THE DESIGN AND ANALYSIS OF PARALLEL ALGORITHMS

Selim G. Akl

The Design and Analysis of Parallel Algorithms

Selim G. Akl

Queen's University
Kingston, Ontario, Canada

江苏工业学院图书馆
藏书章

Prentice Hall, Englewood Cliffs, New Jersey 07632

LIBRARY OF CONGRESS
Library of Congress Cataloging-in-Publication Data

Akl, Selim G.
The design and analysis of parallel algorithms / by Selim G. Akl.
p. cm.
Bibliography: p.
Includes index.
ISBN 0-13-200056-3
✓1. Parallel programming (Computer science) 2. Algorithms.
I. Title.
QA76.6.A38 1989
004'.35--dc19 88-25019
CIP

Editorial/production supervision,
Ann Mohan
Cover design: Lundgren Graphics Ltd.
Manufacturing buyer: Mary Noonan

© 1989 by Prentice-Hall, Inc.
A Division of Simon & Schuster
Englewood Cliffs, New Jersey 07632

All rights reserved. No part of this book may be
reproduced, in any form or by any means,
without permission in writing from the publisher.

Printed in the United States of America
10 9 8 7 6 5 4 3 2 1

ISBN 0-13-200056-3

Prentice-Hall International (UK) Limited, *London*
Prentice-Hall of Australia Pty. Limited, *Sydney*
Prentice-Hall Canada Inc., *Toronto*
Prentice-Hall Hispanoamericana, S.A., *Mexico*
Prentice-Hall of India Private Limited, *New Delhi*
Prentice-Hall of Japan, Inc., *Tokyo*
Simon & Schuster Asia Pte. Ltd., *Singapore*
Editora Prentice-Hall do Brazil, Ltda., *Rio de Janeiro*

Preface

The need for ever faster computers has not ceased since the beginning of the computer era. Every new application seems to push existing computers to their limit. So far, computer manufacturers have kept up with the demand admirably well. In 1948, the electronic components used to build computers could switch from one state to another about 10,000 times every second. The switching time of this year's components is approximately 1/10,000,000,000th of a second. These figures mean that the number of operations a computer can do in one second has doubled, roughly every two years, over the past forty years. This is very impressive, but how long can it last? It is generally believed that the trend will remain until the end of this century. It may even be possible to maintain it a little longer by using optically based or even biologically based components. What happens after that?

If the current and contemplated applications of computers are any indication, our requirements in terms of computing speed will continue, at least at the same rate as in the past, well beyond the year 2000. Already, computers faster than any available today are needed to perform the enormous number of calculations involved in developing cures to mysterious diseases. They are essential to applications where the human ability to recognize complex visual and auditory patterns is to be simulated in real time. And they are indispensable if we are to realize many of humanity's dreams, ranging from reliable long-term weather forecasting to interplanetary travel and outer space exploration. It appears now that parallel processing is the way to achieve these desired computing speeds.

The overwhelming majority of computers in existence today, from the simplest to the most powerful, are conceptually very similar to one another. Their architecture and mode of operation follow, more or less, the same basic design principles formulated in the late 1940s and attributed to John von Neumann. The ingenious scenario is very simple and essentially goes as follows: A control unit fetches an instruction and its operands from a memory unit and sends them to a processing unit; there the instruction is executed and the result sent back to memory. This sequence of events is repeated for each instruction. There is only *one* unit of each kind, and only *one* instruction can be executed at a time.

With parallel processing the situation is entirely different. A parallel computer is one that consists of a collection of processing units, or *processors*, that cooperate to solve a problem by working simultaneously on different parts of that problem. The number of processors used can range from a few tens to several millions. As a result, the time required to solve the problem by a traditional uniprocessor computer is significantly reduced. This approach is attractive for a number of reasons. First, for many computational problems, the natural solution is a parallel one. Second, the cost and size of computer components have declined so sharply in recent years that parallel computers with a large number of processors have become feasible. And, third, it is possible in parallel processing to select the parallel architecture that is best suited to solve the problem or class of problems under consideration. Indeed, architects of parallel computers have the freedom to decide how many processors are to be used, how powerful these should be, what interconnection network links them to one another, whether they share a common memory, to what extent their operations are to be carried out synchronously, and a host of other issues. This wide range of choices has been reflected by the many theoretical models of parallel computation proposed as well as by the several parallel computers that were actually built.

