Supercomputers

V. RAJARAMAN

Supercomputers

About the Series

The educational monograph series of the JNCASR attempts to cultivate a general appreciation of some of the frontier areas of science and engineering, amongst its readers. Each book in this series, written by an acknowledged authority, presents an emerging or a frontier topic from scientific, historical and utilitarian angles. The series though addressed primarily to graduate and postgraduate students of science and engineering also enjoys a general readership interested to keep abreast of what is going on in the world of science.

Editorial Board

S. K. Biswas *Coordinator*
V. Krishnan
N. Mukunda
J.V. Narlikar
G. Padmanaban
V. Rajaraman
K.J. Rao
C. N. R. Rao
J. Srinivasan
M. S. Valiathan

Titles published

1. *Superconductivity Today* — T. V. Ramakrishnan and C. N. R. Rao
2. *Bohr and Dirac — Images of Twentieth Century Physics* — N. Mukunda

Forthcoming Title

Genetic Engineering — G. Padmanaban

Supercomputers

V. RAJARAMAN
Jawaharlal Nehru Centre for Advanced Scientific Research
Bangalore-560012, India

JAWAHARLAL NEHRU CENTRE FOR
ADVANCED SCIENTIFIC RESEARCH, BANGALORE

PUBLISHING FOR ONE WORLD
WILEY EASTERN LIMITED
NEW DELHI BANGALORE BOMBAY CALCUTTA GUWAHATI
HYDERABAD LUCKNOW MADRAS PUNE

Copyright © 1993 { Wiley Eastern Limited
Jawaharlal Nehru Centre for Advanced Scientific Research

WILEY EASTERN LIMITED

NEW DELHI: 4835/24 Ansari Road, Daryaganj, New Delhi 110 002

BANGALORE: 27, Bull Temple Road, Basavangudi, Bangalore 560 004

BOMBAY: Post Box No. 4124, Saraswati Mandir School, Kennedy Bridge, Nana Chowk, Bombay 400 007

CALCUTTA: 40/8, Ballygunge Circular Road, Calcutta 700 019

MADRAS: No. 6, First Main Road, Gandhi Nagar, Madras 600 020

HYDERABAD: 1-2-412/9, Gaganmahal, Near A.V College, Domalguda, Hyderabad 500 029

PUNE: Flat No. 2, Building No. 7, Indira Co-op Housing Society Ltd. Indira Heights, Paud Fatta, Erandawane, Karve Road, Pune 411 038

LUCKNOW: 18, Pandit Madan Mohan Malviya Marg, Lucknow 226 001

GUWAHATI: Pan Bazar, Rani Bari, Guwahati 781 001

This book or any part thereof may not be reproduced in any form without the written permission of the publisher

This book is not to be sold outside the country to which it is consigned by Wiley Eastern Limited

ISBN: 81-224-0496-0

Published by H.S. Poplai for Wiley Eastern Limited, 4835/24, Ansari Road, Daryaganj, New Delhi 110 002 and printed at S.P. Printers, E-120, Sector 7, Noida. Printed in India.

Foreword

The **Jawaharlal Nehru Centre for Advanced Scientific Research** was established by the Government of India in 1989 as part of the centenary celebrations of Pandit Jawaharlal Nehru. Located in Bangalore, it functions in close academic collaboration with the Indian Institute of Science.

The Centre functions as an autonomous institution devoted to advanced scientific research. It promotes programmes in chosen frontier areas of science and engineering and supports workshops and symposia in these areas. It also has programmes to encourage young talent.

In addition to the above activities, the Centre has undertaken a programme of high quality publications at three levels:

(a) *Popular Science and General Books*—intended for the general public.

(b) *Educational Monographs*—short accounts of interesting areas in science and engineering addressed to students at the graduate and postgraduate levels.

(c) *Advanced Monographs*—devoted to specialised topics in current research intended for the international research community.

This monograph is one of the series being brought out as part of the publication and activities of the Centre. The Centre pays due attention to the choice of authors and subjects and style of presentation, to make these monographs attractive, interesting and useful to students as well as teachers. It is our hope that these publications will be received well both within and outside India.

C.N.R. RAO
President

Preface

Of late there has been a lot of interest in our country on Supercomputers. They have become part of the vocabulary of all educated persons as press reports appear regularly in newspapers about these computers and how they are being used to solve a whole range of very interesting problems from predicting the monsoons to synthezising life-saving drugs. There is thus a widespread curiosity to know what are supercomputers, in what way they are different from other computers and why they are considered strategic machines by the advanced western countries which have imposed strict export controls on them. The purpose of this educational monograph is to answer these questions and review the current state-of-the-art of supercomputers.

This book is meant for students in their final year M.Sc. or B.E. courses, (with a basic knowledge of programming in a high level language such as Fortran) who would like to know about supercomputers. It should also interest other scientists and engineers who would like to know about this subject.

The book begins with an introductory chapter which explains what is a supercomputer and why such a machine is needed to solve some problems. Chapter 2 discusses the architectural features of supercomputers which distinguish them from other computers. The third chapter deals with programming supercomputers. The need to vectorize programs to make effective use of supercomputers is brought out and some simple methods for vectorizing programs are explained.

Chapter 4 is on parallel computers. Parallel computers have emerged as a cost effective replacement for traditional vector supercomputers in some application areas. This chapter discusses briefly how problems may be solved in parallel and the architectural features of computers which can solve efficiently problems in parallel. The fifth chapter presents details of some of the commercially available supercomputers and parallel computers. The last chapter deals with some interesting applications of supercomputers.

The book is a broad based introduction to this subject. Many topics are discussed at an elementary level without delving deep into the detailed technical aspects of the problem. A set of references is provided at the end of the book to guide an interested reader to the material needed for deeper study.

In writing a book of this type, I naturally gained a number of ideas from numerous articles in journals and books on this subject. I thank all these authors, too numerous to acknowledge individually. Many colleagues and students generously assisted me by reading a draft of this book and suggested improvements. I thank all of them. My colleagues at the Indian Institute of Science, Dr.N.Balakrishnan, Dr. Mathew Jacob, Mr.R.Krishnamurthy, Mr.T.S.Mohan, Dr.A.Patel, Dr.S.Ramasesha, Dr.V.Viswanathan and Dr.S.Yashonath generously helped me by reading the manuscript carefully and suggesting many improvements which I have incorporated in the book. I sincerely thank them for their assistance.

I thank Prof.C.N.R.Rao, President, Jawaharlal Nehru Centre for Advanced Scientific Research, Bangalore, who encouraged me to write this monograph and for continually evincing interest in all my endeavours. I thank Ms. Mallika who cheerfully typed the manuscript in LaTeX format and drawing the figures in Xfig. I thank Mr.T.S.Mohan and Mr.S.Sundaram, who besides contributing their technical expertise in installing and running all the hardware and software, spent a lot of time in editing and in drawing the more difficult figures. I thank them for their very cheerful and willing efforts. Finally I express my heartfelt appreciation to my wife Dharma for reading the

manuscript, suggesting many improvements to make it understandable, editing and cleaning up the manuscript on LaTeX and Xfig and for her enthusiastic support which enabled me to write this book.

V.RAJARAMAN

Bangalore
1st July 1992

Contents

1

Introduction

In the last three years a lot of interest has been generated in our country on supercomputers. Supercomputers have become part of the vocabulary of all educated persons as press reports and cartoons regularly appear in the newspapers on India's order for a supercomputer and the delays in procuring it due to export controls on high technology by the countries manufacturing these machines. There is thus a widespread curiosity to know what is a supercomputer and how it is different from other computers. The purpose of this small book is to answer this question and to explain the logical organization of supercomputers and how they are programmed.

1.1 Defining a Supercomputer

The fastest and the most expensive computers available at any given time are generally called *supercomputers*. This is not a satisfactory definition but we are forced to use such a definition as computer technology has been evolving rapidly. In the past 20 years there has been a thousand fold increase in the speed of arithmetic operations of computers and it is thus difficult to give a time invariant definition of the speed of a supercomputer. Although the cost of computers for a specified speed and size has been going down, typically the cost of supercomputers has remained constant around US$ 10 million. Currently (1992), for a computer to be called a supercomputer it must have the following characteristics:

High Computing Speed: The computing speed of a supercomputer is measured in *megaflops*. A *mega* is a million and *flops* is an abbreviation for *f*loating point *o*perations *p*er *s*econd. A floating point operation is an arithmetic operation (add, subtract, multiply or di-

vide) on operands which are real numbers with fractional parts. The operands are expressed as a pair *(mantissa, exponent)*. For example, the number 185.67827 which equals 0.18567827 x 10^3 is expressed as (0.18567827, 3) where 0.18567827 is the mantissa and 3 is the exponent. In supercomputers the number of digits in the mantissa is 15 whereas it is 8 in other computers. By other computers we mean personal, mini and mainframe computers normally used in computing laboratories. The range of the exponent in supercomputers is also high, around ±5000 compared to ±99 in other computers. Thus the arithmetic precision and range of numbers used in supercomputers is significantly higher than those used in other computers.

The peak megaflop rating of modern supercomputers is around 1000 megaflops. In other words, supercomputers should perform around 1000 million floating point arithmetic operations per second with operands which have 15 digit mantissa and 4 digit exponent. Today higher speed supercomputers are being designed which will have a peak speed of about 10,000 megaflops. The peak megaflop rating is calculated by assuming that all units of a supercomputer work simultaneously at their highest speed on a single program. It is thus an idealization. The steady average speed obtainable is more important for a user and is much smaller. In fact the best average sustained megaflop rating obtained in solving a typical problem of inverting a 100 x 100 matrix of real numbers is of the order of 50 megaflops which is much smaller than the peak megaflop rating of 1000 megaflops. We will explain later why there is a wide disparity between the peak megaflop rating and the average sustained megaflop rating. In contrast with supercomputers, the average megaflop rating of, for instance, VAX 8810 a mainframe computer made by the Digital Equipment Corporation (DEC) is around 1 and that of an IBM PC is 0.01.

High Precision of Stored Numbers: We mentioned that supercomputers use 15 digits for mantissa which is double the number used normally in computers. One may wonder why such a high precision for representing numbers is needed. The reason is the possibility of ac-

cumulation of the small rounding errors made by the computer while performing arithmetic operations. Larger the number of arithmetic operations, larger will be the accumulation of errors. A computer carrying out 100 million operations per second will also be making 100 million rounding errors per second! Special care should thus be taken to prevent accumulation of rounding errors. More significant digits are used to represent real numbers in supercomputers to provide better factor of safety in arithmetic computation.

All numbers are stored as binary numbers in computers. Each real number is stored in a supercomputer's memory as a 64 bit unit called a *word*. Out of these 64 bits 49 are used to represent the mantissa (giving around 15 digit significance) and 15 bits are used to store the exponent and its sign. This provides an exponent range of $10^{\pm 5000}$.

Large, Fast Main Memory: The size of the main memory and its speed are important parameters of a supercomputer. As pointed out in the previous paragraph a word of a supercomputer is 64 bits long. The width of one data path from the main memory to the processing unit of a supercomputer is 64 bits. The size of the main memory is atleast 8 million words and recent machines provide 256 million words of main memory. This is to be contrasted with mainframe computers whose main memories are around 8 million 32 bit words. Besides the size, the time to access data from the main memory should be comparable to the arithmetic speed. Thus the memory is organized as an interleaved set of memory banks. Typically around 64 banks are interleaved allowing 64 words to be read in almost the same time as it takes to read one word. This time is around 50 nsec giving an average access of one word from the main memory to the processor every 0.78 nsec. (1 nsec = 1 nanosec = 10^{-9} sec). This is to be contrasted with mainframe computers whose average access time of data from main memory to the processor is at least 100 times slower.

Large, Fast Secondary memory: As the computation speed of supercomputers is high the data to be processed must be readily available in the main memory. For a processing speed of 150 megaflops a maximum of 150 million pairs of operands would be needed per

second and all of them cannot be stored in the main memory. As data is retrieved from the main memory and processed new data should be moved to the main memory from the secondary memory. The size of the secondary memory should be large and the speed with which data is to be transferred to main memory should be compatible with the speed of the main memory. Normally atleast 40 Gigabytes (Giga = 10^9). Gigabytes usually abbreviated as GB) storage is provided In supercomputers. The rate at which data is transferred from the secondary memory (usually magnetic disk memory) is around 40 Megabytes per second (Mega = 10^6, Megabytes abbreviated as MB). Disks in mainframe computers provide a data transfer rate of around 5MB per second.

To summarize, a computer may be classified as a supercomputer now (1992), if its average arithmetic speed is of the order of 50 Megaflops with 64 bit operands; it has a large capacity main memory of the order of 256 million 64 bit words with an access time compatible with the arithmetic speed and it has around 40 GB of secondary memory with a data transfer rate to main memory of around 100 MB per second. The main point to note is that mere megaflop rating is not sufficient to call a computer a supercomputer; there must be matching high speed high capacity main memory and high capacity disk drives with very high speed data transfer capability.

1.2 Why do we need Supercomputers?

Of late simulation has emerged as an important method in science, complementing theoretical analysis and experimental observations. Numerical simulation has many advantages:

- It is cheaper than setting up big experiments or building prototypes of physical systems.
- It is possible in a numerical model to change many parameters and observe their effect. Experiments do not allow easy change of many parameters.
- Numerical modelling is versatile. A wide range of problems can be simulated on a computer.

- Observations and interactions allow models to be refined. Such refinements provide a better understanding of physical problems which cannot be obtained from experiments.

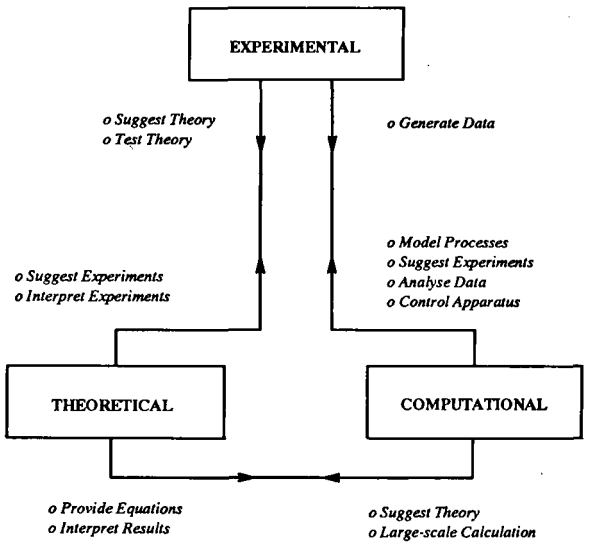


Fig 1.1. Interaction between theory, experiments and simulation

Numerical simulation is also known as "numerical experimentation" as the philosophy is similar to conducting experiments. In some instances it may in fact be the only feasible substitute for experiments, for example, nuclear fusion experiments and finding out the damage to an aircraft and type of injuries to passengers when an aircraft crashlands.

The role of experiments, theoretical models and numerical simulation models is shown in Fig 1.1. It is seen that experiments, theory and numerical simulation now form three interacting methods in sci-

ence and engineering.

With the advance in science, the models used incorporate more detail. This has increased the demand for computing speed and storage capacity. For example, in order to design supersonic aircraft[7] it is necessary to realistically simulate turbulent aerodynamic flows round its wings and body. This is a general non-linear problem modelled by using partial differential equations and cannot be solved analytically. Such a problem is solved by numerical simulation which requires the

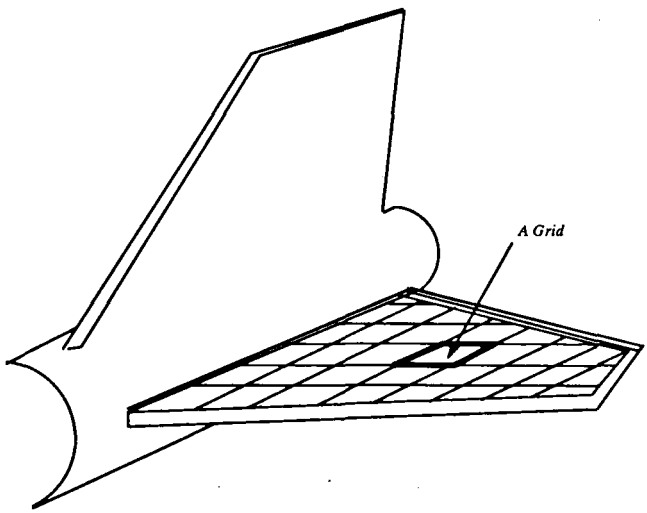


Fig 1.2. Grids on the surface of an aircraft wing used for simulation

surface of the body and wings to be modelled by 10^7 tiny squares bounded by parallel lines (or a grid) as shown in Fig 1.2.

The partial differential equations are discretized to difference equations which are in turn solved as a set of simultaneous algebraic equations. For each grid point 5 to 30 real numbers are stored to represent

vector quantities such as velocity, acceleration, and pressure. Sets of these equations are normally solved numerically using iterative methods (i.e., trial and error methods). In such iterative methods several trials (100 to 1000) are needed for each grid point before the results converge. The calculation of each trial value normally requires around 100 to 500 floating point arithmetic operations. Thus the total number of floating point operations required for each simulation run is approximately given by:

Number of floating point operations per simulation
= Number of grid points ×
Number of values per grid point ×
Number of trials ×
Number of operations per trial
= $10^7 \times 20 \times 100 \times 500$
= 10^{13}

If each floating point operation takes 1 microsecond (corresponding to 1 megaflop speed) then the time taken for each simulation run is 10^7 seconds which is 115 days!

A computer with 1 megaflop speed will thus take 115 days to complete one simulation run if it is operated 24 hours a day. In other words, this problem cannot be solved using such a computer. If the sustained megaflop rating of a supercomputer is 200 megaflops, then each simulation run will take 13.8 hours, which is still quite high, but manageable.

There are many other problems which also require large computational time. A common application is to model global weather. The behaviour of the earth's atmosphere affects global weather. The behaviour is modelled by partial differential equations in which the most important variables are the wind speed, air temperature, humidity and atmospheric pressure. The objective of a numerical weather model is to predict the status of the atmosphere at a particular area at a future time based on the current and past observations of the atmosphere. This is done by approximating the partial differential equations by a system of difference equations in which physical quan-

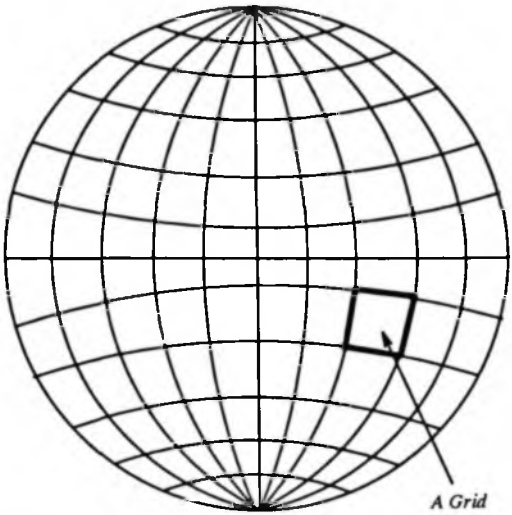


Fig 1.3. Grid for numerical weather model for the Earth

tities are specified at points on a three dimensional grid over the surface of the earth as shown in Fig. 1.3. In the horizontal plane the grid points are defined by 87 parallels of latitude between the north and south poles and 144 meridian circles equally spaced around the globe. In the vertical direction nine layers describe the atmosphere and five physical variables are used for the description. In this example also around 10^{10} floating point calculations are needed for each solution.

In general, the problem complexity is described by the formula:

$$PC = G \times V \times T \times A$$

where

PC = Problem Complexity
G = Geometry of the grid system
V = Variables per grid point

T = Number of steps per simulation for solving problem
A = Number of floating point operations per variable.

For the supersonic aircraft design example: $G = 10^7$, $V = 20$, $T = 100$ and $A = 500$ giving

$$PC = G \times V \times T \times A = 10^{13}$$

operations. For the weather modelling problem:

$$G = 144 \times 87 \times 9 = 112752,\ V = 5,\ T = 200\ A = 400$$

giving $PC \sim 10^{11}$ operations.

Currently a large number of realistic applications require 10^{12} to 10^{14} operations per solution. If each solution has to be done in about an hour, then the sustained speed of a computer should be $10^{14}/60 \times 60$ operations per second which is equal to 27,700 Megaflops!

If computing time per solution exceeds 100 hours we call it an *intractable* problem. Thus with a machine giving 100 megaflop sustained speed a problem requiring 10^{14} floating point operations for a solution is intractable. From the above examples it is clear why we need supercomputers with speeds in the thousands of megaflop range to solve problems of current interest to scientists and engineers.

1.3 How do Supercomputers achieve their speed?

One method of increasing the speed of computers is to use faster semiconductor components to build units of a computer. Higher speed components cost more and normally dissipate more heat. For example, low speed personal computers use integrated circuits based on silicon semiconductors. Currently Gallium Arsenide semiconductor devices which are faster are used in some supercomputers. The rate of growth of speed by using better devices and technology is relatively slow. For instance, the time to add two floating point numbers in a high performance computer in 1980 was 20 nanoseconds and in 1990 it was 6 nanoseconds.

Another method to increase the speed of computation is to design the computer so that the different units of the computer work

simultaneously. For instance, while the processor is computing, data which may be needed later could be fetched from memory and simultaneously an I/O operation can be going on. Such an overlap of operations is achieved by using both hardware and software features. This method is called an architectural method.

