UNIT – 1

Lesson 1: Evolution of Computer Systems & Trends towards parallel Processing

Contents:

1.0 Aims and Objectives

The main aim of this lesson is to learn the evolution of computer systems in detail and various trends towards parallel processing.

1.1 Introduction

Over the past four decades the computer industry has experienced four generations of development. The first generation used Vacuum Tubes (1940 – 1950s) to discrete diodes to transistors (1950 – 1960s), to small and medium scale integrated circuits (1960 – 1970s) and to very large scale integrated devices (1970s and beyond). Increases in device speed and reliability and reduction in hardware cost and physical size have greatly enhanced computer performance. The relationships between data, information, knowledge and intelligence are demonstrated. Parallel processing demands concurrent execution of many programs in a computer. The highest level of parallel processing is conducted among multiple jobs through multiprogramming, time sharing and multiprocessing

1.2 Introduction to Parallel Processing

Basic concepts of parallel processing on high-performance computers are introduced in this unit. Parallel computer structures will be characterized as Pipelined computers, array processors and multiprocessor systems.

1.2.1 Evolution of Computer Systems

Over the past four decades the computer industry has experienced four generations of development.

1.2.2 Generations Of Computer Systems

First Generation (1939-1954) - Vacuum Tube

- 1937 - John V. Atanasoff designed the first digital electronic computer.
- 1939 - Atanasoff and Clifford Berry demonstrate in Nov. the ABC prototype.

- 1941 - Konrad Zuse in Germany developed in secret the Z3.
- 1943 - In Britain, the Colossus was designed in secret at Bletchley Park to decode German messages.
- 1944 - Howard Aiken developed the Harvard Mark I mechanical computer for the Navy.
- 1945 - John W. Mauchly and J. Presper Eckert built ENIAC(Electronic Numerical Integrator and Computer) at U of PA for the U.S. Army.
- 1946 - Mauchly and Eckert start Electronic Control Co., received grant from National Bureau of Standards to build a ENIAC-type computer with magnetic tape input/output, renamed UNIVAC(in 1947 but run out of money, formed in Dec. 1947 the new company Eckert-Mauchly Computer Corporation (EMCC).
- 1948 - Howard Aiken developed the Harvard Mark III electronic computer with 5000 tubes
- 1948 - U of Manchester in Britain developed the SSEM Baby electronic computer with CRT memory
- 1949 - Mauchly and Eckert in March successfully tested the BINAC stored-program computer for Northrop Aircraft, with mercury delay line memory and a primitive magentic tape drive; Remington Rand bought EMCC Feb. 1950 and provided funds to finish UNIVAC
- 1950- Commander William C. Norris led Engineering Research Associates to develop the Atlas, based on the secret code-breaking computers used by the Navy in WWII; the Atlas was 38 feet long, 20 feet wide, and used 2700 vacuum tubes
- In 1950, the first stored program computer,EDVAC(Electronic Discrete Variable Automatic Computer), was developed.
- 1954 - The SAGE aircraft-warning system was the largest vacuum tube computer system ever built. It began in 1954 at MIT's Lincoln Lab with funding from the Air Force. The first of 23 Direction Centers went online in Nov. 1956, and the last in 1962. Each Center had two 55,000-tube computers built by IBM, MIT, AND Bell Labs. The 275-ton computers known as "Clyde" were based on Jay Forrester's Whirlwind I and had magnetic core memory, magnetic drum and magnetic tape storage. The Centers were connected by an early network, and pioneered development of the modem and graphics display.

Second Generation Computers (1954 -1959) – Transistor

- 1950 - National Bureau of Standards (NBS) introduced its Standards Eastern Automatic Computer (SEAC) with 10,000 newly developed germanium diodes in its logic circuits, and the first magnetic disk drive designed by Jacob Rabinow
- 1953 - Tom Watson, Jr., led IBM to introduce the model 604 computer, its first with transistors, that became the basis of the model 608 of 1957, the first solid-state computer for the commercial market. Transistors were expensive at first.
- TRADIC(Transistorized digital Computer), was built by Bell Laboratories in 1954.
- 1959 - General Electric Corporation delivered its Electronic Recording Machine Accounting (ERMA) computing system to the Bank of America in California; based on a design by SRI, the ERMA system employed Magnetic Ink Character Recognition (MICR) as the means to capture data from the checks and introduced automation in banking that continued with ATM machines in 1974.

- The first IBM scientific ,transistorized computer, IBM 1620, became available in 1960.

Third Generation Computers (1959 -1971) - IC

- 1959 - Jack Kilby of Texas Instruments patented the first integrated circuit in Feb. 1959; Kilby had made his first germanium IC in Oct. 1958; Robert Noyce at Fairchild used planar process to make connections of components within a silicon IC in early 1959; the first commercial product using IC was the hearing aid in Dec. 1963; General Instrument made LSI chip (100+ components) for Hammond organs 1968.
- 1964 - IBM produced SABRE, the first airline reservation tracking system for American Airlines; IBM announced the System/360 all-purpose computer, using 8-bit character word length (a "byte") that was pioneered in the 7030 of April 1961 that grew out of the AF contract of Oct. 1958 following Sputnik to develop transistor computers for BMEWS.
- 1968 - DEC introduced the first "mini-computer", the PDP-8, named after the mini-skirt; DEC was founded in 1957 by Kenneth H. Olsen who came for the SAGE project at MIT and began sales of the PDP-1 in 1960.
- 1969 - Development began on ARPAnet, funded by the DOD.
- 1971 - Intel produced large scale integrated (LSI) circuits that were used in the digital delay line, the first digital audio device.

Fourth Generation (1971-1991) - microprocessor

- 1971 - Gilbert Hyatt at Micro Computer Co. patented the microprocessor; Ted Hoff at Intel in February introduced the 4-bit 4004, a VSLI of 2300 components, for the Japanese company Busicom to create a single chip for a calculator; IBM introduced the first 8-inch "memory disk", as it was called then, or the "floppy disk" later; Hoffmann-La Roche patented the passive LCD display for calculators and watches; in November Intel announced the first microcomputer, the MCS-4; Nolan Bushnell designed the first commercial arcade video game "Computer Space"
- 1972 - Intel made the 8-bit 8008 and 8080 microprocessors; Gary Kildall wrote his Control Program/Microprocessor (CP/M) disk operating system to provide instructions for floppy disk drives to work with the 8080 processor. He offered it to Intel, but was turned down, so he sold it on his own, and soon CP/M was the standard operating system for 8-bit microcomputers; Bushnell created Atari and introduced the successful "Pong" game
- 1973 - IBM developed the first true sealed hard disk drive, called the "Winchester" after the rifle company, using two 30 Mb platters; Robert Metcalfe at Xerox PARC created Ethernet as the basis for a local area network, and later founded 3COM
- 1974 - Xerox developed the Alto workstation at PARC, with a monitor, a graphical user interface, a mouse, and an ethernet card for networking
- 1975 - the Altair personal computer is sold in kit form, and influenced Steve Jobs and Steve Wozniak
- 1976 - Jobs and Wozniak developed the Apple personal computer; Alan Shugart introduced the 5.25-inch floppy disk
- 1977 - Nintendo in Japan began to make computer games that stored the data on chips inside a game cartridge that sold for around $40 but only cost a few dollars to manufacture. It introduced its most popular game "Donkey Kong" in 1981, Super Mario Bros in 1985

- 1978 - Visicalc spreadsheet software was written by Daniel Bricklin and Bob Frankston
- 1979 - Micropro released Wordstar that set the standard for word processing software
- 1980 - IBM signed a contract with the Microsoft Co. of Bill Gates and Paul Allen and Steve Ballmer to supply an operating system for IBM's new PC model. Microsoft paid $25,000 to Seattle Computer for the rights to QDOS that became Microsoft DOS, and Microsoft began its climb to become the dominant computer company in the world.
- 1984 - Apple Computer introduced the Macintosh personal computer January 24.
- 1987 - Bill Atkinson of Apple Computers created a software program called HyperCard that was bundled free with all Macintosh computers.

Fifth Generation (1991 and Beyond)

- 1991 - World-Wide Web (WWW) was developed by Tim Berners-Lee and released by CERN.
- 1993 - The first Web browser called Mosaic was created by student Marc Andreesen and programmer Eric Bina at NCSA in the first 3 months of 1993. The beta version 0.5 of X Mosaic for UNIX was released Jan. 23 1993 and was instant success. The PC and Mac versions of Mosaic followed quickly in 1993. Mosaic was the first software to interpret a new IMG tag, and to display graphics along with text. Berners-Lee objected to the IMG tag, considered it frivolous, but image display became one of the most used features of the Web. The Web grew fast because the infrastructure was already in place: the Internet, desktop PC, home modems connected to online services such as AOL and CompuServe.
- 1994 - Netscape Navigator 1.0 was released Dec. 1994, and was given away free, soon gaining 75% of world browser market.
- 1996 - Microsoft failed to recognize the importance of the Web, but finally released the much improved browser Explorer 3.0 in the summer.

1.2.3 Trends towards Parallel Processing

From an application point of view, the mainstream of usage of computer is experiencing a trend of four ascending levels of sophistication:

- Data processing
- Information processing
- Knowledge processing
- Intelligence processing

Computer usage started with data processing, while is still a major task of today's computers. With more and more data structures developed, many users are shifting to computer roles from pure data processing to information processing. A high degree of parallelism has been found at these levels. As the accumulated knowledge bases expanded rapidly in recent years, there grew a strong demand to use computers for knowledge processing. Intelligence is very difficult to create; its processing even more so.

Todays computers are very fast and obedient and have many reliable memory cells to be qualified for data-information-knowledge processing.

Computers are far from being satisfactory in performing theorem proving, logical inference and creative thinking.

From an operating point of view, computer systems have improved chronologically in four phases:

- batch processing
- multiprogramming
- time sharing
- multiprocessing

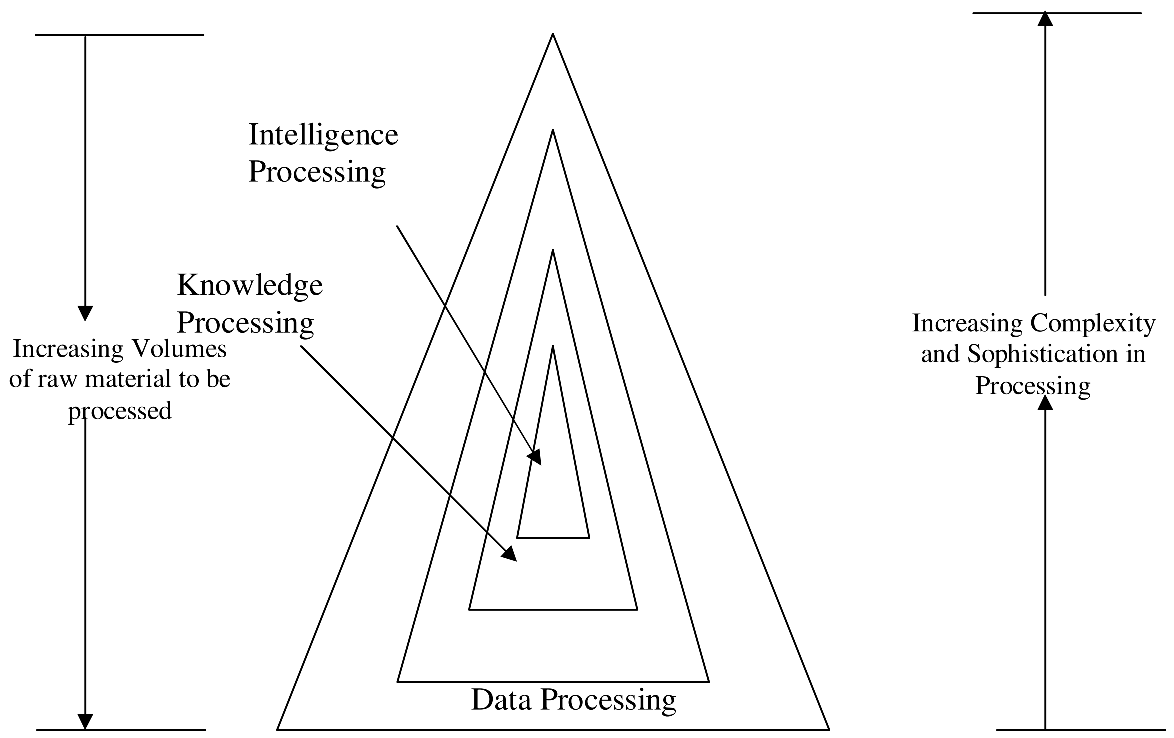

Intelligence Processing
Knowledge Processing
Increasing Volumes of raw material to be processed
Increasing Complexity and Sophistication in Processing
Data Processing

Figure 1.1 The spaces of data, information, knowledge and intelligence from the viewpoint of computer processing

In these four operating modes, the degree of parallelism increase sharply from phase to phase.
We define parallel processing as
Parallel processing is an efficient form of information processing which emphasizes the exploitation of concurrent events in the computing process. Concurrency implies parallelism, simultaneity, and pipelining. Parallel processing demands concurrent executiom of many programs in the computer. The highest level of parallel processing is conducted among multiple jobs or programs through multiprogramming, time sharing, and multiprocessing.

Parallel processing can be challenged in four programmatic levels:

- Job or program level
- Task or procedure level
- Interinstruction level
- Intrainstruction level

The highest job level is often conducted algorithmically. The lowest intra-instruction level is often implemented directly by hardware means. Hardware roles increase from high to low levels. On the other hand, software implementations increase from low to high levels.

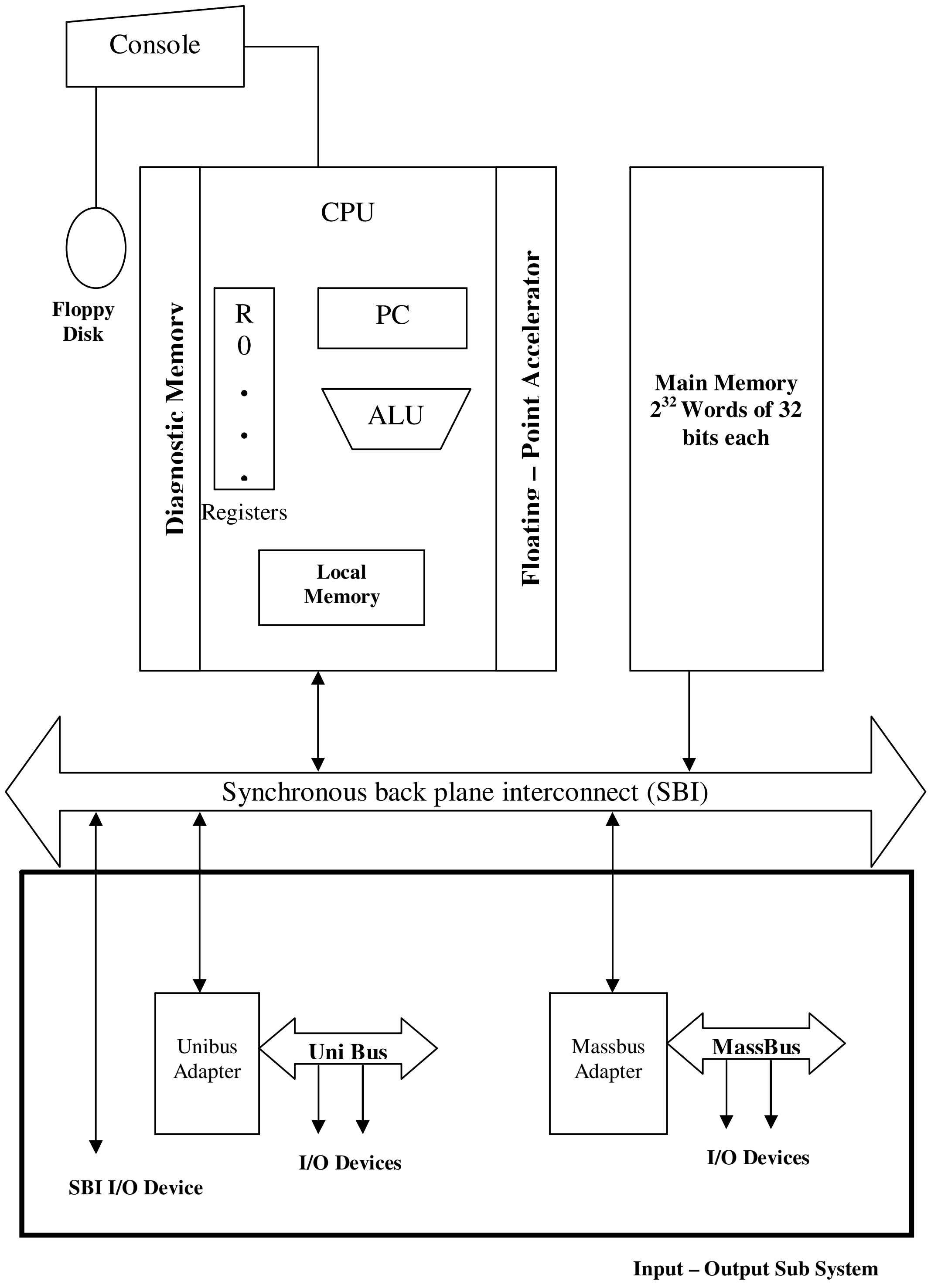


Figure 1.2 The system architecture of the super mini VAX – 11/780 microprocessor system

The trend is also supported by the increasing demand for a faster real-time, resource sharing and fault-tolerant computing environment.

It requires a broad knowledge of and experience with all aspects of algorithms, languages, software, hardware, performance evaluation and computing alternatives.

To achieve parallel processing requires the development of more capable and cost effective computer system.

1.3 Let us Sum Up

With respect to parallel processing, the general architecture trend is being shifted from conventional uniprocessor systems to multiprocessor systems to an array of processing elements controlled by one uniprocessor. From the operating system point of view computer systems have been improved to batch processing, multiprogramming, and time sharing and multiprocessing. Computers to be used in the 1990 may be the next generation and very large scale integrated chips will be used with high density modular design. More than 1000 mega float point operation per second are expected in these future supercomputers. The evolution of computer systems helps in learning the generations of computer systems.

1.4 Lesson-end Activities

1. Discuss the evolution and various generations of computer systems.
2. Discuss the trends in mainstream computer usage.

1.5 Points for Discussions

- The first generation used Vacuum Tubes (1940 – 1950s) to discrete diodes to transistors (1950 – 1960s), to small and medium scale integrated circuits (1960 – 1970s) and to very large scale integrated devices (1970s and beyond).

1.6 References

1. Advanced Computer Architecture and Parallel Processing by Hesham El-Rewini M. Abd-El-Barr Copyright © 2005 by John Wiley & Sons, Inc.
2. www.cs.indiana.edu/classes

Lesson 2 : Parallelism in Uniprocessor Systems

Contents:

2.0 Aims and Objectives

The main aim of this lesson is to know the architectural concepts of Uniprocessor systems. The development of Uniprocessor system will be introduced categorically.

2.1 Introduction

The typical Uniprocessor system consists of three major components: the main memory, the Central processing unit (CPU) and the Input-output (I/O) sub-system. The CPU contains an arithmetic and logic unit (ALU) with an optional floating-point accelerator, and some local cache memory with an optional diagnostic memory. The CPU, the main memory and the I/O subsystems are all connected to a common bus, the synchronous backplane interconnect (SBI) through this bus, all I/O device scan communicate with each other, with the CPU, or with the memory.

A number of parallel processing mechanisms have been developed in uniprocessor computers and they are identified as multiplicity of functional units, parallelism and pipelining within the CPU, overlapped CPU and I/O operations, use of a hierarchical memory system, multiprogramming and time sharing, multiplicity of functional units.

2.2 Parallelism in Uniprocessor Systems

A typical uniprocessor computer consists of three major components: the main memory, the central processing unit (CPU), and the input-output (I/O) subsystem.
The architectures of two commercially available uniprocessor computers are given below to show the possible interconnection of structures among the three subsystems.
There are sixteen 32-bit general purpose registers, one of which serves as the program
Counter (pc).there is also a special CPU status register containing information about the current state of the processor and of the program being executed. The CPU contains an arithmetic and logic unit (ALU) with an optional floating-point accelerator, and some local cache memory with an optional diagnostic memory.

2.2.1 Basic Uniprocessor Architecture

The CPU, the main memory and the I/O subsystems are all connected to a common bus, the synchronous backplane interconnect (SBI) through this bus, all I/O device scan communicate with each other, with the CPU, or with the memory. Peripheral storage or I/O devices can be connected directly to the SBI through the unibus and its controller or through a mass bus and its controller.

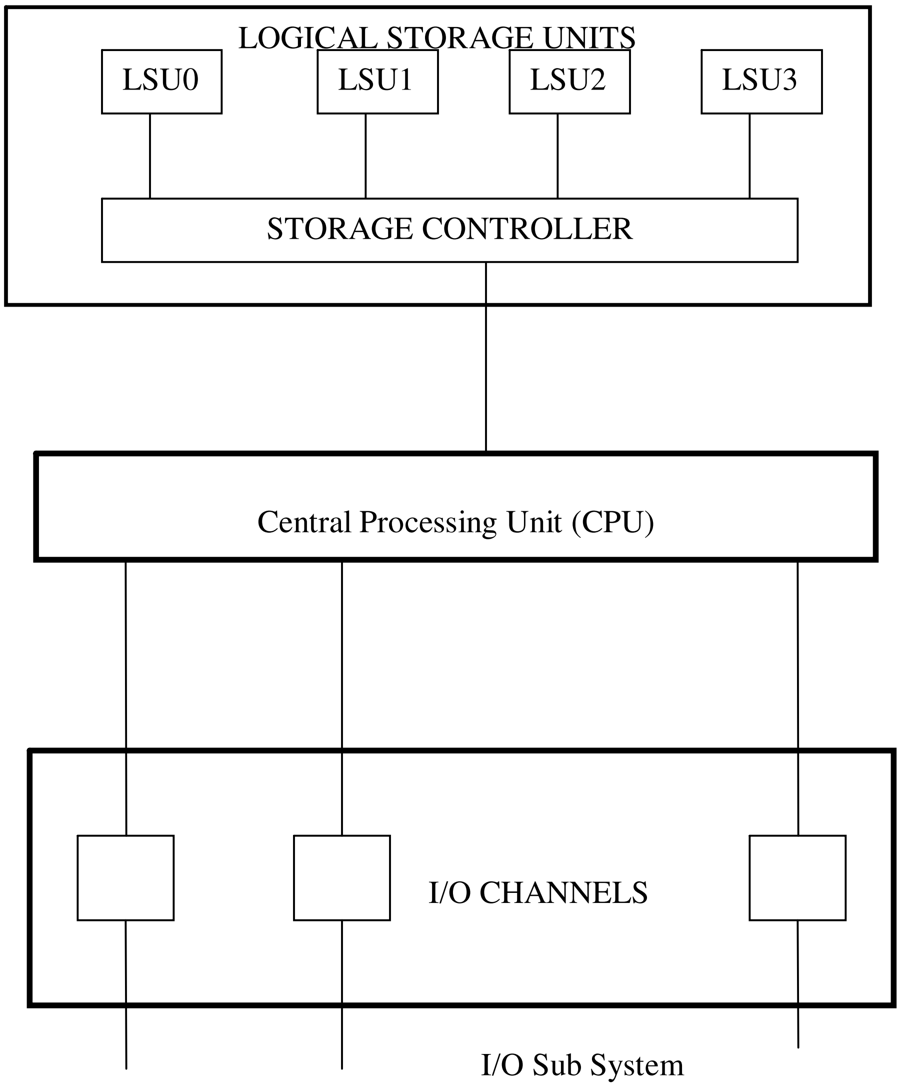


Figure 2.1 The System Architecture of the mainframe IBM System

The CPU contains the instruction decoding and execution units as well as a cache. Main memory is divided into four units, referred to as logical storage units that are four-way interleaved. The storage controller provides mutltiport connections between the CPU and the four LSUs. Peripherals are connected to the system via high speed I/O channels which operate asynchronously with the CPU.

2.2.2 Parallel Processing Mechanism

A number of parallel processing mechanisms have been developed in uniprocessor computers.
We identify them in the following six categories:

- multiplicity of functional units
- parallelism and pipelining within the CPU
- overlapped CPU and I/O operations
- use of a hierarchical memory system
- multiprogramming and time sharing
- multiplicity of functional units

Multiplicity of Functional Units

The early computer has only one ALU in its CPU and hence performing a long sequence of ALU instructions takes more amount of time. The CDC-6600 has 10 functional units built into its CPU.

These 10 units are independent of each other and may operate simultaneously.
A score board is used to keep track of the availability of the functional units and registers being demanded. With 10 functional units and 24 registers available, the instruction issue rate can be significantly increased.

Another good example of a multifunction uniprocessor is the IBM 360/91 which has 2 parallel execution units. One for fixed point arithmetic and the other for floating point arithmetic. Within the floating point E-unit are two functional units:one for floating point add- subtract and other for floating point multiply – divide. IBM 360/91 is a highly pipelined, multifunction scientific uniprocessor.

Parallelism And Pipelining Within The Cpu

Parallel adders, using such techniques as carry-look ahead and carry –save, are now built into almost all ALUs. This is in contrast to the bit serial adders used in the first generation machines. High speed multiplier recording and convergence division are techniques for exploring parallelism and the sharing of hardware resources for the functions of multiply and divide. The use of multiple functional units is a form of parallelism with the CPU.
Various phases of instructions executions are now pipelined, including instruction fetch,decode,operand fetch, arithmetic logic execution, and store result.

Overlapped CPU and I/O Operations

I/O operations can be performed simultaneously with the CPU competitions by using separate I/O controllers, channels, or I/O processors.
The direct memory access (DMA) channel can be used to provide direct information transfer between the I/O devices and the main memory. The DMA is conducted on a cycle stealing basis, which is apparent to the CPU.