Parallelism is sure to change the way we think about and use computers. It promises to put within our reach solutions to problems and frontiers of knowledge never dreamed of before. The rich variety of architectures will lead to the discovery of novel and more efficient solutions to both old and new problems. It is important therefore to ask: How do we solve problems on a parallel computer? The primary ingredient in solving a computational problem on any computer is the solution method, or *algorithm*. This book is about *algorithms for parallel computers*. It describes how to go about designing algorithms that exploit both the parallelism inherent in the problem and that available on the computer. It also shows how to analyze these algorithms in order to evaluate their speed and cost.

The computational problems studied in this book are grouped into three classes: (1) sorting, searching, and related problems; (2) combinatorial and numerical problems; and (3) problems arising in a number of application areas.These problems were chosen due to their fundamental nature. It is shown how a parallel algorithm is designed and analyzed to solve each problem. In some cases, several algorithms are presented that perform the same job, each on a different model of parallel computation. Examples are used as often as possible to illustrate the algorithms. Where necessary, a sequential algorithm is outlined for the problem at hand. Additional algorithms are briefly described in the Problems and Bibliographical Remarks sections. A list of references to other publications, where related problems and algorithms are treated, is provided at the end of each chapter.

The book may serve as a text for a graduate course on parallel algorithms. It was used at Queen's University for that purpose during the fall term of 1987. The class met for four hours every week over a period of twelve weeks. One of the four hours was devoted to student presentations of additional material, reference to which was found in the Bibliographical Remarks sections. The book should also be useful to computer scientists, engineers, and mathematicians who would like to learn about parallel

此为试读，需要完整PDF请访问：www.ertongbook.com

models of computation and the design and analysis of parallel algorithms. It is assumed that the reader possesses the background normally provided by an undergraduate introductory course on the design and analysis of algorithms.

The most pleasant part of writing a book is when one finally gets a chance to thank those who helped make the task an enjoyable one. Four people deserve special credit: Ms. Irene LaFleche prepared the electronic version of the manuscript with her natural cheerfulness and unmistakable talent. The diagrams are the result of Mr. Mark Attisha's expertise, enthusiasm, and skill. Dr. Bruce Chalmers offered numerous trenchant and insightful comments on an early draft. Advice and assistance on matters big and small were provided generously by Mr. Thomas Bradshaw. I also wish to acknowledge the several helpful suggestions made by the students in my CISC-867 class at Queen's. The support provided by the staff of Prentice Hall at every stage is greatly appreciated

Finally, I am indebted to my wife, Karolina, and to my two children, Sophia and Theo, who participated in this project in more ways than I can mention. Theo, in particular, spent the first year of his life examining, from a vantage point, each word as it appeared on my writing pad.

Selim G. Akl
Kingston, Ontario

Contents

此为试读，需要完整PDF请访问：www.ertongbook.com

1

Introduction

1.1 THE NEED FOR PARALLEL COMPUTERS

A battery of satellites in outer space are collecting data at the rate of 10^{10} bits per second. The data represent information on the earth's weather, pollution, agriculture, and natural resources. In order for this information to be used in a timely fashion, it needs to be processed at a speed of at least 10^{13} operations per second.

Back on earth, a team of surgeons wish to view on a special display a reconstructed three-dimensional image of a patient's body in preparation for surgery. They need to be able to rotate the image at will, obtain a cross-sectional view of an organ, observe it in living detail, and then perform a simulated surgery while watching its effect, all without touching the patient. A minimum processing speed of 10^{15} operations per second would make this approach worthwhile.

The preceding two examples are representative of applications where tremendously fast computers are needed to process vast amounts of data or to perform a large number of calculations quickly (or at least within a reasonable length of time). Other such applications include aircraft testing, the development of new drugs, oil exploration, modeling fusion reactors, economic planning, cryptanalysis, managing large databases, astronomy, biomedical analysis, real-time speech recognition, robotics, and the solution of large systems of partial differential equations arising from numerical simulations in disciplines as diverse as seismology, aerodynamics, and atomic, nuclear, and plasma physics. No computer exists today that can deliver the processing speeds required by these applications. Even the so-called *supercomputers* peak at a few billion operations per second.

Over the past forty years dramatic increases in computing speed were achieved. Most of these were largely due to the use of inherently faster electronic components by computer manufacturers. As we went from relays to vacuum tubes to transistors and from small to medium to large and then to very large scale integration, we witnessed—often in amazement—the growth in size and range of the computational problems that we could solve.