Besides overlapping operations of various units of a computer the arithmetic unit itself may be designed to exploit parallelism inherent in the problem being solved. For example, if two vectors are to be added, all pairs of components of the vector may be added simultaneously by a set of adders thereby reducing the time taken to add the vectors. This type of parallelism is called *data parallelism.* Time taken to add two vectors may also be reduced by designing an adder known as a pipeline adder which uses *temporal parallelism.* (We will explain this in the next chapter). Still another method of increasing speed of computation is to organize a set of computers to work simultaneously and cooperate to carry out tasks in a program. These methods are also classified as architectural methods.

Table 1.1. Methods used to increase the speed of computers

- Use faster devices such as GaAs to build computers
- Use architectural methods to exploit parallelism
  The Architectural Methods are:
  - Overlap operation of different units of a computer
  - Execute many instructions simultaneously with multiple functional units
  - Increase speed of arithmetic logic unit by exploiting data and/or temporal parallelism.

As the increase in speed of electronic components is limited by physical constraints, supercomputers primarily use architectural methods to exploit parallelism. Table 1.1 summarizes the architectural methods used to increase the speed of computers. In the subsequent chapters of this book we will explain the architectural methods.

2

Architecture of Vector Supercomputers

From their advent supercomputers have used a technique called *vector pipelined processing* to attain high speeds. This technique exploits *temporal parallelism* (i.e. time oriented parallelism) inherent in the problem being solved. In this chapter we will explain how temporal parallelism is perceived and utilized in the architecture of supercomputers.

2.1 Pipeline Processing

We will illustrate the idea of *pipeline processing* or assembly line processing with an example[6]. Assume that an examination paper has 4 questions to be answered and 1000 answer books are to be graded. Let us label the four questions in the answer books to be graded as Q_1, Q_2, Q_3, Q_4. Assume that the time taken to grade the answer to each question Q_1, Q_2, Q_3, Q_4 is equal to 5 minutes. If one teacher grades all the answers he/she will take 20 minutes to grade a paper. As there are 1000 answer papers, the total time taken will be 20,000 minutes. If we want to increase the speed of grading, then we may employ 4 teachers to cooperatively grade each answer book. The four teachers are asked to sit in a line (Fig 2.1). The answer books are placed in front of the first teacher. This constitutes the input. The first teacher takes an answer book, grades Q_1 and passes it on to the second teacher who grades Q_2; the second teacher passes the paper on to the third teacher who grades Q_3 and passes it on to the last teacher in the line. The last teacher grades Q_4, adds up the marks obtained by the student, and puts the paper in a tray kept for collecting the graded answer books which is the output.

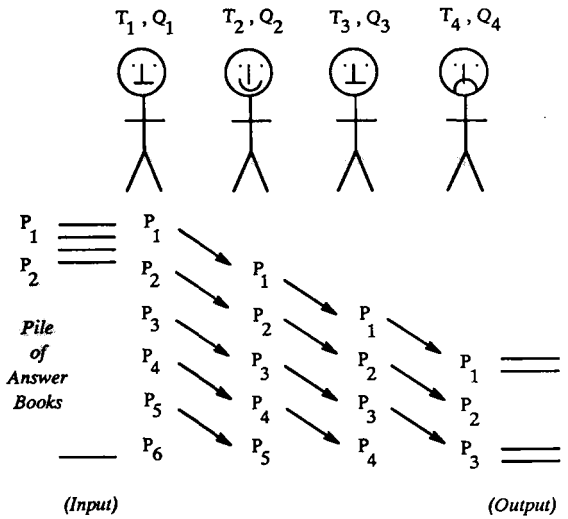


Fig 2.1. Four teachers grading papers in a pipeline

It is seen from Fig 2.1 that when the first answer book is being graded three teachers are idle. When the second answer book is being graded, two teachers are idle. However, from the fourth answer book all teachers are busy, with teacher 1 grading Q_1 of book 4, teacher 2 grading Q_2 of book 3, teacher 3 grading Q_3 of book 2 and teacher 4 grading Q_4 of book 1. As the time to grade each question is 5 minutes, the first answer book will take 20 minutes to be graded but subsequent papers will take only 5 minutes each. The total time taken to grade 1000 answer books will be $20 + (999 \times 5) = 5015$ minutes. This is about one fourth of the time taken by one teacher.

In this example we define a *job* as that of correcting an answer book. This job has been divided into four *tasks*. The four tasks are correcting the answer to Q_1, Q_2, Q_3 and Q_4 respectively. As this method breaks up a job into a set of tasks to be executed overlapped

in time it is said to use *temporal parallelism.* This method of processing is appropriate if:

1. The jobs to be carried out are identical
2. A job can be divided into many independent tasks. In other words, each task can be done by a processor independent of other tasks.
3. The time taken for performing each task is the same.
4. The time taken to transmit a job from one processor to the next is negligible compared to the time needed to execute a task.
5. The number of tasks into which a job is broken up is much smaller compared to the number of jobs to be carried out.

We can quantify the above points as worked out below:
Let the number of jobs = n
Let the time taken to do a job = p
Let a job be divisible into k tasks each taking time = p/k
Time required to complete n jobs with a pipeline of k stages

$$= p + (n-1)(p/k)$$
$$= p(k+n-1)/k$$

Speedup due to pipeline processing is

$$= np/\{p(k+n-1)/k\}$$
$$= k/[1+\{(k-1)/n\}]$$

If the number of jobs n is much larger than the number of stages in the pipeline k, then $(k-1)/n << 1$ and the speedup is nearly equal to k.

The main problems encountered in implementing pipeline processing are:

1. **Synchronization:** Each stage in the pipeline must take equal time for completion of a task so that a job can flow between stages without holdup. If the time taken by stage 1 is say t_1 and is less than the time taken by stage 2, t_2, then a job has to wait for a time $(t_2 - t_1)$ before entering stage 2. This job will

thus need, between stages, a temporary storage area (called a buffer) where it can wait.

2. **Bubbles in Pipeline:** If some tasks are absent in a job "bubbles" form in the pipeline. In the example of teachers grading papers, if an answer book has only two questions answered, two teachers will be forced to be idle when that paper is being corrected.
3. **Fault Tolerance:** The system does not tolerate faults. If one of the stages in the pipeline fails for some reason, the entire pipeline halts.
4. **Intertask Communications** The time to transmit a job between pipeline stages should be much smaller compared to the time taken to do a task.

Inspite of these disadvantages, this method is a very effective technique as it is easy to perceive in many problems how jobs can be broken up into tasks to use temporal parallelism. Further, the pipeline can be made very efficient by tuning each stage in the pipeline to do a specific task well. Pipelining is the main technique used by vector supercomputers to attain their high speed. We will examine in the next section how this is done.

2.2 Vector Processing

Consider a procedure for adding two floating point numbers x and y. As was pointed out a floating point number consists of two parts: a mantissa and an exponent. Let x be represented by the tuple $(mant\ x, exp\ x)$ and y by the tuple $(mant\ y, exp\ y)$. Let the result z be represented by the tuple $(mant\ z, exp\ z)$ The job of adding x and y can be broken up into the following four tasks:

Task 1: Compute $(exp\ x - exp\ y) = m$

Task 2: If $m > 0$ shift $mant\ y$, m positions right and fill the leading bits of $mant\ y$ with zeros. Set $exp\ z = exp\ x$
If $m < 0$ shift $mant\ x$, m positions right and fill the leading bits of $mant\ x$ with zeros. Set $exp\ z = exp\ y$. If $m = 0$ do nothing. Set $exp\ z = exp\ x$.

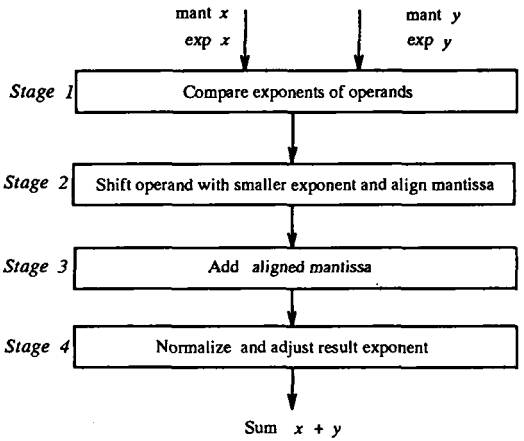


Fig 2.2. A pipelined adder unit with 4 stages

Task 3: Add *mant x* and *mant y*. Let (*mant x* + *mant y*) = *mant z*

Task 4: If *mant z* > 1 then shift *mant z* right by 1 bit and add 1 to *exp z*. If one or more most significant bits of *mant z* = 0 shift *mant z* left until leading bit of *mant z* is non zero. Let the number of shifts be *p*. Subtract *p* from *exp z*.

An electronic adder can be designed with 4 stages, each stage doing one of the tasks explained above. Such an adder is called a *pipelined adder* and is shown in the block diagram of Fig 2.2. To use this adder a sequence of operand pairs to be added (a_1, b_1), (a_2, b_2), (a_3, b_3) ..., (a_n, b_n) are fed to the adder (See Fig 2.3) and are shifted into the pipeline one pair at a time. Let T seconds be the time taken by each stage. One pair of operands is shifted into the pipeline from an input register every T seconds. The shifting is controlled by a series of pulses called a *clock*. The time elapsed between successive pulses is the clock period which in this example is T seconds (See Fig 2.3). The sum $c_1 = a_1 + b_1$ comes out of the pipeline after 4

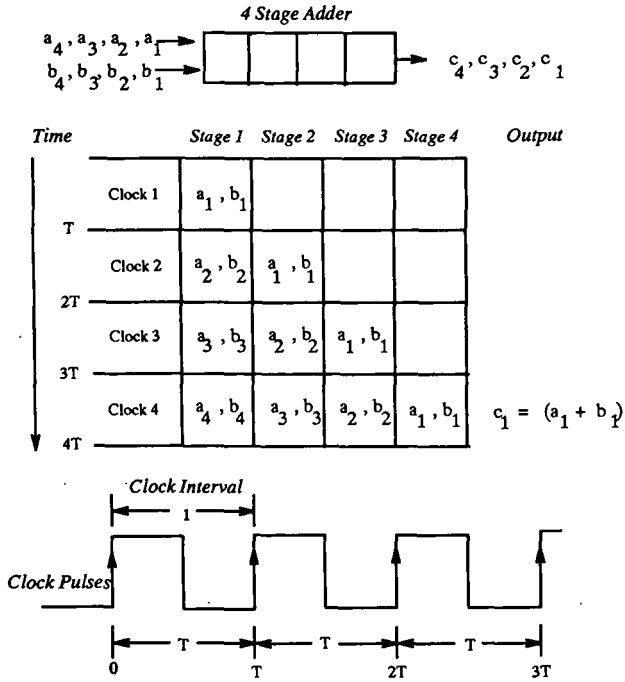


Fig 2.3. Adding vectors in a pipelined adder

clock periods which is $4T$ seconds. However, sums

$$c_2 = (a_2 + b_2),\ c_3 = (a_3 + b_3) \ldots c_n = (a_n + b_n)$$

come out at time $5T$, $6T \ldots (4 + n - 1)T$ seconds.

An ordered sequence of operands $(a_1, a_2, a_3, \ldots, a_n)$ is known as a *vector*. In the above example we add vectors $(a_1, a_2, a_3, \ldots, a_n)$ and $(b_1, b_2, b_3, \ldots, b_n)$ to obtain the sum vector $(c_1, c_2, c_3, \ldots, c_n)$. In order to obtain the potential speedup of a pipelined arithmetic unit,

it is necessary to add vectors with number of components much larger than 4, where 4 is the number of stages in the pipeline.

All vector supercomputers use pipelined processing of vectors extensively to attain high speeds. A typical vector pipeline unit of supercomputers manufactured by Cray Research Inc., USA, a pioneer in supercomputer design, is shown in Fig. 2.4. Here pipelined units consisting of a number of stages work synchronously for arithmetic computations. The computer has a centralized clock which generates a pulse once every T seconds (T is around 10^{-8}sec). In a pipelined addition unit one addition is carried out every T seconds by the system (after an initial time required to fill the pipeline). In a model known as Cray YMP system the value of T is 6 nano seconds (nano $= 10^{-9}$). Thus one floating point arithmetic operation is carried out in 6 nanoseconds giving a peak speed of $(1/6 \times 10^{-9})flops$ which is 166 megaflops. The size of the vector registers is 64 words in Cray computers.

Referring to Fig. 2.4 observe that a pipelined adder unit adds vectors A and B. Observe also that the sum (A+B) can be fed to another pipelined multiply unit. The pipelined multiply unit has as one input, components of (A+B) and as another input, components of a vector C. As components of (A+B), namely, (a_1+b_1), (a_2+b_2),..., $(a_n + b_n)$ are generated, they are multiplied by components of C, namely, c_1, c_2,... c_n to produce $(a_1+b_1)*c_1$, $(a_2+b_2)*c_2$, $(a_3+b_3)*c_3$, ..., $(a_n+b_n)*c_n$ respectively and stored in the vecor register marked D. This method of feeding the output of one pipelined arithmetic unit to another is called *vector chaining.* If a program has instructions of the type $D = (A + B) * C$ this chaining can be used. With chaining the speed of computation is doubled as an add and a multiply are done in one clock period.

Besides vector chaining, many independent pipelined arithmetic units can be provided in a supercomputer. In Fig 2.4, for example, there is a second pipelined adding unit using registers P and Q. This unit can be used independently and simultaneously with the other pipelined units provided the program being executed can use this

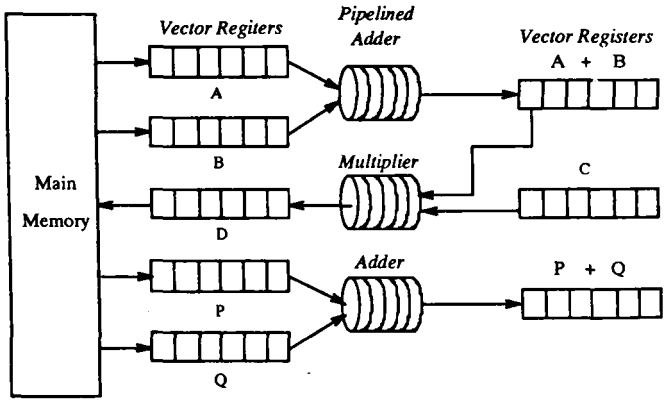


Fig 2.4. Organization of vector pipelined arithmetic units

facility.

2.3 Logical Structure of a Supercomputer

A block diagram of the various units of a supercomputer is shown in Fig 2.5. The main logical units of a supercomputer are:

1. Input/Output processors
2. Main memory
3. Instruction registers and processor
4. Scalar registers and processor
5. Vector registers and processor
6. Secondary storage system
7. Frontend computer system

The most important characteristic of supercomputers is the logical organization which facilitates the parallel operation of the subsystems. Parallelism is obtained by overlapping the operation of various units, for example, the I/O processor, the instruction processor and

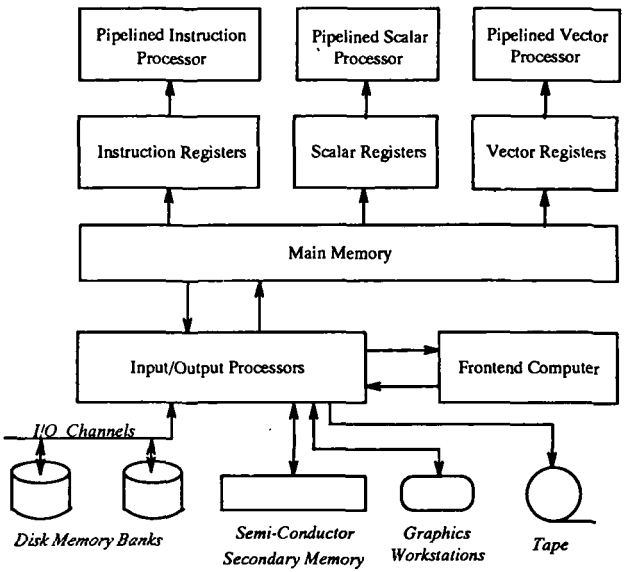


Fig 2.5. Block diagram of units of a supercomputer

the vector processor. Each of these processors also use parallelism obtained with pipelined processing.

Input-Output System: The I/O processing system consists of 4 to 16 small processors with their own instruction set, memory, arithmetic units and control. These processing units control reading/writing of programs and data from secondary storage to main memory, coordinate the operation of graphic workstations and also the frontend computer system. Each I/O processor has a memory of several thousand bytes and an instruction repertoire enabling them to run I/O programs independent of the CPU of the supercomputer. All I/O processors can work simultaneously. They have the ability to transfer blocks of words or a single word from the main memoryto the peripherals and vice-versa. They can also interrupt the CPU if

needed.

Main Memory: The main memory is a very important part of supercomputers. The memory is normally organized to store upto 256 megawords where each word is 64 bits long. A large storage is extremely beneficial in solving large scale scientific problems as speed improvement of more than 20 can be achieved if all data is stored in the main storage rather than in the secondary storage. This is due to the fact that data transfer rate from the main memory to the processor is at least 100 times faster compared to that from the secondary memory to the processor. Access time to a word in main memory is of the order of 50 nsec which is several times the basic clock cycle used by the CPU. To improve the speed of retrieval of words from the memory, it is arranged as a number of parallel "banks". Requests for reading words from memory can then be issued to each of the banks successively so that retrievals go on simultaneously. If the memory is organized as eight banks then request to read from the same bank will come after the other seven banks are read. Effectively the speed of retrieval will be increased eight times. The basic assumption made is that successive elements to be retrieved are in different banks and that data in consecutive addresses are to be retrieved. The organization of memory into banks and issue of requests to banks is illustrated in Fig. 2.6. It is clear that the idea of pipelined retrieval of elements from the memory is used in this case also.

The number of independent paths to the main memory has a direct bearing on the capability of a supercomputer to overlap execution of instructions. For example, Cray YMP supercomputer has two fetch paths and one store path all of which can operate simultaneously. Thus in executing the loop

```
      DO 25 I = 1,64
         C(I) = A(I) + B(I)
   25 CONTINUE
```

Cray YMP will do the following:

	Bank	Bank	Bank	Bank	Bank	Bank	Bank	Bank
Address Array Elements	1 A(1)	2 A(2)	3 A(3)	4 A(4)	5 A(5)	6 A(6)	7 A(7)	8 A(8)
Address Array Elements	9 A(9)	10 A(10)	11 A(11)	12 A(12)	13 A(13)	14 A(14)	15 A(15)	16 A(16)

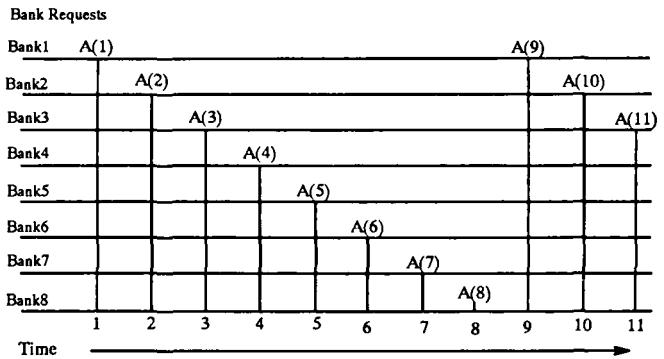


Fig 2.6. Organization of memory into banks

1. Fetch the 64 components of vector A represented by $A(1:64)$ and the 64 components of B, $B(1:64)$ simultaneously to two vector registers
2. As soon as $A(1)$ and $B(1)$ arrive start adding and place the result in a third vector register
3. As soon as $C(1)$ is ready start storing vector $C(1:64)$ concurrently into the main memory. Fig. 2.7 illustrates this.

Reliability of data read from the memory is of paramount importance. To ensure this, a Single Error Correction Double Error Detection (SECDED) coding scheme is used for all data stored in

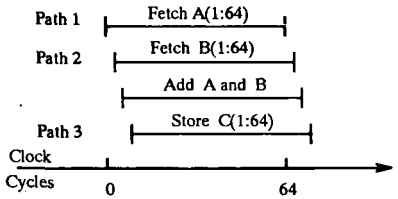


Fig 2.7. Addition and store times with multiple fetch and store paths to memory

the main memory. In order to detect and correct errors eight extra bits called check bits are appended to each 64 bit stored word in the main memory. When a word is received from the main memory to CPU, the 64 data bits are used to generate eight check bits. These eight check bits generated by the system are compared with the eight check bits retrieved along with the word. If they match then there is no error. If they do not match there may be one or more errors. A single error can be corrected or two errors can be detected by using the eight check bits. The coding scheme used is the one suggested by R.W.Hamming and is called the *Hamming Code.*

Instruction Buffers and Processor: The rate at which instructions are processed is faster than the rate at which instructions can be retrieved from the main memory. This speed mismatch between CPU and main memory has remained through various generations of computers. Computer architects have attempted to alleviate this problem by a technique called *instruction caching.* In this technique a small fast memory called a *cache memory* is provided in the CPU. A group of instructions are fetched before they are needed and stored in the cache. The instructions are taken one after another from the cache and executed. The fast cache memory can be partitioned into two segments. While instructions from one segment are being executed the other segment can be filled up with some more instructions from the main memory. Such a scheme will reduce overall instruction fetch time. Another advantage of having instructions in an instruction cache is that it enables repetitive use of these instructions from

the cache for executing programs with loops. If all the instructions belonging to a loop can be placed in the cache, then until the number of repetitions of the loop are completed there will be no need to fetch instructions from memory. As many programs in science and engineering are written in Fortran and use DO loops extensively, supercomputers provide a large instruction cache where all the loop instructions are stored. In the Cray YMP architecture, for example, instructions are divided into 16 bit "parcels". Four instruction caches called instruction buffers are provided each storing 128 parcels. Eight words (64 bits long) are read from memory each clock cycle and fill the least recently used instruction buffer. Backward and forward referencing of instructions within the buffer is possible for loop execution. If an instruction needed is not in the buffer then a word has to be accessed from the main memory. It takes 16 clock cycles to get an instruction from the main memory compared to one clock cycle for an instruction which is already in the instruction buffer.