Use of Hierarchical Memory System

The CPU is 1000 times faster than memory access. A hierarchical memory system can be used to close up the speed gap. The hierarchical order listed is

- registers
- Cache
- Main Memory
- Magnetic Disk
- Magnetic Tape

The inner most level is the register files directly addressable by ALU.
Cache memory can be used to serve as a buffer between the CPU and the main memory. Virtual memory space can be established with the use of disks and tapes at the outer levels.

Balancing Of Subsystem Bandwidth

CPU is the fastest unit in computer. The bandwidth of a system is defined as the number of operations performed per unit time. In case of main memory the memory bandwidth is measured by the number of words that can be accessed per unit time.

Bandwidth Balancing Between CPU and Memory

The speed gap between the CPU and the main memory can be closed up by using fast cache memory between them. A block of memory words is moved from the main memory into the cache so that immediate instructions can be available most of the time from the cache.

Bandwidth Balancing Between Memory and I/O Devices

Input-output channels with different speeds can be used between the slow I/O devices and the main memory. The I/O channels perform buffering and multiplexing functions to transfer the data from multiple disks into the main memory by stealing cycles from the CPU.

Multiprogramming

Within the same interval of time, there may be multiple processes active in a computer, competing for memory, I/O and CPU resources. Some computers are I/O bound and some are CPU bound. Various types of programs are mixed up to balance bandwidths among functional units.
Example
Whenever a process P1 is tied up with I/O processor for performing input output operation at the same moment CPU can be tied up with an process P2. This allows simultaneous execution of programs. **The interleaving of CPU and I/O operations among several programs is called as Multiprogramming.**

Time-Sharing

The mainframes of the batch era were firmly established by the late 1960s when advances in semiconductor technology made the solid-state memory and integrated circuit feasible. These advances in hardware technology spawned the minicomputer era. They were small, fast, and inexpensive enough to be spread throughout the company at the divisional level.

Multiprogramming mainly deals with sharing of many programs by the CPU. Sometimes high priority programs may occupy the CPU for long time and other programs are put up in queue. This problem can be overcome by a concept called as Time sharing in which every process is allotted a time slice of CPU time and thereafter after its respective time slice is over CPU is allotted to the next program if the process is not completed it will be in queue waiting for the second chance to receive the CPU time.

2.3 Let us Sum Up

The architectural design of Uniprocessor systems has been discussed with the help of 2 examples system architecture of the supermini VAX-11/780 Uniprocessor system. And System Architecture of the mainframe IBM system 370/Model 168 Uniprocessor computer. Various components such as main memory, Unibus Adapter, mass Bus adapter SBI I/O device have been discussed.

A number of parallel processing mechanisms have been developed in Uniprocessor computers and the categorization made to understand various parallelism.

2.4 Lesson-end Activities

1. Illustrate how parallelism can be implemented in uniprocessor architecture.
2. How system bandwidth can be balanced? Discuss.

2.5 Points for Discussions

The CPU, the main memory and the I/O subsystems are all connected to a common bus, the synchronous backplane interconnect (SBI) through this bus, all I/O device scan communicate with each other, with the CPU, or with the memory. Peripheral storage or I/O devices can be connected directly to the SBI through the unibus and its controller or through a mass bus and its controller.

The hierarchical order of memory systems are listed

- registers
- Cache
- Main Memory
- Magnetic Disk
- Magnetic Tape

Band Width: The bandwidth of a system is defined as the number of operations performed per unit time.

The interleaving of CPU and I/O operations among several programs is called as Multiprogramming.

Time sharing is mechanism in which every process is allotted a time slice of CPU time and thereafter after its respective time slice is over CPU is allotted to the next program if the process is not completed it will be in queue waiting for the second chance to receive the CPU time.

2.6 References

- Parallel Processing Computers – Hayes
- Computer Architecture and Parallel Processing – Kai Hwang
- Operating Systems - Donovan

Lesson 3: Parallel Computer Structures

Contents:

3 Aims and Objectives

The main objective of this lesson is to learn the parallel computers three architectural configurations called pipelined computers, Array Processors, and Multiprocessor Systems.

3.1 Introduction

Parallel computers are those systems that emphasize parallel processing. The process of executing an instruction in a digital computer involves 4 major steps namely Instruction fetch, Instruction decoding, Operand fetch, Execution.
In a pipelined computer successive instructions are executed in an overlapped fashion.
In a non pipelined computer these four steps must be completed before the next instructions can be issued.

An array processor is a synchronous parallel computer with multiple arithmetic logic units called processing elements (PE) that can operate in parallel in lock step fashion.
By replication one can achieve spatial parallelism. The PEs are synchronized to perform the same function at the same time.

A basic multiprocessor contains two or more processors of comparable capabilities. All processors share access to common sets of memory modules, I/O channels and peripheral devices.

3.2 Parallel Computer Structures

Parallel computers are those systems that emphasize parallel processing. We divide parallel computers into three architectural configurations:

- Pipeline computers
- Array processors
- multiprocessors

3.2.1 Pipeline Computers

The process of executing an instruction in a digital computer involves 4 major steps

- Instruction fetch

- Instruction decoding
- Operand fetch
- Execution

In a pipelined computer successive instructions are **executed in an overlapped fashion.**
In a non pipelined computer these four steps **must be completed before the next instructions can be issued.**

- Instruction fetch : Instruction is fetched from the main memory
- Instruction decoding: Identifying the operation to be performed.
- Operand Fetch: If any operands is needed is fetched.
- Execution : Execution of the Arithmetic and logical operation

An instruction cycle consists of multiple pipeline cycles. The flow of data (input operands, intermediate results and output results) from stage to stage is triggered by a common clock of the pipeline. The operations of all stages are triggered by a common clock of the pipeline.
For non pipelined computer, it takes four pipeline cycles to complete one instruction. Once a pipe line is filled up, an output result is produced from the pipeline on each cycle. The instruction cycle has been effectively reduced to 1/4th of the original cycle time by such overlapped execution.

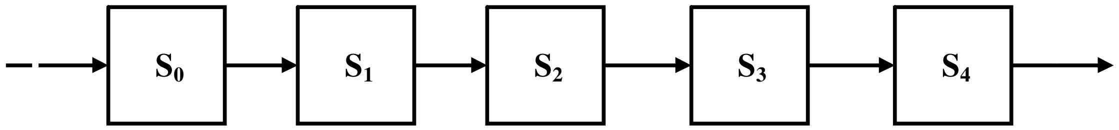
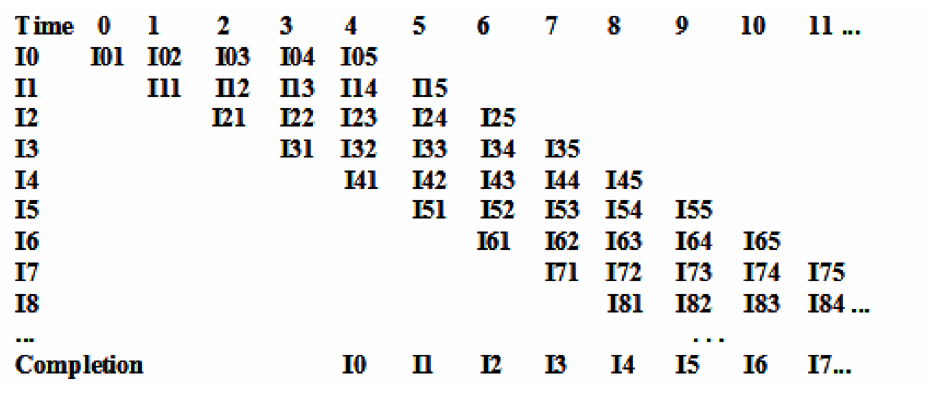


Figure 3.1 A pipelined Processor



Time	0	1	2	3	4	5	6	7	8	9	10	11 ...
I0	I01	I02	I03	I04	I05							
I1		I11	I12	I13	I14	I15						
I2			I21	I22	I23	I24	I25					
I3				I31	I32	I33	I34	I35				
I4					I41	I42	I43	I44	I45			
I5						I51	I52	I53	I54	I55		
I6							I61	I62	I63	I64	I65	
I7								I71	I72	I73	I74	I75
I8									I81	I82	I83	I84 ...
...										...		
Completion					I0	I1	I2	I3	I4	I5	I6	I7...

Figure 3.2 Space Diagram for a Pipelined Processor

3.2.2 Array Processors

An array processor is a synchronous parallel computer with multiple arithmetic logic units called processing elements (PE) that can operate in parallel in lock step fashion.

By replication one can achieve spatial parallelism. The PEs are synchronized to perform the same function at the same time.

Scalar and control type of instructions are directly executed in the control unit (CU). Each PE consists of an ALU registers and a local memory. The PEs are interconnected by a data-routing network. Vector instructions are broadcasted to the PEs for distributed execution over different component operands fetched directly from local memories. Array processors designed with associative memories are called as associative processors.

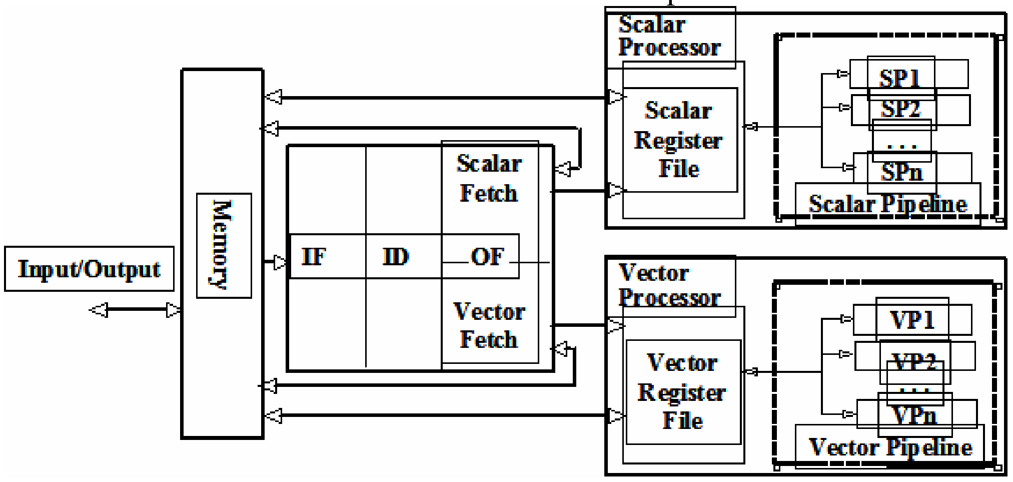


Figure 3.3 Functional structure of a modern pipeline computer with scalar and vector capabilities

3.2.3 Multiprocessor Systems

A basic multiprocessor contains two or more processors of comparable capabilities. All processors share access to common sets of memory modules, I/O channels and peripheral devices. The entire system must be controlled by a single integrated operating system providing interactions between processors and their programs at various levels.

Multiprocessor hardware system organization is determined by the interconnection structure to be used between the memories and processors. Three different interconnection are

- Time shared Common bus
- Cross Bar switch network
- Multiport memories

3.3 Let us Sum Up

A pipeline computer performs overlapped computations to exploit temporal parallelism.

An array processor uses multiple synchronized arithmetic and logic units to achieve spatial parallelism.

A multiprocessor system achieves asynchronous parallelism through a set of interactive processors with shared resources.

3.4 Lesson-end Activities

1.Discuss how instructions are executed in a pipelined processor.
2.What are the 2 methods in which array processors can be implemented? Discuss.

3.5 Points for Discussions

The fundamental difference between an array processor and a multiprocessor system is that the processing elements in an array processor operate synchronously but processors in a multiprocessor systems may not operate synchronously.

3.6 References

From Net : Tarek A. El-Ghazawi, Dept. of Electrical and Computer Engineering, The George Washington University

Lesson 4 : Architectural Classification Schemes

Contents:

4 Aims and Objectives

The main objective is to learn various architectural classification schemes, Flynn's classification, Feng's classification, and Handler's Classification.

4.1 Introduction

The Flynn's classification scheme is based on the multiplicity of instruction streams and data streams in a computer system. Feng's scheme is based on serial versus parallel processing. Handler's classification is determined by the degree of parallelism and pipelining in various subsystem levels.

4.2 Architectural Classification Schemes

4.2.1 Flynn's Classification

The most popular taxonomy of computer architecture was defined by Flynn in 1966. Flynn's classification scheme is based on the notion of a stream of information. Two types of information flow into a processor: instructions and data. The instruction stream is defined as the sequence of instructions performed by the processing unit. The data stream is defined as the data traffic exchanged between the memory and the processing unit.
According to Flynn's classification, either of the instruction or data streams can be single or multiple.
Computer architecture can be classified into the following four distinct categories:

- single-instruction single-data streams (SISD);
- single-instruction multiple-data streams (SIMD);
- multiple-instruction single-data streams (MISD); and

- multiple-instruction multiple-data streams (MIMD).

Conventional single-processor von Neumann computers are classified as SISD systems. Parallel computers are either SIMD or MIMD. When there is only one control unit and all processors execute the same instruction in a synchronized fashion, the parallel machine is classified as SIMD. In a MIMD machine, each processor has its own control unit and can execute different instructions on different data. In the MISD category, the same stream of data flows through a linear array of processors executing different instruction streams. In practice, there is no viable MISD machine; however, some authors have considered pipelined machines (and perhaps systolic-array computers) as examples for MISD. An extension of Flynn's taxonomy was introduced by D. J. Kuck in 1978. In his classification, Kuck extended the instruction stream further to single (scalar and array) and multiple (scalar and array) streams. The data stream in Kuck's classification is called the execution stream and is also extended to include single (scalar and array) and multiple (scalar and array) streams. The combination of these streams results in a total of 16 categories of architectures.

4.2.1.1 SISD Architecture

- A serial (non-parallel) computer
- Single instruction: only one instruction stream is being acted on by the CPU during any one clock cycle
- Single data: only one data stream is being used as input during any one clock cycle
- Deterministic execution
- This is the oldest and until recently, the most prevalent form of computer
- Examples: most PCs, single CPU workstations and mainframes

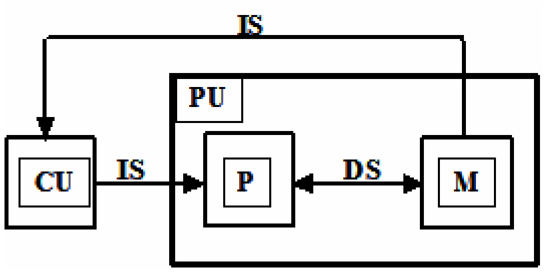
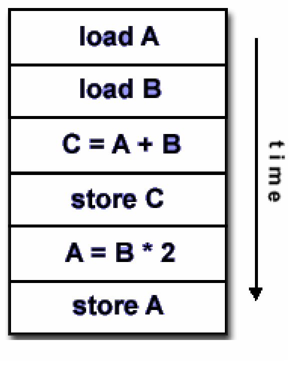


Figure 4.1 SISD COMPUTER



4.2.1.2 SIMD Architecture

- A type of parallel computer
- Single instruction: All processing units execute the same instruction at any given clock cycle
- Multiple data: Each processing unit can operate on a different data element
- This type of machine typically has an instruction dispatcher, a very high-bandwidth internal network, and a very large array of very small-capacity instruction units.
- Best suited for specialized problems characterized by a high degree of regularity, such as image processing.
- Synchronous (lockstep) and deterministic execution
- Two varieties: Processor Arrays and Vector Pipelines
- Examples:
    - Processor Arrays: Connection Machine CM-2, Maspar MP-1, MP-2
    - Vector Pipelines: IBM 9000, Cray C90, Fujitsu VP, NEC SX-2, Hitachi S820

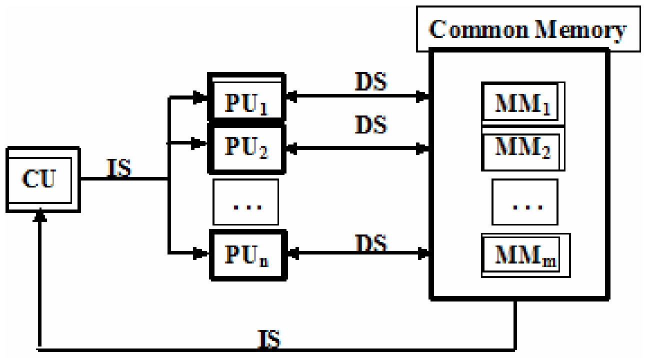
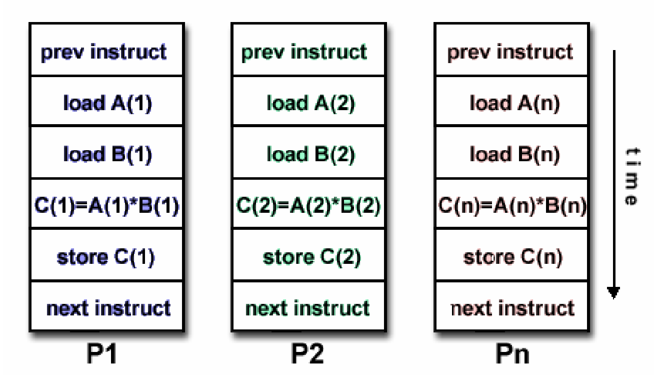


Figure 4.2 SIMD COMPUTER

CU-control unit
PU-processor unit
MM-memory module
SM-Shared memory
IS-instruction stream
DS-data stream



4.2.1.3 MISD Architecture

There are n processor units, each receiving distinct instructions operating over the same data streams and its derivatives. The output of one processor become input of the other in the macro pipe. No real embodiment of this class exists.

- A single data stream is fed into multiple processing units.
- Each processing unit operates on the data independently via independent instruction streams.
- Few actual examples of this class of parallel computer have ever existed. One is the experimental Carnegie-Mellon C.mmp computer (1971).
- Some conceivable uses might be:
    - multiple frequency filters operating on a single signal stream
    - multiple cryptography algorithms attempting to crack a single coded message.

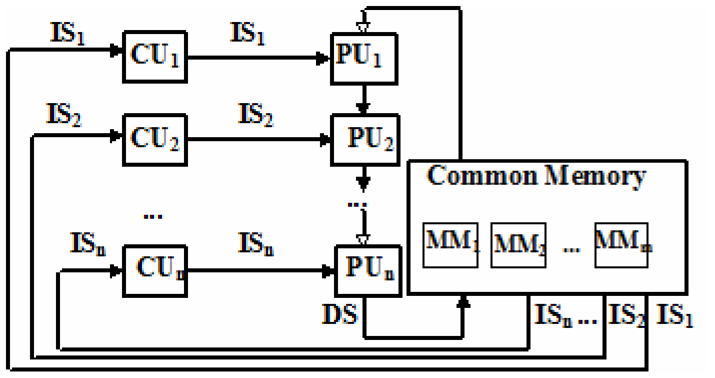
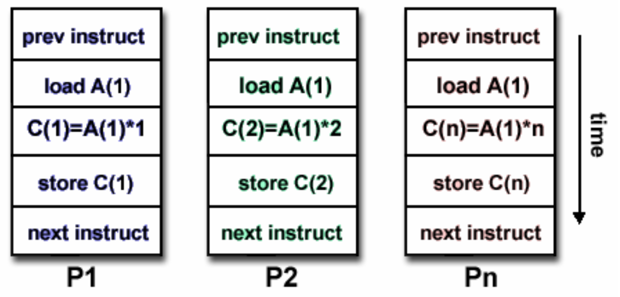


Figure 4.3 MISD COMPUTER



4.2.1.4 MIMD Architecture

Multiple-instruction multiple-data streams (MIMD) parallel architectures are made of multiple processors and multiple memory modules connected together via some interconnection network. They fall into two broad categories: shared memory or message passing. Processors exchange information through their central shared memory in shared memory systems, and exchange information through their interconnection network in message passing systems.

- Currently, the most common type of parallel computer. Most modern computers fall into this category.
- Multiple Instruction: every processor may be executing a different instruction stream
- Multiple Data: every processor may be working with a different data stream
- Execution can be synchronous or asynchronous, deterministic or non-deterministic
- Examples: most current supercomputers, networked parallel computer "grids" and multi-processor SMP computers - including some types of PCs.

A shared memory system typically accomplishes interprocessor coordination through a global memory shared by all processors. These are typically server systems that communicate through a bus and cache memory controller.
A message passing system (also referred to as distributed memory) typically combines the local memory and processor at each node of the interconnection network. There is no global memory, so it is necessary to move data from one local memory to another by means of message passing.

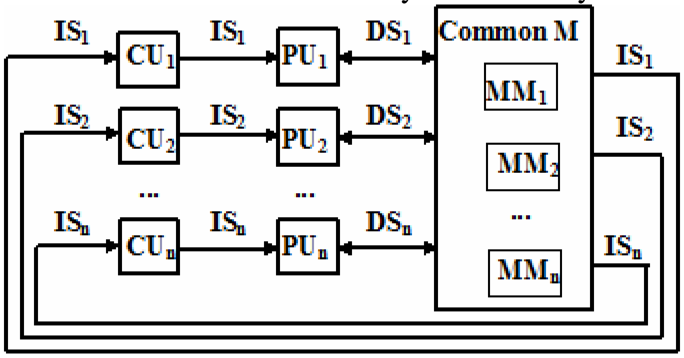
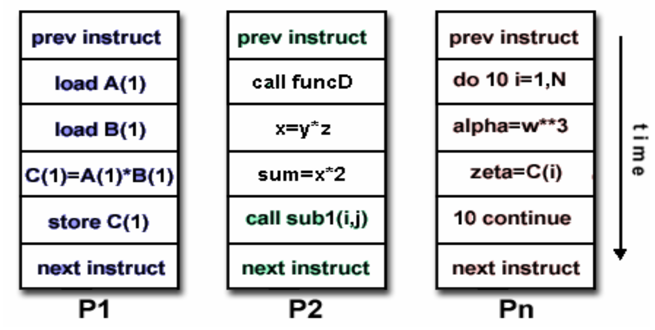


Figure 4.4 MIMD COMPUTER



	Computer Class	Computer System Models
1.	SISD	IBM 701, IBM 1620, IBM 7090, PDP VAX11/ 780
2.	SISD (With multiple functional units)	IBM360/91 (3); IBM 370/168 UP
3.	SIMD (Word Slice Processing)	Illiac – IV ; PEPE
4.	SIMD (Bit Slice	STARAN; MPP; DAP

	processing)	
5.	**MIMD (Loosely Coupled)**	**IBM 370/168 MP; Univac 1100/80**
6.	**MIMD(Tightly Coupled)**	**Burroughs- D – 825**

Table 4.1 Flynn's Computer System Classification

4.2.2 Feng's Classification

Tse-yun Feng suggested the use of degree of parallelism to classify various computer architectures.

Serial Versus Parallel Processing

The maximum number of binary digits that can be processed within a unit time by a computer system is called the maximum parallelism degree P.

A bit slice is a string of bits one from each of the words at the same vertical position.
There are 4 types of methods under above classification

- Word Serial and Bit Serial (WSBS)
- Word Parallel and Bit Serial (WPBS)
- Word Serial and Bit Parallel(WSBP)
- Word Parallel and Bit Parallel (WPBP)

WSBS has been called bit parallel processing because one bit is processed at a time.
WPBS has been called bit slice processing because m-bit slice is processes at a time.
WSBP is found in most existing computers and has been called as Word Slice processing because one word of n bit processed at a time.
WPBP is known as fully parallel processing in which an array on n x m bits is processes at one time.

Mode	**Computer Model**	**Degree of parallelism**
WSPS **N = 1** **M = 1**	**The 'MINIMA'**	**(1,1)**
WPBS **N=1** **M>1**	**STARAN** **MPP** **DAP**	**(1,256)** **(1,16384)** **(1,4096)**
WSBP **n>1** **m=1** **(Word Slice Processing)**	**IBM 370/168 UP** **CDC 6600** **Burrough 7700** **VAX 11/780**	**(64,1)** **(60,1)** **(48,1)** **(16/32,1)**
WPBP **n>1** **m>1** **(fully parallel Processing)**	**Illiav IV**	**(64,64)**

Table 4.2 Feng's Computer Classification

4.2.3 Handler's Classification

Wolfgang Handler has proposed a classification scheme for identifying the parallelism degree and pipelining degree built into the hardware structure of a computer system. He considers at three subsystem levels:

- Processor Control Unit (PCU)
- Arithmetic Logic Unit (ALU)
- Bit Level Circuit (BLC)

Each PCU corresponds to one processor or one CPU. The ALU is equivalent to Processor Element (PE). The BLC corresponds to combinational logic circuitry needed to perform 1 bit operations in the ALU.
A computer system C can be characterized by a triple containing six independent entities

T(C) = <K x K', D x D', W x W' >

Where K = the number of processors (PCUs) within the computer
D = the number of ALUs under the control of one CPU
W = the word length of an ALU or of an PE
W' = The number of pipeline stages in all ALUs or in a PE
D' = the number of ALUs that can be pipelined
K' = the number of PCUs that can be pipelined

4.3 Let us Sum Up

The architectural classification schemes has been presented in this lesson under 3 different classifications Flynn's, Feng's and Handler's. The instruction format representation has also be given for Flynn's scheme and examples of all classifications has been discussed.

4.4 Lesson-end Activities

1.With examples, explain Flynn's computer system classification.
2.Discuss how parallelism can be achieved using Feng's and Handler's classification.

4.5 Points for Discussions

Single Instruction, Single Data stream (SISD)
A sequential computer which exploits no parallelism in either the instruction or data streams. Examples of SISD architecture are the traditional uniprocessor machines like a PC or old mainframes.
Single Instruction, Multiple Data streams (SIMD)
A computer which exploits multiple data streams against a single instruction stream to perform operations which may be naturally parallelized. For example, an array processor or GPU.

Multiple Instruction, Single Data stream (MISD)
Unusual due to the fact that multiple instruction streams generally require multiple data streams to be effective..

Multiple Instruction, Multiple Data streams (MIMD)
Multiple autonomous processors simultaneously executing different instructions on different data. Distributed systems are generally recognized to be MIMD architectures; either exploiting a single shared memory space or a distributed memory space.

4.6 References

- http://en.wikipedia.org/wiki/Multiprocessing
- Free On-line Dictionary of Computing, which is licensed under the GFDL.

