OXFORD APPLIED MATHEMATICS
AND COMPUTING SCIENCE SERIES

Parallel Algorithms and Matrix Computation

JAGDISH J. MODI

Jagdish J. Modi
University of Cambridge

Parallel Algorithms and Matrix Computation

CLARENDON PRESS · OXFORD
1988

Oxford University Press, Walton Street, Oxford OX2 6DP

Oxford New York Toronto
Delhi Bombay Calcutta Madras Karachi
Petaling Jaya Singapore Hong Kong Tokyo
Nairobi Dar es Salaam Cape Town
Melbourne Auckland

and associated companies in
Berlin Ibadan

Oxford is a trade mark of Oxford University Press

Published in the United States
by Oxford University Press, New York

British Library Cataloguing in Publication Data
Modi, Jagdish
Parallel algorithms and matrix computation.
1. Computer systems. Parallel-processor systems. Programming. Algorithms
I. Title II. Series
005.1
ISBN 0-19-859655-3
ISBN 0-19-859670-7 (pbk.)

Library of Congress Cataloging in Publication Data
Modi, Jagdish.
Parallel algorithms and matrix computation/Jagdish Modi.
(Oxford applied mathematics and computing science series)
Bibliography. Includes index.
1. Parallel processing (Electronic computers) 2. Algorithms.
3. Matrices. I. Title. II. Series.
QA76.5.M543 1988 004'.35—dc 19 88-9687
ISBN 0-19-859655-3
ISBN 0-19-859670-7 (pbk.)

Typeset in Northern Ireland by The Universities Press (Belfast) Ltd
Printed in Great Britain
at the University Printing House, Oxford
by David Stanford
Printer to the University

This book is dedicated with
affection and respect to my parents,
my sister, and my younger brother

Preface

The introduction of parallel computers some twenty years ago both provided an opportunity and prompted a need to develop parallel algorithms. In the process of this development, new perspectives have opened up, and a general theory of parallel algorithms, connecting certain established principles in logic and mathematics, will undoubtedly emerge. The elaboration of parallel methods has permitted not only more freedom of expression in problem analysis, but more generally has stimulated wide-ranging intellectual inquiry. Advanced applications, including image processing, robotics, speech recognition, and artificial intelligence, which have overtaken the capacity of the older sequential machines, can expect to benefit from this scientific impetus.

The purpose of this book is to bring together and further articulate the fundamental concepts in parallel computing. I will be focusing particularly on numerical linear algebra in view of its primary importance to both industry and academic research. I have divided the book into two parts. The first part (Chapters 1 to 3) discusses general principles and algorithmic techniques, treating a number of examples, and contains the essentials of a final-year undergraduate course in computer science or computational mathematics. This is complemented in Part 2 (Chapters 4 to 7) by a detailed exposition of some of the key areas of application in linear algebra. While the first part of the book will be of interest to a wide audience including those acquainted only with serial computing, the second part will demand a level of mathematical maturity typical of a graduate in mathematics, physical sciences, or engineering.

The book is structured in the following way. Chapter 1 introduces the fundamentals of parallel computing. Here I provide an up-to-date account of, *inter alia*, the classification of parallel computers, the interplay between machine architecture and algorithm design, performance measurement, and the classification of parallel algorithms. Chapter 2 puts the emphasis on algorithmic design and the variety of newly discovered techniques

which fully exploit the features of parallel computers. Chapter 3 is devoted entirely to parallel sorting techniques, demonstrating how parallelism may be incorporated into some already well-established serial methods.

The remaining chapters concentrate on matrix computations. The discussion of the solution of linear algebraic equations (Chapter 4) is limited to a selection of methods designed to illustrate research into the use of parallelism. In Chapter 5, the Jacobi method for the symmetric eigenvalue problem is discussed in detail. The remarkable stability of the orthogonal transformations, together with the simultaneous application of independent plane rotations, leads to a consideration of these techniques in other applications, notably QR factorization (Chapter 6) and singular-value decomposition and the linear least-squares problem (Chapter 7).

Throughout the book, I have tended to stress the identification of parallelism rather than its implementation on particular architectures, though numerous results, especially in Part 2, apply to the Distributed Array Processor (DAP).

