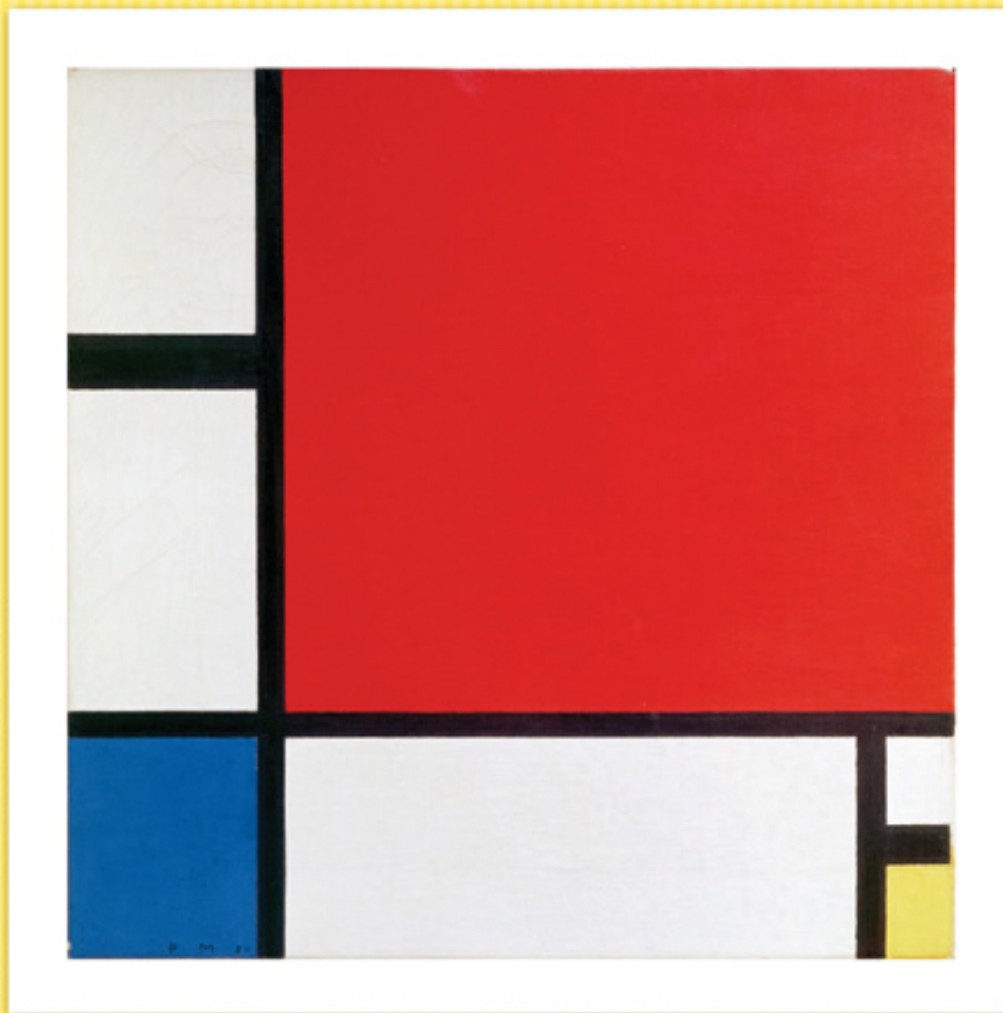


The art of  
**Parallel Programming**  
second edition



© 2006 Mondrian/Holtzman Trust c/o IcariphonInternational.com

**Bruce P. Lester**

*The Art of Parallel Programming*  
*Second Edition*  
Bruce P. Lester

Please visit the book's website: <http://www.artofparallelprogramming.com/>

Published by 1stWorld Publishing  
Fairfield, Iowa 52556

Library of Congress Control Number: 2006902678

Hardcover ISBN: 1595408398

© 2006 Bruce P. Lester

All rights reserved. No part of this book may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying or recording, or by any information storage and retrieval system, without permission in writing from the author.