Lesson 5 : Parallel Processing Applications

Contents:

5.0 Aims and Objectives

The main objective of this lesson is introducing some representative applications of high-performance computers. This helps in knowing the computational needs of important applications.

5.1 Introduction

Fast and efficient computers are in high demand in many scientific, engineering and energy resource, medical, military, artificial intelligence and the basic research areas. Large scale computations are performed in these application areas. Parallel processing computers are needed to meet these demands.

5.2 Parallel Processing Applications

Fast and efficient computers are in high demand in many scientific, engineering, energy resource, medical, military, AI, and basic research areas.
Parallel processing computers are needed to meet these demands.
Large scale scientific problem solving involves three interactive disciplines;

- Theories
- Experiments
- Computations

Theoretical scientists develop mathematical models that computer engineers solve numerically, the numerical results then suggest new theories. Experimental science provides data for computational science and the latter can model processes that are hard to approach in the laboratory.
Computer Simulation has several advantages:

- It is far cheaper than physical experiments
- It can solve much wider range of problems that specific laboratory equipments can

Computational approaches are only limited by computer speed and memory capacity, while physical experiments have many special practical constraints.

5.2.1 Predictive Modelling and Simulations

Predictive modelling is done through extensive computer simulation experiments, which often involve large-scale computations to achieve the desired accuracy and turnaround time.

A) Numerical Weather Forecasting

Weather modelling is necessary for short range forecasts and do long range hazard predictions such as flood, drought and environment pollutions.

B) Oceanography and Astrophysics

Since ocean can store and transfer heat and exchange it with the atmosphere. Understanding of oceans helps us in

- Climate Predictive Analysis
- Fishery Management
- Ocean Resource Exploration
- Costal Dynamics and Tides

C) Socioeconomics and Government Use

Large computers are in great demand in the areas of econometrics, social engineering, government census, crime control, and the modelling of the world's economy for the year 2000.

5.2.2 Engineering Design and Automation

Fast computers have been in high demand for solving many engineering design problems, such as the finite element analysis needed for structural designs and wind tunnel experiments for aero dynamics studies.

A) Finite Element Analysis

The design of dams, bridges, ships, supersonic jets, high buildings, and space vehicles requires the resolution of a large system of algebraic equations.

B) Computational Aerodynamics

Large scale computers have made significant contributions in providing new technological capabilities and economies in pressing ahead with aircraft and spacecraft lift and turbulence studies.

C) Artificial Intelligence and Automation

Intelligent I/O interfaces are being demanded for future supercomputers that must directly communicate with human beings in images, speech, and natural languages. The various intelligent functions that demand parallel processing are:

- Image Processing
- Pattern Recognition
- Computer Vision
- Speech Understanding
- Machien Interface
- CAD/CAM/CAI/OA
- Intelligent Robotics
- Expert Computer Systems
- Knowledge Engineering

D) Remote Sensing Applications

Computer analysis of remotely sensed earth resource data has many potential applications in agriculture, forestry, geology, and water resource.

5.2.3 Energy Resources Exploration

Energy affects the progress of the entire economy on a global basis. Computer can play the important role in the discovery of oil and gas and the management of their recovery, in the development of workable plasma fusion energy and in ensuring nuclear reactor safety.

A) Seismic Exploration

Many oil companies are investing in the use of attached array processors or vector supercomputer for seismic data processing, which accounts for about 10 percent of the oil finding costs. Seismic explorations sets off a sonic wave by explosive or by jamming a heavy hydraulic ram into the ground about the spot are used to pick up the echoes.

B) Reservoir Modelling

Super computers are used to perform three dimensional modelling of oil fields.

C) Plasma Fusion Power

Nuclear fusion researchers to use a computer 100 times more powerful than any existing one to model the plasma dynamics in the proposed Tokamak fusion power generator.

D) Nuclear Reactor Safety

Nuclear reactor designs and safety control can both be aided by computer simulation studies. These studies attempt to provide for :

- On-Line analysis of reactor conditions
- Automatic control for normal and abnormal operations
- Quick assessment of potential mitigation accidents

5.2.4 Medical, Military and Basic research

Fast computers are needed in the computer assisted tomography, artificial heart design, liver diagnosis, brain damage estimation, and genetic engineering studies. Military defence needs to use supercomputers for weapon design, effects, simulation and other electronic warfare.

A) Computer Aided Tomography

The human body can be modelled by computer assisted tomography (CAT) scanning.

B) Genetic Engineering

Biological system can be simulated on super computers.

C) Weapon Research and Defence

Military Research agencies have used the majority of existing supercomputers.

D) Basic Research Problem

Many of the aforementioned application areas are related to basic scientific research.

5.3 Let us Sum Up

The above details are some of the parallel processing applications, without using super computers, many of these challenges to advance human civilization could be hardly realized.

5.4 Lesson-end Activities

1. How parallel processing can be applied in Engineering design & Simulation? Give examples.
2. How parallel processing can be applied in Medicine & military research? Give examples.

5.5 Points for discussions

Computer Simulation has several advantages:

- It is far cheaper than physical experiments.
- It can solve much wider range of problems that specific laboratory equipments can.

Computational approaches are only limited by computer speed and memory capacity, while physical experiments have many special practical constraints.
Various Parallel Processing Applications are

- Predictive Modelling and Simulations
- Engineering Design and Automation
- Energy Resources Exploration
- Medical, Military and Basic research

5.6 References

- Materials of super computer applications can be found in Rodrique et al (1980)

UNIT – II

Lesson 6 : Solving Problems in parallel , Utilizing Temporal Parallelism

Contents:

6.0 Aims and Objectives

The main objective of this lesson is given by an example of how a simple job can be solved in parallel in many different ways.

6.1 Introduction

The simple example given here is correction of answer sheets by teachers and it explains the concept of parallelism and how tasks are allocated to processors for getting maximum efficiency in solving problems in parallel. The term temporal means pertaining to time and this method breaks up a job into a set of tasks to be executed overlapped in time and thus it is temporal parallelism. In case of data parallelism the input data is divided into independent sets and processed simultaneously.

6.2 Utilizing Temporal Parallelism

The term temporal means pertaining to time. An example is considered suppose 1000 candidates appear in an examination. There are answers to 4 questions in each answer book and a teacher has to correct the answers based upon the following instructions.

Instructions given a teacher to correct an answer book

Step 1 : Take an answer book from the pile of answer books.
Step 2 : Correct the answer to Q1 namely A1.
Step 3 : Repeat for Q2, Q3 and Q4 as A2, A3 and A4
Step 4 : Add marks given for each answer.
Step 5 : Put answer books in a pile of corrected answer books.

Step 6 : Repeat steps 1 to 5 until no more answer books are left in the input.

If a teacher takes 20 minutes to correct a paper, then 20,000 minutes will be taken and if to speed up the correction the following methods can be applied.

6.2.1 Method 1: Temporal Parallelism

The four teachers can sit cooperatively to correct an answer book. The first teacher corrects answer for Q1 and passes it to the second teacher who corrects for Q2. (The first teacher immediately takes up the second paper) and passes it to the third teacher who corrects for Q3 and passes it to the fourth teacher who corrects for Q4. After correction of 4 papers all the teacher will be busy.
Time taken to correct A1= Time taken to correct A2= Time taken to correct A3=
Time taken to correct A4 = 5 minutes, then time taken to correct one single paper will be 5 minutes. The total time taken to correct 1000 papers will be 20 + (999 * 5) = 51015 minutes. This is about 1/4th of the time taken by one teacher.

This method is called as parallel processing as 4 teacher work in parallel. The type of parallelism used in this method is called as temporal parallelism. The term temporal means pertaining to time. This method breaks up the job into set of tasks to be executed overlapped in time it is said to use temporal parallelism. It is also known as assembly line processing or pipeline processing or vector processing.

This method of parallel processing is appropriate if
- The jobs to be carried out are identical.
- A job can be divided into many independent tasks and each can be performed separately
- The timer taken for each task is same.
- The time taken to send a job from one teacher to the next is negligible compared to the time needed to do the task
- The number of tasks is much smaller compared to the total number of jobs to be done.

Assuming
Let the number of jobs = n
Let the time to do a job = p. let each job be divisible into k tasks and let each task be done by a different teacher
Let the time for doing each task = plk.
Time to complete n hobs with no pipelining processing = np.
Time taken to complete n jobs with pipelining organization of k teacher =

$$=P+(n-1)\,\frac{p}{k}$$

$$= p\,\frac{(k+n-1)}{k}$$

speedup due to pipeline processing = $\dfrac{np}{P(k+n-1)/k} = \dfrac{k}{1 + [(k-1) / n]}$

The main problems encountered in implementing this method are :

- **Synchronization :** identical time should be taken for doing each task in the pipeline so that a job can flow smoothly in the pipeline without holdup.
- **Bubbles in pipeline :** if some tasks are absent then idle time will be encountered.
- **Fault tolerance :** the system does not have tolerate faults. If one teacher goes out then the entire pipeline is upset.
- **Inter task communication:** The time to pass answer books between teachers in the pipeline should be much smaller compared to the time taken to correct an answer by a teacher.
- **Scalability:** The number of teacher working n the pipeline cannot be increased beyond a certain limit.

6.2.2 Utilizing Data Parallelism

6.2.2.1 Method 2: Data Parallelism

The answer books are divided into four piles and each pile is given to a teacher. Assume each teacher takes 20 minutes to correct an answer book so that every teacher gets 250 answer books and the time taken to correct all the 1000 papers is 5000 minutes. This type of parallelism is called as data parallelism as the input data is divided into independent sets and processed simultaneously.

P1 to P250 allotted to T1
P251 to P500 allotted to T2
P501 to P750 allotted to T3
P751 to P1000 allotted to T4

Assuming

Let the number of jobs = n
Let the time to do a job = p
Let there be k teachers
Let the time to distribute the jobs to k teachers be kq.
Time to complete n jobs by single teacher = np
Time taken to complete n jobs by k teachers

$$=kq \frac{np}{k}$$

Speedup due to parallel processing = $\dfrac{k}{1 + (k^2q/np)}$

The speed is not directly proportional to the number of teachers as the time to distribute jobs to teachers (which is an unavoidable overhead) increases as the number of teacher is increased.

The main advantages:

- There is no synchronization required between teachers. Each teacher can correct papers at their own place.
- The problem of bubble is absent
- It is more fault tolerant. Each teacher can act as per their own way.
- There is no communication required between teachers as teacher works independently.

The main disadvantages:

- The assignment of jobs to each teacher is pre-decided. This is called a static assignment.
- The set of jobs must be partitionable into subsets of mutually independent jobs. Each subset should take the same time to complete.
- Each teacher must be capable of correcting answers to all questions but in pipeline only one question allotted for one teacher.
- The time taken to divide a set of jobs into equal subsets of jobs should be small.

6.2.2.2 Method 3 : Combined Temporal and Data Parallelism

The previous 2 methods are combined to get this method. This method almost halves the time taken by a single pipeline. Even though this method reduces the time to complete the set of jobs it also has the drawbacks of both temporal and data parallelism. This method is effective only if the number of jobs given to each pipeline is much larger that the number of stages in the pipeline.

6.2.2.3 Method 4 : Data Parallelism with Dynamic Assignment

In this method the head examiner gives one answer paper to each teacher and keeps the rest with him. All teachers simultaneously correct the paper given to them. A teacher who completes the correction goes to the head examiner for another paper, which is given to him for correction. If second teacher completes then he queues up in front of the head examiner and waits for his turn to get an answer sheet. This procedure is repeated until all the papers are corrected.

The main advantages are :

- Balancing the work assigned to each teacher dynamically as the work progresses. A teacher who completes gets another paper immediately.
- Overall time to correct paper will be minimized.
- The problem of Bubble is absent.

The main disadvantages are

- The examiner can attend only one teacher if more teachers have completed and the other teacher has to be in queue
- The head examiner can become a bottleneck. If he leaves then the teachers has to wait until he comes back again.
- The head examiner himself is idle between handing out papers.

- T is difficult to increase the number of teachers as it will increase the probability of many teachers completing their jobs simultaneously thereby leading to long queues of teachers waiting to get an answer paper.

6.2.2.4 Method 5 : Data Parallelism with quasi Dynamic Scheduling

The teacher may be given small bunch of papers for correction rather than giving one by one and he held up in queue before the head examiner.
The assignment of jobs to teacher in method 3 is static schedule and the assignment is done initially and not changed. Here the assignment is done dynamically who corrects the papers first will get more answer sheets.

6.3 Let us Sum Up

There are many ways of solving a problem in parallel. The main advantages and disadvantages with reference to each method have been discussed and this helps in choosing the method based on the situation.

6.4 Lesson-end Activities

1.What are the method available for achieving data parallelism. Elaborate.
2. What are the method available for achieving temporal parallelism. Elaborate

6.5 Points for discussions

The term temporal means pertaining to time. This method breaks up the job into set of tasks to be executed overlapped in time it is said to use temporal parallelism. It is also known as assembly line processing or pipeline processing or vector processing.

In case of data parallelism, the input data is divided into independent sets and processed simultaneously.

6.6 References

- Frenkel, K.A editor Special issue on Parallelism, Communications of ACM
- Rodrigue.G Parallel Computations, Academic Press

Lesson 7: Comparison of temporal and data parallel processing & Data parallel processing with specialized processors

Contents:

7.0 Aims and Objectives

The main objective of this lesson is to differentiate between temporal and data parallel processing and to discuss few more methods of data parallel processing with specialized processors.

7.1 Introduction

In case of temporal parallel processing jobs are divided into set of independent tasks, and they are given equal time, Bubbles in jobs leads to idling of processors, task assignment is static and it is not tolerant to processor faults.

In case of data parallelism full jobs are assigned to processing, and they may take different time and no concept of synchronization needed and it tolerant to processor faults.

Various other methods such as Specialist data parallelism in which head examiner despatches answer sheets to examiner, In coarse grained temporal parallel processing the answer sheets are divided and assigned to input tray and output tray , In case of agenda parallelism the answer book is used as an agenda of questions.

7.2 Comparison of Temporal and Data Parallel Processing

S.No	Temporal Parallel Processing (pipelining Idea)	Data Parallel processing
1	Job is divided in to set of independent tasks and tasks are assigned for processing	Full jobs are assigned for processing
2.	Tasks should take equal time. Pipeline stages thus to be synchronized	Jobs may take different times. No need to synchronize beginning of jobs.
3	Bubbles in jobs lead to idling of	Bubbles do not cause idling of

	processors	processors
4	Task assignment static	Job assignment may be static, dynamic or quasi dynamic
5	Not tolerant to processor faults	Tolerates processor faults
6	Efficient with fine grained tasks	Efficient with coarse grained tasks and quasi dynamic scheduling
7	Scales well as long as the number of data items to be processed is much larger than the number of processors in the pipeline and the time taken to communicate task from one processor to the next is negligible.	Scales well as long as the number of jobs is much greater than the number of processors and processing time is much higher than the time to distribute data to processors

7.2.1 Data Parallel Processing with Specialized Processors

Data parallel processing is more tolerant but requires each teacher to be capable of correcting answers to all questions with equal ease.

7.2.1.1 Method 6 : Specialist Data Parallelism

In this a head examiner dispatches the answer sheet to teachers.
1. Give one answer paper to T1, T2, T3, T4 (Teacher Ti corrects only the answer to question Qi)
2. When a corrected answer paper is turned check if all the questions are graded. If yes add marks and put the paper in the output pile.
3. If no, check with questions not graded.
4. For each I, if the Ai is ungraded and teacher Ti, or if any other teacher Tp is idle and answer paper remains in input pile with Ap uncorrected send it to him.
5. Repeat steps 2,3,4 until no answer paper remains in the input pile and all teachers are idle.

The main disadvantages are :
- The load is not balanced.
- If some answers take much longer time to grade than others then some of the teachers will be busy while others are idle.
- The head examiner will waste lot of time in checking which questions are not answered and which teachers are idle.
- The maximum possible speedup will not be attained

7.2.1.2 Method 7 : Coarse grained specialist temporal parallelism

Here all teachers work independently and simultaneously. Many teachers spend lot of time in waiting for other teachers to complete their work.
Procedure 2.3 Coarse grained specialist temporal processing
Answer papers are divided into 4 equal piles and put in the in-trays of each teacher. Each teacher repeats 4 times simultaneously steps 1 to 5.
For teachers Ti (I = 1 to 4) do in parallel

1. Take an answer sheet paper from in-tray.
2. Grade answer Ai to question Qi and put in out-tray
3. Repeat steps 1 and 2 till no papers left in in-tray
4. Check if teacher T(i+1) Mod 4's in-tray is empty.
5. As soon as it is empty, empty own out-tray into the in-tray of that teacher.

7.2.1.3 Method 8 : Agenda Parallelism

In this method answer book is thought as an agenda of questions to be graded. All teachers are asked to work on the first item on the agenda, grade the answer to the first questions in all papers. A head examiner gives one paper to each teacher and asks him to grade the answer A1 to Q1. When a teacher finishes this he is given another paper in which he again grades A1. When A1 of all papers are grades, then A2 is taken up by all teachers, this is repeated until all questions in all papers are graded. This is a data parallel method with dynamic schedule and fine grain tasks. A chart showing various methods of parallel processing has been depicted.

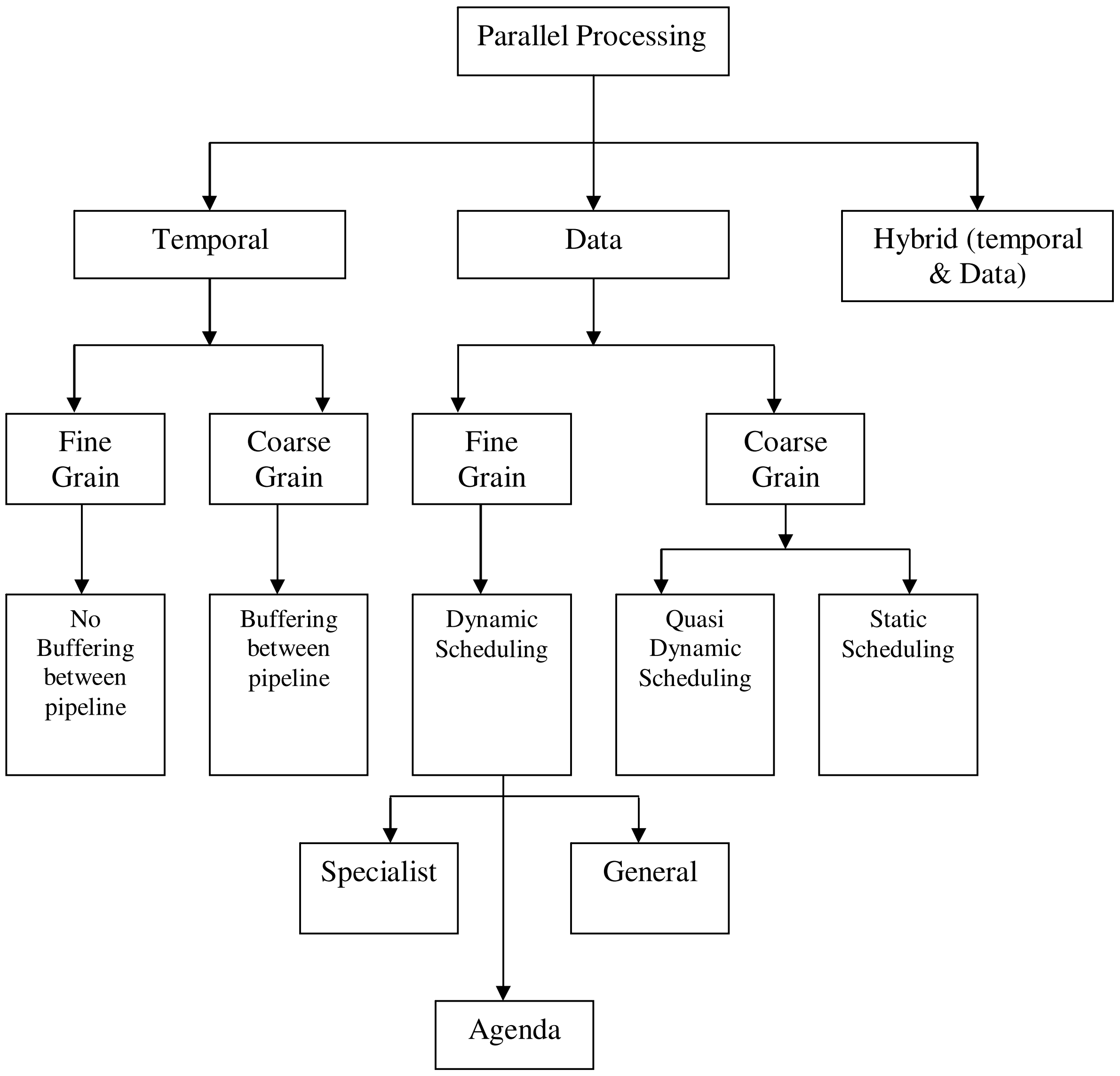


Figure 7. 1 Chart Showing Various Methods of Parallel Processing

7.3 Let us Sum Up

Comparative study of data parallel processing and temporal parallel parallelism has been made and other data processing with specialized processors has been carried out.
In method 6, a head examiner dispatches the answer sheet to teachers.
In method 7, all teachers work independently and simultaneously. Many teachers spend lot of time in waiting for other teachers to complete their work.
In method 8, answer book is thought as an agenda of questions to be graded.

7.4 Lesson-end Activities

1.Discuss how parallelism can be achieved using specialized processors.
2.Compare data parallelism with temporal parallelism.

7.5 Points for discussions

Besides the various issues, there are also constraints placed by the structure and interconnection of computers in a parallel computing system. Thus picking a suitable method is also governed by the architecture of the parallel computer using which a problem is solved.

7.6 References

Parallel Computer Architecture and programming – V. Rajaraman and C.Siva Ram Murthy

Lesson 8 : Inter Task Dependency

Contents:

8.0 Aims and Objectives

The main aim of this lesson is to bring important types of problems encountered in parallel computers in which parallelism can be exploited.

8.1 Introduction

A problem of recipe of Chinese vegetable fried rice preparation is taken, and the task graph is drawn for each tasks. The tasks may be independent of one another or some times may be dependent of the previous task. The task graph is drawn shown and the number of cooks and their time allotment table is also drawn.

8.2 Inter Task Dependency

In general tasks of a job are inter-related. Some tasks can be done simultaneously and independently while others have to wait for the completion of previous tasks. The inter-relation of various tasks of a job may be represented graphically as a Task Graph.

- The Circles represents Tasks.
- A line with arrow connecting 2 circles shows dependency of tasks.
- The direction of arrow shows precedence.
- A task at the head end of an arrow can be done after all tasks at their respective tails are done.

Procedure Recipe for Chinese vegetable fried rice

T1	Clean and wash rice
T2	Boil water in a vessel with 1 teaspoon salt
T3	Put rice in boiling water with some oil and cook till soft
T4	Drain rice and cool
T5	Wash and scrape carrots
T6	Wash and string French beans
T7	Boil water with ½ teaspoon salt into 2 vessels

T8	Drop carrots and French beans separately in boiling water and keep for 1 minute.
T9	Drain and cool carrots and French beans.
T10	Dice carrots.
T11	Dice French beans.
T12	Peel onion and dice into small pieces. Wash and chop spring onions
T13	Clean cauliflower. Cut into small pieces
T14	Heat oil in iron pan and fry diced onion and cauliflower for 1 minute in heated oil.
T15	Add diced carrots, French beans to above and fry for 2 minutes
T16	Add cooled rice, chopped spring onions and Soya sauce to the above and stir and fry for 5 minutes

There are 16 tasks in cooking Chinese vegetable fried rice. Some of these tasks can be carried out simultaneously and others have to be done in sequence. For instance T1, T2, T5,T6, T7, T12 and T13 can be done simultaneously whereas T8 cannot be done unless T5, T6,T7 are done.

If suppose this dish has to be prepared for 50 people and 4 cooks are ready to do it. Task assignments for the cooks must be such that they work simultaneously and synchronize.