It is a pleasure to acknowledge the assistance of many friends and colleagues. I express my sincere thanks to Professors D. J. Wheeler and D. Parkinson and Dr R. K. Livesley for providing constructive criticism and invaluable guidance. I am grateful for their detailed comments on specific chapters to Professors J. K. Iliffe, L. M. Delves, R. O. Davies, C. C. Paige, Drs J. S. Rollett, P. M. Flanders, M. A. Sabin, and Mr S. J. Hammarling.

I am indebted to Drs H. R. Pratt and E. J. Primrose for greatly enhancing my text, and to Mrs M. Ward for—with immense patience—typing the manuscript. I would also like to express my appreciation to the Oxford University Press, and in particular to Mr J. Bentin, for their consistent co-operation. Last but not least I would like to thank my parents, whose unfailing generosity I can never hope to repay.

Cambridge J.J.M.
February 1988

Contents

Part 1

Fundamentals of Parallel Computation

1 General principles of parallel computing

1.1 Parallel computing, past and present

The 1980s will go down in history as the decade in which parallel computing first had a significant impact in the scientific and commercial worlds. A substantial improvement in speed over serial computation has already been secured, by up to a factor of 1000, for a number of major applications, but the scope for further developments remains considerable. Machines are currently being built that exhibit an extensive degree of artificial intelligence, and incorporate a significant amount of parallelism. For example, the Connection Machine, based on learning algorithms, has 65,536 intelligent memory cells capable of communicating with other cells and thus representing a semantic network. As technological sophistication increases, more complex systems aimed at special-purpose applications such as the processing of speech and natural language will undoubtedly emerge. The concepts of both parallel computation and artificial intelligence are, however, far from original, and date back to the earliest developments in computing.

In an often-quoted statement, Lady Lovelace (1843), discussing Babbage's machine, wrote that 'The Analytical Engine has no pretensions to *originate* anything. It can do *whatever we know how to order it to perform*' (her italics). But she was also well aware that it can execute conditional instructions and therefore adapt its own operation during the course of a calculation: 'The engine is capable, under certain circumstances, of feeling about to discover which of two or more possible contingencies has occurred, and of then shaping its future course accordingly.' Menabrea (1842), in his *Sketch of the Analytical Engine Invented by Charles Babbage,* stresses accuracy, rapidity, and the saving of intellectual drudgery as the advantages of the machine. But Turing (1950) pointed out that in fact 'the Analytical Engine was

a universal digital computer', and therefore with adequate storage capacity and speed could mimic any machine, even a hypothetical one that 'thinks for itself' (although, as he said: 'Probably this argument did not occur to the Countess or to Babbage'). In this celebrated paper *Computing Machinery and Intelligence,* Turing essentially gives an affirmative answer to the question: 'Can machines think?'

In Menabrea's report we also read that 'when a long series of identical computations is to be performed, such as those required for the formation of numerical tables, the machine can be brought into play so as to give several results at the same time, which will greatly abridge the whole amount of the processes'. It appears that Babbage was conscious of the basic speed advantage parallelism can offer. But if the concept of parallelism is far from novel, no more than a very limited amount—in the form of instruction pipelining—was envisaged in the classical Von Neumann concept of the serial computer and incorporated in the earliest machines such as EDSAC 1 (1949) and UNIVAC (1951). The application of parallelism at various levels (job level, algorithm level, statement level, etc.) is comparatively recent.

Figure 1.1 shows the increase in arithmetic speed through the development of serial computers since 1950. Improvements in hardware have brought tremendous advances, for example the successive replacement of vacuum-tube switches by transistors, large-scale integrated circuits (LSI), very large-scale integrated circuits (VLSI), and ultra large-scale integrated circuits (ULSI). By the late 1970s, although the transmission of electronic signals through wires had become so fast—almost approaching the speed of light—this nonetheless proved to be the limiting factor compared with electronic switch speed. Minimizing the length of wire unfortunately heats up the switches, and consequently taxes the physical limitations of thermal conductivity. The Cray-1 *vector processor,* which is discussed later, opted for small-scale integration with an elaborate cooling system to prevent meltdown. A twenty-fold speed-up every decade (Fig. 1.1) has thus resulted primarily from progress in hardware technology and—to a lesser extent—from slave processing, interleaving of memory, and the incorporation of parallelism at lower levels.