This material has been written and published solely for educational purposes. The author and the publisher shall have neither liability nor responsibility to any person or entity with respect to any loss, damage or injury caused or alleged to be caused directly or indirectly by the information or programs contained in this book.

The first edition of this book was published by: Pearson Education, Inc.

Cover image:  
Piet Mondrian, *Composition with Red, Blue and Yellow*, 1930  
Oil on canvas, 46 x 46 cm

Printed in the United States of America

# PREFACE

Parallel computing is playing an increasingly important role in computer science, and offers great promise for future progress of computer technology. For this reason, most universities with extensive curricula in computer science are offering courses in this area. *The Art of Parallel Programming* is intended as a text for a first course in parallel computing, taught at the advanced undergraduate or graduate level. The only prerequisites for the text are a basic knowledge of programming and data structures.

The author believes the best way to present a comprehensive overview of the entire field of parallel computing is through a study of parallel *programming*. Programming plays a central role in computing, and brings together elements from many different areas, including algorithms, languages, and computer architecture. To write a parallel computer program, one must first formulate an efficient parallel algorithm. Then the algorithm must be expressed in a parallel programming language. To make the program perform well, the particular characteristics of the target parallel computer architecture must also be considered. Finally, one must have good techniques of debugging and performance evaluation to create a finished product.

Throughout this text, there is a continual interplay between parallel algorithms, languages, architecture, and performance evaluation. This is in contrast to many other textbooks in the field of parallel computing, which contain a more complete presentation of only one of these areas, such as parallel algorithms or parallel architectures. Many textbooks in parallel computing are simply a survey of concepts, which may soon become outdated or forgotten by the students. Whereas, *The Art of Parallel Programming* develops a new level of programming skill in the students, which is something of lasting value that can be applied in other areas, such as systems programming or distributed database implementation.

This text is organized according to the major programming techniques for parallel programs, including both shared-memory and distributed-memory parallel computers. As each new programming technique is introduced, the necessary parallel language support features are also introduced, and several

parallel algorithms are used as examples to illustrate the technique. This text definitely uses a "hands-on" approach for the students. Every chapter has extensive programming projects, in which the students write and debug their own parallel programs.

From the very beginning of their computer science education, students are taught to conceptualize a computer program as a *sequence* of steps. This *sequential* view of computing is carried throughout most of the computer science curriculum. The purpose of studying *parallel programming* is to expand this view to a more comprehensive one. To successfully write parallel programs, the student must be able to conceive of many hundreds or thousands of sequential computing activities all operating at the same time, and interacting with each other complex ways. The expected outcome of a course based on *The Art of Parallel Programming* is to culture the mind of the students to achieve this new level of comprehension of computing.

## Software Accompanying the Text

Accompanying this text is a complete software package that allows students to compile and execute their own parallel programs. This software can be downloaded free of charge from the textbook website:

<http://www.artofparallelprogramming.com>

This software package has the following major components:

- C Compiler
- Parallel Computer Simulation System

The compiler allows the C program to contain some of the standard primitives widely used for parallel programming. We refer to this parallel form of the C programming language as "C\*". The software accompanying this text allows students to write their own C\* programs and simulate the program performance on a wide variety of parallel computer architectures. All of the example programs in this text also are written in the C\* language. The use of C as the base language for this text is one of the major changes over the first edition, which was based on the language Pascal.

The availability of this software package is one of the features that makes this text unique, as the basis for a course in parallel programming. The software is designed for student use and is interactive and user-friendly. The interactive system compiles C\* programs and flags the syntax errors. The student can then specify some of the characteristics of the target parallel computer, including the

total number of processors and whether it is a shared-memory multiprocessor or a distributed-memory multicomputer. For multicomputers, the overall topology can be chosen from among some of the common topologies such as mesh, hypercube, torus, and ring.