Task	T1	T2	T3	T4	T5	T6	T7	T8
Time	5	10	15	10	5	10	8	1
Task	T9	T10	T11	T12	T13	T14	T15	T16
Time	10	10	10	15	12	4	2	5

Table 8. 1 Time for each task

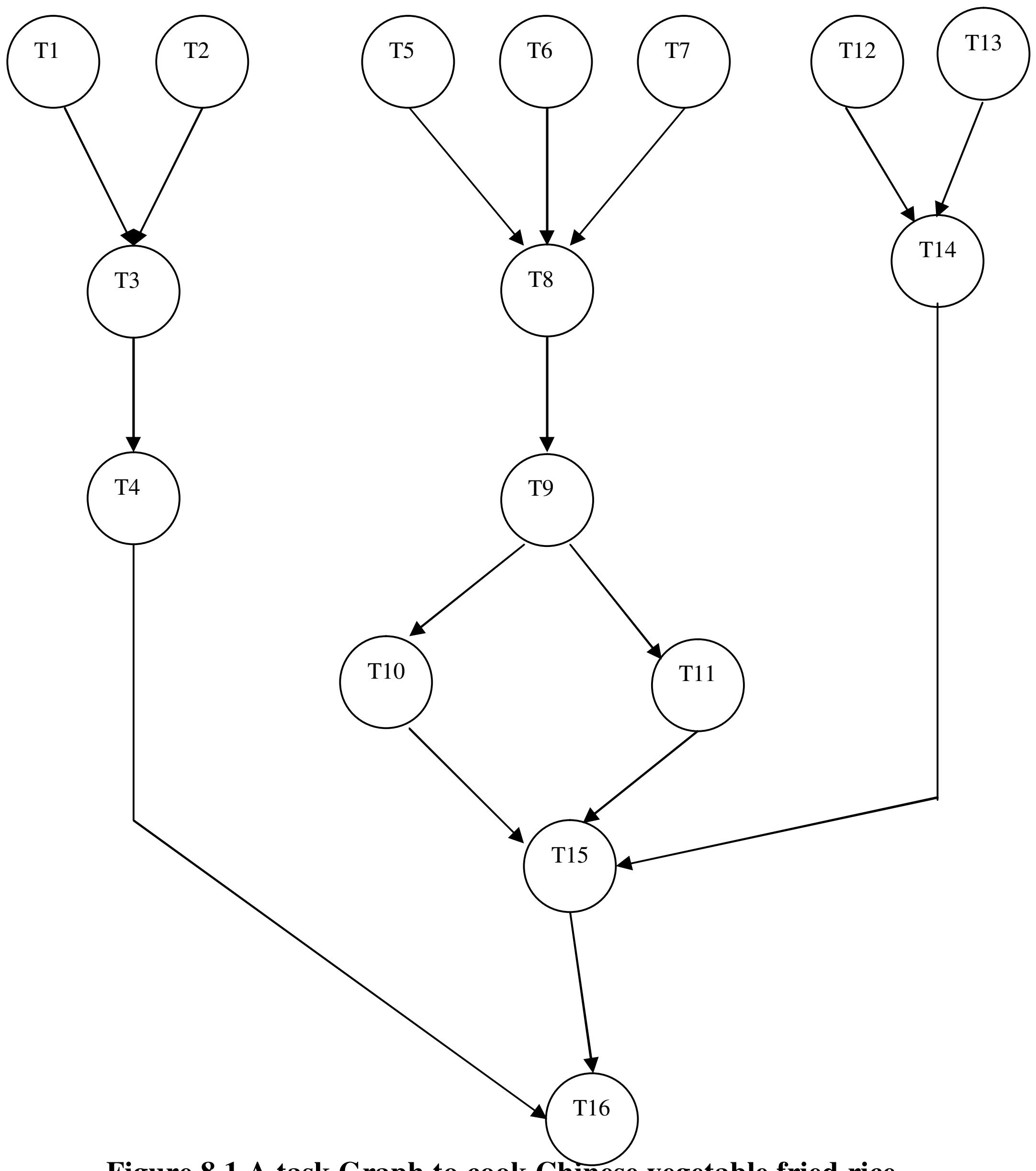


Figure 8.1 A task Graph to cook Chinese vegetable fried rice

Cook 1	T1	T2			T3	T4	T16
	5	10	15	10	5		
Cook 2	T5	T6	T8	T9	T10	T15	Idle
	5	10	1	10	10	2	
Cook 3	T7	T13	T14	Idle			
	8	12	4				
Cook 4	T12	Idle	T11	Idle			
	15	11	10				

Figure 8.2 Assignment of Tasks to Cooks

Procedure Assigning tasks in a task graph to cooks

Step 1 : Find tasks which can be carried out in parallel in level 1. Sum their total tile. In the above case the sum of tasks 1,2,5,6,7,12,13 = 65 minutes. There are 4 cooks so the tasks assigned for each cook will be 65 / 4 = 16 minutes
The assignment based on the logic is
Cook1 → (T1,T2) (15 minutes)
Cook 2 → (T5,T6) (15 minutes)
Cook 3 → (T7, T13) (20 minutes)
Cook 4 → (T12) (15 minutes)

- Step 2: Find tasks that can be carried out in level 2. They are T3, T8, and T14. There are 4 cooks. T14 cannot start unless T12 and T13 are completed. Thus cook1 is allocated T3; Cook2 → T8; Cook 3 → T14; Cook 4 → No task. AT the end of step 2 Cook 1 has worked for 30 minutes.
- Step 3 : The tasks which can be carried out in parallel are T4 and T9. T4 has to follow T3 and T9 has to follow T8. Cook1 → T4 and cook2 → T9
- Step 4: The next allocated tasks are T10 and T11 each taking 10 minutes. They follow completion of T9. Assignments are Cook 2 → T10 and Cook4 → T11. Cook 4 cannot start T11 till cook 2 completes T9.
- Step 5 : In the next level T15 can be allocated and as it has to follow completion of T10 and T11. This can be allocated to T9.
- Step 6 : Only T16 is left and it cannot start till T4, T15 and T14 are complete.T4 is last to finish and T16 is allocated to Cook 2

The problem is an example of parallel computing. These bring important problems encountered during parallelism.

8.3 Let us Sum Up

The lesson indicated the problem encountered during parallelism and the graph depicted the dependent tasks and independent tasks. The cook has been assigned the task in task graph and it represented the time requirement for each cook to carry out the task successfully.

8.4 Lesson-end Activities

1.List out the various graphical notations available to draw a task graph.
2.How tasks can be allocated in parallel? Give your own example procedure & task graph.

8.5 Points for discussions

The inter-relation of various tasks of a job may be represented graphically as a Task Graph.

- The Circles represents Tasks.
- A line with arrow connecting 2 circles shows dependency of tasks.
- The direction of arrow shows precedence.
- A task at the head end of an arrow can be done after all tasks at their respective tails are done.

8.6 References

Parallel Computer Architecture and programming – V. Rajaraman and C.Siva Ram Murthy

Lesson 9: Instruction Level parallel Processing, Pipelining of processing elements

Contents:

9.0 Aims and Objectives

The main aim of this lesson is to execute a number of instructions in parallel by scheduling them suitably on a single processor.

9.1 Introduction

One of the important methods of increasing the speed of PEs is pipelined execution of instructions. An architecture SMAC2P is used to explain the concept of instruction cycle which is broken into 5 steps. They are Fetch, Decode, Execute, Memory Access and Store register.

9.2 Instruction Level Parallel Processing

9.2.1 Pipelining of Processing Elements

One of the important methods of increasing the speed of PEs is pipelined execution of instructions.

Pipelining is an effective method of increasing the execution speed of processors provided the following “ideal” conditions are satisfied:

1. It is possible to break an instruction into a number of independent parts, each part taking nearly equal time to execute.
2. There is so called locality in instruction execution. If the instructions are not executed in sequence but “jump around” due to many branch instructions, then pipelining is not effective.
3. Successive instruction is such that the work done during the execution of an instruction can be effectively used by the next and successive instruction. Successive instructions are also independent of one another.
4. Sufficient resources are available in a processor so that if a resource is required by successive instructions in the pipeline it is readily available.

In actual practice these “ideal” conditions are not always satisfied. The non-ideal situation arises because of the following reasons:

1. It is not possible to break up an instruction execution into a number of parts each taking exactly the same time. For ex., executing a floating point division will normally take much longer than say, decoding an instruction. It is, however, possible to introduce delays (if needed) so that different parts of an instruction take equal time.
2. All real programs have branch instructions which disturb the sequential execution of instructions. Fortunately statistical studies show that the probability of encountering a branch instruction is low; around 17%. Further it may be possible to predict when branches will be taken a little ahead of time and take some pre-emptive action.
3. Successive instructions are not always independent. The results produced by an instruction may be required by the next instruction and must be made available at the right time.
4. There are always resource constraints in a processor as the chip size is limited. It will not be cost effective to duplicate some resources indiscriminately. For ex., it may not be useful have more than two floating point arithmetic units.

The challenges is thus to make pipelining work under these non-ideal conditions.

The computer is a Reduced Instruction Set Computer (RISC) which is designed to facilitate pipelined instruction execution. The computer is similar to SMAC2. It has the following units.

Step 1 : Fetch an instruction from the instruction memory (FI)

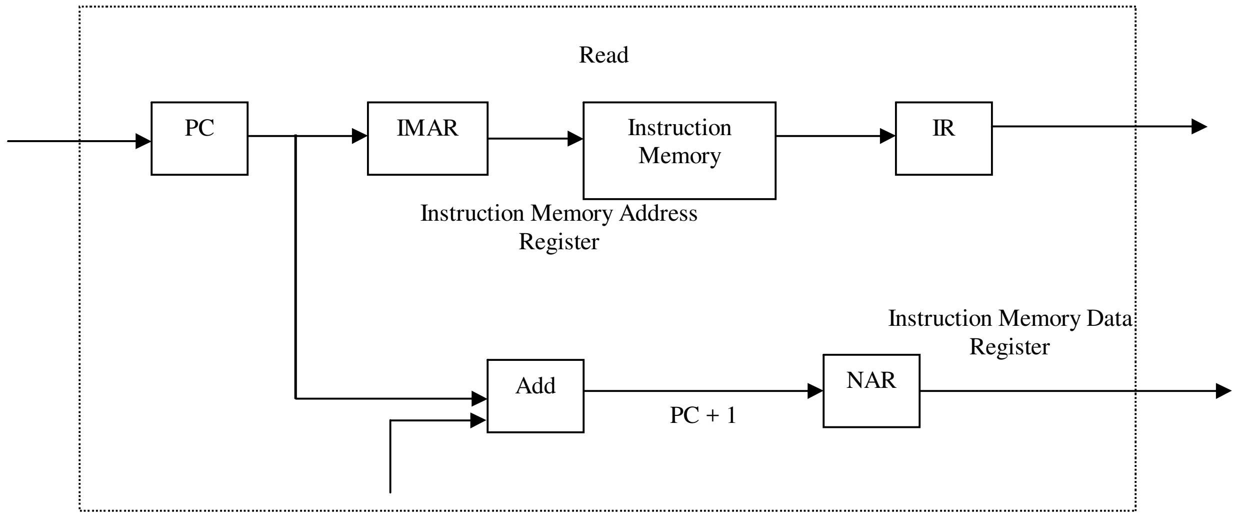

Read
PC
IMAR
Instruction Memory
IR
Instruction Memory Address Register
Instruction Memory Data Register
Add
PC + 1
NAR

Figure 9.1 Block diagram for Fetch Instruction

- A data cache(or memory) and an instruction cache (or memory). It is assumed that the instructions to be executed are stored in the instruction memory and the data to be read or written are stored in the data memory. These two memories have their own address and data registers. (IMAR- Memory Address Register of instruction memory, DMAR – Memory Address Register of data memory, IR - Data Register of Instruction memory, MDR - Data Register of Data memory)

- A Program Counter (PC) which contains the address of the next instruction to be executed. The machine is word addressable. Each word is 32 bits long.
- A register file with 32 registers. These are general purpose registers used to store operands and index values.
- The instructions are all of equal length and the relative positions of operands are fixed. There are 3 instruction types shown in the fig
- The only instructions which access memory are load and store instructions. A load instruction reads a word from the data memory and stores it in a specified in the register file and a store instruction stores the contents of a register in the data memory. This is called Load-Store Architecture.
- An ALU which carries out one integer arithmetic or one logic operation in one clock cycle.

Solid lines indicate Data paths and dashed lines indicate control signal.

IMAR $\leftarrow$ PC

IP $\leftarrow$ IMEM[IMAR]

NAR $\leftarrow$ PC + 1

All the above operations are carried out in one cycle.

Step 2 : Decode Instructions and Fetch Register (DE)

The instructions are decoded and the operands fields are used to read values of specified registers. The operations carried out during this step are shown below:

B1 $\leftarrow$ Reg [$IR_{21...25}$]

B2 $\leftarrow$ Reg [$IR_{16...20}$]

IMM $\leftarrow$ $IR_{0...15}$(Mtype instruction)

All these operations sre carried out in one clock cycle. Observe that the outputs from the register file are stored in two buffer registers B1 and B2.

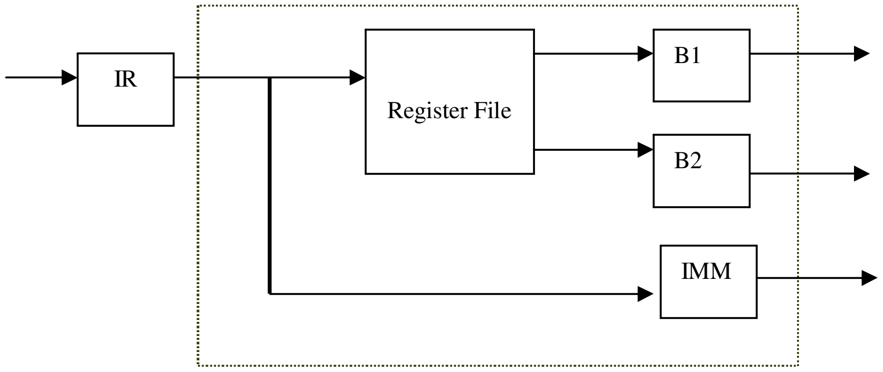


Figure 9.2 Data flow in decode and fetch register step

Step 3 : Execute instruction and calculate effective address(EX)

The ALU operations are carried out. The instructions where ALU is used may be classified as shown below:

- Two registers are operands

B3 $\leftarrow$ B1 <operation> B2

Where operation is ADD, SUB, MUL or DIV and B3 is the register where ALU output is stored

- Increment/Decrement (INC/DEC/BCT)
  B3 ← B1 + 1 (Increment)
  B3 ← B1 – 1 (Decrement or BCT)
- Branch on equal (JEQ)
  B3 ← B1 – B2
  If B3 = 0 set zero flag in Status Register = 1
- Effective address calculation(for LD/ST)
  For load and store instructions the effective address is stored in B3
  B3 ← B2 + IMM

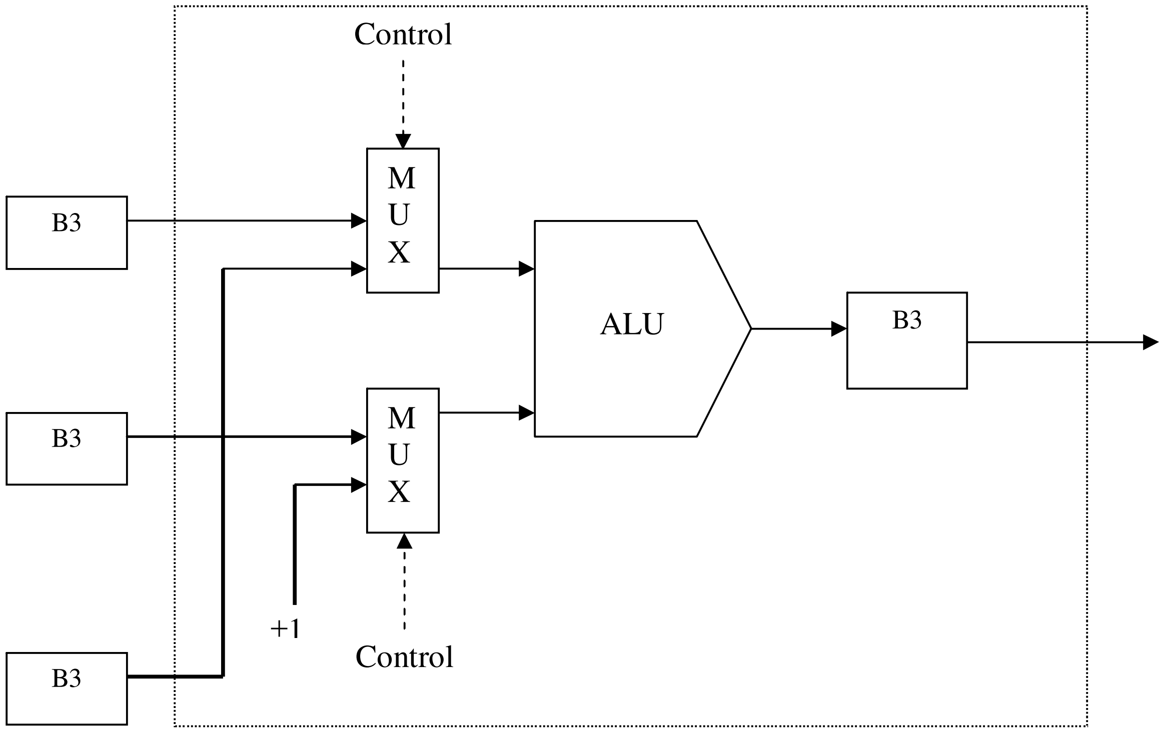


Figure 9.3 Data Flow for ALU operation

Step 4 : Memory Access (MEM)

Load/Store and address determination are carried out here. The address for load or store operation was computed in the previous step and is in B3. The following operations are carried out for load/store.

DMAR ← B3

Where DMAR is data memory address register

MDR ← DMEM [DMAR] load instruction (LD)

DMEM[DMAR] ← MDR ← B1 store instruction (ST)

For branch instructions the address of the next instruction to be executed must be found. The four branch instructions in our computer are JMP, JMI, JEQ and BCT

The PC value in these four cases is as follows:

For JMP

$PC \leftarrow IR_{0\ldots25}$

For JMI

If (negative flag = 1)$PC \leftarrow IR_{0\ldots25}$

Else $PC \leftarrow NAR$

For JEQ and BCT

If (zero flag = 1) $PC \leftarrow IR_{0\ldots15}$

Else $PC \leftarrow NAR$

For other instructions:

$PC \leftarrow NAR$

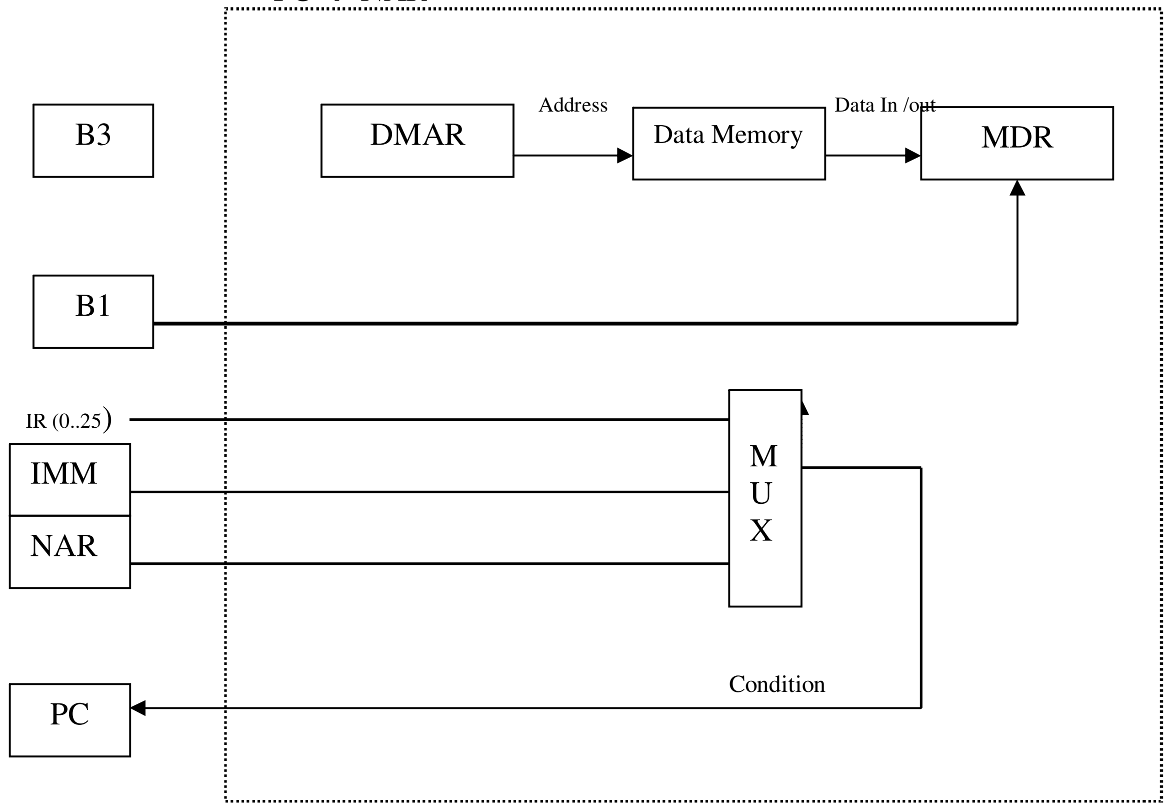


Figure 9.4 Data flow for load / store, next address step

Step 5: Store register (SR)

The result of ALU operation or load immediate or load from memory instruction is stored back in the specified register in the register file. The 3 cases are:

Store ALU output in register file

$Reg[IR_{11\ldots15}] \leftarrow B3$

Load immediate (LDI)

$Reg[IR_{21\ldots25}] \leftarrow IMM$ (IMM=IR(0…15))

Load data retrieved from data memory in register file (LD)

$Reg[IR_{21\ldots25}] \leftarrow MDR$

Conditional branch instruction JEQ,JMI and BCT require 3 cycles. Instructions INC, DEC and LDI do not need a memory access cycle and can complete in 3 cycles.

The proportion of these instructions in actual programs has to be found statistically by executing a large number of programs and counting the number of such instructions.

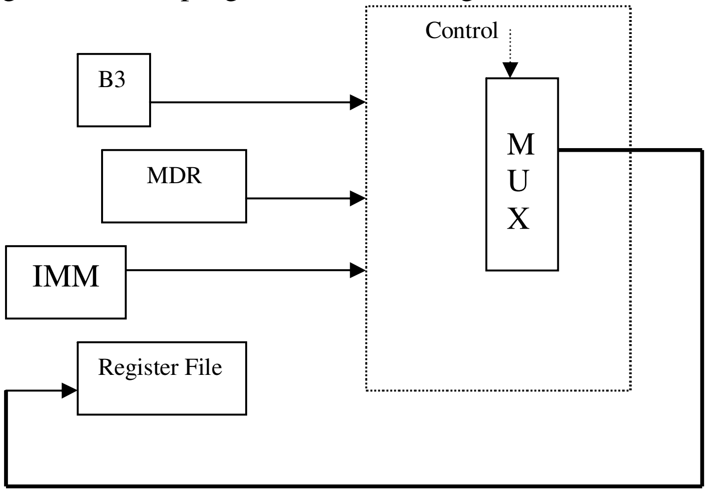


Figure 9.5 Data Flow for the store register step

9.3 Let us Sum Up

Pipelining is an effective method of increasing the execution of processors provided the following conditions are satisfied. They are break up of an instruction to independent parts, no locality in instruction execution, sufficient resources are available. The Instruction cycle has been clearly explained with neat diagrammatic representation such as Fetch an instruction from the instruction memory, Decode Instructions and Fetch Register (DE), Execute Instructions and calculates effective address, Memory access and Store register.

9.4 Lesson-end Activities

1.With neat diagrams, explain how parallel processing can be applied in instruction pipelining.

9.5 Points for discussions

- Instruction Execution Cycle Steps
- Fetch an instruction from the instruction memory
- Decode Instructions and Fetch Register (DE)
- Execute Instructions and calculates effective address
- Memory access and Store register.

9.6 References

Parallel Computer Architecture and programming – V. Rajaraman and C.Siva Ram Murthy

Lesson 10 : Delays in Pipelining Execution, Difficulties in Pipelining

Contents:

10.0 Aims and Objectives

The main objective of this lesson is to study about various pipeline Hazards and the preventive measures.

10.1 Introduction

Limits to pipelining: Hazards prevent next instruction from executing during its designated clock cycle
– Structural hazards: HW cannot support this combination of instructions (single person to fold and put clothes away)
– Data hazards: Instruction depends on result of prior instruction still in the pipeline (missing sock)
– Control hazards: Pipelining of branches & other instructions that change the PC
– Common solution is to stall the pipeline until the hazard is resolved, inserting one or more "bubbles" in the pipeline

10.2 Delays in Pipeline Execution

Delays in pipeline execution of instruction due to non ideal conditions are called Pipeline hazards.
The non –ideal conditions are :

- Available resources in a processor are limited.
- Successive instructions are not independent of one another.
- All programs have branches and loops

Each of the above conditions causes delays in pipeline. It can be classified as

Structural Hazards. They arise from resource conflicts when the hardware cannot support all possible combinations of instructions in simultaneous overlapped execution.

Data Hazards. They arise when an instruction depends on the result of a previous instruction in a way that is exposed by the overlapping of instructions in the pipeline. **Control Hazards.** They arise from the pipelining of branches and other instructions that change the PC.

10.2.1 Delay Due To Resource Constraints (Structural Hazard)

When a machine is pipelined, the overlapped execution of instructions requires pipelining of functional units and duplication of resources to allow all possible combinations of instructions in the pipeline.
If some combination of instructions cannot be accommodated because of a resource conflict, the machine is said to have a structural hazard.

Common instances of structural hazards arise when

- Some functional unit is not fully pipelined. Then a sequence of instructions using that unpipelined unit cannot proceed at the rate of one per clock cycle.
- Some resource has not been duplicated enough to allow all combinations of instructions in the pipeline to execute.

Example1:
A machine may have only one register-file write port, but in some cases the pipeline might want to perform two writes in a clock cycle.

Example2:
A machine has shared a single-memory pipeline for data and instructions. As a result, when an instruction contains a data-memory reference(load), it will conflict with the instruction reference for a later instruction (instr 3):

Clock cycle number								
Instr	1	2	3	4	5	6	7	8
Load	IF	ID	EX	MEM	WB			
Instr 1		IF	ID	EX	MEM	WB		
Instr 2			IF	ID	EX	MEM	WB	
Instr 3				IF	ID	EX	MEM	WB

To resolve this, we stall the pipeline for one clock cycle when a data-memory access occurs. The effect of the stall is actually to occupy the resources for that instruction slot. The following table shows how the stalls are actually implemented.

Clock cycle number

Instr	1	2	3	4	5	6	7	8	9
Load	IF	ID	EX	MEM	WB				
Instr 1		IF	ID	EX	MEM	WB			
Instr 2			IF	ID	EX	MEM	WB		
Stall				bubble	bubble	bubble	bubble	bubble	
Instr 3					IF	ID	EX	MEM	WB

Instruction 1 assumed not to be data-memory reference (load or store), otherwise Instruction 3 cannot start execution for the same reason as above.

To simplify the picture it is also commonly shown like this:

Clock cycle number									
Instr	1	2	3	4	5	6	7	8	9
Load	IF	ID	EX	MEM	WB				
Instr 1		IF	ID	EX	MEM	WB			
Instr 2			IF	ID	EX	MEM	WB		
Instr 3				stall	IF	ID	EX	MEM	WB

10.2.2 Delay Due To Data Dependency (Data Hazard)

A major effect of pipelining is to change the relative timing of instructions by overlapping their execution. This introduces data and control hazards. Data hazards occur when the pipeline changes the order of read/write accesses to operands so that the order differs from the order seen by sequentially executing instructions on the unpipelined machine.