The idea of instruction caching works because of the sequential nature of programs written in Fortran. There is a strong locality of reference to instructions in Fortran and similar sequential languages.

Besides instruction buffer, a special arithmetic unit to compute addresses of operands is also provided in the instruction processor. These arithmetic units perform pipelined integer arithmetic operations with 32 bit addresses in Cray YMP (32 bits are used for addressing in Cray YMP). Besides address computation this integer arithmetic unit can also be used for fast computation with small integers.

Scalar Registers and Processor: Even though vector arithmetic is crucial for obtaining the high speed of supercomputers, it is also very important to provide fast scalar arithmetic.(By a scalar we mean individual floating point numbers). Here again, the pipelining idea is used. The scalars are stored in buffer registers in CPU which feed the pipeline. Buffering is essential as accessing data from the memory is very slow compared to access from buffer registers. In Cray YMP, for example, there are eight 64 bit registers (called S-registers) to hold

scalar operands. These are backed up by 64 registers each with 64 bits which communicate with the main memory. A scalar instruction performs an operation such as addition obtaining two operands from two scalar registers and storing the result in another scalar register.

Vector Registers and Processors: Vector arithmetic forms the core of supercomputers. A set of vector registers (called V registers) are provided to store vector operands. In Cray YMP, for example, there are eight V registers. Each register can store 64 components of a vector with each component being 64 bits long. Many pipelined units to perform various operations such as add/subtract, multiply, shifting, finding reciprocal etc. using vector operands are provided. These units are called functional units. In Cray YMP, for example, there are 14 pipelined functional units in a CPU. Out of these, five are exclusively for vector operations, three are shared for vector as well as scalar operations, four are exclusively for scalar operations and two are for operations on addresses.

In Cray YMP a six stage pipeline is used for addition or subtraction and a seven stage pipeline for multiplication. Division is performed by first approximating the reciprocal of the divisor and multiplying it by the dividend. Table 2.1 shows the performance of some of the vector functional units of Cray YMP.

Table 2.1. Cray YMP vector functional units

Operation	Start-up time (No.of clock pulses)	Time taken for N results
Add/subtract	6	6+N
Multiply	7	7+N
Divide	38	38+3N
One Add Subtract & Multiply (called vector triad)	7	7 + N

More than one pipeline vector unit may be used simultaneously in most supercomputers. The Cray YMP allows three pipelines to be used simultaneously.

As explained earlier in this chapter, operands from the main memory are retrieved and stored in vector registers from which they are sent to the pipelined arithmetic unit. One pair of operands is sent per clock cycle to the arithmetic unit. The result is stored in a vector register. The contents of this register is stored later on in the main memory. If the number of components in a vector is shorter than the number of words in the vector register (64 words in the case of Cray YMP) there is nothing special to be done. If the number of components in the vector exceeds 64, then the extra components have to wait for another vector instruction to start execution. Thus there is a small drop in speed of arithmetic when the number of components in the vectors increase from 64 to 65 as shown in Fig 2.8. For very long vectors the buffers have to be filled many times. If one pair of operands can be retrieved from the main memory in each clock interval, then these operands can be fed directly to the pipelined arithmetic unit without any intermediate buffer store. The result can also be stored back immediately in the main memory. As there is no buffer size limitation, there is no limit to the length of vectors which can be handled in such a system. Further, this method will perform well with long vectors. The disadvantage, however, is the need for a very high speed memory from which a pair of operands can be retrieved each clock cycle. Cyber 205 and ETA 10 supercomputers which used to be manufactured by Control Data Corporation, USA, used this idea but had to use a slower clock. Such an architecture is called a *memory to memory* architecture system as the operands and results are directly retrieved from the memory and stored back in the memory. In contrast the Cray architecture is called a *Load-Store* architecture as the only operations involving memory are loading operands from memory to CPU registers and storing back results from CPU registers to the memory. Load-Store architecture is currently the preferred architecture as it is faster.

Secondary Storage Units: The size of the main memory available in supercomputers is not sufficient to store data and programs for many large scale scientific applications. Further, supercomputers are shared by many users and all the users' programs and data should

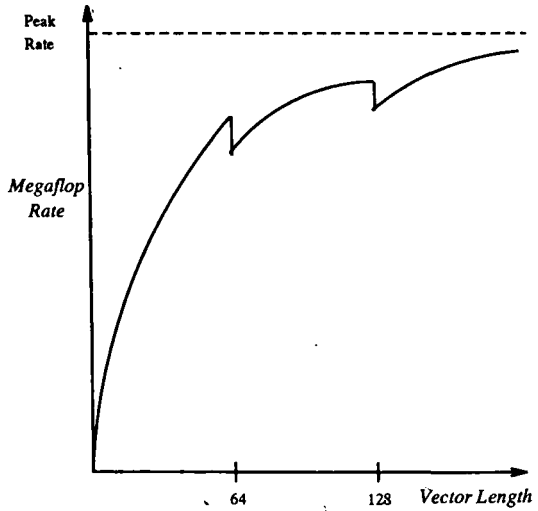


Fig 2.8. Vector processing rate as a function of length of vector

be kept ready for processing. Secondary memory systems normally consisting of large magnetic disk storage units with a capacity of over 30GB are required for this purpose. The data access speed from the disks should be of the order of 100 MB per second.

Cray YMP provides one more level of secondary storage which is a random access storage similar to the main memory but which uses cheaper and slower solid state storage system with a capacity of upto 512 Mwords and a data transfer rate of 128Mwords per second. This transfer rate is significantly faster than electro-mechanical magnetic disk units.

Manufacturers of most mainframe computers provide a much larger address space than actual physical main memory capacity. Many computers provide address length of 32 bits which gives a potential

memory capacity of 4 Gwords. Programs may be written assuming that 4G words of memory space is available. The physical main memory is, however, limited to tens of hundred megawords due to cost and technological limitations. The programs and data are distributed between the main memory and the secondary memory. Moving the relevant portion of a program and data needed during execution of the program to the main memory from the secondary memory and taking it back to the secondary memory is managed by the operating system of the computers. This is called a *virtual memory system.* The operating system divides a program and data into discrete blocks known as *pages* and brings only those pages into the main memory which are currently needed. As soon as they are used, they are sent to the secondary store and new pages which are needed immediately are brought to the main memory. The system will work very efficiently as long as all the pages of data and program needed during execution are available in the main storage. Whenever the pages that are needed are not in the main memory and have to be fetched from the secondary memory there is a delay and consequently the program takes longer to execute. Virtual memory is a convenience to programmers but entail overheads and also possibility of inefficient execution.Some manufacturers such as CDC and NEC of Japan provide a virtual memory system in their supercomputers. Cray computers, however, never provided a virtual memory system as Cray designers were convinced that virtual memory's disadvantages outweigh its advantages. They try to provide as large a main memory as possible within the technological constraints prevailing at a given time. Early models of Cray computers had only 4 Mwords of main memory. Current models have increased the capacity of the main memory to 256 Mwords.

The requirement of secondary storage, in particular, high speed disk, has rapidly been increasing. As the speed of a machine increases the volume of data needed by applications also increases. Currently a balanced configuration of a supercomputer would require anywhere between 25 to 100 GB of high speed disk storage.

Frontend Computer System: Supercomputers are designed to be excellent number crunchers. They are most efficiently operated in a non-interactive mode. A set of user programs are queued up as a batch and one user's program at a time is taken up for execution. From the point of view of the user, however, this is not desirable, particularly during program development phase. In this phase programs are tested, modified and optimized, and a user finds it desirable to have an interactive communication with the computer. It will be uneconomical to allow a single user to use a supercomputer interactively as human response time is much slower than that of a supercomputer. Thus a large number of users time share the computing facility and a fair amount of computer's resources are required to manage multiple users. The high speed arithmetic facility of a supercomputer is not needed for this management function. In fact, it will be a misuse of a supercomputer if it spends a lot of time to manage users' resource allocation problems. This resource management job can be done by a cheaper machine attached to the supercomputer. Such a computer is called a *frontend computer.* Most of the program development and debugging are done using the frontend computer in a time-shared mode. Production programs are transferred from the frontend machine to the supercomputers' secondary store from where the supercomputer takes one program after another for execution. All slow peripherals such as printers and plotters are also connected to the frontend computer.

A limited interaction with the supercomputer is normally allowed with a separate frontend interface to which a graphics terminal is connected. This is provided for visualization, animation etc. Such use is however very expensive.

The frontend connected to a supercomputer is normally a large main frame computer with large volume disk storage (around 30 GB), tapes, printers, plotters etc. The Cray series machines normally use an IBM mainframe or CDC's Cyber mainframe or Amdahl's mainframe or a DEC-VAX mainframe computer as the frontend.

2.4 Technology of Vector Supercomputers

Vector supercomputers are very expensive costing millions of dollars and are manufactured only by four companies in the world today - Cray in USA, and Fujitsu, Hitachi and NEC in Japan. One of the main reasons for their high cost is the difficulty in fabricating electronic circuits to the exacting standards and high speeds demanded by supercomputers. In supercomputers, the basic clock speeds are around 6 nseconds. The time taken by electricity to travel 10 cms in a wire is 0.33 nsec. Distance between components must thus be minimized so that wire lengths are small. Seymour Cray who designed and built the world's first supercomputers built them in a cylindrical shape to reduce wire lengths. (In fact, a sphere would have been ideal as it has the minimum surface area for the volume enclosed. However, fabricating a computer as a sphere is difficult). To reduce wire lengths electronic circuits have to be densely packed. Dense packing makes it difficult for electronic circuits to dissipate heat. The heat generated in the electronic circuits switching at nanosecond rates is quite high. For example, assume that a switch dissipates 100 microwatts. If an integrated circuit package has 100,000 switches it will dissipate 10 watts. If there are 10,000 packages in 1 m^3 100 Kilowatts will be dissipated. If the system is not cooled, the circuits will burn. Thus it is necessary to cool the system efficiently to maintain the integrated circuits at a constant temperature of around $20°C$. Thus Cray YMP requires special cooling systems with copper pipes carrying ice cold water on which the electronic circuits are mounted. In fact Seymour Cray is reputed to have said that supercomputer designers are glorified plumbers! Many methods are used for cooling. The simplest one is to blow cold air between the circuits. This method removes moderate amounts of heat and is not sufficient for machines such as Cray YMP built in the 80s. The next method is to run copper pipes carrying cold water on which the semiconductor integrated circuits are mounted. This method is used in some of the supercomputers such as the one made by NEC of Japan. Instead of water, freon refrigerant is passed through pipes in Cray YMP. Cray 2 uses an ingenious technique to dissipate heat. This computer's CPU

is immersed in liquid fluon (a flouro carbon) which is an electrical insulator but a good thermal conductor. Fluon is circulated through a heat-exchanger and the temperature of the fluid is kept constant at $72°F$. As all the semiconductor components are in contact with the fluid there is good heat transfer and there is no need for copper tube plumbing as in Cray YMP system.

Yet another cooling technique was tried by ETA 10 systems. The entire CPU board of ETA 10 supercomputer was immersed in a liquid nitrogen bath. This method, besides removing the heat emitted by the circuits, kept the semiconductor at a low temperature improving its switching speed.

The semiconductor circuits in most of the supercomputers used silicon. Earlier systems used high speed bipolar switches which emitted a lot of heat. Later they were replaced by Complementary Metal Oxide Silicon (CMOS) devices. These dissipate less heat compared to bipolar circuits but are slower. Recent Bipolar CMOS circuits switch at high speed with low power dissipation. The dominance of silicon as a semiconductor is being currently challenged by Gallium Arsenide (GaAs) semiconductor devices. GaAs has a higher switching speed compared to silicon. However, it is a brittle material and integrated circuits with large number of GaAs devices are just being commercially mass produced. It is expected that the next generation of Cray supercomputers named Cray 3 would use GaAs semiconductor switching devices. Convex computers, a manufacturer of parallel computers, has already delivered their new generation machines using GaAs technology. The GaAs based supercomputers are expected to use a faster 2 nanosecond clock compared to the 4 nanosecond clock used by the fastest supercomputers available now (1992).

3

Computing with Vector Supercomputers

In the last chapter we saw that vector supercomputers attain their high speed of computing by using pipelined functional units. Further we saw that in order to efficiently use such pipelined functional units we need a long sequence of operands or operand pairs on which the function is applied. Thus an application program will be able to use the potential speed offered by a supercomputer only if it has long sequence of operands called *vectors* to be processed. For example, a Fortran program using the loop

```
      DO 100 I = 1, 80
         C(I) = A(I) + B(I)
100 CONTINUE
```

can use the pipeline capability effectively as 80 components of vector A are added to 80 components of vector B. In contrast, the program

```
      DO 50 I = 1,5
         P(I) = Q (I) + R(I)
 50 CONTINUE
```

cannot effectively use the pipeline capability as the length of the vectors is small compared to the number of stages in the pipeline.

Given a Fortran program it may often be possible to transform it either manually or automatically to an equivalent Fortran program which is capable of using the pipeline capability efficiently. Such a transformation is called *Vectorization.* In this chapter we will explain first the basic ideas used in vectorization. We will also discuss the

general question of optimizing programs to reduce their total execution time.

3.1 Vector Instructions

A vector operand is an ordered set of n elements, where n is called the length of the vector. Each element in a vector is a scalar quantity which may be an integer, a real number, a character or a bit. There are four types of vector operations [3]:

Type 1: f_1 : $V \longrightarrow V$
Type 2: f_2 : $V \longrightarrow S$
Type 3: f_3 : V (*operator*) $V \longrightarrow V$
Type 4: f_4 : V (*operator*) $S \longrightarrow V$

where V is a vector and S is a scalar.

Table 3.1. Vector Operations

(X, Y and Z are n component vectors)

Vector operation type		*Vector operation carried out*
Type 1	Vector square root	$Y = \text{SQRT}(X)$
	Absolute value	$Y = \text{ABS}\ (X)$
	Vector Sine	$Y = \text{SIN}\ (X)$
Type 2	Vector minimum	$Min(Y_1\ Y_2, \ldots, Y_n) = Y_p$
	Sum of components	$\sum_{i=1}^{n} y_i$ where $Y = (y_1, y_2, \ldots, y_n)$
Type 3	Vector add	$Z = X + Y$
	Vector multiply	$Z = X * Y$
	Vector OR	$Z = X(OR)Y$
Type 4	Add scalar to vector	$Z = X + a$ $z_1 = x_1 + a,$ $z_2 = x_2 + a,$ $\vdots$ $z_n = x_n + a$

Type 1 and Type 2 are unary operations whereas Type 3 and Type 4 are binary operations. Some examples of these types of vector

operations are given in Table 3.1. These types of operations are implemented in supercomputers using pipelined units and there are hardware instructions to carry them out. In Fig. 3.1 we show how these operations are implemented using pipelined units.

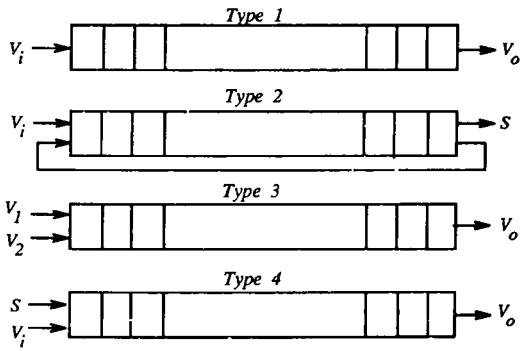


Fig 3.1. Pipelines for four types of vector instructions

As vector supercomputers have built-in hardware to operate on vector operands efficiently it is essential while programming these machines to use vector operands and operators wherever it is possible. Consider, for example,

```
      DO 100 I = 1, 50
         A(I) = B(I) + D(I)
         C(I) = K * P(I+5)
100   CONTINUE
```

The above program may be rewritten using vector operands and vector instructions as:

```
A(1:50) = B(1:50) + D(1:50)
C(1:50) = K * P(6:55)
```

The following program:

```
DO 25 I = 1,N
```

```
      DO 25 J = 1,100
         P(J) = Q(J)* T(J,I) + C(I)
   25 CONTINUE
```

may be replaced by:

```
      DO 25     I = 1, N
        P(1:100) = Q(1:100)*T(1:100,I)+C(I)
   25 CONTINUE
```

The program using vector operands use the operators +, * etc. as operations on vector operands. When the program is compiled, instructions will be issued to the pipelined arithmetic units to perform the operations. The main point is to write programs which use the hardware pipeline features to the maximum extent possible. If a user wants to execute an existing program on a computer with vector pipelines, he/she should convert them to a form which effectively uses the vector facilities of the computer. This conversion is done automatically by a program called a *vectorizer*. The vectorization process and the function of the vectorizer will be discussed at greater length in sections 3.3 and 3.4.

3.2 Vectorization of Programs

If we examine a typical supercomputer the speed of electronic circuits for scalar processing is around a tenth of the speed attainable with vector processing. In other words, if an operation such as multiply takes t_s seconds to perform in the scalar mode then the time taken to perform the same operation in vector mode t_v is t_s/n where n is around 10. Suppose a program is executed on a scalar computer and takes time T. (See Fig. 3.2). Assume that $T = T_s + T_v$ where T_s is the time taken to execute the parts of the program which uses scalar operations and T_v is the time taken by the vectorizable part of the program. If the program is run on a supercomputer with vector capability and n is the speedup due to vector processing then the time taken to execute the program is:

Time taken $= T_s + (T_v/n)$

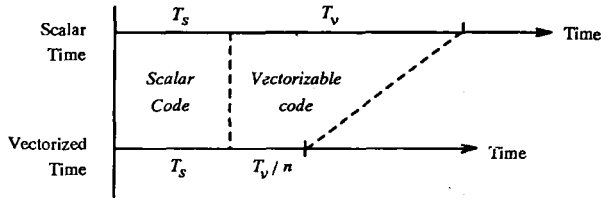


Fig 3.2. Scalar and vectorized time comparison

The speedup obtained by using the supercomputer is:

$$\text{Speedup} = (T_s + T_v)/(T_s + (T_v/n))$$

Let us define *Vectorization ratio* as:

$v = T_v/(T_s + T_v)$

$$\text{Speedup} = S_p(v,n) = n(T_s + T_v)/(nT_s + T_v)$$
$$= n/(n(1-v) + v)$$

If the code is fully vectorizable, ie. $T_s = 0$ then $v = 1$ and speedup $S_p(v,n) = n$ (100% speedup). If there is no vectorizable code in the program then $v = 0$. In this case there is no speedup as $S_p(v,n) = 1$. $S_p(v,n)$ is plotted for $n = 10$, and various vectorization ratios in Fig. 3.3. Observe that the improvement in speed increases rapidly beyond $v = 0.9$. When vectorization is less than 0.8 the improvement in performance is relatively low.

Consider a computer with a vector megaflop rating of 100. Assume its scalar speed is 10 megaflops with 0.8 vectorization ratio. The speed up is (10/2.8). Thus the effective megaflop of the machine is $10 \times (10/2.8) = 35.7$megaflops. If the vectorization ratio is 0.975 the effective megaflop is 83.8. From this calculation it is clear that a supercomputer will operate at reasonable speed only if the vectorization ratio is very high. In other words the vectorizable part of the program must take much longer time to execute compared to the part which is not vectorizable.

Most programs will have a scalar part. The scalar part cannot be entirely eliminated and around 10 to 15% of the code will be

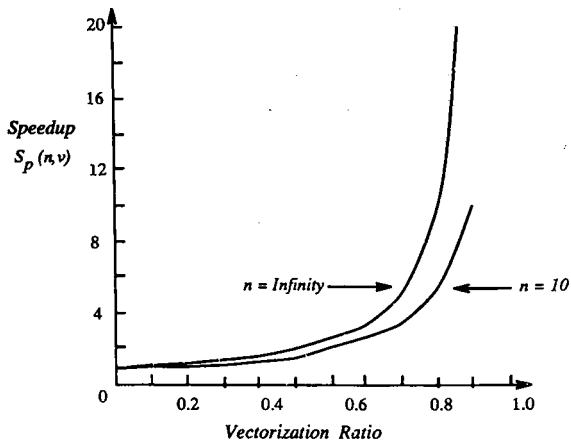


Fig 3.3. Speedup for various vectorization ratios

scalar code. Thus good performance cannot be achieved by merely increasing the vector speed without increasing the scalar speed. This is commonly known as Amdahl's law which informally[1] states:

> "When a computer has two distinct modes of operation, a high speed mode and a low speed mode, the overall speed is dominated by the low speed mode unless the low speed mode can be totally eliminated".

Amdahl's law is somewhat pessimistic as it is usually not possible to completely eliminate scalar processing. The redeeming feature is that in many programs 10 to 15% of the code consumes 90% of the execution time. This 10 to 15% of the code can be often vectorized to get a vectorization ratio exceeding 0.95. Thus the overall program execution time is reduced. If a program takes 10 minutes to execute, and out of this 10% of the code which takes 9 minutes can be vectorized to 98% then the total time taken would be 1 + (9/9.8) or 1.918 minutes if the ideal speedup is 10. Even though the entire code may not be vectorizable to 98% the effective time reduction is substantial.

