DESIGNING EFFICIENT ALGORITHMS FOR PARALLEL COMPUTERS

Michael J. Quinn

DESIGNING EFFICIENT ALGORITHMS FOR PARALLEL COMPUTERS

Michael J. Quinn
University of New Hampshire

McGraw-Hill Book Company

New York St. Louis San Francisco Auckland Bogotá Hamburg Johannesburg London
Madrid Mexico Milan Montreal New Delhi Panama Paris São Paulo
Singapore Sydney Tokyo Toronto

This book was set in Times Roman by Aldine Press.
The editors were Kaye Pace and Larry Goldberg;
the cover was designed by Albert Cetta;
the cover photograph was taken by A. Diakopoulos/Stock, Boston;
the production supervisor was Diane Renda. The drawings were done by ANCO/Boston.
R. R. Donnelley & Sons Company was printer and binder.

Figure Credits

Figures 2-1 and 4-18: Aho, Hopcroft, and Ullman, *The Design and Analysis of Computer Algorithms*, ©1974, Addison-Wesley, Reading, Massachusetts, page 5, Figure 1.3, and page 264, Figure 7.5. Reprinted with permission.

Figures 2-6 and 2-8: Reprinted from *Computational Aspects of VLSI* by Jeffrey D. Ullman, ©1984, with the permission of the publisher, Computer Science Press, Inc., 1803 Research Blvd., Rockville, MD 20850 USA.

Figures 2-11, 2-12, 2-13, and 2-14: Reprinted from *Parallel Computers* by R. W. Hockney and C. R. Jesshope, ©1981, with the permission of the publisher, Adam Hilger, Bristol, England.

Figure 4-9: Robert Sedgewick, *Algorithms*, ©1983, Addison-Wesley, Reading, Massachusetts, page 465 (figure). Reprinted with permission.

Figure 4-12: Donald E. Knuth, *The Art of Computer Programming, Volume 3, Sorting and Searching*, ©1973, Addison-Wesley, Reading, Massachusetts, page 237, Figure 56. Reprinted with permission.

DESIGNING EFFICIENT ALGORITHMS FOR PARALLEL COMPUTERS

1 2 3 4 5 6 7 8 9 0 DOCDOC 8 9 4 3 2 1 0 9 8 7

ISBN 0-07-051071-7

Library of Congress Cataloging-in-Publication Data

Quinn, Michael J. (Michael Jay)
Designing efficient algorithms for parallel computers.

(Supercomputing & artificial intelligence)
Bibliography; p.
Includes index.
1. Parallel processing (Electronic computers)
2. Algorithms. I. Title. II. Series.
QA76.5.Q56 1987 004'.35 86-27230
ISBN 0-07-051071-7

PREFACE

A few years ago parallel computers could be found cnly in research laboratories. Now they are available commercially. For this reason we are entering an exciting period, when work on parallel algorithms can progress beyond design and analysis into implementation and use. This book has two primary goals: to familiarize the reader with classical results and to provide practical insights into how algorithms are made to run efficiently on processor arrays, multiprocessors, and multicomputers.

Chapter 1 puts parallel computing in perspective, showing the need for higher-performance computers and summarizing methods used in the past to increase computer performance. Chapter 2 begins with a presentation of a number of fundamental processor organizations and continues with a description of three parallel computer architectures: the processor array, the multiprocessor, and the multicomputer. Chapter 3 addresses many of the efficiency issues confronting the designer of parallel algorithms.

The chapters after Chapter 3 are more specialized. Chapter 4 presents some important results in parallel sorting from the large body of work done in this area. Chapter 5 discusses dictionary operations and illuminates trade-offs between the complexity of the underlying sequential algorithm and the potential for keeping a large number of processors busy doing useful work. Matrix multiplication is a fundamental component of many numerical and nonnumerical algorithms. Results in parallel matrix multiplication appear in Chapter 6. Chapter 7 describes parallel numerical algorithms to solve recurrence relations, partial differential equations, and systems of linear equations. Chapter 8 surveys parallel algorithms for searching graphs and finding connected components, minimum spanning trees, and shortest paths in graphs.