Consider the pipelined execution of these instructions:

		1	2	3	4	5	6	7	8	9
ADD	R1, R2, R3	IF	ID	EX	MEM	WB				
SUB	R4, R5, R1		IF	IDsub	EX	MEM	WB			
AND	R6, R1, R7			IF	IDand	EX	MEM	WB		
OR	R8, R1, R9				IF	IDor	EX	MEM	WB	
XOR	R10,R1,R11					IF	Idxor	EX	MEM	WB

All the instructions after the ADD use the result of the ADD instruction (in R1). The ADD instruction writes the value of R1 in the WB stage (shown black), and the SUB instruction reads the value during ID stage (IDsub). This problem is called a data hazard. Unless precautions are taken to prevent it, the SUB instruction will read the wrong value and try to use it.

The AND instruction is also affected by this data hazard. The write of R1 does not complete until the end of cycle 5 (shown black). Thus, the AND instruction that reads the registers during cycle 4 (IDand) will receive the wrong result.

The OR instruction can be made to operate without incurring a hazard by a simple implementation technique. The technique is to perform register file reads in the second half of the cycle, and writes in the first half. Because both WB for ADD and IDor for OR are performed in one cycle 5, the write to register file by ADD will perform in the first half of the cycle, and the read of registers by OR will perform in the second half of the cycle.

The XOR instruction operates properly, because its register read occur in cycle 6 after the register write by ADD.

The next page discusses forwarding, a technique to eliminate the stalls for the hazard involving the SUB and AND instructions.

We will also classify the data hazards and consider the cases when stalls can not be eliminated. We will see what compiler can do to schedule the pipeline to avoid stalls.

The problem with data hazards, introduced by this sequence of instructions can be solved with a simple hardware technique called forwarding.

		1	2	3	4	5	6	7
ADD	R1, R2, R3	IF	ID	EX	MEM	WB		
SUB	R4, R5, R1		IF	IDsub	EX	MEM	WB	
AND	R6, R1, R7			IF	IDand	EX	MEM	WB

The key insight in forwarding is that the result is not really needed by SUB until after the ADD actually produces it. The only problem is to make it available for SUB when it needs it.

If the result can be moved from where the ADD produces it (EX/MEM register), to where the SUB needs it (ALU input latch), then the need for a stall can be avoided. Using this observation , forwarding works as follows:

The ALU result from the EX/MEM register is always fed back to the ALU input latches. If the forwarding hardware detects that the previous ALU operation has written the register corresponding to the source for the current ALU operation, control logic selects the forwarded result as the ALU input rather than the value read from the register file.

Forwarding of results to the ALU requires the additional of three extra inputs on each ALU multiplexer and the addition of three paths to the new inputs.

The paths correspond to a forwarding of:
(a) the ALU output at the end of EX,

(b) the ALU output at the end of MEM, and
(c) the memory output at the end of MEM.

Without forwarding our example will execute correctly with stalls:

		1	2	3	4	5	6	7	8	9
ADD	R1,R2, R3	IF	ID	EX	MEM	WB				
SUB	R4,R5, R1		IF	stall	stall	IDsub	EX	MEM	WB	
AND	R6,R1, R7			stall	stall	IF	IDand	EX	MEM	WB

As our example shows, we need to forward results not only from the immediately previous instruction, but possibly from an instruction that started three cycles earlier. Forwarding can be arranged from MEM/WB latch to ALU input also. Using those forwarding paths the code sequence can be executed without stalls:

		1	2	3	4	5	6	7
ADD	R1, R2, R3	IF	ID	EXadd	MEMadd	WB		
SUB	R4, R5, R1		IF	ID	EXsub	MEM	WB	
AND	R6, R1, R7			IF	ID	EXand	MEM	WB

The first forwarding is for value of R1 from EXadd to EXsub .
The second forwarding is also for value of R1 from MEMadd to EXand.
This code now can be executed without stalls.

Forwarding can be generalized to include passing the result directly to the functional unit that requires it: a result is forwarded from the output of one unit to the input of another, rather than just from the result of a unit to the input of the same unit.

10.2.3 Pipeline Delay Due To Branch Instructions (Control Hazard)

Control hazards can cause a greater performance loss for DLX pipeline than data hazards. When a branch is executed, it may or may not change the PC (program counter) to something other than its current value plus 4. If a branch changes the PC to its target address, it is a taken branch; if it falls through, it is not taken.
If instruction i is a taken branch, then the PC is normally not changed until the end of MEM stage, after the completion of the address calculation and comparison (see diagram).
The simplest method of dealing with branches is to stall the pipeline as soon as the branch is detected until we reach the MEM stage, which determines the new PC. The pipeline behavior looks like :

Branch	IF	ID	EX	MEM	WB					

Branch successor		IF(stall)	stall	stall	IF	ID	EX	MEM	WB	
Branch successor+1						IF	ID	EX	MEM	WB

The stall does not occur until after ID stage (where we know that the instruction is a branch).
This control hazards stall must be implemented differently from a data hazard, since the IF cycle of the instruction following the branch must be repeated as soon as we know the branch outcome. Thus, the first IF cycle is essentially a stall (because it never performs useful work), which comes to total 3 stalls.
Three clock cycles wasted for every branch is a significant loss. With a 30% branch frequency and an ideal CPI of 1, the machine with branch stalls achieves only half the ideal speedup from pipelining!
The number of clock cycles can be reduced by two steps:

- Find out whether the branch is taken or not taken earlier in the pipeline;
- Compute the taken PC (i.e., the address of the branch target) earlier.

Both steps should be taken as early in the pipeline as possible.
By moving the zero test into the ID stage, it is possible to know if the branch is taken at the end of the ID cycle. Computing the branch target address during ID requires an additional adder, because the main ALU, which has been used for this function so far, is not usable until EX.

10.2.4 Branch Prediction schemes

There are many methods to deal with the pipeline stalls caused by branch delay. We discuss four simple compile-time schemes in which predictions are static - they are fixed for each branch during the entire execution, and the predictions are compile-time guesses.

- Stall pipeline
- Predict taken
- Predict not taken
- Delayed branch
-

Stall pipeline

The simplest scheme to handle branches is to freeze or flush the pipeline, holding or deleting any instructions after the branch until the branch destination is known. Advantage: simple both to software and hardware (solution described earlier)

Predict Not Taken

A higher performance, and only slightly more complex, scheme is to predict the branch as not taken, simply allowing the hardware to continue as if the branch were not executed. Care must be taken not to change the machine state until the branch outcome is definitely known.

- The complexity arises from:
  we have to know when the state might be changed by an instruction;
- we have to know how to "back out" a change.

The pipeline with this scheme implemented behaves as shown below:

Untaken Branch	IF	ID	EX	MEM	WB		

Instr							
Instr i+1		IF	ID	EX	MEM	WB	
Instr i+2			IF	ID	EX	MEM	WB

Taken Branch Instr	IF	ID	EX	MEM	WB			
Instr i+1		IF	idle	idle	idle	idle		
Branch target			IF	ID	EX	MEM	WB	
Branch target+1				IF	ID	EX	MEM	WB

When branch is not taken, determined during ID, we have fetched the fall-through and just continue. If the branch is taken during ID, we restart the fetch at the branch target. This causes all instructions following the branch to stall one clock cycle.

Predict Taken

An alternative scheme is to predict the branch as taken. As soon as the branch is decoded and the target address is computed, we assume the branch to be taken and begin fetching and executing at the target address.
Because in DLX pipeline the target address is not known any earlier than the branch outcome, there is no advantage in this approach. In some machines where the target address is known before the branch outcome a predict-taken scheme might make sense.

Delayed Branch

In a delayed branch, the execution cycle with a branch delay of length n is

Branch instr
sequential successor 1
sequential successor 2
. . . . .
sequential successor n
Branch target if taken

Sequential successors are in the branch-delay slots. These instructions are executed whether or not the branch is taken.
The pipeline behavior of the DLX pipeline, which has one branch delay slot is shown below:

Untaken branch instr	IF	ID	EX	MEM	WB				
Branch delay instr(i+1)		IF	ID	EX	MEM	WB			
Instr i+2			IF	ID	EX	MEM	WB		
Instr i+3				IF	ID	EX	MEM	WB	
Instr i+4					IF	ID	EX	MEM	WB

Taken branch instr	IF	ID	EX	MEM	WB				
Branch delay instr(i+1)		IF	ID	EX	MEM	WB			
Branch target			IF	ID	EX	MEM	WB		
Branch target+1				IF	ID	EX	MEM	WB	
Branch target+2					IF	ID	EX	MEM	WB

10.2.5 Difficulties in Pipelining

The another difficulty which makes pipeline processing challenging is due to the interruption of normal flow of a program due to events such as illegal instruction codes, page faults and I/O calls. These are called as Exception conditions.
Various list of exception conditions has been listed out. It has been mentioned that whether after the exception the normal process can be started or not.
The problem of restarting computation is complicated by the fact that several instructions will be in various stages of completion in the pipeline. If the pipeline processing can be stopped when an exception condition is detected in such a way that all instructions which occur before the one causing the exception are completed and all instructions which were in progress at the instant exception occurred can be restarted from the beginning, the pipeline is said to have precise exceptions.

Exception Type	Occurs during pipeline stage		Resume or Terminated
	Yes / No	Which Stage ?	
I/O Request	No	--	Resume
OS request by user program	No	--	Resume
User initiates break point during execution	No	--	Resume
User Tracing Program	No	--	Resume
Arithmetic Overflow or underflow	Yes	EX	Resume
Page fault	Yes	FI, MEM	Resume
Misaligned Memory access	Yes	FI, MEM	Resume
Memory protection violation	Yes	FI, MEM	Resume
Undefined instruction	Yes	DE	Resume
Hardware failure	Yes	Any	Resume
Power failure	Yes	Any	Resume

Table : 10.1 Exception Types in Computer

10.3 Let us Sum Up

It discussed the various delays in pipeline execution and the different hazards are structural hazards, data hazards and control hazards and the prevention mechanism for the hazards.

10.4 Lesson-end Activities

1.What delays are encountered in pipeline execution? How can they be overcome?
2. Discuss the difficulties involved in pipelining. Also, explain the prevention measures.

10.5 Points for discussions

- Structural Hazard
- Data Hazard
- Control Hazard
- Branch prediction Buffer

10.6 References

Materials from Net : Shankar Balachandran, Dept. of Computer Science and Engineering,IIT-Madras,shankar@cse.iitm.ernet.in

UNIT – III

Lesson 11 : Principles of Linear Pipelining, Classification of Pipeline Processors

Contents:

11.0 Aims and Objectives

The main objective of this lesson is to known the basic properties of pipelining, classification of pipeline processors and the required memory support.

11.1 Introduction

Pipeline is similar to the assembly line in industrial plant. To achieve pipelining one must divide the input process into a sequence of sub tasks and each of which can be executed concurrently with other stages. The various classification or pipeline line processor are arithmetic pipelining, instruction pipelining, processor pipelining have also been briefly discussed.

11.2 Pipelining

Pipelining offers an economical way to realize temporal parallelism in digital computers. To achieve pipelining, one must subdivide the input task into a sequence of subtasks, each of which can be executed by a specialized hardware stage.

- Pipelining is the backbone of vector supercomputers
- Widely used in application-specific machine where high throughput is needed
- Can be incorporated in various machine architectures (SISD,SIMD,MIMD,.....)

Easy to build a powerful pipeline and waste its power because:

- Data can not be fed fast enough
- The algorithm does not have inherent concurrency.
- Programmers do not know how to program it efficiently.

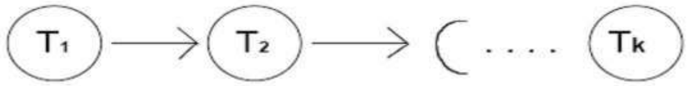
Types of Pipelines

- Linear Pipelines
- Non-linear Pipelines
- Single Function Pipelines
- Multifunctional Pipelines
  - » Static
  - » Dynamic

11.2.1 Principles of Linear Pipelining

A. Basic Principles and Structure

Let T be a task which can be partitioned into K subtasks according to the linear precedence relation:
T= {T1,T2,..........,Tk} ; i.e.,
a subtask Tj cannot start until { Ti " i < j } are finished. This can be modelled with the linear precedence graph:



A linear pipeline (No Feedback!) can always be constructed to process a succession of subtask with a linear precedence graph.

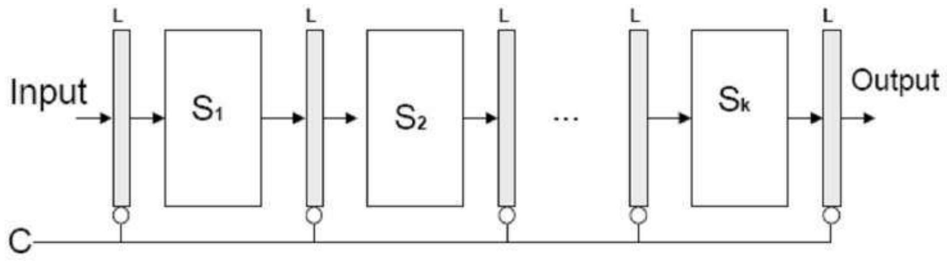


Figure 11.1 Basic Structure and Control of a Linear Pipeline

Processor (L=latch, C=clock, Si=the ith stage.)

- Stages are pure combinational circuits used for processing.
- Latches are fast registers to hold the intermediate data between the stages.
- Informational flow is controlled by a common clock with some clock period "t", and the pipeline runs at a frequency of 1/t
- t is selected as: t = MAX{ti} + tL = tM + tL where, ti= propagation delay of stage Si tL=latch delay
- Pipeline clock period is controlled by the stage with the max delay.
- Unless the stage delays are balanced, one big and slow stage can slow down the whole pipe

Space-Time Diagrams
Consider a four stage linear pipeline processor and a sequence of tasks.
T1, T2,...
Where each task has 4 subtasks (1st subscript if for task, 2nd is for subtask) as follows:
T1=>{T11,T12,T13,T14}
T2=>{T21,T22,T23,T24}
…
T2=>{Tn1,Tn2,Tn3,Tn4}
A space-time diagram can be constructed to illustrate how the overlapping execution of the tasks as follows

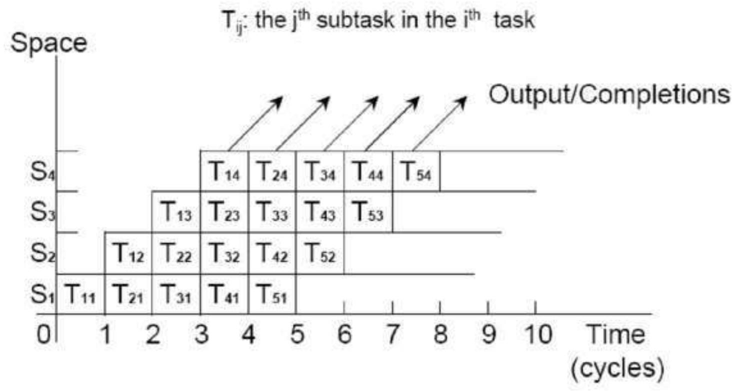

T_{ij}: the j^{th} subtask in the i^{th} task
Space
Output/Completions
S4
S3
S2
S1
T_{11} T_{21} T_{31} T_{41} T_{51}
T_{12} T_{22} T_{32} T_{42} T_{52}
T_{13} T_{23} T_{33} T_{43} T_{53}
T_{14} T_{24} T_{34} T_{44} T_{54}
0 1 2 3 4 5 6 7 8 9 10
Time (cycles)

Figure 11.2 Space - Time Diagram

Performance Measure for Linear Pipelined Processors

Speedup Sk- the speedup of a k-stage linear pipeline processor(over an equivalent non-pipelined) is given by Sk=(T)/(Tk)=

$$\frac{\text{Execution time for the non-pipelined processor}}{\text{Execution time for the pipelined processor}}$$

- With a non-pipelined processor, each task takes k clocks, thus for n tasks T1=n. k clocks
- With a pipelined processor we need k clocks to fill the pipe and generate the first result (n-1) clocks to generate the remaining n-1 results

Thus, Tk=k+(n-1), and,

$$S_k = \frac{n\,.\,k}{k+n-1}$$

The maximum speedup, attained after an infinite # of takes is

$$\lim_{n \to \infty} Sk(n) = \lim_{n \to \infty} \frac{k}{(K/n)+1-(1/n)} = \boxed{S_{k\,max} = K}$$

Efficiency "E" - the ration of the busy time span over the overall time span (note : E is easy to see from spacetime)

» Overall time span =(# of stages) * (total # of clocks)
= k*(k+n-1) clock.stage
» Busy time span = (# of stages) * (# of tasks)
= k*(n) clock.stage

And

$$E = \frac{(k*n)}{k*(k+n-1)} \quad \text{,i.e.} \quad \boxed{E = \frac{n}{k+n-1}}$$

And

$$E_{max} = \lim_{n \to \infty} \frac{1}{(k/n) + 1 - (1/n)} \quad \text{,i.e.} \quad \boxed{E_{max} = 1}$$

Note that :
Overall time span <=> Allocated computational power
Busy time span <=> Utilized computational power and
Thus E <=> Pipeline Utilization

note also that : $E = \frac{S_k}{k}$

where Sk = actual speedup and k can be viewed as the maximum speedup
Throughput "W" - is the number of tasks that can be completed, by the pipeline, per unit time
» For a k-stage pipeline with a clock period t , n tasks take (k+n-1) t time units
Thus

$$W = \frac{n}{(k+n-1) * \tau} = E / \tau$$

and

$$W_{max} = 1 / \tau$$

To illustrate the operation principles of a pipeline computation, the design of a pipeline floating point adder is given. It is constructed in four stages. The inputs are
$A = a \times 2^p$
$B = b \times 2^q$
Where a and b are 2 fractions and p and q are their exponents and here base 2 is assumed.
To compute the sum
$C = A + B = c \times 2^r = d \times 2^s$
Operations performed in the four pipeline stages are specified.

1. Compare the 2 exponents p and q to reveal the larger exponent r =max(p,q) and to determine their difference t =p-q
2. Shift right the fraction associated with the smaller exponent by t bits to equalize the two components before fraction addition.
3. Add the preshifted fraction with the other fraction to produce the intermediate sum fraction c where 0 <= c <1.

4. Count the number of leading zeroes, say u, in fraction c and shift left c by u bits to produce the normalized fraction sum $d = c \times 2^{u}$, with a leading bit 1. Update the large exponent s by subtracting $s = r - u$ to produce the output exponent.

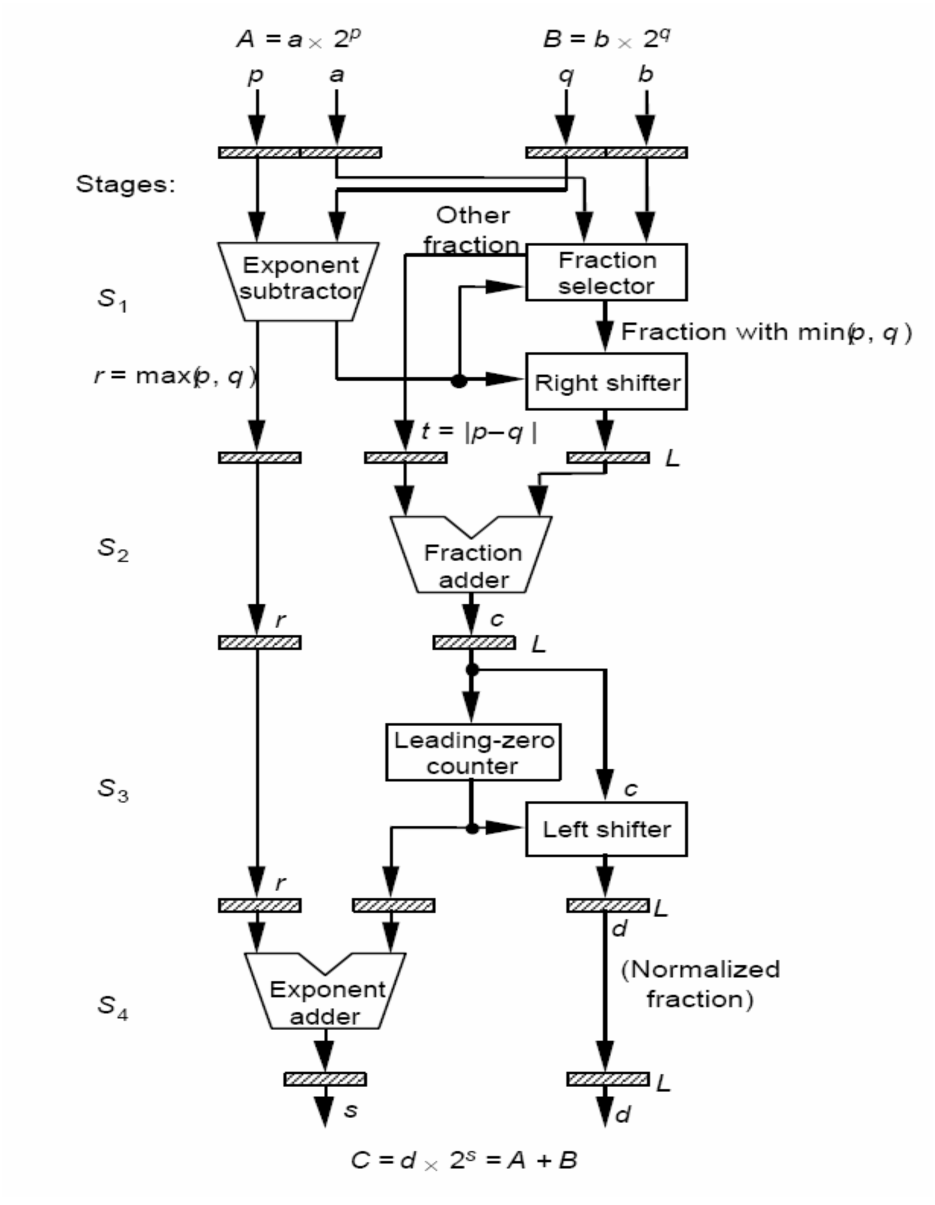


Figure 11.3 A pipelined floating-point adder with four processing stages

11.2.2 Classification of Pipeline Processors

Arithmetic Pipelining

The arithmetic and logic units of a computer can be segmentized for pipeline operations in various data formats. Well known arithmetic pipeline examples are

- Star 100
- The eight stage pipes used in TI-ASC

- 14 pipeline stages used in Cray-1
- 26 stages per pipe in the cyber-205

Instruction Pipelining

The execution of a stream of instructions can be pipelined by overlapping the execution of the current instruction with fetch, decode and operand fetch of subsequent instructions. This technique is known as instruction look-ahead.

Processor pipelining

This refers to pipelining processing of the same data stream by a cascade of processors each of which processes a specific task. The data stream passes the first processor with results stored in memory block which is also accessible by the second processor. The second processor then passes the refined results to the third and so on.

The principle pipeline classification schemes are :
Unification Vs Multifunction pipelines

A pipeline with fixed and dedicated function such as floating adder is called unifuncitonal pipeline. Eg : Cray-1
A multifunction pipe may perform different functions, either at different times or at the same time, by interconnecting different subsets of stages in the pipeline.
Eg : TI-ASC

Static Vs Dynamic Pipeline

A static pipeline has only one functional configuration at a time.
A dynamic pipeline permits several functional configurations to exist simultaneously.

Scalar Vs Vector Pipelines

A scalar pipeline processes a sequence of scalar operands under the control of DO loop.
A vector pipeline is designed to handle vector instructions over vector operands.

11.3 Let us Sum Up

The basics of pipelining has been discussed such as structure of a linear pipeline processor, space time diagram of a linear pipeline processor for over lapped processing of multiple tasks. Four pipeline stages have been explained with a pipelined floating point adder. Various classification schemes for pipeline processors have been explained.

11.4 Lesson-end Activities

1.Discuss the classification schemes of pipeline processors.
2. Discuss the Principles of Linear Pipelining with floating point adder.

11.5 Points for discussions

- Pipelining is the backbone of vector supercomputers. It is Widely used in application-specific machine where high throughput is needed
- Can be incorporated in various machine architectures (SISD,SIMD,MIMD,.....)

11.6 References

- 31R6 - Computer Design by Leslie S. Smith
- Tarek A. El-Ghazawi, Dept. of Electrical and Computer Engineering, The George Washington University

Lesson 12 : General Pipeline and Reservation Tables, Arithmetic Pipeline Design Examples

Contents:

12.0 Aims and Objectives

The main objective of this lesson is to learn about reservation tables and how successive pipeline stages are utilized for a specific evaluation function.

12.1 Introduction

The interconnection structures and data flow patterns in general pipelines are characterized either feedforward or feedbackward connections, in addition to the cascaded connections in a linear pipeline. A 2D chart known as reservation table shows how the successive pipelines stages are utilized for a specific function evaluation in successive pipeline cycles. Multiplication of 2 numbers is done by repeated addition and shift operations.

12.2 General Pipeline and Reservation Tables

Reservation tables are used how successive pipeline stages are utilized for a specific evaluation function.

Consider an example of pipeline structure with both feed forward and feedback connections. The pipeline is dual functional denoted as function A and function B. The pipeline stages are numbered as S1, S2 and S3. The feed forward connection connects a stage S_i to a stage S_j such that $j \geq i + 2$ and feedback connection connects to S_i to a stage S_j such that $j <= i$.