3.3 What is Vectorization?

When a new application program is developed for a vector supercomputer, the algorithm chosen must be such that it fully exploits the vector facility provided by the hardware. The programming language chosen must have language features to express vectors and operations with vector operands. The new standard evolved for Fortran, called Fortran 90, has recognized this need and allows vector operands and operations. A compiler for Fortran 90 will compile a Fortran 90 program to a machine language program which uses vector machine instructions available in these computers.

This situation of writing a new program for an application is more an exception. Usually scientists use programs supplied by others for their applications. These programs are written most often in Fortran for mainframe computers which have no vector processing capability. These programs must be modified to use the vector processing capabilities of supercomputers if they are to run faster. This process of modifying a program is know as *vectorization* of the program. Vectorization of programs can be either *automatic* or *manual.* In automatic vectorization a program called a *vectorizer* supplied by the manufacturer of the supercomputer is used to translate the Fortran program to one which uses vector instructions of the computer. In manual vectorization a programmer examines the program and decides which parts are vectorizable and rewrites those parts. Manual vectorization is difficult and slow and is done only when automatic vectorization is not effective.

3.4 The Vectorization Process

The questions we will now ask are "how is vectorization performed?" and "when is vectorization effective?". The primary idea in vectorization is to recognize DO loops in Fortran programs in which all the component values of the vector operands are readily available to be streamed through the pipelined functional units of the supercomputer. These DO loops are replaced by equivalent vector instructions. For example, the loop

```
      DO 25 I = 1,100
         A(I) = B(I) + C(I)
         D(I) = A(I) * 2
   25 CONTINUE
```

is vectorized and expressed in Fortran 90 array notation as:

```
A(1:100) = B(1:100) + C(1:100)
D(1:100) = A(1:100) * 2
```

Observe that all the values of vector A are calculated by the first vector instruction and are properly used in the second vector instruction without altering the meaning of the loop (See Fig 3.4). Consider

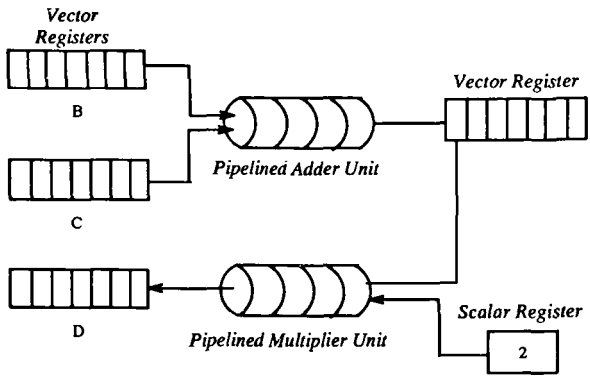


Fig 3.4. Usage of successive vector instructions

the loop:

```
      DO 20 I = 1, N
         B(I) = C(I) * 2
20       D(I) = B(I+1) + A(I)
```

A naive vectorization of the above loop as

```
B(1:N) = C(1:N) * 2
D(1:N) = B(2:N+1) + A(1:N)
```

will be incorrect as the vector calculation will yield

```
B(1) = C(1)* 2
B(2) = C(2)* 2
B(3) = C(3)* 2
     .
     .
     .
B(N) = C(N)* 2
D(1) = B(2) + A(1) = C(2)*2 + A(1)
D(2) = B(3) + A(2) = C(3)*2 + A(2)
     .
     .
     .
D(N) = B(N+1)+A(N) = C(N+1)*2+A(N)
```

whereas the following was intended:

```
B(1) = C(1)* 2
D(1) = B(2)+A(1)
B(2) = C(2)* 2
D(2) = B(3)+ A(2)
     .
     .
     .
D(N) = B(N+1)+ A(N)
```

The correct vectorized code is:

```
D(1:N) = B(2:N+1) + A(1:N)
B(1:N) = C(1:N) * 2
```

which preserves the meaning of the original loop. Some more examples of vectorization are given below.

Consider the loop:

```
DO 50 I = 1,100
   A(I) = B(I) * C(I)
   D(I) = DULT(A(I) * X(I))
```

```
      E(I) = D(I)/B(I) + A(I)
50 CONTINUE
```

Assume that the function *DULT* is not a vectorizable function (i.e., it cannot be computed by using a pipeline). The vectorized code is:

```
   A(1:100) = B(1:100) * C(1:100)
   DO 50  I = 1,100
       D(I) = DULT(A(I) * X(I))
50 CONTINUE
   E(1:100) = D(1:100)/B(1:100)+A(1:100)
```

The following loop is not vectorizable:

```
   DO 90 I = 2,50
      A(I) = B(I-1)
      B(I) = C(I)
90 CONTINUE
```

The actual assignments by this loop are:

```
   A(2) = B(1)
   B(2) = C(2)
   A(3) = B(2)
   B(3) = C(3)
        .
        .
        .
   A(50)= B(49)
   B(50)= C(50)
```

If we write a vectorized code as:

```
   A(2:50) = B(1:49)
   B(1:50) = C(1:50)
```

the computer will first assign

```
   A(2) = B(1)
   A(3) = B(2)
```

```
        .
A(50)= B(49)
```

and then assign

```
B(1) = C(1)
B(2) = C(2)
        .
        .
        .
B(50)= C(50)
```

Thus vector A will have old values of vector B and not its updated values. This error has occurred due to *data dependency* in the loop. This data dependency may be eliminated manually by rewriting the code as:

```
A(2) = B(1)
B(2:50) = C(2:50)
A(3:50) = B(2:49)
```

The code given above is vectorized. Some intelligent vectorizers do this vectorization automatically. The loop

```
      DO 150 I = 2,100
150     A(I) = A(I-1) + P
```

is not vectorizable as values of components of A are values of previous components which are calculated sequentially. This is called a *recursive* DO loop and is not vectorizable by automatic vectorizers. A programmer may, however, observe that when the loop is expanded one gets:

```
A(2) = A(1)+P
A(3) = A(2)+P = A(1)+2*P
A(4) = A(3)+P = A(1)+3*P
```

and rewrite the program as:

```
      DO 150 I = 2, 100
150     A(I) = A(1)+P*(I-1)
```

which is vectorizable.

All possible vectorizations which preserve the meaning of the original Fortran program are automatically performed by the vectorizer supplied by a manufacturer. Part of Fortran programs which have dependent or recursive DO loops are highlighted by the automatic vectorizer. This information is valuable to a user to enable him/her to alter the code if appropriate to facilitate vectorization.

Program Profiler: In practice many scientists and engineers use programs written by others for an application. Many of these programs are written in Fortran and have more than 10,00,000 statements of Fortran code. It is impossible to manually go through such programs, understand them and vectorize them. Computer aids are thus necessary to locate the subroutines in the program which take most of the execution time. As was pointed out earlier 90% of the execution time of many programs is spent in 10% of the source code. It is thus worthwhile locating this 10% of the source code. Supercomputer manufacturers supply a program called a *profiler* to assist in this effort. The profiler adds instructions to the program for measuring the processing time taken by each subroutine in the program, how many times each subroutine is called and the percentage of the total execution time spent by the program in each subroutine. Thus the practical method used to vectorize programs is to start with the source program and execute it on the supercomputer along with the profiler program. A typical output of a profiler is shown in Table 3.2 It is seen from Table 3.2 that the subroutines *funx* and *caisph* account for 94.1% of the total execution time of the program. Thus reducing the execution time of these two subroutines would give the greatest benefit for the least effort.

After getting the execution profile of the source program, the program is recompiled with the vectorizer turned on for vectorizing the entire program. Even though it is beneficial to vectorize only those subroutines which take the largest percentage of execution time, the program is run once fully vectorized to compare the scalar and vector times of each subroutine. After vectorizing the program the profiler

Table 3.2. Profile of a Fortran Program (Unvectorized)

Subroutine name	Percentage time spent in subroutine	Time spent in subroutine (sec)	No.of subroutine invocations
funx	69.8	167.15	100
calsph	24.3	58.25	100
pion	4.8	11.43	305
valcal	0.5	1.3	100
molwt	0.2	0.5	100
kroot	0.2	0.4	1
picky	0.2	0.45	1
Total	100%	239.48	-

is re-run to obtain the new profile. It is shown in Table 3.3. It is interesting to note that vectorization has reduced the execution time of the first three routines: *funx* runs seven times faster after vectorization, *calsph* runs about two and a half times faster and *pion* about five times faster. In contrast routines *valcal*, *molwt*, *krost* and *picky* run slower with vectorization. Although vectorization yields very good time reduction in a number of cases, there are cases where the overhead of vectorizing short loops can cause these loops to execute slower than when scalars are used.

Table 3.3. Profile of a Fortran Program (Vectorized)

Subroutine name	Percent time spent in subroutine	Time spent in subroutine	No.of subroutine invocations
funx	44.7	24.25	100
calsph	42.6	23.12	100
pion	4.1	2.21	305
valcal	3.0	1.61	100
molwt	2.8	1.53	100
kroot	1.2	0.67	1
picky	1.6	0.85	1
Total	100	54.24	-

For such short loops a method known as *loop unrolling* would lead to faster execution. For example the DO loop

```
      DO 25  I = 1,4
25       A(I) = B(I)
```

could be written as:

```
A(1) = B(1)
A(2) = B(2)
A(3) = B(3)
A(4) = B(4)
```

The code given above is said to be "unrolled". Unrolling eliminates the overhead of incrementing the index in each iteration and comparing its value with the final value of the index for loop termination.

In the example being considered, the program is recompiled again after vectorizing only the first three subroutines. If further time reduction is desired one may look in detail at the subroutines *funx* and *calsph* and see if any loops have been left unvectorized by the automatic system. To look at the detailed execution of loops another profiler known as *loop profiler* is used. This profiler creates a table similar to Table 3.2 for each of the loops in a subroutine. Such a profiler also highlights those loops which have been automatically vectorized and those loops which have not been vectorized. The user can attempt to vectorize such loops manually by restructuring the code.

As stated earlier it is most often found in programs that 90% of the execution time of a program is spent in 10% of the code. Thus it is more cost effective to spend one's time in manually vectorizing this 10% of the code. The reasons are:

- The largest speedup is possible by concentrating on this part of the code.
- When a small part of a program is manually rewritten there is less chance of introducing errors in the program.

- As only a small part of the program is restructured the new program is close to the original program and is thus easy to maintain.
- One may look deeper into improving the algorithm for a small part of the overall program to tune it to a vector computer.

Manual vectorization requires careful examination of dependencies in loops, the type of subscript expressions in loops and understanding the manner in which memory banks are accessed by multidimensional arrays. Discussion on these points is beyond the scope of this book and the interested reader is referred to [1] and [2] given at the end of this book.

Subroutine in-lining: Another technique for reducing the execution time of programs is called *subroutine in-lining*. This is replacing a subroutine or function call in a calling routine by the actual code of the called subroutine. Consider the following simple example:

```
      DO 200 I = 1,100
      CALL INIT(P(I))
      CALL PAL(X(I),Y(I),Z(I),P(I))
200   CONTINUE
        .
        .
        .
      SUBROUTINE INIT(A)
        A = 1.0
        RETURN
      END
      SUBROUTINE PAL(A,B,C,D)
        D = D + A * B + C * C
        RETURN
      END
```

The above code may be replaced by

```
      DO 200 J = 1,100
        P(J) = 1.0
```

```
            P(J) = P(J)+X(J)*Y(J)+Z(J)*Z(J)
200 CONTINUE
```

The main reasons for subroutine in-lining in the above example are the reduction in the overhead of subroutine calling in each loop and the possibility of vectorization with this transformation. If the subroutine is invoked from within a loop, vectorization of the loop is inhibited.

The subroutine calling overhead is computer-dependent. It is normally of the order of tens of microseconds. In-lining of subroutine code is justifiable if the number of calls to the subroutine is (as seen from the profiler information of Table 3.3) of the order of hundreds of thousands. It is also justifiable if the profiler shows that the subroutine is taking a large proportion of computer's execution time and by in-lining vectorization becomes feasible.

The role of Fortran: We have been extensively using Fortran language in our examples so far. The main reason is that inspite of the emergence of many improved programming languages such as Pascal and C, Fortran is still the most popular language among scientists and engineers. The reasons for this are many. Fortran compilers are very efficient; many programs for common applications have been written in Fortran and are widely available; most scientists have learnt Fortran as their first language to write programs and are reluctant to change unless there is an immense advantage in doing so. Fortran is a simple language to learn and use and finally many good books are available to learn Fortran. As new languages are invented and come into vogue the demise of Fortran is predicted but Fortran continues to thrive. Periodically Fortran has been improved and standardized. Fortran 77 was standardized by the International Standards Organization and more recently a new standard called Fortran 90 has emerged in 1990. The new standard has absorbed some of the good qualities of more recent languages such as Pascal while retaining compatibility with earlier Fortran compilers. Besides this, Fortran has recognized the emergence of supercomputers and has enhanced Fortran syntax and semantics to include constructs to facilitate vector

processing. Thus we note that any one who wants to use a supercomputer has to be well conversant with programming in Fortran primarily because of the availability of:

1. A large number of application programs written in Fortran
2. Efficient compilers and optimizers for Fortran particularly for supercomputers and
3. Availabillity of special constructs in Fortran 90 to operate on vector operands.
4. Many of the new languages such as C have advanced features such as pointers which inhibit vectorization. Codes written in these languages to exploit vector hardware may end up like Fortran using simple *for* loops and array data structures.

3.5 Scalar Optimization of Programs

Besides vectorization, it is necessary to optimize the non-vector part of a Fortran program to reduce its execution time. Due to many advancements in knowledge about languages and compilers, automatic optimizing compilers have improved considerably over the years. Supercomputer manufacturers are particularly sensitive to this and improve their optimizers regularly. Inspite of this, it is essential for a serious Fortran programmer who wishes to reduce execution time on a supercomputer to follow some simple rules which assist the optimizing compilers to recognize optimization opportunities. Some rules which are effective and found in optimizing compilers are:

- Evaluating constant expressions
- Moving code independent of loop index out of a loop
- Eliminating common subexpressions
- Eliminating unnecessary store statements.

Numerous examples of such optimization and an extensive discussion of this topic is outside the scope of this book and the interested readers may refer to the books [1] and [2] in the bibliography.

Besides vectorizers and optimizing compilers, supercomputer manufacturers also provide libraries of commonly used mathematical procedures such as matrix inversion, eigenvalue calculation, curve fitting etc. These libraries are vectorized and optimized to work efficiently on the manufacturer's machine and are sometimes written using assembly language. If the examination of a program profiler's output shows that a large proportion of the program's execution time is taken by the subroutines written by the user for such common mathematical procedures, then the user written procedures should be replaced by the manufacturer's optimized subroutines and linked to the rest of the program. In some cases this method reduces execution time of a program by a factor of four! Thus manufacturers' routines should be used whenever they lead to significant reduction in computing time.

In conclusion we note that in order to attain the high speed of computation promised by supercomputers it is essential to vectorize and optimize the program. It is not possible to fully automate the vectorization process and some manual vectorization may be required. A profiler should be used to find the parts of the code which consume most of the execution time and devote manual vectorization and optimization effort to these parts of the code. These parts are normally quite small in terms of number of lines of code and thus this effort would not be enormous. These methods are used to get the best out of existing programs. When new applications are developed for vector supercomputers the algorithms should be matched to the architecture. Vector operands and operations should be used extensively.

4

Parallel Computers

We saw that vector supercomputers use pipelining as the method of exploiting temporal parallelism in solving problems. Apart from temporal parallelism, data parallelism is inherent in solving many problems and can be easily exploited to increase the speed of computing. It is also possible to combine both types of parallelism to further increase the speed of a computer and this is done in most supercomputers marketed now. In this chapter we will discuss how parallelism in algorithms is exploited in building parallel supercomputers. We will also look at a generalized structure of parallel computers. Finally we will compare vector computers with parallel computers.

4.1 Array Processors

In a pipelined (vector) computer, two vectors $(a_1, a_2, a_3 \ldots a_n)$ and $(b_1, b_2, b_3, \ldots, b_n)$ are added by streaming in pairs of operands through a pipelined adder. The adder is designed with several stages or segments, each segment performing a specialized operation on a different pair of input operands. If a pipelined adder has, say, four stages and the pipeline is full then four pairs of operands will be in different stages of addition in the pipeline (see Fig.2.3). If each stage in the pipeline takes T seconds to do its work, then the first sum takes $4T$ seconds but subsequent sums take T seconds each. As we saw in chapter 2, pairs of operands are streamed through the pipeline, one pair of operands entering once every T seconds. This method of designing an adder is said to exploit temporal parallelism. If we write a DO loop for addition

```
      DO  25 I = 1, 100
25       C(I) = A(I) + B(I)
```

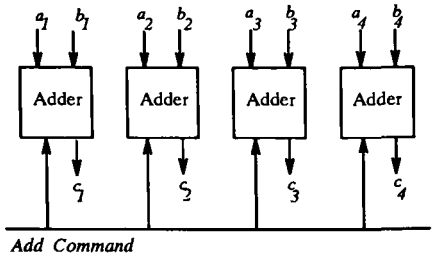


Fig 4.1. An array adder

then we can think of the index I as representing time and the vectors $A(I)$ and $B(I)$ as being sequentially streamed through the adder pipeline.

Another method of adding vectors is to have an array of n adders and to distribute one pair of operands $(a_1, b_1), (a_2, b_2) \ldots (a_n, b_n)$ to each adder. An *add* instruction may then be broadcast to all the adders. All the adders then simultaneously start adding the operand pair assigned to them (Fig. 4.1). If each adder takes T seconds to add then the time to add a vector is T seconds if we ignore the time taken to distribute the operands to the adders. Such an organization of adders is called an *array adder*. We can generalize the system and arrange a set of Processing Elements (PEs) with each PE capable of performing not only addition but other operations such as multiplication, division, exponentiation etc. Such an organization of PEs is called an *Array Processor*.

An array processor uses *data parallelism*. In the vector addition example a single instruction, namely ADD, is performed simultaneously on multiple data items, namely, pairs of components of two vectors. This type of computer is thus called a Single Instruction Multiple Data (SIMD for short) type computer.

If we write a DO loop for addition

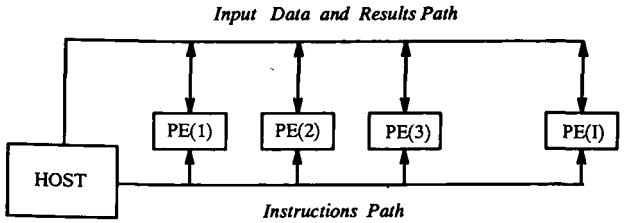


Fig 4.2. An array of processors for data parallel computation

```
      DO  25 I = 1, 100
          C(I) = A(I) + B(I)
   25 CONTINUE
```

then we can think of the index I as representing a PE number and each component of the vectors A(I) and B(I) as being added in parallel in the respective PE(I). In fact the DO loop may be written to explain this point as:

```
DO ACROSS  PE(I)
   FOR I = 1, 100
   C(I) = A(I) + B(I)
END DO
```

An array processor (see Fig 4.2) is normally attached to a host computer. The host computer stores the program to be executed. Components of the data arrays to be processed are first despatched to each of the PEs. The host computer then broadcasts the instructions to operate on components of the array to all the PEs. All the PEs independently and simultaneously perform the arithmetic operation on the data arrays stored in their respective *private* data memory. The results are then sent back to the host by each of the PEs. These are combined and output by the host. (Each PE in the array can itself use pipelining for performing arithmetic operations. In such a case the vectors to be processed should be very long so that each PE in the array is sent a vector of reasonable length to process).

The idea used in an array processor can be extended to design a computer which efficiently executes data parallel algorithms. In this structure a set of identical Computing Elements (CEs for short) are connected to a host computer. A computing element consists of a CPU and a private memory which stores both data and instructions. The host compiles a program to be executed and stores one copy of the object code in the memory of each CE. The data set is partitioned equally and one partition is stored in the memory of each CE. On a command from the host all CEs execute simultaneously the program stored in their respective memories using the data stored in each CE. The results are then output by the individual CEs or by the host. This method of organizing a parallel computer is called a Single Program Multiple Data (SPMD) architecture.

SPMD mode of working is of great practical interest as many problems can be easily partitioned to use data parallelism. Some examples are:

- Averaging a set of n numbers: If there are k CEs, (n/k) numbers can be allocated to each CE. Each CE averages the data allocated to it. The CEs work simultaneously and independently. The average computed by each CE is returned to the host. The host computes the average of the averages received by it and outputs the result. This method will be effective if the computing time taken by each CE is much larger than the time required by each CE to receive the data and program from the host.

  One interesting side-effect of data parallel computing is the reduction in rounding errors. As the number of arithmetic operations performed by each CE is only (n/k) the rounding error is reduced compared to the case in which one CE does all n arithmetic operations.