The C\* software system will simulate the running of the student's program as it would behave on the chosen target architecture. The system has a powerful interactive debugger that allows students to set program breakpoints, step through the program, and display the values of program variables. The student can also monitor the program performance through processor utilization statistics and performance profiles. The simulation system charges overheads for process creation time, memory contention, and interprocessor communication delays. Comparison with performance of actual parallel computers has shown that the simulation system is quite realistic.

For purposes of a course in parallel programming, the use of this C\* interactive system offers many advantages over the use of a real parallel computer. With a real parallel machine, the student is required to master a whole new and complex programming environment, which takes a considerable amount of time. Many universities now have a parallel computing cluster comprised of many individual workstations networked together. If this is used for a parallel programming course, then the students only learn how to use this one architecture, which is fairly limited in comparison to the whole universe of parallel architectures. When using the C\* interactive system, the students learn to program both shared-memory and distributed-memory parallel architectures with up to 1000 processors.

With the use of this text and the accompanying software package, the instructor is given a completely integrated and self-sufficient package for teaching a course in parallel programming. The text includes instruction in the C\* language and the use of the interactive system to run C\* programs. All of the sample algorithms in the text are written in C\*, and the student exercises and programming projects can be done in C\* and tested using the interactive system.

The main philosophy behind this textbook is that to gain a real understanding of parallel programming, students must have the opportunity to write and test their own parallel programs. There are many important issues in parallel programming that do not even arise in sequential programming. Just reading about these issues and studying some abstract parallel algorithms in a book brings one level of understanding of these issues. However, when this reading is supplemented with the experience of writing parallel programs and seeing how they actually perform, then a whole new level of understanding is achieved. In courses based on this text, it is anticipated that students will spend the majority of their

time working on programming projects and exercises using the C\* interactive system.

## MPI Standard Function Library

The Message-Passing Interface (MPI) Standard contains complete specifications for a library of function calls to be used in parallel computer programs. Since MPI is a widely accepted industrial standard, the software accompanying this text includes its own MPI Function Library. One major chapter of this textbook is completely devoted to providing an overview of MPI. While studying this chapter, students will have the opportunity to write and run parallel programs using the MPI Function Library.

The C\* Compiler and Parallel Computer Simulation System that accompanies this text has three major modes of operation:

- Shared-memory Mode
- Distributed-memory Mode
- MPI Mode

The user selects the mode. In each mode, the available primitives for parallel programming are different. In shared-memory mode, parallel activities are created using *forall* statements, which is a parallel form of the C *for* loop. Access to shared data is controlled using *spinlocks*. In distributed-memory mode, parallel processes are created using *remote procedure calls*. Interprocess communication is done using *streams*. In MPI mode, the C program must use MPI function calls for all parallel programming activity. Therefore, students gain an understanding of all three of these major arenas for parallel programming.

There are several popular parallel programming textbooks that are completely based on MPI alone. In our opinion, this is not the best way to teach a first course in parallel programming. The MPI Standard has more than 125 individual functions, each with an average of 5-10 parameters. This kind of vastness and complexity may be useful as an industrial standard, but it is problematic for students in a one-semester course.

In the *Art of Parallel Programming*, the first nine chapters of the text use a small set of simple parallel programming primitives that are easily learned. This allows the students to focus full attention on developing skills of organizing, testing, and debugging parallel programs that perform well. After some level of maturity is gained with these basic skills, then MPI is introduced in Chapter 10. At this point, the students are prepared to deal with the complexity of MPI.

## Organization of the Text

The text contains twelve chapters. It should be possible to cover the entire text in a one-semester course. The instructor also has the option of omitting some chapters, and focusing more attention on the other chapters by assigning more of the exercises and programming projects. For students with some prior knowledge of parallel computer architecture or concurrent programming, the instructor may choose to skip some of the material in the first half of the text, and spend more time on the latter chapters that deal with the more sophisticated programming techniques.