Time	t0	t1	t2	t3	t4	t5	t6	t7
S1	A			A			A	
S2		A						A
S3			A		A	A		

Table 12.1 Reservation Table for function A

Time	t0	t1	t2	t3	t4	t5	t6
S1	B				B		
S2			B			B	
S3		B		B			B

Table 12.2 Reservation Table for function B

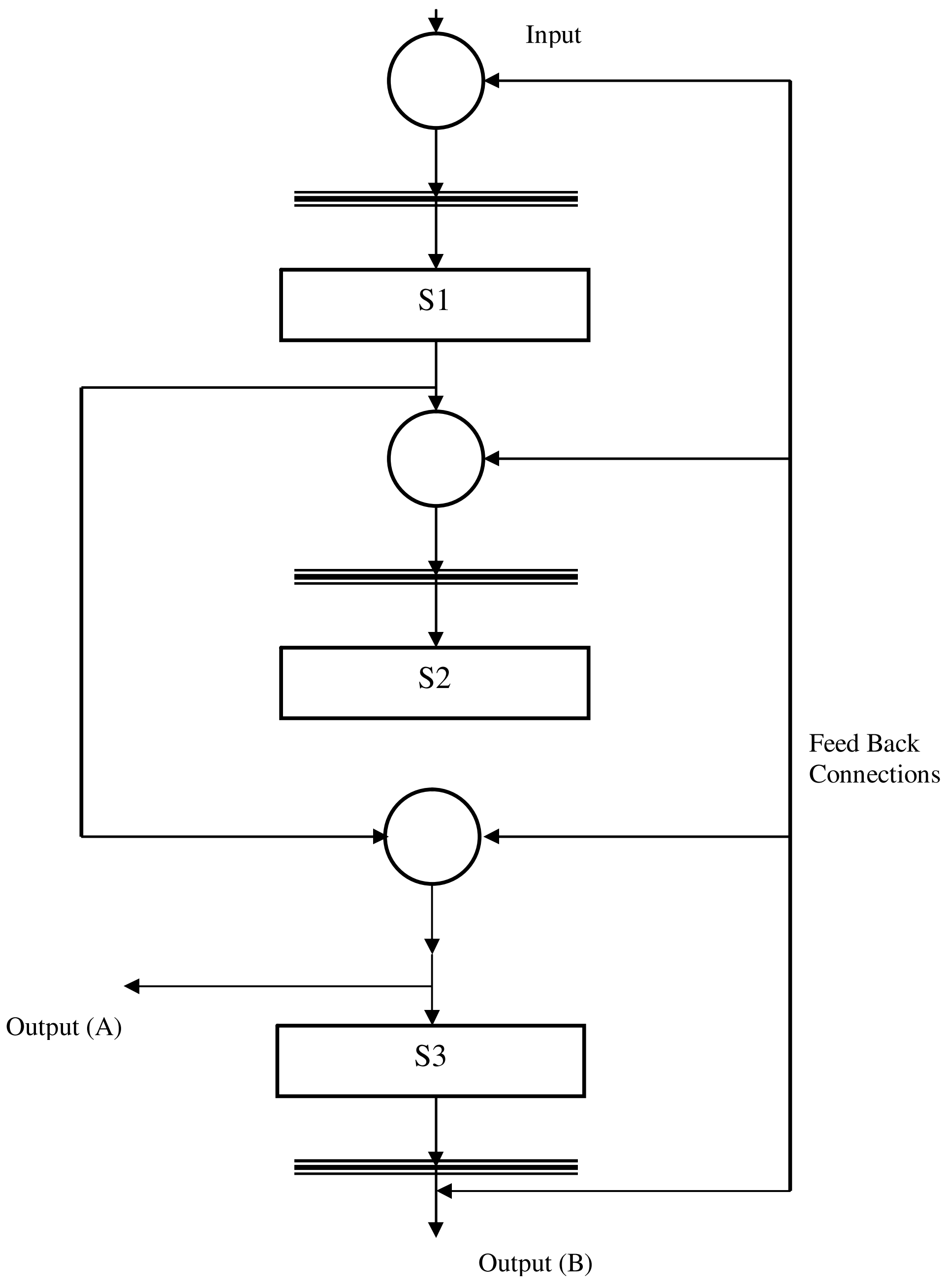


Figure 12.1 A sample Pipeline

The row corresponds to the 2 functions of the sample pipeline. The rows correspond to pipeline stages and the columns to clock time units. The total number of clock units in the table is called the evaluation time. A reservation table represents the flow of data through the pipeline for one complete evaluation of a given function. A market entry in the (i,j)th square of the table indicates the stage Si will be used j time units after initiation of the function evaluation.

12.2.1 Arithmetic Pipeline Design Examples

The multiplication of 2 fixed point numbers is done by repeated add-shift operations, using ALU which has built in add and shift functions. Multiple number additions can be realized with a multilevel tree adder. The conventional carry propagation adder (CPA) adds 2 input numbers say A and B, to produce one output number called the sum A+B carry save adder (CSA) receives three input numbers, say A,B and D and two output numbers, the sum S and the Carry vector C.

	A	=		1	1	1	1	0	1
	B	=		0	1	0	1	1	0
	D	=		1	1	0	1	1	1
	C	=	1	1	0	1	1	1	
	S	=		0	1	1	1	0	0
	A+B+D	=	1	1	1	0	0	1	0

A CSA can be implemented with a cascade of full adders with the carry-out of a lower stage connected to the carry-in of a higher stage. A carry-save adder can be implemented with a set of full adders with all the carry-in terminals serving as the input lines for the third input number D, and all the carry-out terminals serving as the output lines for the carry vector C.

This pipeline is designed to multiply two 6 bit numbers. There are five pipeline stages. The first stage is for the generation of all 6 x 6 = 36 immediate product terms, which forms the six rows of shifted multiplicands. The six numbers are then fed into two CSAs in the second stage. In total four CSAs are interconnected to form a three level merges six numbers into two numbers: the sum vector S and the carry vector C. The final stage us a CPA which adds the two numbers C and S to produce the final output of the product A x B.

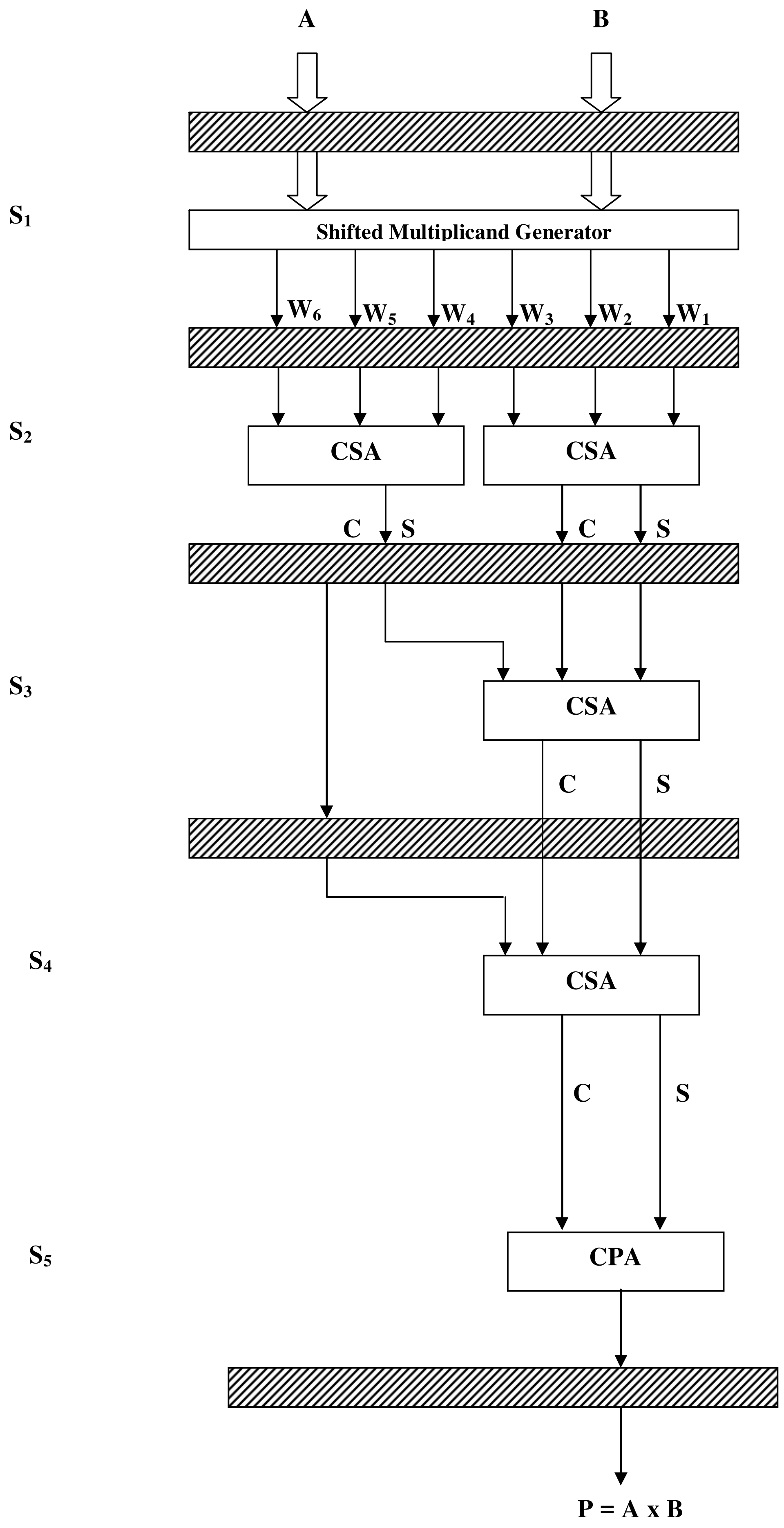

A
B
S_1
Shifted Multiplicand Generator
W_6
W_5
W_4
W_3
W_2
W_1
S_2
CSA
CSA
C
S
C
S
S_3
CSA
C
S
S_4
CSA
C
S
S_5
CPA
P = A x B

Figure 12.2 A Pipelined Multiplier built with a CSA tree

12.3 Let us Sum Up

Many interesting pipeline utilization can be revealed by the reservation table. It is possible to have multiple marks in a row or in a column. A CSA (carry save adder) is used to perform multiple number additions.

12.4 Lesson-end Activities

1. Give the reservation tables and sample pipeline for any two functions.
2. With example, discuss carry propagation adder (CPA) and carry save adder (CSA).

12.5 Points for discussions

- The conventional carry propagation adder (CPA) adds 2 input numbers and produces an output number called as Sum.
- A carry save adder (CSA) receives three input numbers A, B, D and outputs of 2 numbers the sum vector and the carry vector.

12.6 References

- Computer Architecture and Parallel Processing – Kai Hwang

Lesson 13 : Data Buffering and Busing Structure, Internal Forwarding and Register Tagging, Hazard Detection and Resolution, Job Sequencing and Collision Prevention.

Contents:

13.0 Aims and Objectives

The objective of this lesson is to be familiar with busing structure, register tagging and various pipeline hazards and its preventive measures and job sequencing and Collision prevention.

13.1 Introduction

Buffers are used to speed close up the speed gap between memory accesse=s for either instructions or operands. Buffering can avoid unnecessary idling of the processing stages caused by memory access conflicts or by unexpected branching or interrupts. The concepts of busing is discussed which eliminates the time delay to store and to retrieve intermediate results or to from the registers.

The computer performance can be greatly enhanced if one can eliminate unnecessary memory accesses and combine transitive or multiple fetch-store operations with faster register operations. This is carried by register tagging and forwarding. A pipeline hazard refers to a situation in which a correct program ceases to work correctly due to implementing the processor with a pipeline.

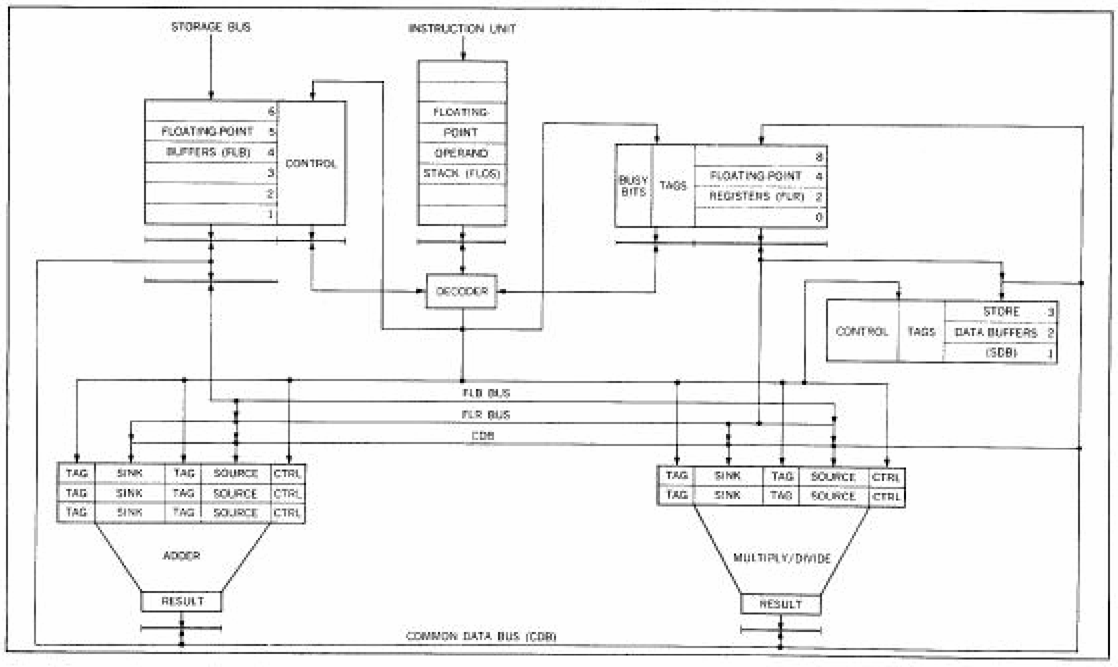
There are three fundamental types of hazard:

- Data hazards,
- Branch hazards, and
- Structural hazards.

13.2 Data Buffering and Busing Structure

Another method to smooth the traffic flow in a pipeline is to use buffers to close up the speed gap between the memory accesses for either instructions or operands and arithmetic and logic executions in the functional pipes. The instruction or operand buffers provide a continuous supply of instructions or operands to the appropriate pipeline units. Buffering can avoid unnecessary idling of the processing stages caused by memory access conflicts or by unexpected branching or interrupts. Sometimes the entire loop instructions can be stored in the buffer to avoid repeated fetch of the same instructions loop, if the buffer size is sufficiently large. It is very large in the usage of pipeline computers.

Three buffer types are used in various instructions and data types. Instructions are fetched to the instruction fetch buffer before sending them to the instruction unit. After decoding, fixed point and floating point instructions and data are sent to their dedicated buffers. The store address and data buffers are used for continuously storing results back to memory. The storage conflict buffer is used only used when memory



Data registers and transfer paths, including CDB and reservation stations.

Figure 13.1 Data Buffers, transfer paths, reservation stations and common data bus

Busing Buffers

The sub function being executed by one stage should be independent of the other sub functions being executed by the remaining stages; otherwise some process in the pipeline must be halted until the dependency is removed. When one instruction waiting to be executed is first to be modified by a future instruction, the execution of this instruction must be suspended until the dependency is released.

Another example is the conflicting use of some registers or memory locations by different segments of a pipeline. These problems cause additional time delays. An efficient internal busing structure is desired to route the resulting stations with minimum time delays.

In the AP 120B or FPS 164 attached processor the busing structure are even more sophisticated. Seven data buses provide multiple data paths. The output of the floating point adder in the AP 120B can be directly routed back to the input of the floating point adder, to the input of the floating point multiplier, to the data pad, or to the data memory. Similar busing is provided for the output of the floating point multiplier. This eliminates the time delay to store and to retrieve intermediate results or to from the registers.

13.2.1 Internal Forwarding and Register Tagging

To enhance the performance of computers with multiple execution pipelines

1. **Internal Forwarding** refers to a short circuit technique for replacing unnecessary memory accesses by register -to-register transfers in a sequence of fetch-arithmetic-store operations
2. **Register Tagging** refers to the use of tagged registers, buffers and reservations stations for exploiting concurrent activities among multiple arithmetic units.

The computer performance can be greatly enhanced if one can eliminate unnecessary memory accesses and combine transitive or multiple fetch-store operations with faster register operations. This concept of internal data forwarding can be explored in three directions. The symbols Mi and Rj to represent the ith word in the memory and jth fetch, store and register-to-register transfer. The contents of Mi and Rj are represented by (M_i) and R_j

Store-Fetch Forwarding

The store the n fetch can be replaced by 2 parallel operations, one store and one register transfer.
2 memory accesses

$M_i \leftarrow (R_1)$ (store)
$R_2 \leftarrow (M_i)$ (Fetch)

Is being replaced by
Only one memory access

$M_i \leftarrow (R_1)$ (store)
$R_2 \leftarrow (R_1)$ (register Transfer)

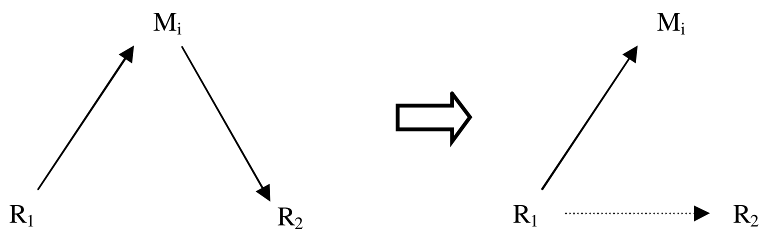
Fetch-Fetch Forwarding

The following fetch operations can be replaced by one fetch and one register transfer. One memory access has been eliminated.
2 memory accesses

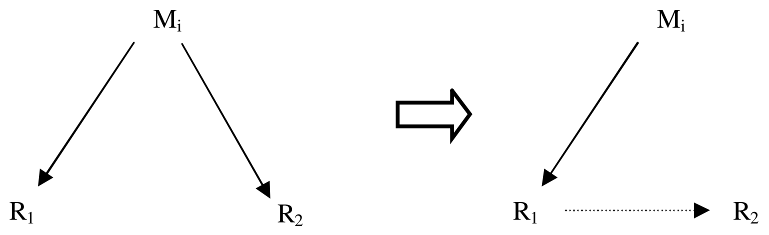
$R_1 \leftarrow (M_i)$ (fetch)
$R_2 \leftarrow (M_i)$ (Fetch)

Is being replaced by
Only one memory access

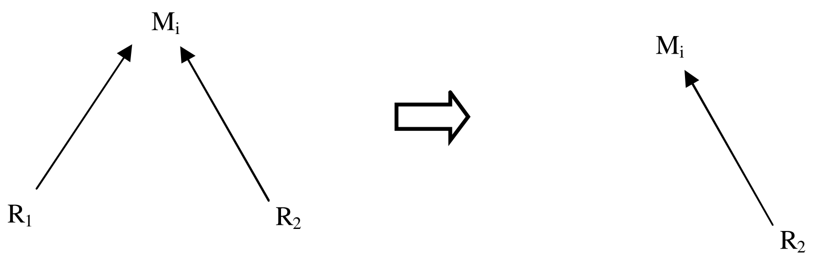
$R_1 \leftarrow (M_i)$ (Fetch)
$R_2 \leftarrow (R_1)$ (register Transfer)



Store – Fetch Forwarding



Fetch – Fetch Forwarding



Store-Store overwriting

Figure 13.2 Internal Forwarding Examples thick arrows for memory accesses and dotted arrows for register transfers

Store-Store Overwriting

The following two memory updates of the same word can be combined into one; since the second store overwrites the first.

2 memory accesses

$M_i \leftarrow (R_1)$ (store)

$M_i \leftarrow (R_2)$ (store)

Is being replaced by
Only one memory access

$M_i \leftarrow (R_2)$ (store)

The above steps shows how to apply internal forwarding to simplify a sequence of arithmetic and memory access operations

13.2.2 Hazard Detection and Resolution

Defining hazards

The next issue in pipelining is ***hazards***. A pipeline hazard refers to a situation in which a correct program ceases to work correctly due to implementing the processor with a pipeline.
There are three fundamental types of hazard:

- Data hazards,
- Branch hazards, and
- Structural hazards.

Data hazards can be further divided into

- Write After Read
- Write After Write
- Read After Write

Structural Hazards

These occur when a single piece of hardware is used in more than one stage of the pipeline, so it's possible for two instructions to need it at the same time.
So, for instance, suppose we'd only used a single memory unit instead of separate instruction memory and data memories. A simple (non-pipelined) implementation would work equally well with either approach, but in a pipelined implementation we'd run into trouble any time we wanted to fetch an instruction at the same time a `lw` or `sw` was reading or writing its data.
In effect, the pipeline design we're starting from has anticipated and resolved this hazard by adding extra hardware.
Interestingly, the earlier editions of our text used a simple implementation with only a single memory, and separated it into an instruction memory and a data memory when they introduced pipelining. This edition starts right off with the two memories.
Also, the first Sparc implementations (remember, Sparc is almost exactly the RISC machine defined by one of the authors) *did* have exactly this hazard, with the result that load instructions took an extra cycle and store instructions took two extra cycles.

Data Hazards

This is when reads and writes of data occur in a different order in the pipeline than in the program code. There are three different types of data hazard (named according to the order of operations that must be maintained):

RAW
A Read After Write hazard occurs when, in the code as written, one instruction reads a location after an earlier instruction writes new data to it, but in the pipeline the write occurs after the read (so the instruction doing the read gets stale data).

WAR
A Write After Read hazard is the reverse of a RAW: in the code a write occurs after a read, but the pipeline causes write to happen first.

WAW
A Write After Write hazard is a situation in which two writes occur out of order. We normally only consider it a WAW hazard when there is no read in between; if there is, then we have a RAW and/or WAR hazard to resolve, and by the time we've gotten that straightened out the WAW has likely taken care of itself.
(the text defines data hazards, but doesn't mention the further subdivision into RAW, WAR, and WAW. Their graduate level text mentions those)

Control Hazards

This is when a decision needs to be made, but the information needed to make the decision is not available yet. A Control Hazard is actually the same thing as a RAW data hazard (see above), but is considered separately because different techniques can be employed to resolve it - in effect, we'll make it less important by trying to make good guesses as to what the decision is going to be.
Two notes: First, there is no such thing as a RAR hazard, since it doesn't matter if reads occur out of order. Second, in the MIPS pipeline, the only hazards possible are branch hazards and RAW data hazards.

Resolving Hazards

There are four possible techniques for resolving a hazard. In order of preference, they are:
Forward. If the data is available somewhere, but is just not where we want it, we can create extra data paths to ``forward'' the data to where it is needed. This is the best solution, since it doesn't slow the machine down and doesn't change the semantics of the instruction set. All of the hazards in the example above can be handled by forwarding.
Add hardware. This is most appropriate to structural hazards; if a piece of hardware has to be used twice in an instruction, see if there is a way to duplicate the hardware. This is exactly what the example MIPS pipeline does with the two memory units (if there were only one, as was the case with RISC and early SPARC, instruction fetches would have to stall waiting for memory reads and writes), and the use of an ALU and two dedicated adders.

Stall. We can simply make the later instruction wait until the hazard resolves itself. This is undesirable because it slows down the machine, but may be necessary. Handling a hazard on waiting for data from memory by stalling would be an example here. Notice that the hazard is guaranteed to resolve itself eventually, since it wouldn't have existed if the machine hadn't been pipelined. By the time the entire downstream pipe is empty the effect is the same as if the machine hadn't been pipelined, so the hazard has to be gone by then.

Document (AKA punt). Define instruction sequences that are forbidden, or change the semantics of instructions, to account for the hazard. Examples are delayed loads and delayed branches. This is the worst solution, both because it results in obscure conditions on permissible instruction sequences, and (more importantly) because it ties the instruction set to a particular pipeline implementation. A later implementation is likely to have to use forwarding or stalls anyway, while emulating the hazards that existed in the earlier implementation. Both Sparc and MIPS have been bitten by this; one of the nice things about the late, lamented Alpha was the effort they put into creating an exceptionally "clean" sequential semantics for the instruction set, to avoid backward compatibility issues tying them to old implementations.

13.2.3 Job Sequencing and Collision Prevention

Initiation	the start a single function evaluation
Collision	two or more initiations attempt to use the same stage at the same time

Problem:

To properly schedule queued tasks awaiting initiation in order to avoid collisions and to achieve high throughput.
Reservation Table + Modified State Diagram + MAL

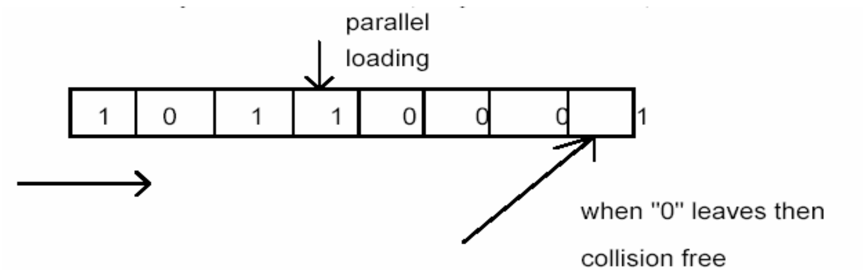
Fundamental concepts:

Latency - number of time units between two initiations (any positive integer 1, 2,…)
Latency sequence – sequence of latencies between successive initiations
Latency cycle – a latency sequence that repeats itself
Control strategy – the procedure to choose a latency sequence
Greedy strategy – a control strategy that always minimizes the latency between the current initiation and the very last initiation

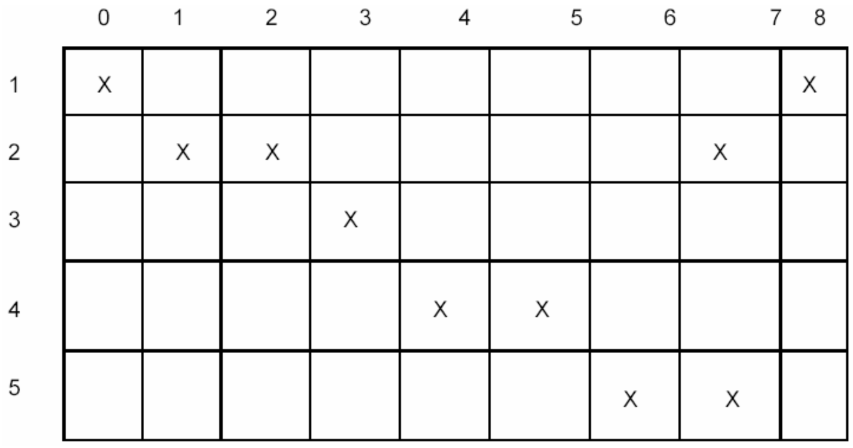
Definitions:

1. A collision occurs when two tasks are initiated with latency (initiation interval) equal to the column distance between two “X” on some row of the reservation table.
2. The set of column distances $F = \{l_1, l_2, \ldots, l_r\}$ between all possible pairs of “X” on each row of the reservation table is called the forbidden set of latencies.
3. The collision vector is a binary vector $C = (C_n \ldots C_2 C_1)$,
Where $C_i=1$ if i belongs to F (set of forbidden latencies) and $C_i=0$ otherwise.