Unfortunately, it is evident that this trend will soon come to an end. The limiting factor is a simple law of physics that gives the speed of light in vacuum. This speed is approximately equal to 3×10^8 meters per second. Now, assume that an electronic device can perform 10^{12} operations per second. Then it takes longer for a signal to travel between two such devices one-half of a millimeter apart than it takes for either of them to process it. In other words, all the gains in speed obtained by building superfast electronic components are lost while one component is waiting to receive some input from another one. Why then (one is compelled to ask) not put the two communicating components even closer together? Again, physics tells us that the reduction of distance between electronic devices reaches a point beyond which they begin to interact, thus reducing not only their speed but also their reliability.

It appears that the only way around this problem is to use *parallelism*. The idea here is that if several operations are performed simultaneously, then the time taken by a computation can be significantly reduced. This is a fairly intuitive notion, and one to which we are accustomed in any organized society. We know that several people of comparable skills can usually finish a job in a fraction of the time taken by one individual. From mail distribution to harvesting and from office to factory work, our everyday life offers numerous examples of parallelism through task sharing.

Even in the field of computing, the idea of parallelism is not entirely new and has taken many forms. Since the early days of information processing, people realized that it is greatly advantageous to have the various components of a computer do different things at the same time. Typically, while the central processing unit is doing calculations, input can be read from a magnetic tape and output produced on a line printer. In more advanced machines, there are several simple processors each specializing in a given computational task, such as operations on floating-point numbers, for example. Some of today's most powerful computers contain two or more processing units that share among themselves the jobs submitted for processing.

In each of the examples just mentioned, parallelism is exploited profitably, but nowhere near its promised power. Strictly speaking, none of the machines discussed is truly a parallel computer. In the modern paradigm that we are about to describe, however, the idea of parallel computing can realize its full potential. Here, our computational tool is a *parallel computer*, that is, a computer with many processing units, or *processors*. Given a problem to be solved, it is broken into a number of subproblems. All of these subproblems are now solved simultaneously, each on a different processor. The results are then combined to produce an answer to the original problem. This is a radical departure from the model of computation adopted for the past forty years in building computers—namely, the sequential uniprocessor machine.

Only during the last ten years has parallelism become truly attractive and a viable approach to the attainment of very high computational speeds. The declining cost of computer hardware has made it possible to assemble parallel machines with millions of processors. Inspired by the challenge, computer scientists began to study parallel computers both in theory and in practice. Empirical evidence provided by homegrown prototypes often came to support a large body of theoretical studies. And very recently, a number of commercial parallel computers have made their appearance on the market.

With the availability of the hardware, the most pressing question in parallel computing today is: How to program parallel computers to solve problems efficiently and in a practical and economically feasible way? As is the case in the sequential world, parallel computing requires algorithms, programming languages and compilers, as well as operating systems in order to actually perform a computation on the parallel hardware. All these ingredients of parallel computing are currently receiving a good deal of well-deserved attention from researchers.

This book is about one (and perhaps the most fundamental) aspect of parallelism, namely, *parallel algorithms*. A parallel algorithm is a solution method for a given problem destined to be performed on a parallel computer. In order to properly design such algorithms, one needs to have a clear understanding of the *model of computation* underlying the parallel computer.

1.2 MODELS OF COMPUTATION

Any computer, whether sequential or parallel, operates by executing instructions on data. A stream of instructions (the algorithm) tells the computer what to do at each step. A stream of data (the input to the algorithm) is affected by these instructions. Depending on whether there is one or several of these streams, we can distinguish among four classes of computers:

1. Single Instruction stream, Single Data stream (SISD)
2. Multiple Instruction stream, Single Data stream (MISD)
3. Single Instruction stream, Multiple Data stream (SIMD)
4. Multiple Instruction stream, Multiple Data stream (MIMD).

We now examine each of these classes in some detail. In the discussion that follows we shall not be concerned with input, output, or peripheral units that are available on every computer.

1.2.1 SISD Computers

A computer in this class consists of a single processing unit receiving a single stream of instructions that operate on a single stream of data, as shown in Fig. 1.1. At each step during the computation the control unit emits one instruction that operates on a datum obtained from the memory unit. Such an instruction may tell the processor, for

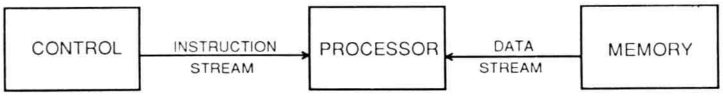

Figure 1.1 SISD computer.

example, to perform some arithmetic or logic operation on the datum and then put it back in memory.

The overwhelming majority of computers today adhere to this model invented by John von Neumann and his collaborators in the late 1940s. An algorithm for a computer in this class is said to be *sequential* (or *serial*).