With the increase in computing capability, certain large-scale computational problems, especially those concerned with pre-

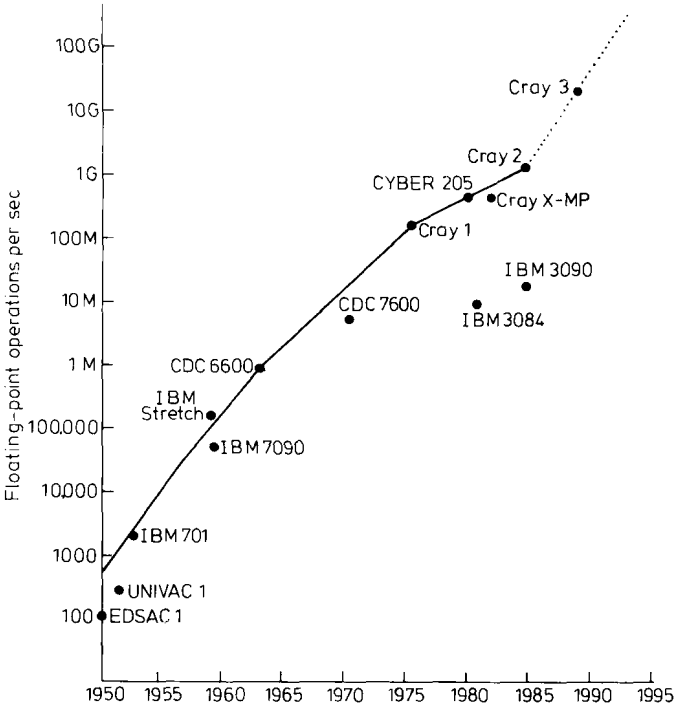

Fig. 1.1 Improvement in peak computer performance 1950–1995

dicting the behaviour of complicated physical systems by means of mathematical modelling, began to approach feasibility, and demands for even greater capacity became more pressing. In fact, this state of affairs is self-perpetuating: more computing power is always going to lead to the demand for yet more computing power. At any given moment, the capabilities of contemporary computers are inevitably one step behind the perceived needs for scientific and technological applications.

While machines of the classical Von Neumann design, with minor modifications, have thus been made very much faster, it seems certain that, to obtain comparable improvements in speed in the future, it will be necessary to incorporate a high degree of parallelism. Speed improvement on serial machines is constrained by an inability to increase the transfer speeds of

operands to and from the data buffers and the functional units. Particularly when lengthy repetitive calculations are required, the cost associated with data transfer becomes considerable: main storage devices currently in use, for instance, have access times of 100–150 nsec, and longer if accessed through a shared highway, whereas typical modern processor times are of the order of 50 nsec. The objective is therefore to strike an optimal balance between processor speed and memory access rate.

One well-established method of improving processing speed, by means of limited parallelism at assembler level, is to start each operation before the previous one is completed. This approach, known as pipelining, is discussed in detail in Section 1.9. Machines like those of the Cray and Cyber series couple the technique of pipelining with independent hardware units for performing distinct operations such as addition and multiplication. These allow a greater number of simultaneous operations, which in turn result in greater speed. The term vector processor is commonly used to describe such a system.

In the 1980s, VLSI technologies have led to the development of multiprocessors. High-speed buffers may be omitted entirely, and a large number of processing units connected directly to the memory banks. Memory is distributed among these processors, with the possible inclusion of a global memory to which all units have access; processor-interconnection schemes based on hypercubes, rings, and lattices have also been designed (Section 1.7). This concept is the basis of parallel processing. The single central processing unit of the classical Von Neumann design is replaced by many processors, which, even if individually somewhat slower, accelerate the speed of processing by operating in parallel. In terms of cost, an n-processing-unit parallel system costs much less than n times a single-unit system. These developments in hardware design were motivated by the use of semiconductor stores as memory components. Semiconductor devices have the same physical and electrical properties as logical devices, and it may be appropriate to consider the memory units and processing units together as a single entity: the 'active memory array' in the phrase coined by Iliffe (1982). A number of parallel machines have been built that incorporate vector processing, multiprocessing, or a combination of both. Figure 1.2 shows the three generations of supercomputers since the 1970s,