Example: Let us consider a Reservation Table with the following set of forbidden latencies F and permitted latencies P (complementation of F).

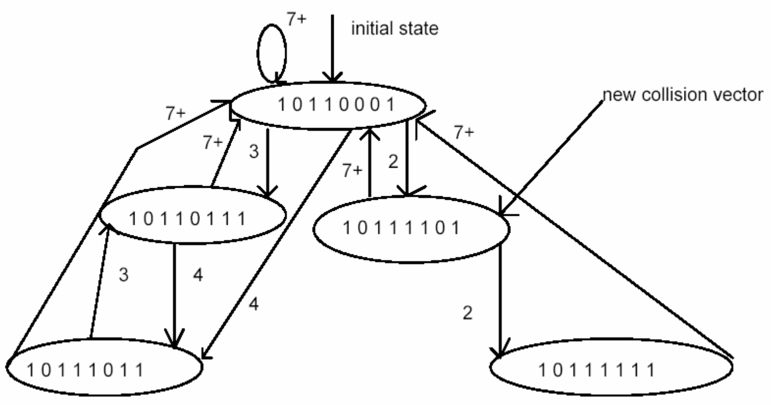


Forbidden list = F = {1,5,6,8}
Collision vector: C={1 0 1 1 0 0 0 1)
8 7 6 5 4 3 2 1



	0	1	2	3	4	5	6	7	8
1	X								X
2		X	X					X	
3				X					
4					X	X			
5							X	X	

```
1 0 1 1 0 0 0 1
1 0 1 1 0 1 1 1
OR      1 0 1 1
1 0 1 1 1 0 1 1
```



```
   1 0 1 1 0 0 0 1      initial
(+)0 0 1 0 1 1 0 0      →2 (initial)
   1 0 1 1 1 1 0 1

   1 0 1 1 0 0 0 1      initial
   0 0 0 1 0 1 1 0      →2 (initial)
   1 0 1 1 0 1 1 1

   1 0 1 1 0 0 0 1      initial
   0 0 0 0 1 0 1 1      →4 (initial)

   1 0 1 1 0 0 0 1      initial
   0 0 0 0 0 0 0 1      →7 (initial)
   1 0 1 1 0 0 0 1
```

Facts:

1. The collision vector shows both permitted and forbidden latencies from the same reservation table.
2. One can use n-bit shift register to hold the collision vector for implementing a control strategy for successive task initiations in the pipeline. Upon initiation of the first task, the collision vector is parallel-loaded into the shift register as the initial state. The shift register is then shifted right one bit at a time, entering 0's from the left end. A collision free initiation is allowed at time instant t+k a bit 0 is being shifted at of the register after k shifts from time t.
A **state diagram** is used to characterize the successive initiations of tasks in the pipeline in order to find the shortest latency sequence to optimize the control strategy. A **state** on the diagram is represented by the contents of the shift register after the proper number of shifts is made, which is equal to the latency between the current and next task initiations.
3. The successive collision vectors are used to prevent future task collisions with previously initiated tasks, while the collision vector C is used to prevent possible collisions with the current task. If a collision vector has a "1" in the ith bit (from the right), at time t, then the task sequence should avoid the initiation of a task at time t+i.
4. Closed logs or cycles in the state diagram indicate the steady – state sustainable latency sequence of task initiations without collisions.
The **average latency** of a cycle is the sum of its latencies (period) divided by the number of states in the cycle.
5. The throughput of a pipeline is inversely proportional to the reciprocal of the average latency. A latency sequence is called **permissible** if no collisions exist in the successive initiations governed by the given latency sequence.

6. The maximum throughput is achieved by an optimal scheduling strategy that achieves the (MAL) minimum average latency without collisions.

Corollaries:

1. The job-sequencing problem is equivalent to finding a permissible latency cycle with the MAL in the state diagram.
2. The minimum number of X's in array single row of the reservation table is a lower bound of the MAL.

Simple cycles are those latency cycles in which each state appears only once per each iteration of the cycle.
A single cycle is a **greedy cycle** if each latency contained in the cycle is the minimal latency (outgoing arc) from a state in the cycle.
A good task-initiation sequence should include the greedy cycle.

Procedure to determine the greedy cycles
1. From each of the state diagram, one chooses the arc with the smallest latency label unit; a closed simple cycle can formed.
2. The average latency of any greedy cycle is no greater than the number of latencies in the forbidden set, which equals the number of 1's in the initial collision vector.
3. The average latency of any greedy cycle is always lower-bounded by the

MAL <= $ALgreedy <= \#of$1'sin the collision vector

13.3 Let us Sum Up

Buffers helped in closing up the speed gap. It helps in avoiding idling of the processing stages caused by memory access. Busing concepts eliminated the time delay. A pipeline hazard refers to a situation in which a correct program ceases to work correctly due to implementing the processor with a pipeline. Various pipeline hazards are Data hazards, Branch hazards, and Structural hazards.

13.4 Lesson-end Activities

1. How buffering can be done using Data Buffering and Busing Structure? Explain.
2. Define Hazard. What are the types of hazards? How they can be detected and resolved?
3. Discuss i. Store-Fetch Forwarding ii. Fetch-Fetch Forwarding iii. Store-Store overwriting
4. Discuss Job Sequencing and Collision Prevention

13.5 Points for discussions

- Register Tagging and Forwarding
    - The computer performance can be greatly enhanced if one can eliminate unnecessary memory accesses and combine transitive or multiple fetch-store operations with faster register operations. This is carried by register tagging and forwarding..
- Pipeline Hazards : Data Hazard, Control Hazard, Structural Hazard

13.6 References

- Pipelining Tarek A. El-Ghazawi, Dept. of Electrical and Computer Engineering, The George Washington University
- Pipelining Hazards, Shankar Balachandran, Dept. of Computer Science and Engineering, IIT-Madras, shankar@cse.iitm.ernet.in
- www.cs.berkeley.edu/~lazzaro
- www.csee.umbc.edu/~younis/CMSC611/CMSC611.htm
- CIS 570 Advanced Computer Systems, r. Boleslaw Mikolajczak

Lesson 14 : Vector Processing Requirements, Characteristics, Pipelined Vector Processing Methods

14.0 Aims and Objectives

The main aim this lesson is to learn the vector processing requirements, its characteristics and the instructions used by vector and to perform a comparative study to know the difference between scalar and vector operations.

14.1 Introduction

A vector processor consists of a scalar processor and a vector unit, which could be thought of as an independent functional unit capable of efficient vector operations. Various pipelined vector processing methods are Horizontal Processing, in which vector computations are performed horizontally from left to right in row fashion.

Vertical processing, in which vector computations are carried out vertically from top to bottom in column fashion. Vector looping, in which segmented vector loop computations are performed from left to right and top to bottom in a combined horizontal and vertical method.

14.2 Vector Processing Requirements

A vector operand contains 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 a floating point number, an integer, a logical value or a character.
A vector processor consists of a scalar processor and a vector unit, which could be thought of as an independent functional unit capable of efficient vector operations.

Vector Hardware

Vector computers have hardware to perform the vector operations efficiently. Operands can not be used directly from memory but rather are loaded into registers and are put back in registers after the operation. Vector hardware has the special ability to overlap or pipeline operand processing.

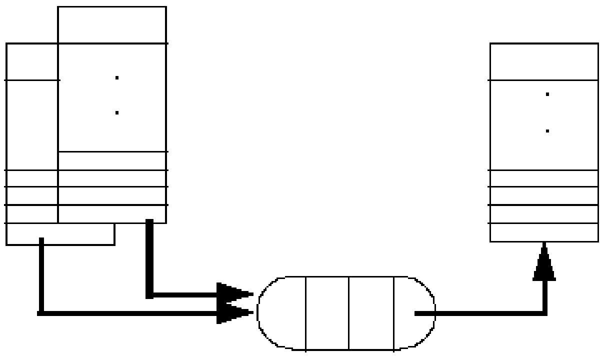


Figure 14.1 Vector Hardware

Vector functional units pipelined, fully segmented each stage of the pipeline performs a step of the function on different operand(s) once pipeline is full, a new result is produced each clock period (cp).

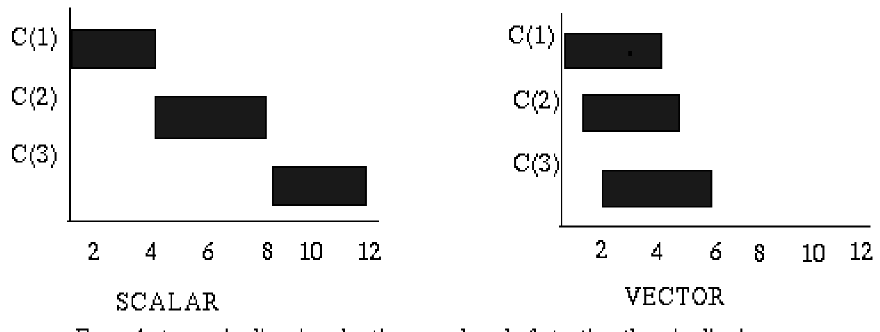
Pipelining

The pipeline is divided up into individual segments, each of which is completely independent and involves no hardware sharing. This means that the machine can be working on separate operands at the same time. This ability enables it to produce one result per clock period as soon as the pipeline is full. The same instruction is obeyed repeatedly using the pipeline technique so the vector processor processes all the elements of a vector in exactly the same way. The pipeline segments arithmetic operation such as floating point multiply into stages passing the output of one stage to the next stage as input. The next pair of operands may enter the pipeline after the first stage has processed the previous pair of operands. The processing of a number of operands may be carried out simultaneously.

The loading of a vector register is itself a pipelined operation, with the ability to load one element each clock period after some initial startup overhead.

Chaining

Theoretical speedup depends on the number of segments in the pipeline so there is a direct relationship between the number of stages in the pipeline you can keep full and the performance of the code. The size of the pipeline can be increased by chaining thus the Cray combines more than one pipeline to increase its effective size. Chaining means that the result from a pipeline can be used as an operand in a second pipeline as illustrated in the next diagram.



For a 4 stage pipeline (neglecting overhead of starting the pipeline)

S(I) = A * X(I) + Y(I)

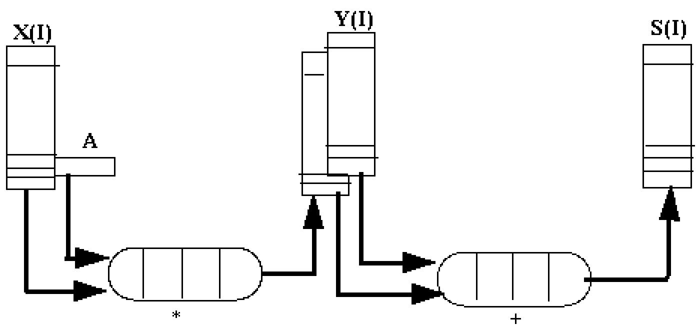


Figure 14.2 Pipeline Chaining

This example shows how two pipelines can be chained together to form an effectively single pipeline containing more segments. The output from the first segment is fed directly into the second set of segments thus giving a resultant effective pipeline length of 8. Speedup (over scalar code) is dependent on the number of stages in the pipeline. Chaining increases the number of stages

14.2.1 Characteristics of Vector Processing

14.2.1.1 Vector Instructions

The ISA of a scalar processor is augmented with vector instructions of the following types:

Vector-vector instructions:

f1: Vi → Vj (e.g. MOVE Va, Vb)

f2: Vj x Vk → Vi (e.g. ADD Va, Vb, Vc)

Vector-scalar instructions:

f3: s x Vi → Vj (e.g. ADD R1, Va, Vb)

Vector-memory instructions:

f4: M $\rightarrow$ V (e.g. Vector Load)
f5: V $\rightarrow$ M (e.g. Vector Store)

Vector reduction instructions:

f6: V $\rightarrow$ s (e.g. ADD V, s)
f7: Vi x Vj $\rightarrow$ s (e.g. DOT Va, Vb, s)

Gather and Scatter instructions:

f8: M x Va $\rightarrow$ Vb (e.g. gather)
f9: Va x Vb $\rightarrow$ M (e.g. scatter)

Masking instructions:

fa: Va x Vm $\rightarrow$ Vb (e.g. MMOVE V1, V2, V3)

Gather and scatter are used to process sparse matrices/vectors. The gather operation, uses a base address and a set of indices to access from memory "few" of the elements of a large vector into one of the vector registers. The scatter operation does the opposite. The masking operations allows conditional execution of an instruction based on a "masking" register.

A Boolean vector can be generated as a result of comparing two vectors, and can be used as a masking vector for enabling and disabling component operations in a vector instruction.
A compress instruction will shorten a vector under the control of a masking of vector.
A merge instruction combines two vectors under the control of a masking vector.

In general machine operation suitable for pipelining should have the following properties:

- Identical Processes (or functions) are repeatedly invoked many times, each of which can be subdivided into subprocesses (or sub functions)
- Successive Operands are fed through the pipeline segments and require as few buffers and local controls as possible.
- Operations executed by distinct pipelines should be able to share expensive resources, such as memories and buses in the system.
- The **operation code** must be specified in order to select the functional unit or to reconfigure a multifunctional unit to perform the specified operation.
- For a memory reference instruction, the **base addresses** are needed for both source operands and result vectors. If the operands and results are located in the vector register file, the designated vector registers must be specified.
- The **address increment** between the elements must be specified.
- The **address offset** relative to the base address should be specified. Using the base address and the offset the relative effective address can be calculated.
- The **Vector length** is needed to determine the termination of a vector instruction.
- The Relative Vector/Scalar Performance and Amdahl Law

14.2.1.2 Comparison - Vector and Scalar Operations

A scalar operation works on only one pair of operands from the S register and returns the result to another S register whereas a vector operation can work on 64 pairs of operands together to produce 64 results executing only one instruction. Computational efficiency is achieved by processing each element of a vector identically eg initializing all the elements of a vector to zero. A vector instruction provides iterative processing of successive vector register elements by obtaining the operands from the first element of one or more V registers and delivering the result to another V register. Successive operand pairs are transmitted to a functional unit in each clock period so that the first result emerges after the start up time of the functional unit and successive results appear each clock cycle.

Vector overhead is larger than scalar overhead, one reason being the vector length which has to be computed to determine how many vector registers are going to be needed (ie the number of elements divided by 64).

Each vector register can hold up to 64 words so vectors can only be processed in 64 element segments. This is important when it comes to programming as one situation to be avoided is where the number of elements to be processed exceeds the register capacity by a small amount eg a vector length of 65. What happens in this case is that the first 64 elements are processed from one register, the 65th element must then be processed using a separate register, after the first 64 elements have been processed. The functional unit will process this element in a time equal to the start up time instead of one clock cycle hence reducing the computational efficiency. There is a sharp decrease in performance at each point where the vector length spills over into a new register.

The Cray can receive a result by a vector register and retransmit it as an operand to a subsequent operation in the same clock period. In other words a register may be both a result and an operand register which allows the chaining of two or more vector operations together as seen earlier. In this way two or more results may be produced per clock cycle.

Parallelism is also possible as the functional units can operate concurrently and two or more units may be co-operating at once. This combined with chaining, using the result of one functional unit as the input of another, leads to very high processing speeds.

Scalar and vector processing examples

```
DO 10 I = 1, 3
JJ(I) = KK(I)+LL(I)
10 CONTINUE
```

14.2.1.3 Scalar and Vector Processing
Scalar Processing

Read one element of Fortran array KK
Read one element of LL
Add the results

Write the results to the Fortran array JJ
Increment the loop index by 1
Repeat the above sequence for each succeeding array element until the loop index equals its limit.

Vector Processing

Load a series of elements from array KK to a vector register and a series of elements from array LL to another vector register (these operations occur simultaneously except for instruction issue time)
Add the corresponding elements from the two vector registers and send the results to another vector register, representing array JJ
Store the register used for array JJ to memory
This sequence would be repeated if the array had more elements than the maximum elements used in vector processing ie 64.

Processing Order and Results

Inherent to vector processing is a change in the order of operations to be performed on individual array elements, for any loop that includes two separate vectorized operations. The following example illustrates this:

```
DO 10 I =1, 3
L(I) = J(I) + K(I)
N(I) = L(I) + M(I)
10 CONTINUE
```

Scalar Version

The two statements within this loop are each executed three times, with the operations alternating;
L(I) is calculated before N(I) in each iteration
the new value of L(I) is used to calculate the value of N(I).

Results Of Scalar Processing

Event	Operation	Values
1	L(1) = J(1)+K(1)	7 = 2 + 5
2	N(1) = L(1)+M(1)	11 = 7 + 4
3	L(2) = J(2)+K(2)	-1 = (-4) + 3
4	N(2) = L(2)+M(2)	5 = (-1) + 6
5	L(3) = J(3)+K(3)	15 = 7 + 8
6	N(3) = L(3)+M(3)	13 = 15 + (-2)

Vector Version

With vector processing the first line within the loop processes all elements of the array before the second line is executed.

Results Of Vector Processing

Event Operation Values
1 L(1) = J(1)+K(1) 7 = 2 + 5
2 L(2) = J(2)+K(2) -1 = (-4) + 3
3 L(3) = J(3)+K(3) 15 = 7 + 8
4 N(1) = L(1)+M(1) 11 = 7 + 4
5 N(2) = L(2)+M(2) 5 = (-1) + 6
6 N(3) = L(3)+M(3) 13 = 15 + (-2)
NB Both processing methods produce the same results for each array element.

Scalar	Vector
The loop repeats the loop control overhead in each iteration	Using pipelines the overhead is reduced
	A vector length register is used to control the vector operations

The pipeline vector computers can be divided into 2 architectural configurations according to where the operands are received in a vector processor.
They are :
Memory -to- memory Architecture, in which source operands, intermediate and final results are retrieved directly from the main memory.
Register-to-register architecture, in which operands and results are retrieved indirectly from the main memory through the use of large number of vector or scalar registers.

14.2.2 Pipelined Vector Processing Methods

Vector computations are often involved in processing large arrays of data. By ordering successive computations in the array, the vector array processing can be classified into three types :

- **Horizontal Processing**, in which vector computations are performed horizontally from left to right in row fashion.
- **Vertical processing**, in which vector computations are carried out vertically from top to bottom in column fashion.
- **Vector looping**, in which segmented vector loop computations are performed from left to right and top to bottom in a combined horizontal and vertical method.

A simple vector summation computation illustrate these vector processing methods

Let { a_i for 1 <= i<= n) ne n scalar contstants, $X_j = (X_{1j}, X_{2j} \ldots\ldots X_{mj})^T$ for j = 1,2,3 ….n ne n column vectors and $Y_j = (Y_{1j}, Y_{2j} \ldots\ldots Y_m)^T$ be a column vector of m components. The computation to be performed is

$Y = a_i.x_1 + a_2.x_2 + \ldots.\ a_n.x_n$

$Y_1 = Z_{11} + Z_{12} + \ldots..Z_{1n}$
$Y_2 = Z_{21} + Z_{22} + \ldots..Z_{2n}$
.
.
.
$Y_m = Z_{m1} + Z_{m2} + \ldots..Z_{mn}$

Horizontal Vector Processing
In this method all components of the vector y are calculated in sequential order, yi for i = 1,2,….m. Each summation involving n-1 additions must be completed before switching to the evaluation the next summation.

Vertical Vector Processing :
The sequence of additions in this method are, compute the partial sum sequentially through the pipeline (in row wise $z_{11}+z_{12}\ldots$)
Computer the partial sum in the column format repeatedly.

Vector Looping Method:

It combines the horizontal and vertical approaches into a block approach.

14.3 Let us Sum Up

The various vector instructions have been discussed and the comparative study between scalar and vector helps in known differences existing between them. The pipeline vector computers are also divided into 2 architectural configurations according to where the operands are received in a vector processor has been discussed and the various pipelined vector processing methods has been mentioned.

14.4 Lesson-end Activities

1. Compare scalar and vector processing.
2. Explain various vector processing methods.
3. What is pipeline chaining? Also, give the characteristics of vector processing.

14.5 Points for discussions

- Pipelining
- Difference between vector and scalar
- Vector Processing methods

14.6 References

- http://www.cs.berkeley.edu/~pattrsn/252F96/Lecture06.ps
- http://www.pcc.qub.ac.uk/tec/courses/cray/stu-notes/CRAY-completeMIF_2.html
- http://www-ugrad.cs.colorado.edu/~csci4576/VectorArch/VectorArch.html

UNIT – IV

Lesson 15 : SIMD Array Processors, Organization, Masking and Data Routing & Inter PE communications

15.0 Aims and Objectives

The aim of this lesson is to know about various organization and control mechanisms of array processors.

15.1 Introduction

A synchronous array of parallel processors is called an array processor which consists of multiple processing elements (PE) under supervision of one control unit (CU). Some array processor may use 2 routing register, one for input and the other for output. Each PEi is either active or in the inactive mode during each instruction cycle. If a PEi is active, it executes the instruction broadcast to it by the CU. If a PE_i is inactive, it will not execute the broadcast instruction. The masking schemes are used to specify the status flag of PE_i.

There are fundamental decisions in determining the appropriate structure of an interconnection network for an SIMD machine. They are made between Operation Modes, Control Strategies, Switching methodologies, Network Topologies.

15.2 SIMD Array Processor

A synchronous array of parallel processors is called an array processor which consists of multiple processing elements (PE) under supervision of one control unit (CU). An array processor can handle single instruction multiple data streams (SIMD). It is also known as SIMD computers.

SIMD appears in 2 basic architectural organization

- Array Processors using random access memory
- Associative Processors

Classification of Parallel Machines

Depending on whether there is one or several of these streams we have 4 classes of computers.

- Single Instruction Stream, Single Data Stream: SISD
- Multiple Instruction Stream, Single Data Stream: MISD
- Single Instruction Stream, Multiple Data Stream: SIMD
- Multiple Instruction Stream, Multiple Data Stream: MIMD

15.2.1 SIMD Computer Organization

- All N identical processors operate under the control of a single instruction stream issued by a central control unit.
- There are N data streams; one per processor, so different data can be used in each processor.

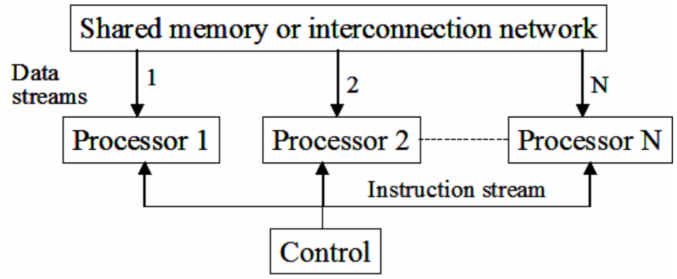


Figure 15.1 Configuration of SIMD Array Processor

- The processors operate *synchronously* and a global clock is used to ensure lockstep operation, i.e., at each step (global clock tick) all processors execute the same instruction, each on a different datum.
- Array processors such as the ICL DAP, Connection Machine CM-200, and MasPar are SIMD computers.
- SIMD machines are particularly useful at exploiting data parallelism to solve problems having a regular structure in which the same instructions are applied to subsets of data.

$$\begin{matrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{matrix} + \begin{matrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{matrix} = \begin{matrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{matrix}$$

The same instruction is issued to all 4 processors (add two numbers), and all processors execute the instructions simultaneously. It takes one step to add the matrices, compared with 4 steps on a SISD machine.

- In this example the instruction is simple, but in general it could be more complex such as merging two lists of numbers.
- The data may be simple (one number) or complex (several numbers).

- Sometimes it may be necessary to have only a subset of the processors execute an instruction, i.e., only some data needs to be operated on for that instruction. This information can be encoded in the instruction itself indicating whether
  - the processor is active (execute the instruction)
  - the processor is inactive (wait for the next instruction)
- SIMD machines usually have 1000's of very simple processors. Shared memory SIMD machines are unrealistic because of the cost and difficulty in arranging for efficient access to shared memory for so many processors. There are no commercial shared memory SIMD machines.
- MIMD machines use more powerful processors and shared memory machines exist for small numbers of processors (up to about 100).

15.2.2 Masking and Data Routing

Each processor PE has its own memory PEM_i. It has a set of working registers and flags names A_i, B_i ,C_i. It contains an Arithmetic and Logic unit S_i and a local index register Ii, an address register D_i and a data routing register R_i. The R_i of each PE_i is connected to the R_j of other PEs via the interconnection network. When data transfer among PEs occurs, it is the content of Ri registers that are being transferred.

Some array processor may use 2 routing register, one for input and the other for output. Each PEi is either active or in the inactive mode during each instruction cycle. If a PEi is active, it executes the instruction broadcast to it by the CU. If a PE_i is inactive, it will not execute the broadcast instruction.

The masking schemes are used to specify the status flag S_i of PE_i. The conventions $S_i = 1$ is chosen for an active PE_i and $S_i = 0$ for an inactive PE_i. In the CU, there is a global index register I and a Masking register M. The M register has N bits.

The physical length of a vector is determined by the number of PEs. The CU performs the segmentation of a long vector into vector loops, the setting of a global address, and the offset increment.

- Two processes are employed
  Master Process:
  - Holds pool of tasks for worker processes to do
  - Sends worker a task when requested
  - Collects results from workers

  Worker Process: repeatedly does the following
  - Gets task from master process
  - Performs computation
  - Sends results to master
- Worker processes do not know before runtime which portion of array they will handle or how many tasks they will perform.
- Dynamic load balancing occurs at run time: the faster tasks will get more work to do.