- Integrating a function $f(x)$: In this case the region between A and B (See Fig 4.3) is divided into k strips with each strip having width $(B - A)/k$. The function $f(x)$ and the beginning and end x-coordinates of a strip to be integrated by a CE

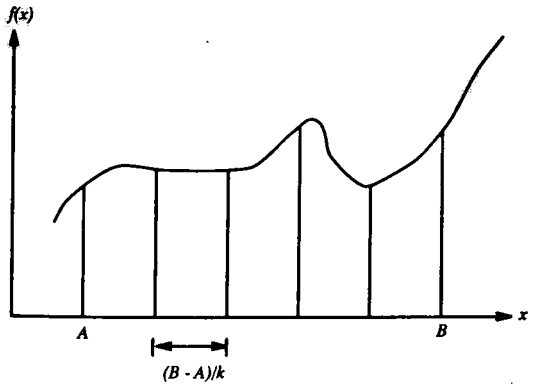


Fig 4.3. Parallel computing of an integral

are despatched to each of the k CEs. Each CE integrates the portion of the function allocated to it. All the CEs work concurrently and independently. Each CE then sends the integral calculated by it to the host which adds all these integrals and outputs the result. The same idea can also be used to compute surface and volume integrals.

- Generating a frequency table for a given data set: Here again the data set can be partitioned into k sets and distributed to the k CEs. Each CE then computes a frequency table with the data set allocated to it. These tables are then gathered by the host, consolidated and printed.
- Sorting a list of names: If a list of names is to be sorted, the list may be partitioned into several sublists and each CE may be given a sublist to sort. The sorted sublists are then sent to the host by each CE. The host then merges these sublists into one sorted list.

The efficiency of data parallel computing with an array of processors depends on the time taken for computation in comparison with the time taken to distribute data to the processors. Let the time

taken to compute a job on a single processor be T. Assume that the data is partitioned and distributed to k processors. Let T_s be the time taken to send data to the processors. Let each processor take T/k seconds computing time. Let T_r be the time taken by the host to receive the results. Then the speedup due to parallel processing is:

$$\text{Speedup} = T/((T/k) + T_s + T_r)$$

When $(T/k) >> (T_s + T_r)$ the speedup is nearly ideal, i.e., k.

4.2 Executing Task Graphs in Parallel Computers

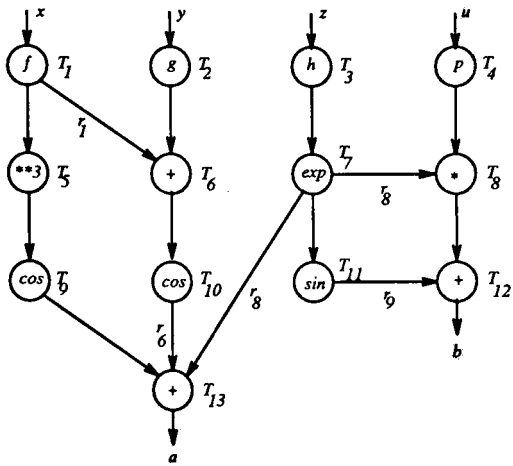


Fig 4.4. Task graph for a complex computation

In the last section we looked at problems which could be broken up into a number of independent tasks and carried out in parallel. In general, problems cannot be broken up into independent tasks. Consider, for example, the following computations:

$$a = \cos(f^3(x)) + \cos(f(x) + g(y)) + e^{h(z)}$$

Table 4.1: Time for each task in a task graph

Task	T_1	T_2	T_3	T_4	T_5	T_6	T_7
Time	3	4	5	4	3	1	3
Task	T_8	T_9	T_{10}	T_{11}	T_{12}	T_{13}	-
Time	2	4	4	4	1	2	-

$b = \sin(e^{h(z)}) + p(u) * e^{h(z)}$

The computations above can be depicted by the *task graph* of Fig. 4.4. The task graph clearly shows the sequence of execution of tasks. Each circle in the task graph represents a task. For example, T_1 is the task of computing $f(x)$. A line with an arrow connecting two circles shows the dependency between tasks. The direction of the arrow shows precedence. In Fig 4.4, task T_6 can be done only after tasks T_1 and T_2 are done. On the other hand tasks T_1, T_2, T_3 and T_4 can be done independent of one another and can be carried out in parallel. Task T_{13} can be done only after tasks T_9, T_{10}, and T_7 are completed. Tasks T_1, T_5, T_9 and T_{13} have to be performed sequentially one after another.

We will now examine how a and b are calculated in parallel, if 4 CEs are available. We will assume that the CEs are identical. In Table 4.1 the time necessary to execute each task is given.

The minimum time in which all the tasks can be completed by 4 CEs working in parallel (if there is no dependency between tasks) is given by:

Minimum time = Sum of execution time of Tasks T_1 to T_{13}/4
= 40 / 4
= 10

The above estimate is an optimistic estimate as it does not take into account the constraints placed by the task graph on the sequencing of tasks. If we inspect the graph we find that the earliest time at which a can be computed is 12 units of time and the earliest time b can be computed is 13 units of time. The tasks T_1 to T_{13} may be

scheduled on 4 CEs as shown in Fig. 4.5. The assignment shown in this figure is optimal.

Programs to compute a and b for the 4 CEs may be written as below:

Program for CE1

do T_1
Send r_1 *to* CE2
do T_5
do T_9
get r_6 *from* CE2
get r_8 *from* CE3
do T_{13}
Write a
Stop

Program CE2

do T_2
get r_1 *from* CE1
do T_6
do T_{10}
Send r_6 *to* CE1
Stop

Program from CE3

do T_3
do T_7
Send r_8 *to* CE4
Send r_8 *to* CE1
do T_{11}
Send r_9 *to* CE4
Stop

Program for CE4

do T_4
get r_8 *from* CE3
do T_8
get r_9 *from* CE3
do T_{12}
Write b
Stop

Observe that as soon as the Program for CE1 performs task T_1, the result r_1 is sent to CE2. Thus there should be a data path between CE1 and CE2. Further, as soon as r_1 is sent by CE1 it can proceed with the rest of its work and perform tasks T_5 and T_9. When these tasks are done, it cannot proceed further as it needs the results r_6 and r_8 from CE2 and CE3 respectively to do task T_{13}. Data paths are needed from CE2 and CE3 respectively to receive r_6 and r_8. CE1 issues commands to get r_6 and r_8 from CE2 and CE3 respectively and waits till the corresponding *Send* commands have been executed by CE2 and CE3. As soon as r_6 and r_8 are received T_{13} is executed. Programs for CE2, CE3 and CE4 are similarly interpreted. For efficiently executing this task graph, data paths between CEs must be

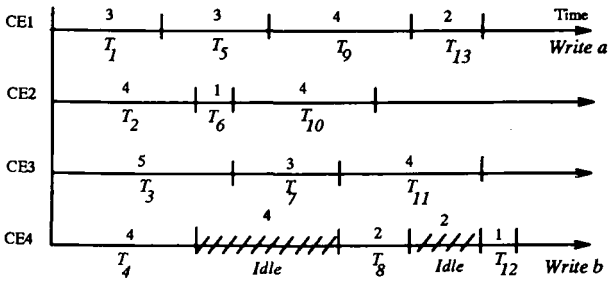


Fig 4.5. Assignment of tasks of task graph to 4 CEs

available as shown in Fig. 4.6. Observe that a CE which receives data from another CE must provide a small storage area called a *buffer* to receive the data.

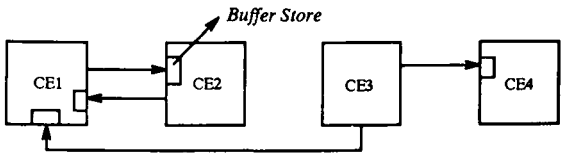


Fig 4.6. Interconnection of computing elements to task graph of Fig 4.3

The data paths connecting CEs shown in Fig 4.6 is specialized to efficiently execute the task graph of Fig. 4.4. For executing densely interconnected graphs it would be ideal to provide data paths between every CE and every other CE. The paths will become numerous if there are many CEs. For n CEs the total number of paths will be $n(n-1)$. For 4 CEs these paths are shown in Fig. 4.7. To reduce the complexity of connections one may connect all the CEs to a *bus* (i.e. a set of parallel data lines and control lines) as shown in Fig. 4.8.

This is a cost effective method but the problem with this method is that at a given time only one data packet can be sent on the bus. Thus if CE2 sends data on the bus, no other CE can use the bus until the

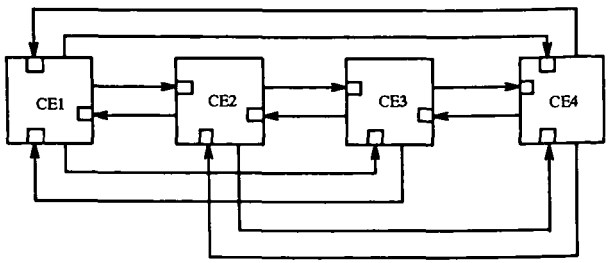


Fig 4.7. A fully connected set of CEs

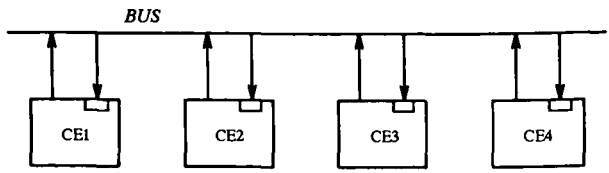


Fig 4.8. CEs connected by a bus

data reaches its destination(s). In a fully interconnected system (See Fig 4.7), any CE can send data to any another CE and receive data from any other CE simultaneously. Another possible interconnection scheme between CEs is shown in Fig. 4.9. This interconnection is called a 2D hypercube interconnection. In this interconnection CE3 cannot directly send data to CE1. If CE3 wants to send data to CE1, it sends it to either CE2 or CE4 with a request to forward it to CE1. Thus the time taken to send the data is longer and also another CE has to interpret this request and do the work of data forwarding. The advantage, however, is the number of interconnections is smaller than a fully connected network. The hypercube structure can be generalized. A 3D hypercube connection of 8 CEs is shown in Fig. 4.10. A general hypercube structure of dimension d has $n = 2^d$ processors. The total number of links (bidirectional) is $n \times d/2$. If any CE wants to communicate with any other CE which is not directly connected to it, the maximum number of intervening CEs will be d. For example, if

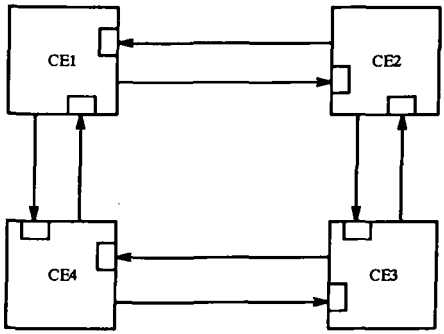


Fig 4.9. A 2D hypercube connection of CEs

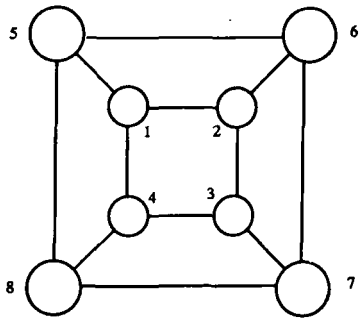


Fig 4.10. A 3D hypercube connection of CEs

CE1 wants to send data to CE7 (in Fig. 4.10) it can either send data to CE2 which will forward it to CE3 which will in turn forward it to CE7 or the route can be CE1-CE4-CE3-CE7 or CE1-CE5-CE8-CE7 etc. In fact, there are 6 paths of length 3 from CE1 to CE7 (length is defined as the number of links traversed). As the number of links is of the order of $n \times d$ and the maximum distance is d the hypercube interconnection of CEs is very popular and many commercial multicomputer systems use this method of interconnection.

4.3 A Generalized Structure of a Parallel Computer

A conceptual structure of a general purpose parallel computer is given in Fig. 4.11. It consists of a set of CEs which are interconnected by a communication network. The task graph to be executed is stored in a task store and the data to be processed in a data store. A scheduler assigns tasks to be processed to the CEs. CEs use the communication network to send or receive data from each other. Results computed by the CEs are despatched to an output unit. A host computer is often used to store the task graph, perform the functions of the scheduler, read input data and output results.

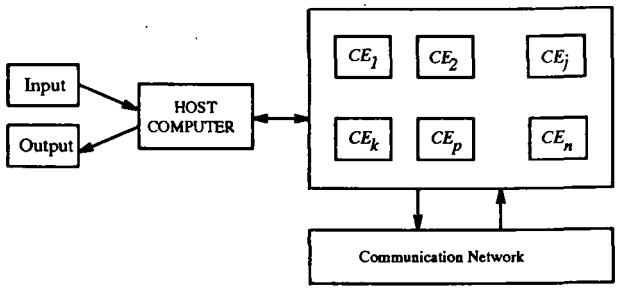


Fig 4.11. Generalized message passing multicomputer

Given this structure of a parallel computer we assume that the task graph and requisite data are stored in the host computer. The scheduler uses an algorithm to statically allocate groups of tasks to CEs. The static allocation keeps in view the available parallelism in the task graph and the expected time needed to execute each task. Each task is a small program which is stored in a CE's private memory. One way of representing a task to be executed by a CE would be:

Name of Task to be carried out	Slots for input data	Slots for results

Once the tasks are allocated to CEs the following steps are followed to execute a program:

Step 1: Send a command from the scheduler to all CEs to start executing tasks.

Step 2: CEs execute those tasks which have all the input data available.

Step 3: The results are sent to the scheduler which routes them to the CEs which need this data.

Step 4: Steps 2 and 3 are repeated until no tasks remain unexecuted and no data (to be processed) is left in the communication network or CEs.

There could be many variations to Step 3. They are:

1. The results are not sent to the scheduler but to a storage unit shared by all CEs. Whichever CE requires a result to execute the task assigned to it reads the results from the shared storage unit.
2. Each CE keeps a program which consists of all the tasks to be executed, the identity of the CEs from which data is to be received and the identity of the CEs to which results are to be sent. After executing a task the results are sent via the communication network to the appropriate CEs which receive them.
3. At the end of execution of a task a CE broadcasts the result on the communication network. All the CEs needing this result capture it and store it in their local memory. An uncaptured result is kept in a result buffer by the scheduler and broadcast again.

The parallel computer hardware organization is normally tailored to match the corresponding execution model.

The scheduling philosophy described above is a static philosophy. Tasks are pre-assigned to CEs. A *dynamic schedule* may be more efficient particularly in cases where the completion times of tasks are strongly data dependent. In dynamic scheduling the task graph is stored in the host computer. The host computer schedules tasks as follows:

Step 1: The host sends tasks ready to be executed to free CEs.

Step 2: The CEs execute tasks assigned to them and send results to the host.

Step 3: The host fills the input data slots of waiting tasks with relevant results and marks them ready.

Step 4: Steps 1, 2 and 3 are repeated until no more tasks are left in the task graph, all CEs are free and no data remains in the communication network.

This method requires a lot of coordination by the host and the host may become the real bottleneck and slow down execution. A combination of the static scheduling method and this method is often used and is called *quasidynamic* schedule.

Observe that in this organization of a parallel computer multiple tasks are carried out on multiple data sets asynchronously. Such machines are called Multiple Instruction Multiple Data (MIMD) parallel computer. This is in sharp contrast with Pipelined Processors, Array Processors and SPMD computers which synchronously carry out a single task on multiple data.

Referring again to Fig. 4.11, the heart of the system is a set of Processing Elements (PEs) or Computing Elements (CEs). This general structure can have many variations in the type of PEs, the disposition of the memory modules and the way the communication network connects the two and the philosophy of scheduling tasks. These variations lead to a rich variety of parallel computers. We will describe now some of the common parallel computers which have been built.

4.4 Shared Memory Multiprocessors

In this type of computer around 8 to 128 microprocessors or very powerful processors are connected to a set of memory modules using a communication network. This set of memory modules can be accessed by any PE and forms the main memory with one common address space. The program to be executed, data and results are stored in it. This type of parallel computer is also known in the literature as

a *tightly coupled multiprocessor* system.

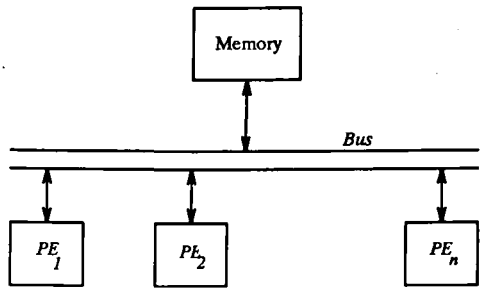


Fig 4.12. A shared bus multiprocessor

The most commonly used communication network to connect PEs to the main memory is a bus. The bus is shared by the processors and the memory (see Fig. 4.12). There will be lines in the bus to address locations in memory and also to address PEs. The bus is a common resource used by the PEs to access memory and other PEs. Two PEs cannot use the bus at the same time. Thus if two or more PEs want to access memory simultaneously, a queue is formed. Even if simultaneous access is not requested, a shared bus limits the number of PEs which can be effectively used in this system as explained below.

Assume that a PE can carry out 1 million instructions per second (1 mips) and that after each instruction is carried out a 64 bit(8 byte) data is to be stored in the shared memory. Let the time to store a result in memory be $0.2\mu sec$. The time to carry out one instruction is $1/10^6$ or $1\mu sec$. If the memory is connected to a bus and the bus speed is 40 megabytes per sec then the time to transport the result (8 bytes) on the bus is equal to $8/(40 * 10^6)$ which equals $0.2\mu sec$.

Assume that 10 PEs share the bus and PEs request access to the bus with equal probability. The minimum time taken to get access to the bus is 0. The maximum time to get access to the bus is 9*0.4 or $3.6\mu sec$. Thus the average time to get access to the bus is $1.8\mu sec$.

The average time to carry out an instruction = Average bus access time + Time on bus + memory access time + average instruction execution time =1.8 + 0.2 + 0.2 + 1 = 3.2μsec.

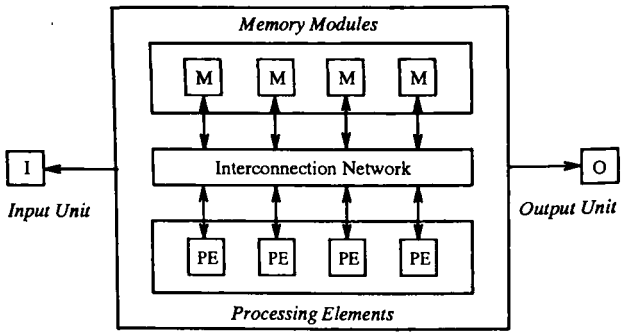


Fig 4.13. General structure of a shared memory multiprocessor

Therefore the effective speed of 10 PEs will be (10/3.2) or 3.125 mips. If all PEs work simultaneously without the need to access a shared memory using a bus then the maximum speed of computation is (10/1.4) or 7.14 mips. Sharing the memory using a bus has reduced the effective speed of the multiprocessor by 55%.

This very elementary calculation shows that the shared bus limits the number of PEs in such a parallel computer. Most shared bus parallel computers which are commercially available use a small fast memory called a cache in each PE and copy currently required contents of the main memory into the cache of each PE. This reduces the time required to access memory by a PE. It, however, introduces another problem. If a PE alters a data item in its cache, then this should be broadcast to all other PEs having the same data item in their caches. This problem called *cache coherence* problem introduces complexity in the design and limits the number of PEs to less than 16 in bus based systems. Inspite of this, shared bus shared memory multiprocessors are popular as they are inexpensive to build and

relatively easy to program.

Instead of using a shared bus we may use an interconnection network (see Fig. 4.13) connecting a set of memory modules with a set of PEs. Such a network allows many *simultaneous* transactions between processors and memories. A larger number of PEs can thus be used in such a computer. The cost of such networks is much higher than that of a bus. In this case also the cache coherence problem limits the number of PEs.

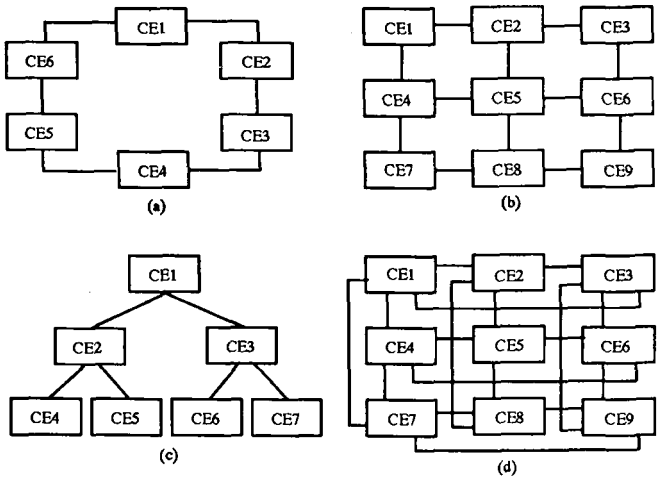
Supercomputers such as Cray combine both pipelined vector processing and shared memory parallel computing. For example, the Cray YMP 132 is a single processor vector computer with 32 Mwords of memory (The 1 in the numbering indicates a single processor and the 32 the memory size). A Cray YMP 864 is an eight processor machine in which each processor is a Cray YMP processor. The 8 processors share a common memory of 64 Mwords. The eight processors can work independently or can cooperate to solve one program stored in the shared memory.

4.5 Message Passing Multicomputers

The parallel computer structure discussed in Section 4.2 (Fig. 4.9) is called a *message passing multicomputer*. The result obtained after executing a task by a CE which is sent to another CE is called a *message*. In Section 4.2 we looked at an interconnection scheme between CEs called a hypercube interconnection. There are other interconnection schemes called a ring, a 2D mesh, a tree, a 2D torus etc., which are shown in Fig. 4.14.