The final three chapters address current trends and past successes. Areas in which parallel computing may have a significant impact in the future include artificial intelligence and logic programming. Chapter 9 describes potential parallelism in the solution of combinatorial search problems. These problems occur in artificial intelligence, operations research, and graph theory, among other areas. Chapter 10 introduces Prolog, a logic programming language, and summarizes approaches to executing logic programs in parallel. Pipelined vector processors have been of great historical importance; two well-known machines, the Cray-1 and Cyber-205, are surveyed in Chapter 11.

The principal audience for this text is intended to be seniors and graduate students in computer science. Suggested prerequisites are calculus, high-level language programming, data structures, operating systems, computer architecture, and the analysis of algorithms.

The book includes many parallel algorithms written in a machine-independent, high-level pseudocode. Experimental results from implementations of parallel algorithms have been included wherever possible. Important results have been presented as theorems, to make them easier to reference. Each chapter ends with a set of exercises. They range from the elementary to the difficult. A Glossary of parallel computing appears after Chapter 11. References are given throughout the text, and a large bibliography appears at the end of the book. A solutions manual is available to instructors only.

I have taught a one-semester graduate-level course in parallel computing at the University of New Hampshire, using earlier drafts of this book. I recommend that you supplement the exercises with actual programming assignments on a parallel computer or a simulator. Programming a parallel computer is a new, difficult, and exciting experience for most students, and they learn a great deal from their efforts. In addition, graduate students should read recent journal articles and conference papers. With these supplements, there is more than enough material for a one-semester course, giving the instructor some latitude. I have usually taken an "historical" approach, covering Chapters 1, 11, 2, and 3 before the midterm examination and Chapters 4, 5, 6, 8, 9, and 10 in less depth after the midterm.

Kai Hwang, B. Jayaraman, and Vipin Kumar provided many helpful suggestions that led to a substantial improvement in the quality of the text between the first and second drafts. Kaye Pace, my editor at McGraw-Hill, always made me feel as if I were her only responsibility. Let me extend my thanks to everyone involved in the production of the book.

I feel fortunate to have had as my dissertation advisor Narsingh Deo, who introduced me to the area of parallel algorithms. I am grateful to Donald Knuth and numerous unknown support people, who made the TEX typesetting system public, and to L. Michael Gray, who installed and maintained the TEX environment at the University of New Hampshire. I had felt for some time that there was a book in me, waiting to get out. Seeing my words transformed into beautifully typeset output was all the catalyst I needed.

Finally, I would like to thank my teachers throughout the years who provided me with such an inspiring example.

Michael J. Quinn

CONTENTS

CHAPTER
ONE
INTRODUCTION

This book is concerned with **parallel computing**, the process of solving problems on parallel computers. Parallel computing is a relatively young field: the Illiac IV, a processor array, became operational in 1975; the first Cray-1, a pipelined vector processor, was delivered in 1976; and low-cost multiprocessors were not available before 1984. The advent of very large-scale integration (VLSI) heralded a new era in computing: not only did it make the personal computer possible, but also it made practical the development of large-scale computing devices consisting of tens, hundreds, even thousands of processors, all working together to perform a computation.

Although the study of parallel computing is a new discipline, it is far from unimportant. Many programs that run well on conventional computers are not easily transformed to programs that efficiently harness the capabilities of parallel computers. Conversely, algorithms that are less efficient in a sequential context often reveal an inherent parallelism that makes them attractive bases for parallel programs.

Many claim we are entering "the decade of the parallel computer." Applications demand computers that are many *orders of magnitude* faster than the fastest computers available today. Parallelism represents the most feasible avenue to achieve this kind of breakthrough, and countries throughout the western world are vigorously developing ever more powerful parallel computers. Some of these computers are quite expensive: the Connection Machine, marketed by Thinking Machines Corporation, contains up to 65,536 processors and costs $3 million. Other parallel computers cost little more than professional work stations. Low-cost multiprocessors and multicomputers have been announced by Ametek, Aretè, Encore, Intel, NCUBE, Sequent, and other companies.

The remainder of this chapter puts the problem of parallel computing in context. Section 1-1 lists a few applications that demand computers much faster than those presently available. Section 1-2 presents a brief history of architectural advances used to increase the performance of computers over the past 35 years. This section also explores the difference between pipelining and parallelism. Section 1-3 introduces the architectural classification schemes of Flynn and Händler. Finally, Section 1-4 examines reasons that have traditionally been given opposing the feasibility of high-level parallel computation. Some of these reasons can now be refuted easily; others are more weighty. It is good to remember that it takes a special kind of creativity to unleash the full power of a parallel architecture. Perhaps that is the best reason to study parallel computing: Success is more difficult and hence more rewarding.