Example 1.1

In order to compute the sum of n numbers, the processor needs to gain access to the memory n consecutive times and each time receive one number. There are also $n - 1$ additions involved that are executed in sequence. Therefore, this computation requires on the order of n operations in total. □

This example shows that algorithms for SISD computers do not contain any parallelism. The reason is obvious, there is only one processor! In order to obtain from a computer the kind of parallel operation defined earlier, it will need to have several processors. This is provided by the next three classes of computers, the classes of interest in this book. In each of these classes, a computer possesses N processors, where $N > 1$.

1.2.2 MISD Computers

Here, N processors each with its own control unit share a common memory unit where data reside, as shown in Fig. 1.2. There are N streams of instructions and one stream of data. At each step, one datum received from memory is operated upon by all the processors simultaneously, each according to the instruction it receives from its control. Thus, parallelism is achieved by letting the processors do different things at the same time on the same datum. This class of computers lends itself naturally to those computations requiring an input to be subjected to several operations, each receiving the input in its original form. Two such computations are now illustrated.

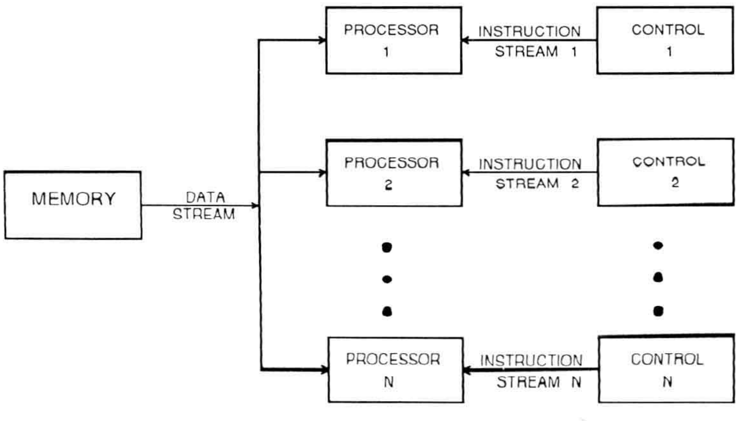

Figure 1.2 MISD computer.

Example 1.2

It is required to determine whether a given positive integer z has no divisors except 1 and itself. The obvious solution to this problem is to try all possible divisors of z: If none of these succeeds in dividing z, then z is said to be *prime*; otherwise z is said to be *composite*.

We can implement this solution as a parallel algorithm on an MISD computer. The idea is to split the job of testing potential divisors among processors. Assume that there are as many processors on the parallel computer as there are potential divisors of z. All processors take z as input, then each tries to divide it by its associated potential divisor and issues an appropriate output based on the result. Thus it is possible to determine in *one step* whether z is prime. More realistically, if there are fewer processors than potential divisors, then each processor can be given the job of testing a different subset of these divisors. In either case, a substantial speedup is obtained over a purely sequential implementation.

Although more efficient solutions to the problem of *primality testing* exist, we have chosen the simple one as it illustrates the point without the need for much mathematical sophistication. □

Example 1.3

In many applications, we often need to determine to which of a number of classes does a given object belong. The object may be a mathematical one, where it is required to associate a number with one of several sets, each with its own properties. Or it may be a physical one: A robot scanning the deep-sea bed "sees" different objects that it has to recognize in order to distinguish among fish, rocks, algae, and so on. Typically, membership of the object is determined by subjecting it to a number of different tests.

The classification process can be done very quickly on an MISD computer with as many processors as there are classes. Each processor is associated with a class and can recognize members of that class through a computational test. Given an object to be classified, it is sent simultaneously to all processors where it is tested in parallel. The object belongs to the class associated with that processor that reports the success of its test. (Of course, it may be that the object does not belong to any of the classes tested for, in which case all processors report failure.) As in example 1.2, when fewer processors than classes are available, several tests are performed by each processor; here, however, in reporting success, a processor must also provide the class to which the object belongs. □

The preceding examples show that the class of MISD computers could be extremely useful in many applications. It is also apparent that the kind of computations that can be carried out efficiently on these computers are of a rather specialized nature. For most applications, MISD computers would be rather awkward to use. Parallel computers that are more flexible, and hence suitable for a wide range of problems, are described in the next two sections.

1.2.3 SIMD Computers

In this class, a parallel computer consists of N identical processors, as shown in Fig. 1.3.

Each of the N processors possesses its own local memory where it can store both

此为试读,需要完整PDF请访问: www.ertongbook.com