For a message passing multicomputer to work efficiently it is necessary to:

1. Minimize the number of messages transmitted beween CEs.
2. Minimize the number of inter-CE links the messages have to traverse from source CE to destination CE.
3. Minimize the time taken to communicate messages between CEs. This must be much less than that taken to compute.



(a) A ring. (b) A 2D mesh. (c) A tree. (d) A 2D torus.

Fig 4.14. Some interconnection networks used in message passing multicomputers

Unlike a shared memory computer a message passing computer does not share a common global memory. It is thus scalable. In other words the number of CEs can be increased provided a proper task allocation is possible. Message passing machines with 1024 CEs have been built and used for solving problems.

Programming message passing multicomputers is more difficult than programming shared memory machines. The major difficulty in programming is the allocation of tasks to CEs and managing message routing between CEs. As the time taken for messages to travel between CEs is normally much higher than the computing time, a program with a poor task allocation which requires many messages to be transmitted to distant CEs will be executed very slowly. Some automated tools to optimize task allocation to reduce execution time have

been proposed. The problem is, however, difficult to solve as task graphs vary from one program to another, whereas the CE interconnection structure is fixed. Another approach is to have a dynamically reconfigurable interconnection between CEs. The reconfiguration is program controlled and is dependent on the task graph.

4.6 Comparison of Vector and Parallel Supercomputers

We saw in Chapter 3 that vector supercomputers will achieve their ideal speed provided hundred percent of a program is vectorizable. In practice this is not possible. However, the most time consuming part of a program is normally vectorizable leading to good speedup of the overall program. The main advantage of vector machines is the ease with which temporal parallelism is used by a vectorizing compiler.

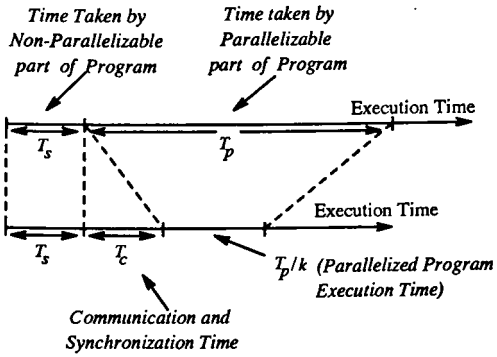


Fig 4.15. Finding execution time of parallel program

Parallel computers, on the other hand, use data parallelism. The speedup depends upon the type of task graph and on how many tasks can be carried out simultaneously. In order to estimate the speedup obtainable with parallel machines we use Fig. 4.15. In this figure T_s indicates the time taken by the sequential part of a program which cannot be executed in parallel and T_p the time taken by the paralleliz-

able part of a program. If we run the program on a parallel computer with k processors the time taken for executing the program is shown in the lower part of Fig. 4.15. The time taken by the sequential part of the program remains the same. The time taken by the part which can be executed in parallel is $(T_p/k)+ T_c$, where T_c is the time taken to synchronize execution of tasks and/or time taken to communicate data and results between tasks.

The speed up is given by:

$$\begin{aligned} Speedup = & \quad (T_p + T_s)/((T_s + T_c) + (T_p/k)) \\ = & \quad k/((k(T_s + T_c)/(T_p + T_s)) + (T_p/(T_p + T_s))) \end{aligned}$$

If we call $T_p/(T_p + T_s) = f$ = the fraction and the time taken to execute the parallelizable code and $T_c/(T_p + T_s) = g$ = fraction of the time spent in synchronization and communication, then

$$\text{Speedup} = k/(f + kg + k(1 - f))$$

If $(f = 1)$ and $(g = 0)$ the speedup is maximum and is k. If $(f = 1)$ then

$$\text{Speedup} = k/(1 + kg) = 1/(g + (1/k))$$

Asymptotically when $k \rightarrow \infty, Speedup \rightarrow (1/g)$.

Thus if the fraction of time g spent in communication and synchronisation is (1/100), regardless of the number of processors, the maximum speedup cannot exceed 100! In Table 4.2 we give speedup for various values of k for given values of f and g. It is seen that the efficiency defined as (Speedup/ k)rapidly decreases with the increase in the number of processors. Thus the figure of 1000 Megaflops speed specified for a parallel supercomputer by some vendors for a computer configured with 1000 processors each of 1 Megaflop speed has to be taken with a pinch of salt!

An interesting question to ask is,"Under what conditions will a parallel supercomputer (of the type we discussed in this chapter) give near linear speedup?" We list some of these conditions below:

Table 4.2. Speedup obtainable from parallel computers

k	f	g	Speedup	Efficiency
2	0.9	0	1.8	0.9
10	0.9	0	5.26	0.526
100	0.9	0	9.17	0.0917
1000	0.9	0	9.9	0.0099
10000	0.9	0	9.99	0.00099
2	1.0	0.1	1.67	0.833
10	1.0	0.1	5.0	0.5
100	1.0	0.1	9.1	0.091
1000	1.0	0.1	9.9	0.0099
10000	1.0	0.1	9.99	0.000099

- If an algorithm can be composed of a set of independent tasks with no need for inter-processor communication
- If an algorithm is designed for an architecture so that while computation is being performed, synchronization and communication are carried out simultaneously and the parallelism available in the algorithm always exceeds the number of available processors.
- If the time taken for computation by the non-parallelizable part is very small compared to the parallelizable part and the parallelizable part has f = 1 and g = 0.

In Table 4.3 we give a comparison of vector computers and parallel computers.

In general it is difficult to satisfy any one of the conditions stated above, particularly if there are very large number of processors in a computer. Parallel supercomputers such as Cray YMP 864 have only 8 computers working in parallel. Each computer is a powerful vector processor. Thus it is relatively easy to exploit parallelism inherent in programs being executed on this machine. Such a machine is thus very efficient and very often delivers the promised speed. On the other hand, a parallel computer using a thousand one megaflop

Table 4.3. Comparison of Parallel and Vector Computers

Type of Computer	*Type of Parallelism exploited*	*Task size*	*Programming ease*
Pipelined Vector computer	Temporal	Small (Fine grain)	Relatively easy (automatic) vectorization
Array Processor	Data Parallelism (SPMD)	Small (Fine grain)	Relatively easy
Shared Memory multicomputer (with around 12 to 16 Processors)	Multiple tasks and Multiple data (General Purpose)	Medium (Small to medium grain)	Due to availability of global space relatively easy. Scalability poor
Message Passing multicomputer	Multiple tasks and multiple data (General Purpose)	Big (Medium grain)	Difficult. Good task allocation and synchronization needed

microcomputers will rarely deliver 1000 megaflop speed if one tries to execute a variety of programs. It is extremely difficult to program such a machine to obtain the promised speed.

5

Available High Performance Computers

In this chapter we will describe some of the high performance computers available in the market now (1992). We will also discuss how the performance of these machines is evaluated. The word "supercomputers" has been commonly misused during the last five years and is losing its distinct meaning. The reason for this is the availability of very high-speed microprocessors at a low cost and the ease with which these microprocessors are intereconnected as a parallel computer with a high nominal megaflop speed. Whereas high speed vector computers such as Cray YMP cost around US $10 million these micro-based parallel machines cost less than US $1 million with a nominal megaflop figure in the same range. There is thus a widespread confusion about the true capabilities of such low cost "supercomputers" (sometimes also known as mini supercomputers) in comparison with existing vector supercomputers. In the absence of a precise definition of the word supercomputers, particularly by computer vendors, we prefer to use the terminology high performance computers in this chapter. The main objectives of this chapter are to describe some of the currently available vector supercomputers and other parallel high performance low cost computers, their strengths and weaknesses. We will also discuss how to scientifically evaluate the true capabilities and performance of supercomputers.

5.1 Vector Supercomputers: Cray & Others

Currently only four manufacturers in the world make high performance "traditional" vector supercomputers. Many people call them "true" supercomputers as they fulfill all the desirable architectural

requirements such as 64 bit words, large main memory (exceeding 256MB), large secondary memory (exceeding 40Gb), high speed data transfer between main memory and the vector processor, extensive pipelining, high speed data transfer between the secondary memory and the main memory (around 100 Mbytes/sec), peak arithmetic speed exceeding 1000 megaflops and sustained arithmetic speed exceeding 50 megaflops. These machines can solve a large class of compute intensive problems very effectively. All these machines have an architecture similar to that of Cray computers. We will describe two of them.

Cray Research Supercomputers: Cray Research, USA, are pioncers in supercomputer design. Starting with Cray 1S in 1978, the latest offering is Cray 3 expected to be delivered in 1992. Currently (1992) they made two classes of supercomputers - Cray YMP and Cray 2. The minimum configuration of a Cray YMP is Cray YMP 14 and the highest is Cray YMP C90. Cray computers are designed by Seymour Cray who designed the first supercomputer. The YMP in the model stands for model *YM*ulti*P*rocessor. The first digit following YMP indicates the number of processors in the system and the other digits indicate the number of megawords of shared main memory. For example, YMP 132 specifies a 1 Processor 32 megaword memory machine (1 word = 64 bits) and YMP 8256, a machine with 8 processors sharing 256 megawords of main memory. The latest model Cray YMP C90 has 16 processors and 256 megawords of memory. The other important characteristics of the computer are illustrated in Table 5.1. Besides the hardware features, the main merit of Cray range of machines is the variety of software available for the machine. Besides a good vectorizing Fortran Compiler, Crayalso provides a library of mathematical subroutines for common operations such as matrix computations, eigen value/vector calculation, polynominal computations etc. These subroutines have been written in assembly language and are optimized for Cray machines. As Cray's internal architecture has remained invariant, the assembly language routines work for all models of Cray. Fortran for Cray has been enhanced to include vector instructions (Some of these have been included in Fortran 90).

Many application programs have been developed for Cray both by the manufacturer and by numerous users and software vendors. These programs are usually written in Fortran and vectorized. Cray has the largest number of application programs in almost all areas of science and engineering. This is one of the important considerations when an organization buys a supercomputer.

Table 5.1. Characteristics of Cray Supercomputers

Characteristics	*Cray YMP C90*	*Cray-2*	*Cray-3*
Peak Speed in Megaflops	16000	2000	16000
Cycle Time in nsecs.	6	4	2.5
Shared Main Memory (Max in megawords)	256	256	2048
Number of Processors	16	4	16
Semiconductor used	Silicon	Silicon	Gallium Arsenide
Cooling Method	Freon	Processor immersed in Fluon	Water

NEC Computers: NEC Corporation of Japan is a large diversified company which embarked on building supercomputers in 1984. NEC introduced a series of machines known as the SX series. The smallest in the series is SX1 and the latest is SX3. The unique feature of the SX series is a built-in frontend computer. The SX series machines can work in both 32 bit and 64 bit modes. Thus the megaflop rating for 32 bit operands is almost double that of the 64bit mode. This feature is useful in many problems. The hardware of the machine is superb. It is mostly air cooled with the CPU being water cooled. The hardware features of the NEC series machines[5] is given in Table 5.2.

The Fortran Compiler and Vectorizer on NEC machines are very good. The operating system, however, is not upto the mark. It is not

Table 5.2. Characteristics of Japanese Vector Supercomputers.

Characteristics	*Fujitsu VP-200*	*NEC SX2*	*Hitachi S-810/20*
Peak speed in Megaflops	855	1333	855
Cycle time in nano seconds	7	6	14
Main Memory Megawords	128	32	32
Vector Registers	8192	8192	10,240
Vector Pipelines	2 load/ store	3 load/ 1 loadstore	8 load/ 4 store
Cooling Method	Forced Air	Water	Air
Operating Sytem	VSP, MVS/XA	ACOS	HAP

compatible with the UNIX operating system which is now universally used. Further it is of an old vintage with restricted facilities. Networking features of NEC machine with the machines of other vendors is also poor. Another weakness of NEC machines is a poor application software library. As it is a relatively new machine there are not many users of this machine. Consequently, very few application programs are available from independent software vendors. Until this applications bottleneck is solved these machines will not be very popular. Besides NEC, there are two other Japanese manufacturers of vector supercomputers. They are Fujitsu and Hitachi. They are slowly gaining a share of the market. Cray, however, holds a pre-eminent position in the high end high performance supercomputer market.

5.2 Vector Computer on a Chip: Intel 80860

During 1989 a new Very Large Scale Integrated Circuit (VLSI) chip incorporating a microcomputer with architectural features mimic-

Table 5.3. Characteristics of Intel i860 microprocessor

Characteristics	*Value*	*Characteristics*	*Value*
Word length	64 bits	Vector registers	16
Cycle time in nsec	25	Virtual Memory size	4 Gigabytes
Peak Floating add/multiply speed (64bit operands)	80 megaflops	No.of pipeline stages in adder/ multiplier	4
On chip memory data cache	8 Kbytes	Number of instructions	75
Instruction cache	4 Kbytes	RISC architecture	Yes

ing a supercomputer was released by Intel Corporation, one of the leading integrated circuit manufacturers in the world. The main features of the Intel 80860 (also known as i860) are given in the Table 5.3.

The processor has a pipelined four stage adder and a four stage multiplier. The operands are stored in a fast register file in the processor (16 registers each 64 bits long) and streamed through the pipelined arithmetic unit. One addition and one multiplication can be performed independently by the two units (See Fig. 5.1). The basic clock is 40 MHZ (one clock pulse every 25 nsec). One addition operation and one multiplication operation are performed every 25 nsec giving a peak megaflop rating of 80 megaflops (for 64 bit operands) provided a program can generate both these operations simultaneously. This is so far the highest speed obtained using a microprocessor chip at a low cost.

So called super minicomputers have been built around i860 processor by many manufacturers including two companies in India: Wipro Infotech Ltd., and DCM Data Products. The characteristics of the Wipro Landmark 860 machine and DCM COSMOS 860 - machine are given in Table 5.4. They cost around Rs.15 lakhs (1991 price) and the cost is quite competitive considering the floating point speed

Table 5.4. Characterisitcs of some i860 based computers made in India (1991)

Characteristics	*Manufacturers*	
	Wipro Infotech	DCM Data Products
CPU	i860	Intel 80486 + i860[a]
I/O Processor Disk: Terminal:	 80286 80186	 80386 (Optional) 80186
Max.Main memory (Mbytes)	64	32
Bus	Multibus II	Proprietary DCM Bus
Multiple i860 possible	yes	yes
Operating System	UNIX System V 4	UNIX System V 4
Fortran 77	Greenhill[b]	DCM or Greenhill
Vectorizer for Fortran	Pacific Sierra[c] VAST-2	Pacific Sierra VAST-2
Networking	Ethernet	Ethernet

[a]Both can work concurrently and independently
[b]Company which developed Fortran for i860
[c]Pacific Sierra Company developed VAST-2 vectorizer for i860

delivered by the machines.

5.3 Shared Memory Systems: Alliant & Convex

Many manufacturers have been designing high performance parallel computing systems using either a commercially available micro processor or special customized processors. An example of the former is the Alliant FX/2800 parallel computers and that of the latter is Convex computers. These machines are often called mini supercomputers.

Alliant FX 2800 Parallel Computer: This computer uses the i860 processor for computing. It has 28 processors (i860) arranged as 14 *Super Computational Elements* (SCE) for parallel processing of large

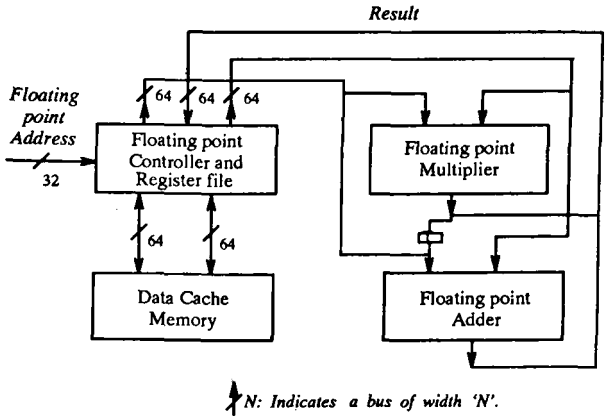


Fig 5.1. Floating point arithmetic unit of i860

compute intensive tasks and 14 *Super Interactive Processors* (SIP) for multiprocessing serial compute jobs, executing system tasks, I/O and 3D graphics processing. A single *Processor Module* (PM) contains four i860 processors arranged as two supercomputational elements and two super interactive processors. Upto seven processing modules are supported by the largest FX/2800 configuration. A separate I/O module contains two super interactive processors combined with two 20 MB/sec Channel interfaces.

All the 14 processor modules share a common memory which can be upto 1 GB. The main memory is connected to the Processors via a cache which can be upto 4 MB. A block diagram of Alliant FX/2800 computer is shown in Fig. 5.2.

An interesting feature of the architecture is the possibility of running a single job using all the 14 computing elements in parallel and simultaneously running 14 other jobs, each using one computing element. The scheduling of various jobs is managed by a proprietary operating system which attempts to optimize the use of all the re-

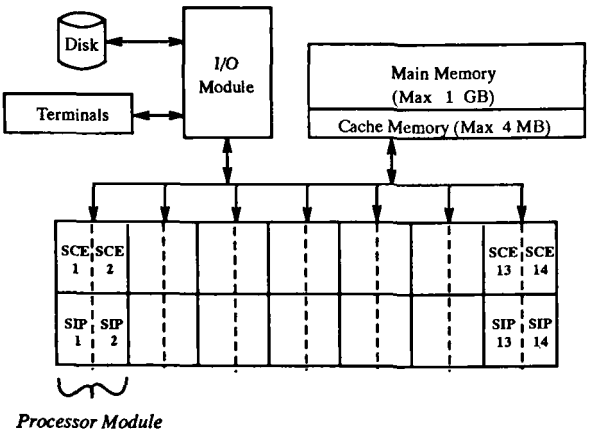


Fig 5.2. Alliant FX/2800 parallel computer

sources of the parallel computing system.

The division of a processing module into an interactive part and a computing part is intended to provide interactive program development, debugging, editing and visualization facilities through the interactive processor which is a high speed processor using an i860. The interactive part also executes operating system tasks such as paging, swapping, file system management, networking and I/O.

The system also provides parallel compilers for Fortran and C. A vectorizer is also provided to take advantage of the i860's pipelining capabilities.

Convex Computer Systems: This is a shared memory multiprocessor made by Convex Computer Corporation, USA. The salient features of the Convex 3800 system are:

- Eight processors sharing a common main memory of upto 4Gbytes and a virtual memory of 2 Gbytes per process. The memory has 8 ports which connect it to the CPUs. Each port can carry

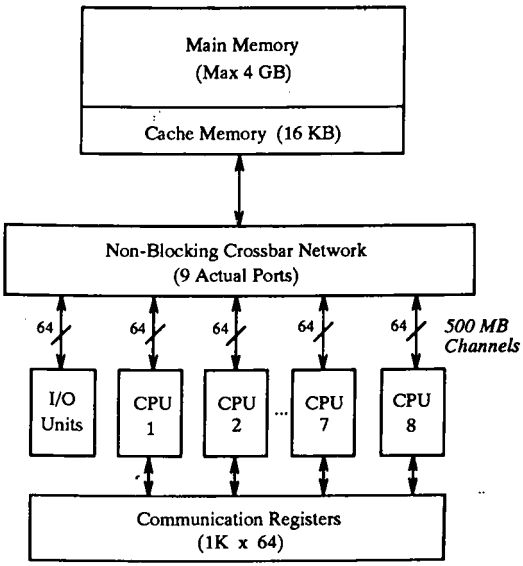


Fig 5.3. Convex C3800 parallel computer

data at a rate of 500 Mbytes/sec. In other words 64 bits data can be transferred in parallel from the main memory to the CPU in 1.6 nsec. The ports are connected to the memory by a crossbar switch. The cross bar allows simultaneous access of each CPU to a different memory bank. If two CPUs try to access the same memory bank there is hardware to temporarily store the request and honour it after a delay (See Fig. 5.3)

The CPUs use Gallium Arsenide semiconductor technology and have a basic clock speed of 20 nsec. It has a Cray like instruction set including vector instructions. The vector operands and results are taken from the memory, processed, and returned to memory (This is known as a memory to memory vector instructions). A set of parallel processing instructions to enable

parallelising DO loops is also provided. The peak megaflop rating of the Convex C 3800 quoted by the manufacturer is 2000 megaflops.

- Convex has an interesting new software scheduler to allocate parallel processes and threads. A thread is a sequence of instructions that represents the smallest schedulable entity. When a parallel region is detected by the compiler, a thread is created. These threads are picked up by idle processors. A thread initiates execution with its own value of the program counter and stack management registers. While leaving a parallel region all processors asynchronously and independently execute join instructions. When a processor executes a join it is free to execute other threads or processes of different programs. The processor schedules itself by picking a waiting thread or process kept in despatch queues maintained in a set of communication registers. When the last processor executes the join, it executes the subsequent scalar part of the program.
- Compiler for Fortran, C and Ada support both vectorizing and parallelizing features.
- A variant of Unix is used as the operating system.

The system seems to have reasonable software and is currently quite popular. Besides C3800 which is the most powerful system in their series, Convex computers have two other less powerful machines which use the same architecture but slower technology. C3400 uses upto 8 processors using BiCMOS technology. The maximum memory available is 2 Gigabytes and the peak speed is 800 megaflops. The C3200 has upto 4 processors, uses ECL and CMOS technology, upto 2 Gigabytes of memory and a peak rating of 200 megaflops.

5.4 Message Passing Multicomputers: iPSC & PARAM

The structure and property of message passing multicomputers were discussed in section 4.5. We will consider two systems of this type.