Part One of the text (Chapters 1-6) is mainly devoted to programming of shared-memory multiprocessors. Part Two (Chapters 7-12) then focuses on distributed-memory multicomputers. Chapter 1 contains a brief introduction to the entire field of parallel computing, including a preview of the material covered in the text. Chapter 2 introduces the most important organizational technique for parallel programs: data parallelism. The chapter also contains coverage of two of the important sources of parallel program performance degradation, process creation overhead and sequential portions of code. Chapter 3 presents an overview of shared-memory multiprocessor architecture, with special emphasis on caching and memory system organization. The important performance issue of memory contention is introduced in this chapter.

Chapters 4 and 5 are concerned with the two major types of parallel process interaction: process communication (Chapter 4) and data sharing (Chapter 5), including the supporting parallel programming languages features. Chapter 4 discusses the syntax and semantics of process communication using streams and their application in pipeline parallelism. Chapter 5 presents an explanation of the use of spinlocks and atomic operations for process data sharing. Special attention is given to the problem of contention for shared data and how to avoid performance problems through decentralization of the locked region. Chapter 6 covers the important organizational technique for parallel programs called *synchronous iteration*, which is the parallel form of iterative numerical programs. The chapter considers the implementation and efficiency of *barrier* synchronization and convergence testing, both of which play a major role in synchronous iteration.

Chapter 7 begins the second half of the text with an overview of distributed-memory parallel computers. The main topics covered in this chapter are multicomputer topology and communications. Chapter 8 then gives an overview of the message-passing programming style and its application for creating efficient programs for multicomputers. Also, this chapter introduces the language support features required for multicomputer programming, including remote procedure calls. The main focus of Chapter 9 is on how to achieve good performance in multicomputer programs. The technique of data partitioning is presented

as the method for overcoming the potential performance degradation caused by communication delays. This chapter also contains extensive material on how evaluate parallel program performance through a combination of complexity analysis and empirical testing.

Chapter 10 provides an overview of the MPI Standard, including process creation, point-to-point communication, group communication, and creation of cartesian topologies. This chapter also discusses the application of parallel computing in image processing and Monte Carlo methods. While studying this chapter, students will write and run some parallel programs using the MPI Function Library that accompanies this text.

Chapters 11 and 12 complete the text with a coverage of *replicated workers*, a method for organizing parallel programs that produce computational tasks in a dynamic and unpredictable way. The two chapters mainly focus on the implementation of "work pools" that allow newly generated tasks to be collected and distributed to worker processes. The main issues that arise in this context are contention, load balancing, and termination detection. Chapter 11 deals with implementation of replicated workers on shared-memory machines, and Chapter 12 with distributed-memory machines.

For reference by the students throughout the course, a complete description of the C\* Compiler and Parallel Computer Simulation System is found in Part Three (Appendices A-D). Appendix A is a reference for the MPI Function Library, as described in Chapter 10. Appendix B describes the C\* language in detail. Appendix C is a User Manual for the C\* interactive system. Appendix D presents details of the C\* compiler error messages.

All the chapters of the text are intended to be covered in sequence. However, there are three chapters that may be omitted without loss of continuity: Chapters 3, 9, and 10. Also, the instructor has the option of covering Chapter 11 immediately after Chapter 6. If students have already had an advanced course in computer architecture covering aspects of parallel computer organization, then the instructor may choose to omit Chapter 3 and the first half of Chapter 7. Many operating system courses cover the topics of concurrent process communication and mutual exclusion. If students have had such a course, the instructor can move quickly through Chapter 2 and omit the first half of Chapter 5.

At the end of each chapter are exercises and programming projects for the students. Most of the exercises are rather straightforward—their purpose is to help the students review and integrate the material presented in the chapter. The programming projects require more creativity and more time to complete.

## Textbook Website

The textbook website for *The Art of Parallel Programming, Second Edition* is: <http://www.artofparallelprogramming.com>. From this website, you can download a free copy of the C\* Compiler and Parallel Computer Simulation System software. This website also contains information on how to obtain a copy of the Instructor's Manual, which contains the following:

- solutions to the exercises found at the end of each chapter
- copies of the example programs used in the textbook