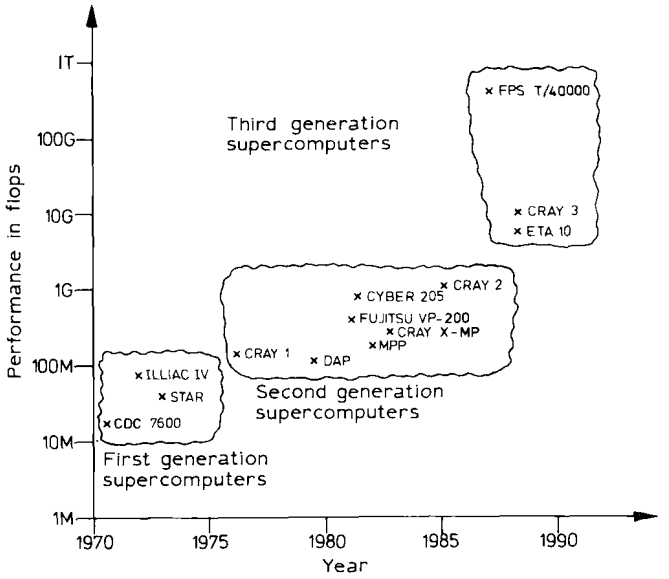

Fig. 1.2 Generations of supercomputers since the 1970s. The figures show approximate performance only, and strongly depend on the particular application

in terms of peak performance measured in *flops* (floating-point operations per second).

The innovation of parallel computing has also added a new dimension to the design of algorithms and programs. Parallel programming is not a simple extension of serial programming. In order to exploit the possibilities offered by parallelism, programmers need to 'think in parallel' and reconsider the solution process. Experience has shown that judgements of efficiency based on serial techniques can easily be overturned in a parallel environment (Section 1.10). Algorithms that are obsolete so far as implementation on serial machines is concerned often show a high degree of parallelism—i.e. contain numerous subcalculations that are independent of one another and may therefore be executed simultaneously—and can outperform the best serial techniques (for example, the Jacobi method for the eigenvalue problem; see Chapter 5). But, in addition to demanding new

techniques of algorithm design, the introduction of extended parallelism also requires us to rethink what we know about systems, parallel languages, non-numerical problems, and numerical methods in general.

This chapter is divided as follows. Initial sections provide background concepts with an informal survey of possible architectures and discuss Flynn's classification of parallel computers. The introduction of parallelism into computer systems has led to various architectures consisting of processing elements, and the interconnections between them. To discuss the issues related to transmission between individual processing elements we need some formal algebra, which is provided by graph theory (Section 1.6). We describe some commonly available networks (Section 1.7), particularly with respect to symmetry, homogeneity and embedding (Section 1.8). Although a vector processor does not constitute a truly parallel system, it nevertheless provides a significant improvement in speed (Section 1.9). The characteristics of parallel computers and parallel algorithms are qualitatively different from those of serial computation, and these issues are dealt with in later sections. The second part of the chapter thus provides a formal study, and leads naturally into a discussion of techniques for developing parallel algorithms in Chapter 2.

1.2 Classification of parallel computers

In a situation of rapid development in parallel computing, it is hardly surprising that no entirely satisfactory taxonomy of machines has yet been established. With some justification, it has been suggested that Flynn's classification, dating as it does from 1966 when parallel computing was in its infancy, does not adequately reflect current architectural designs. Kuck (1982) has attempted to provide a more up-to-date machine taxonomy and the reader is referred to his paper for further discussion of this subject. Nevertheless, as a guideline, Flynn's categories remain useful. He classifies computers into four types:

1. SISD: single instruction stream, single data stream
2. SIMD: single instruction stream, multiple data stream
3. MISD: multiple instruction stream, single data stream
4. MIMD: multiple instruction stream, multiple data stream.