1-1 THE NEED FOR HIGHER-PERFORMANCE COMPUTERS

The increasing power of computers has led to greater visions of what they might be able to do. Hwang and Briggs [1984] point out that mainstream computer usage is gradually becoming more and more sophisticated, progressing from data processing and information processing to knowledge processing and, eventually, intelligence processing. Each level of increasing sophistication demands much more powerful computers. We consider a few examples of current applications that could use extremely powerful computers. Although most of these applications are using computers for "number crunching," other applications are using computers to manipulate symbols or ideas. Hwang and Briggs [1984] are the primary source of information for these examples; their text contains more information about these and other uses.

Weather Prediction

Forecasting the weather on a computer requires the solution of general circulation model equations in a spherical coördinate system. A three-dimensional grid partitions the atmosphere by altitude, latitude, and longitude. Time is the fourth dimension; it, too, is partitioned by specifying a time increment. Given a grid with 270 miles on a side and an appropriate time increment, about 100 billion operations must be performed to compute a 24-hour forecast. This can be done in about 100 minutes on a computer capable of performing 100 million operations per second, such as a Cray-1. A grid this coarse is capable of producing a forecast for New York and Washington, D.C., but not for Philadelphia, located approximately halfway between the other two cities. To get a more accurate forecast for Philadelphia, the grid size would have to be halved in all four dimensions, leading to a 16-fold increase in the number of computations required. A computer capable of 100 million floating-point operations per second (100 megaflops), like the Cray-1, would require 24 hours to complete the 24-hour forecast. Even this new grid size would not be sufficiently fine to allow reliable long-range forecasting. If we want to receive accurate long-range forecasts, much more powerful computers must be developed.

Computational Aerodynamics

Wind tunnel experiments have a number of fundamental limitations. These include the model size, wind velocity, density, temperature, wall interference, and other factors. Numerical flow simulations have none of these limitations. The replacement of wind tunnels by computers has been limited only by the processing speed and memory capacity of the computer being used. The Burroughs Corporation and Control Data Corporation have proposed supercomputers, known as the numerical aerodynamic simulation facilities, with the goal of eliminating the need for wind tunnels. These supercomputers are designed to perform more than a billion floating-point operations per second (gigaflops).

Researchers at the University of Illinois, supported by a grant from the National Science Foundation, have begun using supercomputers to study wind shear. Using computational aerodynamics to "fly" simulated airplanes through microbursts, they hope to learn more about microbursts and the dangers posed to commercial aviation [USA Today 1985].

Artificial Intelligence

Most current computers have a relatively inflexible input/output (I/O) interface. If computers are to become more "user-friendly," they must be able to interact with humans at a higher level, using speech, pictures, and natural language. Allowing voice, pictorial, and natural language input to be handled in real time requires an enormous amount of computing power, much more than is available on standard architectures.

Japan has begun a project to develop fifth-generation computers. One of the goals of the project is to build a computer capable of making 100 million to 1 billion logical inferences per second. Since one logical inference may take anywhere from 100 to 1000 machine instructions to execute, such a machine would have to be able to perform between 10 billion and 1 *trillion* instructions per second.

Remote Sensing

The analysis of earth-resource data broadcast from satellites has many applications in agriculture, ecology, forestry, geology, and land use planning. However, images are often so large that even simple calculations require large amounts of CPU time. For example, a single Thematic Mapper image from the latest Landsat satellite is a 6000-picture element (pixel) by 6000-pixel square. Each pixel is represented by 8 bits; the entire picture is made up of eight images, or bands. A single picture, then, is represented by 288 megabytes of information.

NASA has installed the Massively Parallel Processor (MPP), manufactured by Goodyear Aerospace, to perform satellite image processing.

Nuclear Reactor Safety

The importance of being able to do simulations in real time is evident when you consider the problem of providing computer-aided analysis and simulation of events in a nuclear reactor. The ability to double-check a corrective measure before it is taken could help keep minor malfunctions from turning into major catastrophes. Only supercomputers have the processing speed that would enable such calculations to be done in real time.