Intel Hypercube: Intel Scientific Computers has pioneered in build-

ing low cost message passing multicomputers. The individual CEs in their early model (called iPSC) were Intel 80286 microprocessors. The iPSC system consists of two major functional elements; the *cube* and the *cube manager*. The cube is a set of CEs connected as a hypercube with upto 128 CEs. Each node is an independent microcomputer (80286/80287 set with 512 Kbytes local memory in early models). Each node stores a small program (called a kernel) which manages the messages sent and received by the node, routes messages to appropriate destinations and supports execution of multiple processes. The cube manager serves as the host for the parallel computer supporting the programming environment, communication/control software and the system diagnostic facility. The responsibilities of the cube manager include program compilation, program loading, input/output and error handling.

Later version of iPSC, called iPSC2, uses 80386/80387 as the node processor, a hardware router to route messages automatically between processors and a higher speed communication link compared to iPSC.

The most recent system being built using the same basic message passing hypercube structure is called Touchstone DELTA System. This system incorporates 570 Intel i860 and 386 microprocessors, along with a custom mesh routing chip developed at Caltech. It is claimed to have a peak speed of 32000 megaflops.

C-DAC's PARAM: Centre for Development of Advanced Computing set up by the Government of India at Pune, has developed a high performance parallel multicomputer with 256 CEs. This computer is called PARAM, an acronym for PARAllel Machine, which is also the Sanskrit word for supreme. Each CE in PARAM is a microcomputer called a *transputer*. Transputer is manufactured by INMOS and each CE in PARAM is a T805 transputer with 4MB main memory. This transputer is a 32 bit processor and has a good floating point speed of 4.3 megaflops. The transputer was designed to be a building block of parallel computers with built-in links to communicate with other transputers. Each transputer has 4 communication links and can thus

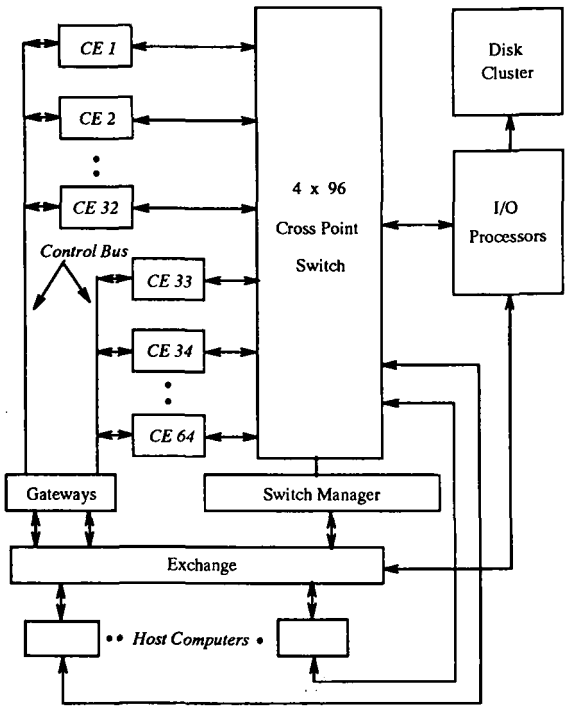


Fig 5.4. Architecture of the 64CE PARAM

be connected to 4 other transputers. Data can be sent and received via the communication link without disturbing the processing unit. Hardware support is also provided to allow the transputer to concurrently execute many processes and to communicate with neighbours. It is thus convenient to build parallel computers with transputers.

PARAM is a message passing multicomputer system. The CEs are interconnected using a crosspoint switch. The four communication ports of each CE is connected to the switch. The switch connections

can be controlled by a switch manager. This allows the CEs to be connected in different configurations such as a tree, a ring, a torus, a hypercube etc. The only restriction is the limit of 4 ports per CE. The architecture of a 64 node mchine is shown in Fig 5.4. A special feature of PARAM is that the 64 CEs can be partitioned into a number of sets. Each set can be allocated to a different user who has a host. The interconnection of the CEs within a set can be specified by the user. Thus if 4 users are each allotted 16 nodes, one user may use a hypercube interconnection, another a ring, third a tree and the fourth user a mesh. Only one user can use a set of CEs at a time.

The maximum number of CEs which are available in PARAM currently is 256. If all 256 CEs are needed for a program, they can be used but by only one user at a time. A maximum of 20GB disk is now provided.

A software environment to program PARAM has been developed and is called PARAS. It has tools for parallel program development such as process allocation to CEs, process management, debugging and profiling. Percieving parallelism in a program and allocating tasks to CEs is the user's responsibility. A new version of PARAM which uses an intel i860 processor as a vector coprocessor shared by 4CEs has now been designed. It is expected to enhance the speed of PARAM.

5.5 Data Parallel Computers: Connection Machine

In the last chapter we described the idea of data parallelism. We described the structure of array processors. The idea of array of processors has been generalized in the *Connection Machine* built by Thinking Machines Corporation of USA. The Connection Machine, CM 2 system has 65,536 physical processors. Each processor has 4096 bits of memory and has an Arithmetic Logic Unit which operates on 1 bit operands. Sixteen processors are integrated into one chip and are connected in a mesh. There are 4096 chips.

These 4096 chips are connected together, as a 12 dimensional hypercube. Routing of messages between chips is done by a hardware

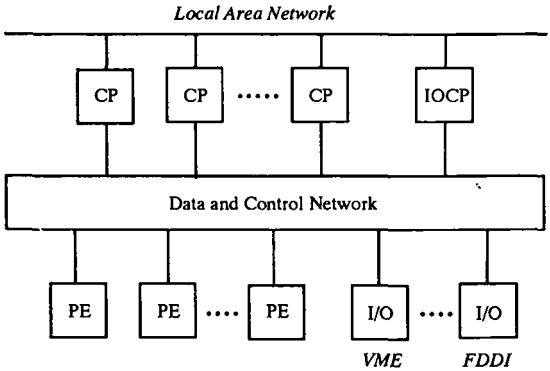


Fig 5.5. The CM 5 architecture

router in each chip. The basic message passing instruction is *send.* Arguments of send specify the length of the message, the message itself and the address of the destination processor. The Connection Machine routing hardware handles a large number of messages efficiently. CM 2 is an SIMD machine.

Model CM 2 has now been replaced by model CM 5. Model CM 5[4] has thousands of processing elements, one or more control processors and I/O units to support disks, graphics terminals and other peripherals. These are connected by two networks called the control network and the data network as shown in Fig. 5.5.

A processing element of CM 5 is a general purpose 32 bit computer with 8 to 32 megabytes of private memory. Thus it can compute by itself. It can also cooperate with other processing elements (as specified by the control processor) and compute in parallel. A high performance arithmetic accelerator can be added to the processing elements to give it the capability to deliver 128 Mflops speed. The arithmetic accelerator is connected to the processor through a 64 bit bus and consists of 4 vector arithmetic units each with 8 Mbytes of data memory. The vector units carry out vector instructions issued

by the processing element. In fact a single CM 5 processor with a vector unit is a minisupercomputer by itself.

The control processor consists of a high performance microprocessor, memory, I/O and an interface to connect it to the control and data networks. The main function of the control processor is scheduling user tasks on processors, allocating resources, ensuring security etc. The control processor runs a version of Unix operating system. Programs are loaded on CM 5 control processor. It broadcasts blocks of instructions to the parallel processing elements and then initiates execution. When all processors are on a single control thread, they are kept closely synchronized. When the processors take different branches, they fetch instructions independently and synchronize only as required by the program.

To maximize system utilization a system administrator may divide the parallel processing elements into groups known as partitions. Each partition is then managed by a control processor.

The main advantage of CM 5 is its special structure and software aids to perform efficiently data parallel computations at both fine and coarse grains. As it is easy to perceive data parallelism in programs the promised speed of 100 Gflops on a thousand processors may be attained with such programs.

5.6 Performance Evaluation of Supercomputers

When a customer wants to buy a supercomputer paying millions of dollars he would like to know how the computer will perform for his applications. Manufacturers normally quote peak megaflop rating which is based on the best possible hardware capability of the computer. There is a wide disparity between the average megaflop speed obtained when applications are run and the peak value. For instance, in some applications, the average megaflops obtained may be 5 while the peak rating may be 2500! The main reason for this is that algorithms and programs, if not properly designed, can perform very poorly on supercomputers. Small differences in code can make a supercomputer run signficantly slower or faster. Thus it is impor-

tant to get a realistic measure of the performance of a supercomputer before it is bought.

There are many methods of evaluating the performance of high performance computers. A very good method is to collect a reasonable variety of (atleast 15 programs) from the users' applications for which the high peformance computer is intended to be used. The programs must be the ones which take a long time on a mainframe to compute. The expected execution time of each program on the fast computer should be atleast 10 minutes.

This batch of programs, called *benchmark programs*, are then executed on the specific configuration of a high performance computer one intends to buy and the execution times are recorded. The programs are run in four different modes:

1. As it is, with no vectorization (parallelization) or optimization
2. Programs vectorized (parallelized) and optimized using the vendor's automatic vectorizer and optimizer
3. Programs further vectorized (parallelized) and optimized manually. Manufacturer's assembly language code for some of the standard routines such as matrix inversion may be substituted in the place of users' routines. This is expected to give the least execution time
4. Program algorithm changed and program re-written keeping in view the specific architecture of the machine on which the program is executed. This may be particularly critical for parallel computers for which special programming may be required.

Even though this is a good method there are many practical problems in following this method. The exact configuration required may not be available. It is expensive as it requires both computing and human resources of the vendor. Vendors are thus reluctant to run benchmark programs of prospective customers unless they feel the customer is a very serious prospect. The entire operation of collecting a set of benchmarks, validating them, running them on existing machines to get sample answers with data and then converting them

to run on a vendor's machine is very time consuming and expensive for the prospective customer also. Thus there have been attempts to standardize a set of commonly used application programs which are representative of the load on supercomputers. We will describe some of these approaches.

A popular benchmark (also known as a *kernel* program as it is the heavily used core of a program) is the LINPACK kernel. This is a program to solve a set of 100 simultaneous algebraic equations. The set of equations is dense, in other words, most of the coefficients of the equations are non zero. The program is coded in Fortran and is run in 2 modes: one with 32 bit real number precision and one with 64 bit precision. The 64 bit precision is more relevant for supercomputers. Executing LINPACK requires very frequent calls of the so-called Basic Linear Algebra Subprograms (BLAS). LINPACK execution time is quoted using only Fortran. Another LINPACK time quoted is by replacing BLAS by optimized assembly language programs for specific machines. LINPACK program was developed by Jack Dongarra of Argonne National Laboratory, USA. LINPACK program has been run for almost all computers starting with a Personal Computer to the most powerful supercomputer. The quoted megaflop rating is known as LINPACK megaflop and it is around 40 for a Cray YMP single processor machine. Even though LINPACK program is not truly representative of all scientific and engineering problems it is well understood and relative performance of computers can be evaluated. Another drawback of LINPACK is its relevance to only single processor computers.

More recently scientists at the Centre for Supercomputer Research and Development (CSRD) at the University of Illinois, USA, have evolved a set of benchmark programs called the PERFECT CLUB benchmarks. This set of benchmarks is the result of extensive work undertaken by CSRD to develop a set of thirteen long-running supercomputer application program representing a spectrum of scientific applications. These benchmarks are becoming widely accepted as standard benchmarks for evaluating supercomputers. In Table 5.5

Table 5.5. Perfect Benchmark Results for Cray Machines

Program	*CRAY YMP 1*		*CRAY YMP 8*	
	Unoptimized speed in Mflops	Optimized Speed in Mflops	Unoptimized Speed in Mflops	Optimized Speed in Mflops
Fluid Dynamics				
ADM	19	62	19	91
ARC3D	140	148	291	989
FLO52	109	111	329	347
OCEAN	32	124	36	275
SPEC77	36	101	36	543
Chemistry & Physics				
BDNA	84	142	121	288
MDG	17	80	17	595
QCD	13	39	13	250
TRFD	56	76	56	440
Engineering Design				
DYFESM	47	136	59	295
SPICE	6	20	6	20
Signal Processing				
MG3D	23	191	23	1094
TRACK	8	18	8	39
Harmonic mean megaflops	20	57	21	116

we give the megaflops obtained on 13 Perfect club benchmarks for Cray YMP. It is clear that there is a wide variability in megaflops obtained. The harmonic mean value gives an estimate of the sustained speed available. Observe also that the peak megaflop is nine times larger than the harmonic mean of the megaflops obtained for the 13 applications in CRAY YMP 8. Thus performance evaluation of supercomputers for specific applications is very difficult without actually implementing and running the program.

Evaluating the performance of parallel computers is still more dif-

ficult. Optimizing the program is highly architecture dependent. An algorithm and a corresponding program which runs very fast on a particular parallel machine may be very inefficient on another. Thus the peak megaflop rating quoted by vendors would be meaningless when one tries to run specific applications. The field is still nascent and it will take some time before acceptable methods of performance evaluation of such machines are established.

6

Applications of Supercomputers

Supercomputers are indispensable tools for scientists and engineers in their research and development. As was pointed out in the first chapter, numerical experimentation using computers is now as important as hypothesis or theory formulation and experimental verification. Numerical experimentation, namely, simulation on a computer of the mathematical model enhances a scientist's ability to reason about complex phenomenon in many ways. Simulation allows the study of phenomenon which are difficult to study experimentally[8]. Examples of such situations are simulating accidents in nuclear reactors, simulating crashes of a motor car or aeroplane or spread of fire in oil wells. Simulation allows study of complex, non-linear models which are difficult to solve analytically. Examples are the study of fusion reactors, formulating atmospheric models and modelling spread of oil spills in oceans. Simulation is also useful to test whether a proposed theory is correct. Examples are numerical study of drug reactions, and synthesis of drugs. We illustrate these with examples in this chapter. Besides their use in science and engineering which may be thought of as traditional applications, many other applications have now emerged which were not foreseen by supercomputer designers. The interesting ones are the use of supercomputers in Hollywood to make cartoons and, by advertising agencies to make innovative graphics based video presentations. Economists now use it for large economic models, security agencies use them to design secret codes and cryptologists to break these codes. The enormous speed of supercomputers, wider availability, reduction in cost and the emergence of excellent high resolution graphics have all led to an increase in the

variety and novelty of supercomputer applications. In this chapter, we will describe some interesting applications of supercomputers to enable the reader to get an appreciation of the versatility of these computers.

6.1 Motor Car Crash Simulation

An interesting application of supercomputers is to simulate the effect of various types of accidents involving a motor car, its driver and passengers. The primary aim is to design a car body in such a way that if it collides with another vehicle on the side, or head on, or crashes against a barrier the driver and the passengers in the car will be safe. In Western countries standards have been specified for motor car bodies to ensure safety which should be adhered to by manufacturers. Thus manufacturers design their vehicles with care so that it is a safe vehicle to drive.

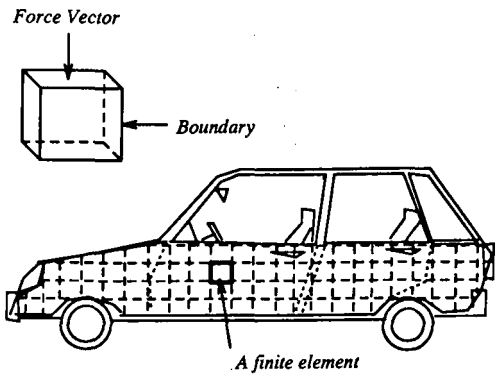


Fig 6.1. Division of a Motor Car body as finite elements

In earlier days (1970s and 80s) manufacturers used to make model cars and actually crash them against a barrier at different speeds and study the effect. This has many disadvantages. It is very expensive to make a car and crash it. Thus the number of experiments carried out would be limited by this cost. If it is found after the crash that

major modifications are needed in the body design there will be a long delay before the new model car is ready to be marketed. This delay will reduce the competitiveness of the manufacturer and hence his sales. Thus extensive crash tests were not feasible.

With the advent of computers a natural question to ask is "Is it possible to model the motor car body mathematically in enough detail and simulate crashes using this mathematical model on a computer?". To answer this question let us examine how to mathematically model the body of the car. This is done by taking the surface of the body and dividing it into small squares called finite elements, as shown in Fig. 6.1.

The force due to collision is modelled as force vectors on each finite element. With the known geometry of the element and the property of the material of the motor car body the stresses and resultant deformation is computed. If the body is divided using a 100 $\times$ 100 grid, each corner has three translational and three rotational degrees of freedom so that there are roughly 6^4 unknown quantities to be computed. The basic matrix equation to compute these is[8]:

$$[K]\{U\} = \{P\}$$

where $[K]$ is the stiffness matrix generated from the known characteristics of the material used for the body $\{U\}$ is the unknown displacements to be computed and $\{P\}$ is the vector of forces applied to the grid points and generated from the type of crash to be simulated.

The computations required to generate $[K]$ $\{P\}$ and subsequently solve for $\{U\}$ are substantial. They are around 10^9 floating point operations for each simulation run. The stiffness matrix is sparse with over 90% of the element values being zeros. This problem takes too long a time on mainframe computers. It is found that the time to run one simulation on a 200 megaflop (peak speed) supercomputer is 6 hours. Inspite of this, manufacturers find this technique more cost-effective than testing with physical car models. The main advantage is that simulation can be performed very early in the design cycle of a motor car. Many body designs can be simulated and only the good

ones kept for futher detailed design and analysis. Fewer real crash tests on physical prototypes are needed. The body design can be optimized to meet very strict safety specifications. This is one of the applications for which car manufacturers abroad use supercomputers extensively.

Similar ideas are used to design bodies of aircratts and space vehicles. The problems are a lot more complex and need very large supercomputers.

6.2 Application in Oil Exploration

Geologists carry out what are known as seismic exploratory studies to find out whether a given region has a potential oil well. These studies are also used to pick out a spot which is the most promising one for digging an oil well. As digging a deep well is very expensive, it is advisable to find ways whereby every drilling exercise is successful. The usual method used by geologists is to dig a deep hole in the earth, plant an explosive there and detonate it. When the bomb explodes, shock waves travel all around and eventually arrive at various points on the surface after a lapse of time. A number of detectors are placed surrounding the hole and the intensity of the signals received and the elapsed time between the detonation and the detection of signals are measured. A large number of such experimental results are stored on storage devices such as magnetic tapes. The intensity of the signals received and elapsed time are functions of the composition of the ground in the region surrounding the excavation.

The question a geologist would ask is "What is the composition of layers of rocks, sand, soil, water, oil etc. which will result in the signals detected by the explosion?". Once this composition is known the geologist can predict the best place to dig for oil. In order to answer this, the composition of the earth in that area is modelled by multiple layers of materials with varying densities as shown in Fig. 6.2. The acoustic wave propagation in the surrounding region when an impulse blast is applied at the specified point is calculated. The governing equation is a 3D accoustic wave equation given by [11]:

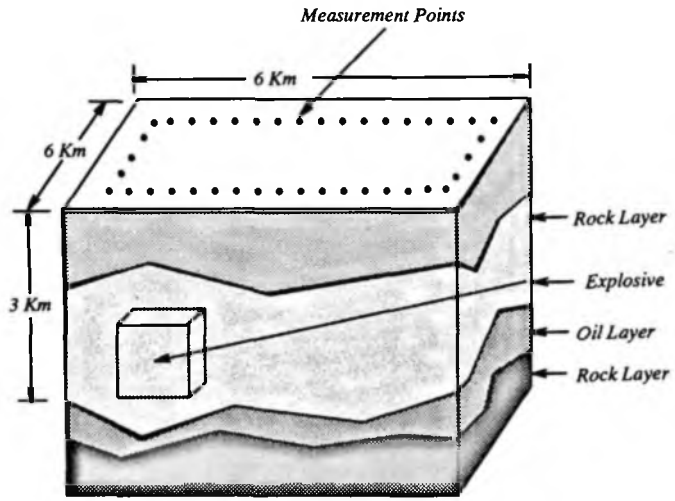


Fig 6.2. Model of earth for exploration

$$\frac{\partial}{\partial x}\left(\frac{1}{\rho}\right)\frac{\partial p}{\partial x} + \frac{\partial}{\partial y}\left(\frac{1}{\rho}\frac{\partial p}{\partial y}\right) + \frac{\partial}{\partial z}\left(\frac{1}{\rho}\frac{dp}{dz}\right) = \frac{1}{k}\frac{\partial^2 p}{\partial t^2} + s(x\,y\,z\,t)$$

where ρ(x, y, z) is density, k(x, y, z) is the bulk modulus and s(x, y, z, t) is the pressure. It is assumed that ρ, k and s are known and p(x, y, z, t) is calculated.

A realistic seismic model for oil exploration requires a 6km × 6km surface and a depth of 3km and simulation should simulate the values of p(x, y, z,t) for 3 seconds. The experimentally detected acoustic waves are sampled at 0.75 or 1 msec interval. Thus the simulation should be run with time intervals of $\Delta t = 1msec$ and 3000 steps are needed to simulate 3 seconds. The values of Δx, Δy, Δz should be around 5 meters for a good finite difference approximation. Thus there will be 6000 × 6000 × 3000 elements in the finite difference model. With assumed values for ρ, k and s, the function $p(x, y, z, t)$,

is computed for the selected values of (x, y, z, t).

These computed results are matched with the experimental values. If they do not match the assumed values of ρ, k, and s are changed and the calculations are repeated. As can be seen, the trial and error method is very tedious and time consuming. Only a supercomputer with a very large main memory and speed around a gigaflop can solve these equations realistically.

