiler directives, MPI's message passing subroutines, or PJ2's Java classes, the concepts and patterns are the same. Only the syntax differs. Once you've learned parallel programming in Java with PJ2, you'll be able to apply the same concepts and patterns in C, Fortran, or other languages with OpenMP, MPI, or other libraries.

To study parallel programming with this book, you'll need the following prerequisite knowledge: Java programming; C programming (for GPU programs); computer organization concepts (CPU, memory, cache, and so on); operating system concepts (threads, thread synchronization).

My pedagogical style is to teach by example. Accordingly, this book consists of a series of complete parallel program examples that illustrate various aspects of parallel programming. The example programs' source code is listed on the right-hand pages, and explanatory narrative is on the left-hand pages. The example source code is also included in the PJ2 download. To write programs well, you must first learn to read programs; so please avoid the temptation to gloss over the source code listings, and carefully study both the source code and the explanations.

Also study the PJ2 Javadoc documentation for the various classes used in the example programs. The Javadoc includes comprehensive descriptions of each class and method. Space does not permit describing all the classes in detail in this book.

The book consists of these parts:

- Part I covers introductory concepts.
- Part II covers parallel programming for multicore computers.
- Part III covers parallel programming for compute clusters.
- Part IV covers parallel programming on GPUs.
- Part V covers big data parallel programming using map-reduce.
- Appendix A compares the performance of Java/PJ2 with C/OpenMP.

Instructors: There are no PowerPoint slides to go with this book. Slide shows have their place, but the classroom is not it. Nothing is guaranteed to put students to sleep faster than a PowerPoint lecture. An archive containing all the book's illustrations in PNG format is available from the book's web site (https://www.cs.rit.edu/~ark/bcbd/); please feel free to use these to develop your own instructional materials.

Alan Kaminsky
July 2016

# Preface to the Second Edition

In the three years since the first edition of *BIG CPU, BIG DATA* was published, I have used the book as a text for several parallel computing courses. I have continued to use the Parallel Java 2 Library (PJ2) to develop parallel programs for my research on cryptography and combinatorial optimization problems, and I have added new classes to PJ2 as I have encountered the need for additional capabilities. The second edition of *BIG CPU, BIG DATA* incorporates feedback from classroom use of the book and adds new material reflecting the new capabilities of PJ2. Most of the text, however, is the same as the first edition.

Major revisions to the book include the following.

I broke Part I, Preliminaries, into several chapters and added a new chapter on supercomputers and their benchmarks.

I changed the design of the minimum vertex cover programs in Chapters 15, 16, and 26 and the Hamiltonian cycle programs in Chapters 17 and 27. The programs now obtain the graph to be analyzed from a *graph specification* object.

I further changed the design of the cluster parallel Hamiltonian cycle program in Chapter 27 to use PJ2's new *cluster work queue* capability.

I added a new Chapter 28 with a cluster parallel big prime number finding program to illustrate PJ2's *on-demand tasks* capability.

The Rochester Institute of Technology Computer Science Department's `tardis` cluster parallel computer, which I used for all the running time measurements in the first edition, has been upgraded. The old `tardis` machine had ten quad-core nodes, each with 8 GB of main memory. The new `tardis` machine has ten dodeca-core (12-core) nodes, each with 64 GB of main memory. I have executed all the programs in the book on the new `tardis` machine and have updated the running time measurements. For the scaling studies, I have scaled the parallel programs up to 12 cores (on a single node) or 120 cores (on the entire cluster).

The RIT Computer Science Department's `kraken` GPU accelerated parallel computer has also been upgraded. The old `kraken` machine had four Nvidia Tesla C2075 448-core GPU cards. The new `kraken` machine has four Nvidia Tesla K40c 2880-core GPU cards. I have executed all the GPU pro-

grams in the book on the new `kraken` machine and have updated the running time measurements.

As with the first edition, there are no PowerPoint slides to go with this book. An archive containing all the book's illustrations in PNG format is available from the book's web site (https://www.cs.rit.edu/~ark/bcbd_2/).

I dedicate the second edition of *BIG CPU, BIG DATA* to the folks who run the Equal Grounds Coffee House in Rochester, New York (http://equal-grounds.com/). Much of the work of writing this edition took place at their tables, using their WiFi connection, fortified with their café mochas, iced lattes, and fruit smoothies.

Alan Kaminsky
January 2019

# Table of Contents

# Source Code Listings

**Part III. Loosely Coupled Cluster** (cont.)



**Part IV. GPU Acceleration**



**Part V. Big Data**



**Appendices**

Everything about big data. All posts. Top.Â  ClickHouse stores data in compressed form. When running queries, ClickHouse tries to do as little as possible, in order to conserve CPU resources. In many cases, all the potentially time-consuming computations are already well optimized, plus the user wrote a well thought-out query. Then all that's left to do is to perform decompression. So why does LZ4 decompression becomes a bottleneck? LZ4 seems like an extremely light algorithm: the data decompression rate is usually from 1 to 3 GB/s per processor core, depending on the data. This is much faster than the typical disk subsystem. Moreov