Military Uses

Many existing supercomputers are being used by agencies doing research for the military. These agencies use supercomputers to design nuclear weapons, simulate their effects, gather intelligence, and process cartographic data in order to generate maps automatically. Since the U.S. Department of Defense continues to support research into parallel computing, clearly it desires even faster computers.

1-2 METHODS USED TO ACHIEVE HIGHER PERFORMANCE

There has always been a demand for faster computers, and computer engineers have used two methods to achieve higher performance. First, they have increased the speed of the circuitry; second, they have increased the number of operations that can take place concurrently, through either pipelining or parallelism. This section highlights the advances made since the inception of the electronic computer, showing how pipelining and parallelism have worked their way into the design of modern high-performance computers. The book by Hockney and Jesshope [1981] serves as the primary source of information on these advances.

Figure 1-1 illustrates how computer performance has increased over the past three decades. There has been roughly a 10-fold increase in the speed of computer arithmetic every 5 years. For example, the speed of floating-point multiplication increased by a factor of about 1,000,000 between the early 1950s and the early 1980s. Only some of the increase can be attributed to an increase in the speed of components. Architectural advances must be credited with the remainder of the increase. For example, the clock of the MPP is only about 1 order of magnitude faster than the clock of the EDSAC1; the rest of the speedup is due to the ability of the MPP to perform 16,384 multiplications in parallel.

The speed of light puts a ceiling on the speed at which electronic components of a certain size can operate. Hence parallelism has been introduced at many levels to improve the performance of computers. Many different architectural

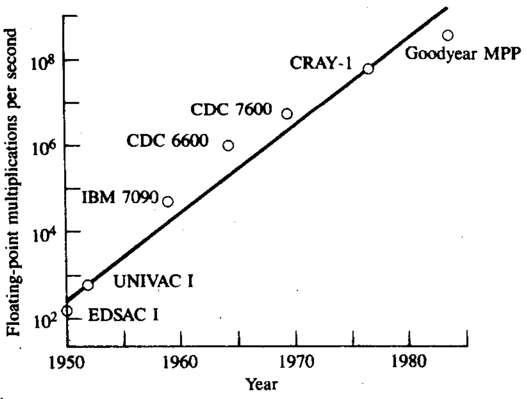

Figure 1-1 The performance of computers has increased by a factor of 10 every 5 years.

advances have been made over the past three decades. Some of these advances are summarized below. Frequently an advance was only possible because a new technology became available. Such new technologies will be mentioned where appropriate.

Bit-Parallel Memory and Bit-Parallel Arithmetic

The first electronic digital computers used a bit-serial main memory. Each bit of a word was read individually from memory. The EDSAC, SEAC, Pilot ACE, EDVAC, and UNIVAC all had mercury delay line (ultrasonic) bit-serial memories.

The first memory to allow all the bits in a word to be accessed in parallel was a cathode ray tube (CRT) system named after F. C. Williams of Manchester University in England. Although Williams used the CRT in bit-serial mode, the SWAC computer at the Institute for Numerical Analysis in England used Williams tubes in a parallel mode with the *k*th bit of memory words stored in the *k*th CRT. Williams tubes in parallel mode were also used in the computer at the Institute for Advanced Study and in the IBM 701 (1953).

Bit-parallel arithmetic became possible once bit-parallel memory was available. The IBM 701 was the first commercial machine to have bit-parallel arithmetic.

A prototype ferrite core memory was constructed by Jay Forrester at M.I.T. in 1950. The IBM 704 (1955) was the first commercial computer to use core memory. Besides allowing bit-parallel memory access, cores had the advantages of zero standby power, reasonable cost, speed for general-purpose applications (especially in large systems), and nonvolatility.

I/O Processors (Channels)

In first-generation computers I/O instructions were executed by the CPU. I/O devices continued to become faster (a magnetic tape drive is about 100 times

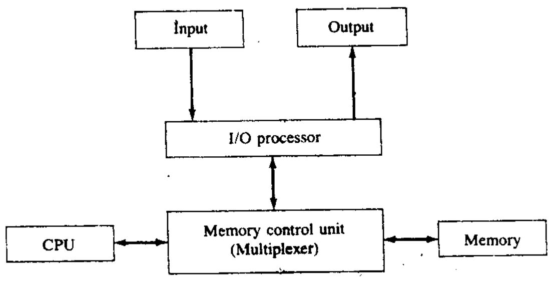

Figure 1-2 I/O processors are one of the architectural advances distinctive of second-generation computers.

faster than a card reader), but even the data transmission speed of a tape drive was far slower than the data manipulation speed of a processor. Because the electromechanical I/O devices were much slower than the electronic CPU, the CPU spent most of its time idling while executing an I/O instruction. This inefficiency frequently was a cause of poor performance [Hayes 1978].