After the calculations are complete the model nearest to the one which gave answers $p(x, y, z)$ close to the experimental observations is chosen. This is used to decide the best place to explore for oil by digging a well.

Even though the computing time is very large and an expensive supercompter is needed to solve the problem, it is much cheaper than digging a well and finding no oil. Computer simulation considerably increases the probability of correct drilling and is thus used extensively by geologists prospecting for oil.

6.3 Movie-making with Supercomputers

One of the innovative uses of high performance computers coupled to high resolution colour graphics terminals is in producing animation and special effects of excellent clarity and realism for the motion picture industry. Hollywood has pioneered [13] in the use of supercomputers such as Cray XMP to revolutionize movie making, particularly, those which combine humans and animated characters such as in the famous movie "Who Framed Roger Rabbit". The dream is to even simulate and synthesize human characters by computers. The most difficult problem wil be the creation of believable human figures and human interaction. Currently exaggerated expressions can be modelled, but not subtle facial features. The personality, persona, or psyche of a character must be accurately portrayed to achieve total realism.

Computer simulation of a scene starts with the conception of scenes by an art director. Using the script of the movie and the

scenes, a storyboard artist draws visual descriptions of the key frames. Simultaneously a production designer isolates the objects of a scene such as spaceships, animals, humanlike figures,trees, houses etc. Each object is designed in great detail and an architectural plan is drafted that contains complete 3D information. The surface shape of each part of an object is drawn as a set of polygons. A large number of polygons are needed to describe complex objects realistically. Both computer storage and time needed for simulation increase rapidly as the number of polygons increase. The polygons are translated into numbers using a digitizing tablet and stored in a data file in the computer. Using this data a computer program constructs 3D models and displays them on a graphics terminal connected to the computer. Sophisticated hardware and software on the graphics terminal and the computer allows the objects to be rotated, enlarged and contracted. This facility is used to make any modifications in the design. A library of designed objects is normally kept in a database for future use. The image of an object created on the graphics terminal is called a *wire-frame* as it is a line drawing and looks as if a wire mesh has been stretched over the surface of the object.

The wireframe is then converted by software to look like the artist's conception of the original object. In doing this the texture of the object, the type of lighting, colour etc., is realistically portrayed. The computer faithfully portrays the type of material such as glass, metal, wood, wool etc. The surfaces are made dull or shiny, smooth or rough, shadows are shown, transparency, translucency, reflection, refraction etc., by objects are simulated. The artist, the software consultant and the producer work closely together to ensure accuracy and realism.

Trial animation is done by the technical director using the wire frame images. The animation should faithfully follow the conceptualisation of the art director. The technical director specifies the starting and end wire frames, the actions which take place in between and length of time for each action. A software package uses these specifications, computes the number of intermediate frames to

be generated, the changes from frame to frame and automatically creates the appropriate actions. This is viewed by the technical director and approved. If not satisfactory the specifications are changed by the technical director and the program is re-run.

When the final video film is made, the software computes each frame using all the characteristics of the object including texture, colour, etc., and creates visually realistic scenes. A high performance computer is needed to create realistic intermediate objects. The film segments are edited and combined with live action footage and special effects to entertain movie fans.

The scope of computer based animation systems is very broad. Movie directors, advertising companies, and television producers have realized the enormous potential of this medium and are investing large amounts on supercomputers and high resolution graphics systems. These systems allow artists and directors to create new worlds far from reality. These creations can defy gravity, turn inside-out, go through solid walls, explode and recombine instantaneously after being shattered to pieces. The productivity of the medium is increasing rapidly. It is possible to produce 24 minutes of film per month or 288 minutes per year which is equal to 4 full length feature films. In contrast, using traditional techniques Walt Disney studio animators produce, on the average, one film every two years.

6.4 Weather Forecasting

A major goal in atmospheric sciences is to forecast weather over long periods accurately. Accurate prediction of monsoons in India, for instance, can make an enormous economic impact. The date of onset of the monsoon in various parts of the country and the pattern of the monsoon can help agriculturists to start their sowing operations in time thereby increasing yield of crops. It was not practical to do this three years ago due to the non availability of supercomputers. With improvements in modelling, better understanding of the monsoon phenomenon, better and more reliable recording of temperature, rainfall, etc., coupled with the availability of supercomputers,

has now made it feasible to predict the onset and the pattern of monsoons two months before the actual onset of the monsoon. The monsoon model is quite ingeneous and uses many empirically determined parameters. The model is continuously being refined based on the difference between observed monsoon and predicted monsoon. In order to predict monsoons and also for medium range weather forecasting a supercomputing facility has been established by the Department of Science and Technology, Government of India at Delhi. This centre has a Cray XMP 28 supercomputer and a VAX front-end computer.

Besides forecasting the pattern of monsoons it is also very useful to predict the detailed weather well ahead of time. Twenty years ago it was thought impossible to forecast weather for long periods such as 2 weeks. Today such forecasts are routinely done by the European Medium Range Weather Forecasting Centre with the help of supercomputers. The weather is governed by physical laws which can be modelled as a set of partial differential equations in which the most important variables are the wind-speed, air temperature, water content and atmosphereic pressure. These three dimensional models can be divided into two classes - grid point or spectral depending on the manner in which horizontal (with respect to earth's surface) fields are represented. In the grid point model fields are represented by values on a discrete grid and derivatives are approximated by finite differences. In the spectral model fields are represented by a finite series of analytic functions such as Sines and Cosines, and derivatives are approximated by differentiating each function in the series [12]. The details of these models are described in text books on weather modelling. Spectral methods are now more commonly used for both numerical weather prediction and climate simulation in USA, Canada and Europe as the predictions based on this model have been closer to the observed weather.

When considering operational weather forecasting one must always keep in mind its timeliness. The sooner the forecast is available, the more useful it is. Supercomputers are essential to ensure timeliness as computation of forecasts are very time consuming. Besides

the time taken by the supercomputer, the elapsed time also includes the time taken to convert raw observations to a form the computer model can use and the time to send the forecasts to the user.

Besides weather forecasting, models of the atmosphere have also been used to simulate the "Greenhouse Effect". Emission of gases such as carbon dioxide, methane and chloroflourocarbons by big industries has started intensive debates on the potential of global warming which could occur due to these gases accumulating in the atmosphere. This is called the Greenhouse Effect. Scientists in many countries are using supercomputers to model the atmosphere for studying the potential impact of emission of these gases on average daily temperature.

Forecasts based on simulation of the effect of these gases on global weather in the next 50 to 100 years have been computed. Various scenarios are considered in simulation such as:

1. Emissions being as they are now
2. Emission increasing each year by 5%
3. Emission decreasing by 5% each year.

These computations would be impossible to carry out without a supercomputer. Simulation results are compared with past observations and authenticated. The results of simulation of the future scenario will give governments of various countries the proper information to assist in proposing a law to control industries to reduce the adverse impact of greenhouse gases.

Another related problem is that of ozone depletion in the upper atmosphere. This is due to the free chlorine in chlorofluorocarbons and the oxygen in nitrous oxide (from motor car exhaust gases) reacting with ozone in the upper atmosphere and breaking down ozone into oxygen. The loss of ozone increases ultraviolet radiation leading to skin cancer in humans and destruction of some cells in plant and animal life. Supercomputers are being used to study how chemical reactions take place in the upper atmosphere, whether both depletion and creation of ozone goes on simultaneously and how ozone

depletion can be halted.

6.5 Magnetic Fusion Energy Research

The Sun and the Stars seem to radiate inexhaustible energy due to fusion reactions which are continuously maintained in the system. The major problem scientists and engineers try to solve is to design a reactor which simulates the conditions in the Sun. The conditions are simulated by the behaviour of plasmas (hot electrons and ions) that while trapped inside intense magnetic fields interact with neutral particles and sources of electromagnetic energy. One of the fields of study is known as Magneto Hydro Dynamics (MHD) which tries to predict the behaviour of a plasma of ions and electrons in a complex magnetic field. A successful research area in plasma physics in recent years has been the analysis of the non-linear evolution of ideal and resistive MHD modes. This success has been achieved by using computer programs which simulate the system in 3 dimensions. The model uses non-linear partial differential equations and requires intensive computational effort. The non-linearity and complexity of these equations preclude analytical solutions and thus computer simulation of these equations is essential. The results of this simulation have played a key role in guiding the theorist in understanding magnetic island[9] development, which is essential in determining sets of parameters of the system which lead to a stable plasma in a confined volume. Earlier experimental results which were not well understood have been clearer after the simulation results obtained by using a supercomputer nearly matched the experimental observations.

6.6 Computational Chemistry

One of the interesting problems solved by structural chemists is to predict the structure of complex molecules. Chemical bonds bind atoms together and one looks for molecules which have some useful properties. Electrons in a molecule have several energy states that can be approximated as a combination of *basis functions*. The basis functions are a set of orthogonal polynominals. When a set of basis functions with appropriate coefficients are picked to meet a minimum

energy criterion they describe the stable arrangement of the atoms in the molecule. A technique known as *ab-initio* method computes the energies of electron states, ignoring those above a specified energy. With N basis functions, the number of interactions are of the order of N^4 (N is of the order of 100). The interactions, or *Gaussian integrals*, typically number in the billions and need a large high speed disk storage device for their storage and high speed computers for calculations. The processing of these integrals usually involves repeated matrix multiplications[14]. Typical matrices in computational chemistry are sparse with less than 1% nonzero elements. Thus algorithms which can exploit sparsity will lead to ten thousand fold increase in speed of computation.

Ab-initio methods have also been used to study chemical properties of various clinically active drug molecules which are difficult to study experimentally. These studies are extremely useful to synthesize new drugs. For example, cyclophosphomide, one of the most widely used and effective anticancer and immuno-suppressive agents acts on a wide variety of tumours and lukemias. The local chemical determinants of cytotoxicity of this drug were identified by using a supercomputer and a hierarchy of molecular computational methods. The number of experiments carried out were much smaller as a number of unpromising cases could be eliminated by examining the simulation results[15]. Thus the total time taken to discover life-saving drugs is reduced considerably.

6.7 Conclusions

In this chapter we described some applications of supercomputers so that a reader will get a flavour of the variety of problems for which these machines are used. Computer technology is progressing very fast. Every two years the speed of computers is doubled and the price is reduced. Computers which were considered supercomputers ten years ago and which cost over ten million dollars and required stringent environmental control and maintenance are now available as desk top workstations. The emergence of massively parallel computers has thrown a big challenge to traditional vector supercomputers

such as Cray YMP. The very definition of supercomputers is now questioned and many Computer Scientists prefer to call them high performance computers. The topic of supercomputers is of potentially great interest to all educated persons as it points to the leading edge of technology and its impact on our lives. We have attempted in this book to give a bird's eye view of this important field. At the end of this book a small bibliography of books which have more detail and research articles for those who want to explore this topic further is given.

Bibliography

[1] Lazou, Christopher; *Supercomputers and their use*, Clarendon Press, Oxford, U.K., 1986.

A book intended for those who want to know details of vector supercomputers, such as Cray, and use them for solving their numeric intensive problems. Fair amount of detail on the architecture of supercomputers.

[2] Levesque, J.M. and Williamson, J.W; *A Guidebook to Fortran on Supercomputers*, Academic Press, California, USA, 1989.

This is for persons well versed in Fortran programming who would like to vectorize their Fortran programs for vector supercomputers. The authors are experienced programmers with Pacific Sierra Associates which specialises in writing vectorizing Fortran Compilers for Supercomputers.

[3] Hwang, K.,and Briggs , F.A; *Computer Architecture and Parallel Processing*, McGraw Hill, N.Y., USA, 1984.

A textbook for postgraduate level students in Computer Science on the hardware and architectural features of high performance computers including vector supercomputers and parallel computers. Not recommended for a beginner.

[4] *CM 5 Technical Summary*, Thinking Machines Corporation Inc, Cambridge, MA, USA, October 1991.

[5] Hwang, K., DeGroot, D.S., *Parallel Processing for Supercomputers and Artificial Intelligence*, McGraw-Hill, N.Y., USA, 1989.

A collection of long articles written by experts in Computer Science on the role of parallel computers in high performance numeric and non-numeric processing. Intended for postgraduate students in Computer Science.

[6] Rajaraman, V., *Elements of Parallel Computing*, Prentice- Hall of India, New Delhi, India-1990.

An introductory book intended for beginners which gives an elementary and self-contained presentation of parallel computing. It explains the notion of parallelism in formulating algorithms, methods of programming parallel machines and their architectural features.

[7] Levine R.D., *Supercomputers*, Scientific American, Vol.46, No.1, pp. 118-35 (1982)

A long popular article which explains why supercomputers are necessary and describes how they solve problems. It is somewhat old but is a very good article for a lay scientist.

[8] Buzbee, B.L., *Gaining Insight from Supercomputing*, Proceedings IEEE, Vol.72, No.1, pp. 19-21 (1984)

An interesting article which gives three situations in which supercomputers are indispensable. This issue of IEEE Proceedings is a special issue on "Supercomputers - Their impact on Science and Technology". It has many interesting articles giving details on how a supercomputer was used to solve important and difficult problems.

[9] Fuss, D., Tull, C.G., *Centralised Supercomputer Support for Magnetic Fusion Energy Research*, Proc.IEEE, Vol.72, No.1, pp. 32-41, (1984)

[10] Gloudeman, J.F, *The anticipated impact of Supercomputers on Finite-Element Analysis*, Proc.IEEE, Vol.72, No.1, pp. 80-84 (1984)

[11] Johnson, O.G., *Three-Dimensional Wave Equation Computation on Vector Computers*, Proc. IEEE, Vol.72, No.1, pp. 90-95 (1984)

Explains application of supercomputers in seismic data processing.

[12] Williamson, D.L., and Swarztrauber,A., *A Numerical Weather Prediction Model - Computational Aspects on Cray -1*, Proc.IEEE, Vol.72, No.1, pp.56-67(1984).

[13] Demos, G., Brown, M.D. and Weinberg P.A., *Digital Scene Simulation - The Synergy of Computer Technology and Human Creativity*, Proc. IEEE, Vol.67, No.1, pp.22-31 (1984).

Describes graphics and animation and how supercomputers are used

in these applications.

[14] Gustaffson, J.L., *Computer-Intensive Processors and Multicomputers*, Chapter 5 in the book by Hwang and DeGroot cited as reference 4 above.

[15] Hausheer, F.H., and Singh, U.C., *Computational Design of Pharmocologic Agents for Cancer and AIDS therapy*, Cray Channels, Vol.12, No.2, pp. 18-21 (1990).

Cray channels is a quarterly publication of Cray Research Inc. It is for limited circulation and contains many interesting articles on applications of Cray supercomputers in diverse areas of Science and Technology.

[16] Bell, G., *The Future of High Performance Computers in Science and Engineering*, Commn. ACM, Vol.32, No.9, pp.1091-1101 (1989).

This article is by a pioneer computer designer, the designer of DEC-PDP and VAX series machines. It is a survey of the state-of-the-art of traditional supercomputers such as Cray and new massively parallel computer, which have now entered the market.

[17] Dongarra, J., Martin, J.J., and Worlton, J., *Computer Benchmarking: Paths and Pitfalls*, IEEE Spectrum, Vol., No.7, pp. 38-43(1989)

Describes various methods of comparing the performance of supercomputers using benchmarks.

[18] Bhatkar, V.P., *etal* (Editors),*Advanced Computing*, Tata McGraw-Hill, New Delhi, 1991.

This 796 page collection of technical reports of the work done at CDAC, Pune and Bangalore describes in great detail the hardware and software features of PARAM parallel computer designed by CDAC engineers.

Index

TITLES OF RELATED INTEREST

UNIX IN EASY STEPS
Azam, M.
Contents : A comprehensive guide to learning UNIX, this book gradually moves from the elementary principles of the subject to the more complex concepts. Adopts an integrated approach in treatment of commands and concepts. The lucidity of the text accompanied by the numerous exercises provide a theoretical and practical knowledge of UNIX.

HARDWARE AND SOFTWARE OF PERSONAL COMPUTERS
Bose, Sanjay K.
Contents : Microcomputer Organization. An Introduction to the 8088 CPU. Hardware Organization of the PC. The Video Display of the PC. The ROM-BIOS Services. The Fundamentals of DOS. The DOS Functions of INT21H. Disks and files under DOS. Memory Allocation, Program Loading and Execution. Interrupt Handling Through DOS. Filters for DOS. Review Questions.

8122403034	1991	271pp	paper	Rs. 75

DIGITAL SYSTEMS: FROM GATES TO MICROPROCESSORS, 2nd Edition
Bose, Sanjay K.
Contents : Boolean Algebra and Combinatorial Circuit Design. Semiconductor Memory Elements. Sequential Circuit Design. Dependency Notation. Microprocessor Architecture and Instruction Execution. Microcomputer Organization. Assembly Language Programming Using the 8085 A. Pin-Out and Timing of the 8085 A. Input/Output (I/O) Techniques. Analog Interfacing. Interrupts. Single-Chip Microcomputers. The Intel 8086/8088 CPU. Microprocessor System Development Aids.

8122404324	1992	448pp	paper	Rs. 100

INTRODUCTION TO COMPUTER SCIENCE
Govindaraju, S., M. Chandrasekaran, A. Abdul Haq, T. R. Narayanan.
Contents : Overview of Computers. Information Representation. Algorithm

and Flowcharting. Theoretical Models of a Computer. An Operating System for Personal Computer Software Packages, Programming in Basic. Information Processing. Computer Graphics. Pascal from Basic. Artificial Intelligence.

8122404251 1992 346pp paper Rs. 70

THE C LANGUAGE TRAINER WITH C++

Jayasri, J.

Contents : Introduction to Computer Languages. Introduction to 'C', Lexical Elements of 'C' Language. Entering and Executing A 'C' Program. Input/ Output in 'C'. Operators and Expressions. Control Structures. 'C' Functions. Arrays and Strings. Pointers. Structures and Unions. 'C' Files. 'C' Preprocessors and Command Line Arguments. Graphics Features in 'C'.

CREATIVE COMPUTING USING BASIC

Kannan, N.

Contents : About Computers. Formation of Algorithms and Flowcharting. Fundamentals of Computers. Beginning to Compute with BASIC. Basic Control Statements. For-Next Loop. Subscripted Variables. Functions and Subroutines. Testing and Debugging. Aspects of Efficiency. Sorting, Searching and Some Business Applications. Problem Solving with Computer. Structured Basic using Microsoft Basic. File Handing. Graphics. Artificial Intelligence.

8122402666 1990 299pp paper Rs. 60

DESIGN OF ELECTRONIC CIRCUITS AND COMPUTER AIDED DESIGN

Shah, M.M.

Contents : PART I : Design of Electronic Circuits : Power Supplies. Filters. Design of Power Supplies. Regulated Power Supplies. Series Voltage Regulator. Controlled Rectifiers. Amplifiers. Jfet Amplifiers. Power Amplifiers. PART II : Computer Aided Design of Electronic Circuits : DC Circuit Analysis. Device Modelling. Design of Circuits.

8122404723 1993 246pp paper Rs. 70

WILEY EASTERN LIMITED

4835/24, Ansari Road, Daryaganj, New Delhi 110 002, INDIA

Bangalore, Bombay, Calcutta, Guwahati, Hyderabad, Lucknow, Madras, Pune

SUPERCOMPUTERS

Of iate there has been a lot of interest in our country on Supercomputers. They have become part of the vocabulary of all educated persons as press reports appear regularly in newspapers about these computers and how they are being used to solve a whole range of very interesting problems from predicting the monsoons to synthesizing life-saving drugs. There is thus a widespread curiosity to know what are supercomputers, in what way they are different from other computers and why they are considered strategic machines by the advanced western countries which have imposed strict export controls on them. The purpose of this educational monograph is to answer these questions and review the current state-of-the-art of supercomputers.

This book explains what is a supercomputer and why such a machine is needed to solve challenging problems in science and engineering. The architecture of supercomputers which distinguishes them from other computers is explained. The need to vectorize programs to make effective use of supercomputers is brought out and some simple methods of vectorizing programs are discussed. The emergence of parallel computers as cost effective replacement for Cray type vector supercomputers is explained. The architecture of parallel machines and the principles used to program them is discussed. The book presents details of some of the commercially available high performance computers. It concludes by describing some interesting applications of supercomputers.

This book is meant for students in their final year M.Sc. or B.E. courses, (with a basic knowledge of programming in a high level language such as Fortran) who would like to know about supercomputers. It should also interest other scientists and engineers who would like to know about this subject.

V. Rajaraman Ph.D. (Wisconsin), is Tatachem Professor of Computer Science and Chairman, Supercomputer Education and Research Centre at the Indian Institute of Science, Bangalore. He is also Honorary Professor at the Jawaharlal Nehru Centre for Advanced Scientific Research, Bangalore. Earlier, Professor Rajaraman taught at the Indian Institute of Technology, Kanpur, from 1963 to 1982.

A pioneer in computer science education and research in India, Professor Rajaraman was awarded the prestigious Shanti Swarup Bhatnagar Prize in 1976. He is also the recipient of the Homi Bhabha Award for Research in Applied Sciences, and the U.P. Government National Award for Excellence in Teaching and Research. He is a Fellow of the Indian Academy of Sciences, the Indian National Science Academy and the Indian National Academy of Engineers. An author of several well established and highly successful computer books, Professor Rajaraman has published many research papers in reputed national and international journals.

WILEY EASTERN LIMITED

New Delhi Bangalore Bombay Calcutta Guwahati
Hyderabad Lucknow Madras Pune

ISBN 81-224-0496-0