The problem was solved by introducing a separate processor to handle I/O operations. This I/O processor, called a **channel**, receives I/O instructions from the CPU but then works independently, freeing the CPU to resume arithmetic processing. The channel has its own instruction set, custom-tailored for I/O operations. Six channels were added to the IBM 704 in 1958; the new computer was renamed the IBM 709. I/O processors are one of the architectural advances distinctive of second-generation computers (Figure 1-2).

Interleaved Memory

An **interleaved memory** is a memory unit divided into a number of **modules**, or **banks**, that can be accessed simultaneously. Each memory bank has its own addressing circuitry. Instruction and data addresses are interleaved to take advantage of the parallel fetch capability. With **low-order interleaving** the low-order bits of an address determine the memory bank containing the address; with **high-order interleaving** the high-order bits of an address determine the memory bank. Figure 1-3 illustrates the difference between low-order interleaving and high-order interleaving.

When a computer is being designed, it is important to match the speeds of the various components. For example, it does no good to have an extremely fast CPU if the memory unit cannot keep it supplied with instructions and data. The IBM STRETCH computer (1961) was the first computer to have an interleaved memory. Memory interleaving enabled a relatively slow magnetic-core memory to keep up with a fast processor. Memory was divided into two memory banks. Thus, the maximum data transfer rate to and from memory was increased by a factor of 2.

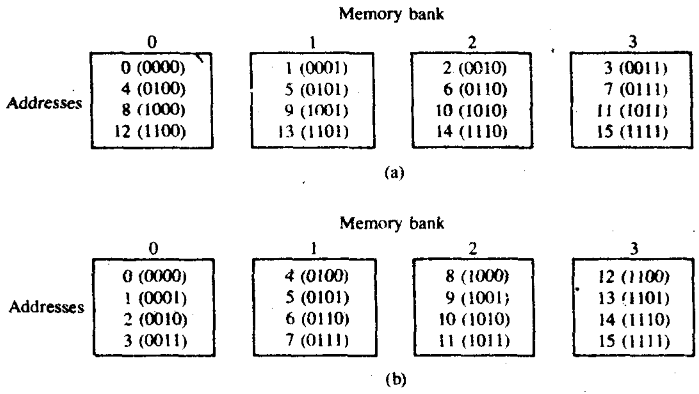

Figure 1-3 Memory interleaving. (a) Low-order interleaving lets the low-order bits of an address determine the memory bank. (b) High-order interleaving lets the high-order bits of an address determine the memory bank.

The ATLAS computer (1963), famous for its innovations in virtual memory, paging, and multiprogramming, had a four-way interleaved memory. The CDC 6600 (1964) divided memory into 32 independent banks. Since the IBM STRETCH, virtually all large computers have used interleaved memory.

Cache Memory

A **cache memory** is a small, fast memory unit used as a buffer between a processor and primary memory. The purpose of a cache memory is to reduce the time the processor must spend waiting for data to arrive from the slower primary memory. The efficiency of a cache memory depends, in part, on the **locality of reference** in the program being run. **Temporal locality** refers to the observed phenomenon that once a particular data or instruction location is referenced, it is often referenced again in the near future [Madnick and Donovan 1974]. **Spatial locality** refers to the observation that once a particular memory location is referenced, a nearby memory location is often referenced in the near future [Madnick and Donovan 1974]. Given a reasonable amount of locality of reference, the majority of the time the processor can fetch instructions and operands from cache memory, rather than primary memory. Only when the instruction or operand is not in the cache memory must the processor idle. Although cache memories began to appear in computers in the early 1960s, they were not economical before the introduction of large-scale integration (LSI) semiconductor memories in the late 1960s. A typical memory hierarchy, showing the position of cache memory, appears in Figure 1-4.