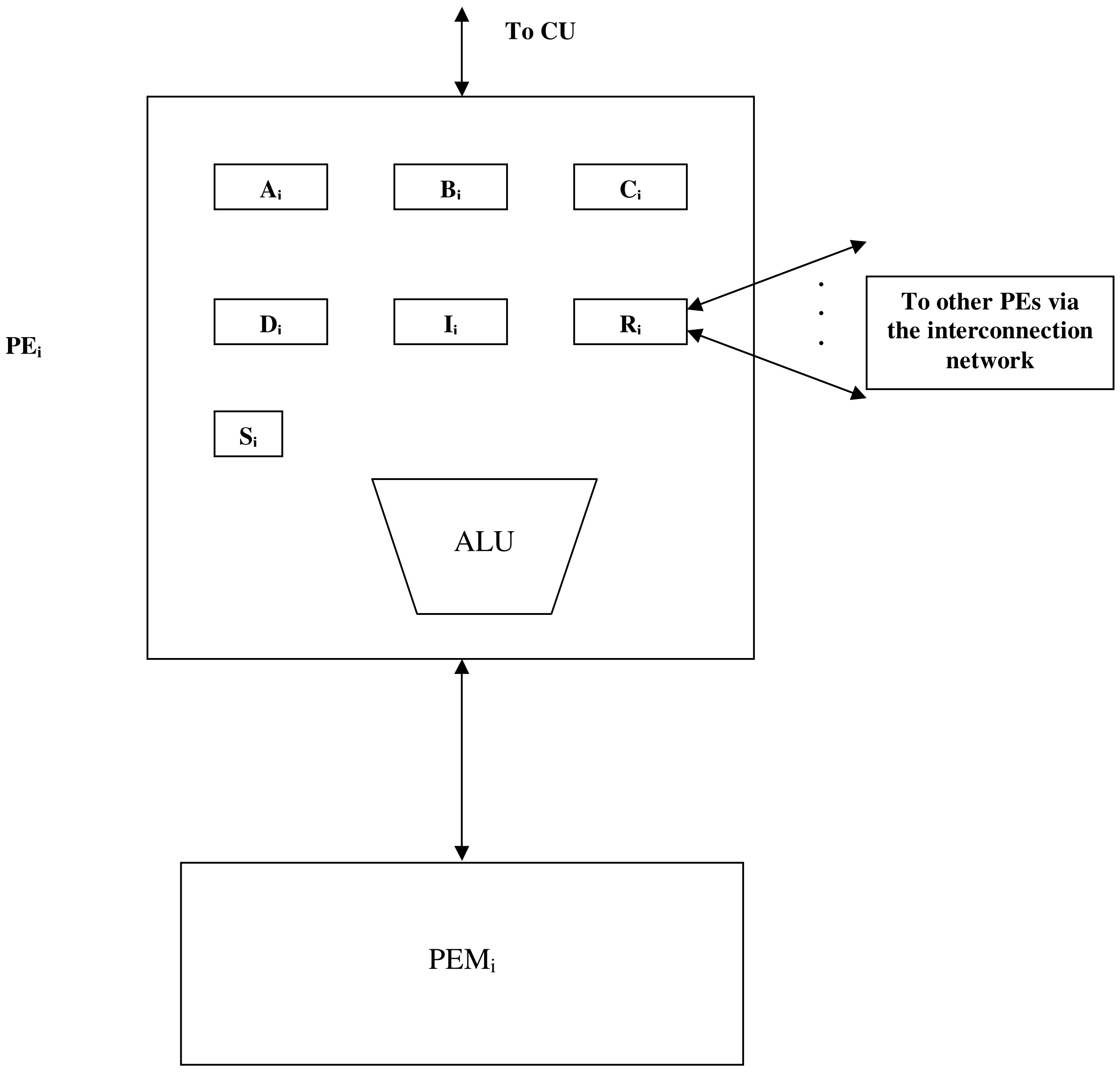


Figure 15.2 Components in a Processing element (PE_i)

15.2.3 Inter PE Communications

These are fundamental decisions in determining the appropriate structure of an interconnection network for an SIMD machine. The decisions are made between

- Operation Modes
- Control Strategies
- Switching methodologies
- Network Topologies

Operation Mode

The operations modes of interconnection networks can be of 3 categories

- Synchronous
- Asynchronous
- Combined.

Synchronous communication is needed for establishing communication paths synchronously for either a data manipulating function or for a data instruction broadcast.
Asynchronous communication is needed for multiprocessing in which connection requests are issued dynamically. A system may also be designed to facilitate both synchronous and asynchronous operations.

Control Strategy

A typical interconnection networks consists of number of switching elements and interconnecting links. Interconnection functions are realized by properly setting control of the switching elements. The control setting function can be of 2 types

- Distributed Control managed by individual switching element.
- Centralized Control managed by a centralized

Switching Technology

The 2 major switching methodologies are

- Circuit switching
- Packet switching
- In circuit switching a physical path is actually established between source and destination before transmission of data.
- This is suitable for bulk data transmission.
- In packet switching, data is put in a packet and routed through interconnection network without establishing a physical connection path.
- This is suitable for short messages.

Network Topology

A network can be depicted by a graph which nodes represent switching points and edges represent communication links. The topologies can be of 2 types

- Static
- Dynamic

In static link between 2 processors are passive and dedicated buses cannot be reconfigured for direct connection to other processors.
In dynamic link, configuration can be made by setting the network's active switching elements.

15.3 Let us Sum Up

The SIMD computer organization has been explained. The various data masking and routing concepts has been explained. The concept of master and slave process has been discussed. These are fundamental decisions in determining the appropriate structure of an interconnection network for an SIMD machine. The decisions are made between Operation Modes, Control Strategies, Switching methodologies, Network Topologies.

15.4 Lesson-end Activities

1. Discuss SIMD computer architecture in detail.
2. Discuss Inter PE communication.

15.5 Points for Discussions

- In circuit switching a physical path is actually established between source and destination before transmission of data.
- This is suitable for bulk data transmission.
- In packet switching, data is put in a packet and routed through interconnection network without establishing a physical connection path.
- Synchronous communication is needed for establishing communication paths synchronously for either a data manipulating function or for a data instruction broadcast.
- Asynchronous communication is needed for multiprocessing in which connection requests are issued dynamically.
- A system may also be designed to facilitate both synchronous and asynchronous operations

15.6 Suggested References

Computer Architecture and Parallel Processing – Kai Hwang

Lesson 16 : SIMD Interconnection Networks, Static Vs Dynamic, Mesh Connected Illiac Network, Cube Interconnection Network, Shuffle-Exchange Omega Network

Contents:

16.0 Aims and Objectives

The main aim of this lesson is to learn Single stage, recirculating networks and Multistage Networks. Important network classes has been included such as Illiac Network, Cube Network, Omega Network.

16.1 Introduction

The SIMD networks classified into 2 categories based on topologies called as Static Networks and Dynamic Networks. The diagrammatical representation of static interconnection networks is shown. Dynamic networks has been further classified as Single stage versus Multistage. Examples of Single stage network is implemented in ILLIAC and examples of Multistage network is Omega network

16.2 SIMD Interconnection Networks

Various interconnection networks have been suggested for SIMD computers.

16.2.1 Static versus Dynamic Networks

The topological structure of an SIMD array processor is mainly characterized by the data routing network used in interconnecting the processing elements.

Interconnection Networks

- Parallel computers with many processors do not use shared memory hardware.
- Instead each processor has its own local memory and data communication takes place via message passing over an *interconnection network*.

- The characteristics of the interconnection network are important in determining the performance of a multicomputer.
- If network is too slow for an application, processors may have to wait for data to arrive.

Interconnection Networks and Message Passing

In this case each processor has its own private (local) memory and there is no global, shared memory. The processors need to be connected in some way to allow them to communicate data.

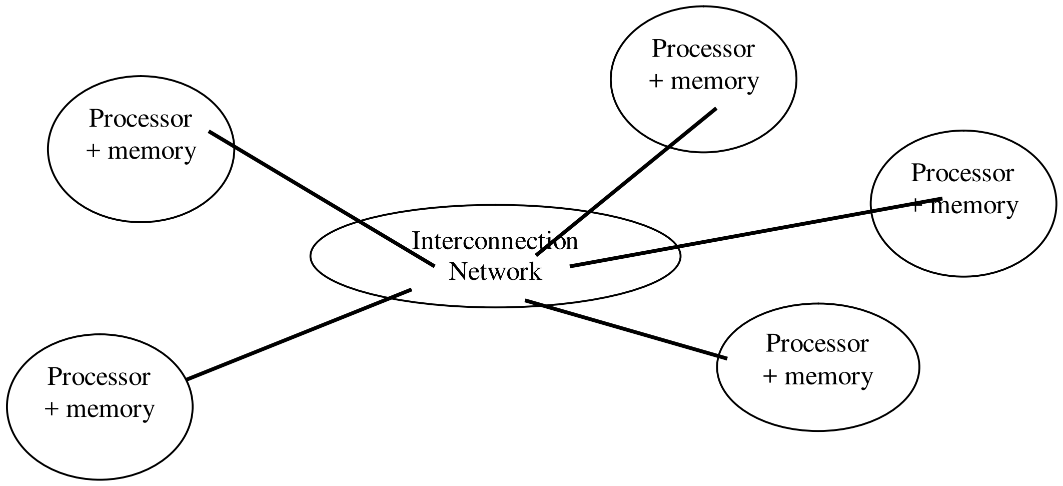


Figure 16.1 Interconnection Networks and Message Passing

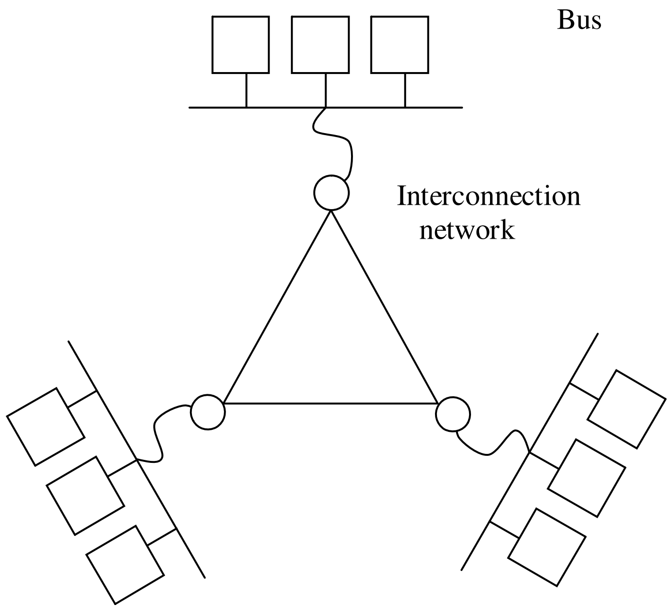


Figure 16.2 Interconnection Networks

The SIMD interconnection networks are classified into the following 2 categories based on network topologies

- Static Networks
- Dynamic Networks

Static Networks

Topologies in the static networks can be classified according to the dimension required for layout.
One dimensional topologies include

- Linear array

Two dimensional topologies include

- The ring
- Star
- Tree
- Mesh
- Systolic Array

Thee dimensional topologies include

- Completely connected chordal ring
- 3 cube
- 3 cube connected cycle

Dynamic Networks

2 classes of dynamic networks are

- single stage
- multi stage

Single Stage Networks

A single stage switching network with N input selectors (IS) and N output selectors (OS). Each is essentially a 1- to-D demultiplexer and each OS is an M-to-1 multiplexer. Cross bar network is a single stage network.

The single stage network is also called as recirculating network. Data items may have to recirculate through the single stage several times before reaching their final destinations. The number of recirculation depends on the connectivity in the single stage network.

Multistage Networks

Multi-stage networks are based on the so called shuffle-exchange switching element, which is basically a 2 x 2 crossbar. Multiple layers of these elements are connected and form the network.
Many stages of interconnected switches form a multistage SIMD networks.
Three characterizing feature describe multistage networks

- The Switch Box
- Network Topology

- Control Structure

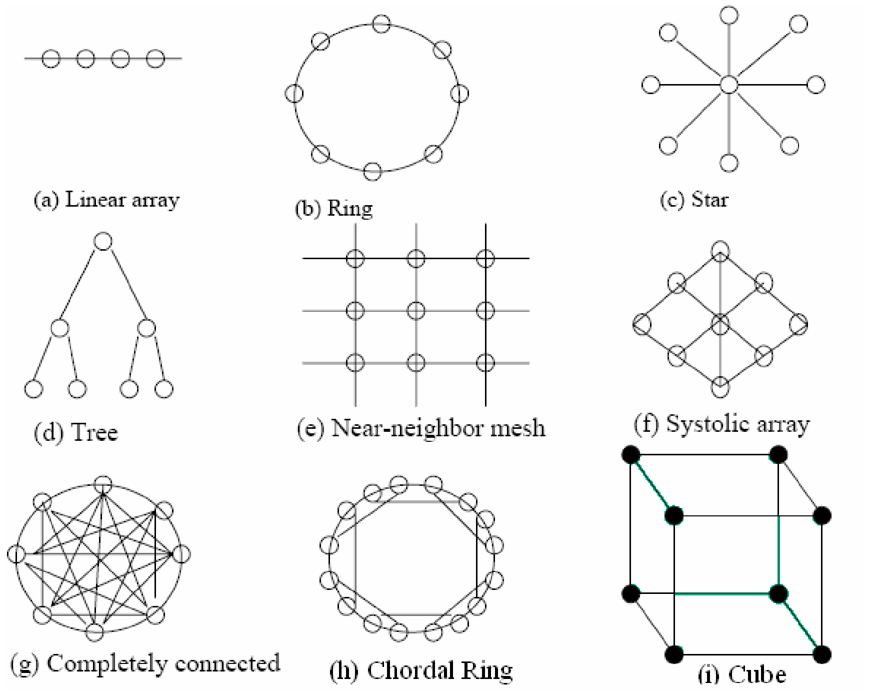


Figure 16.3 Static Interconnection Network Topologies

Each box is essentially an interchange device with 2 inputs and 2 outputs. There are 4 states in a switch box. They are

- Straight
- Exchange
- Upper Broadcast
- Lower broadcast.

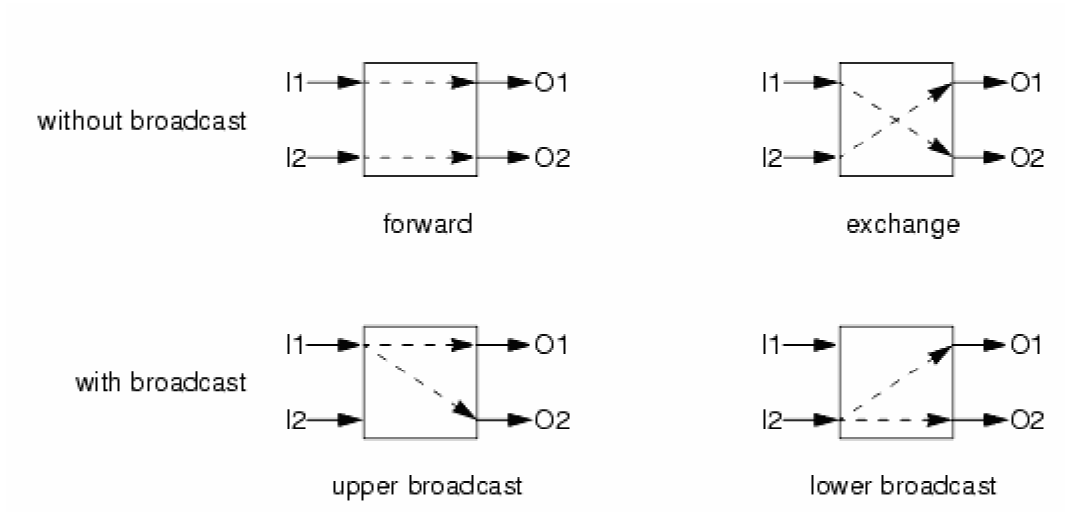


Figure 16.4 A two-by-two switching box and its four interconnection states

Examples of Multistage Networks

- Banyan
- Baseline
- Cube
- Delta
- Flip
- Indirect cube
- Omega

All multistage networks that are based on shuffle-exchange elements, is a blocking network because not all possible input-output connections can be made at the same time, since one path might block another (in contrast to a crossbar, which is nonblocking). Using crossbars instead of the shuffle-exchange elements, it is possible, to build nonblocking networks. Such networks are called CLOS-networks.

A multistage network is capable of connecting an arbitrary input terminal to an arbitrary output terminal. Multistage networks can be

- One sided
- Two Sided

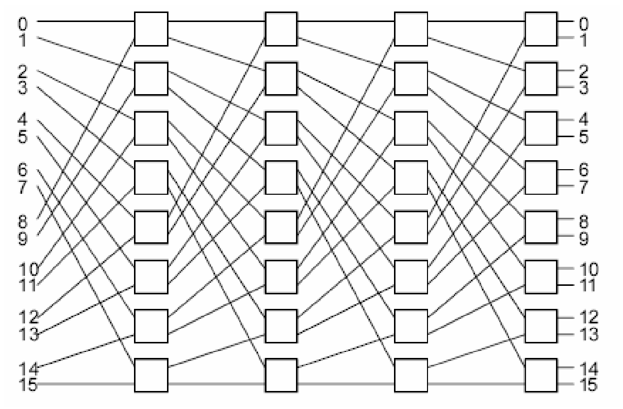
The one sided network called as full switches, have input-output ports on the same side.
The two sided network have an input side and output side and can be divided into three classes

- Blocking
- Arrangeable
- Non- Blocking

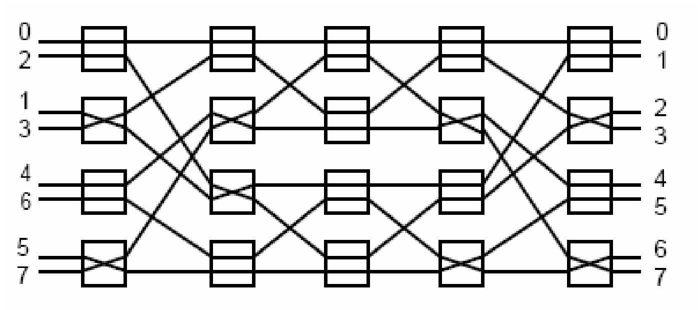
In Blocking networks, simultaneous connections of more than one terminal pair may result conflicts in the use of network communication links.
Examples : Data Manipulator, Flip, N cube, omega

In rearrangeable network, a network can perform all possible connections between inputs and outputs by rearranging its existing connections so that a connection path for a new input-output pair can always be established.
Example : Benes Network

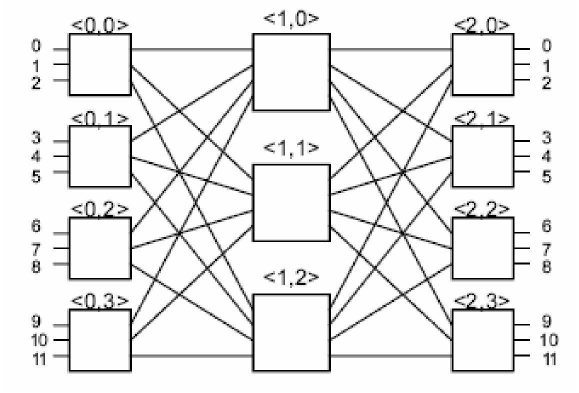
A non –blocking can handle all possible connections without blocking.
Example : Clos Network



(a) Omega Network



(b) Benes Network



(c) Clos Network

Figure 16.5 Several Multistage Interconnection Networks

16.2.2 Mesh-Connected Illiac Networks

In a mesh network nodes are arranged as a q-dimensional lattice, and communication is allowed only between neighboring nodes.
In a *periodic mesh*, nodes on the edge of the mesh have wrap-around connections to nodes on the other side. This is sometimes called a *toroidal mesh.*

Mesh Metrics
For a q-dimensional non-periodic lattice with kq nodes:

- Network connectivity = q
- Network diameter = q(k-1)

- Network narrowness = k/2
- Bisection width = kq-1
- Expansion Increment = kq-1
- Edges per node = 2q

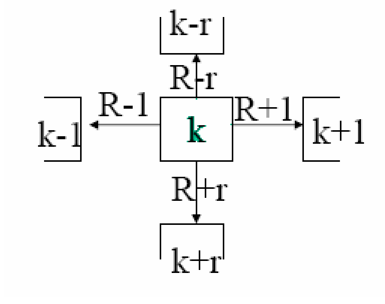


Figure 16.6 Topology

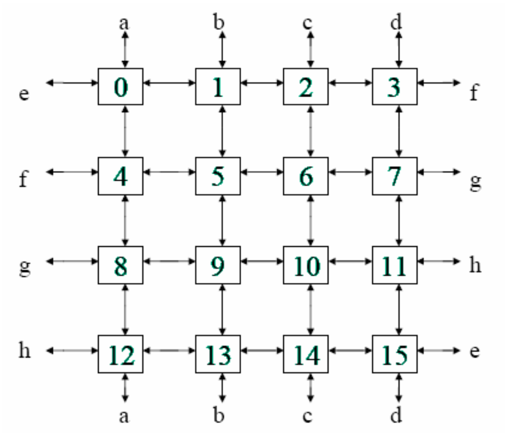


Figure 16.7 Mesh Connections

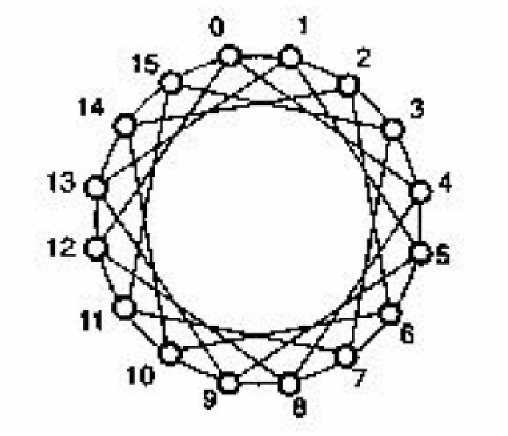


Figure 16.8 Mesh Redrawn

The topology is formerly described by the four routing functions:

- R+ : I→ i+1 mod N=> (0,1,2…,14,15)
- R- : I→i-1 mod N=> (15,14,…,2,1,0)
- R+r: I→i+r mod N=> (0,4,8,12)(1,5,9,13)(2,6,10,14)(3,7,11,15)
- R-r: I→i-r mod N=> (15,11,7,3)(14,10,6,2)(13,9,5,1)(12,8,4,0)

where (i j k)(l m n)=> i-->j,j-->k,k-->i ; l-->m,m-->n,n-->l and r = N

Each PE_i is directly connected to its four neighbors in the mesh network. Horizontally, all PEs of all rows form a linear circular list as governed by the following two permutations, each with a single cycle of order N. The permutation cycles (a b c) (d e) stands for permutation a→b, b→c, c→a and d→e, e→d in a circular fashion with each pair of parentheses.

R_{+1} = (0 1 2 ….N-1)

R_{-1} = (N-1 ….. 2 1 0)

For the example betwork of N = 16 and r = √16 = 4, the shift by a distance of four is specifies by the followimg permutations, each with four cycles of order four each:

R_{+4} = (0 4 8 12)(1 5 9 13)(2 6 10 14)(3 7 11 15)

R_{-4} = (12 8 4 0)(13 9 5 1)(14 10 6 2)(15 11 7 3)

16.2.3 Cube Interconnection Networks

The cube network consists of m functions defined by $cube_i$ (P_{m-I} ..P_{i+1} P_iP_{i-i} ... P_o)

$= P_{m-i} \ldots P_{i+i}P'_iP_{i-i} \ldots P_o$

for 0 <= i < m. When the PE addresses are considered as the corners of an m-dimensional cube this network connects each PE to its m neighbors.

Dimensionality of a cube "n" = log2N

- Communication links connect each pair of PEs with addresses that differ in 1 bit only (distance of 1).
- Examples

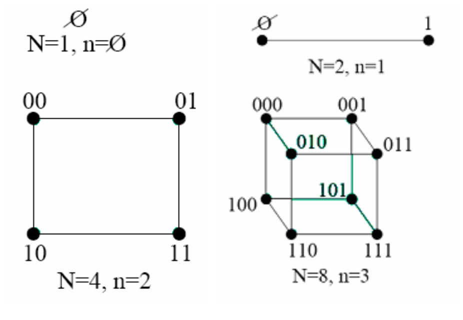


Figure 16.9 A 3 Cube of 8 nodes

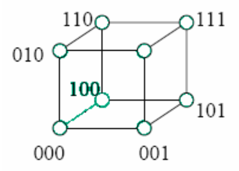


Figure 16.10 A 3-cube with nodes denoted as C2 C1 C0 in binary.

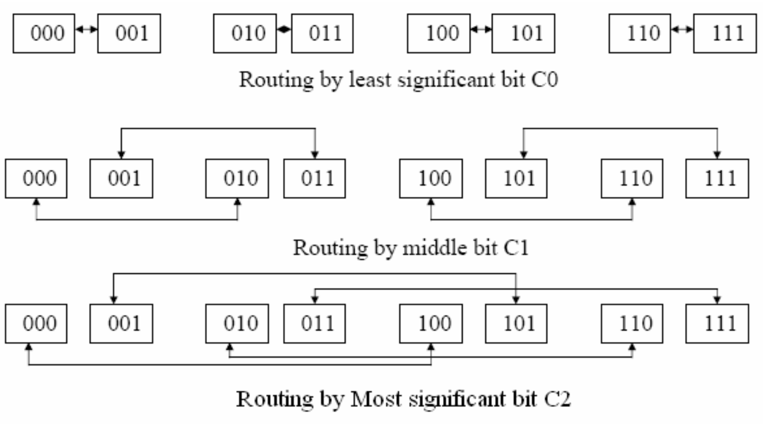

000
001
010
011
100
101
110
111
Routing by least significant bit C0
Routing by middle bit C1
Routing by Most significant bit C2

Figure 16.11 The recirculating Network

It takes <= log N steps to rotate data from any PE to another.
• Routing functions are formerly described as:
Ci (An-1 …. A1 A0)= An-1…Ai+1 A'i Ai-1……A0
V i =0,1,2,…,n-1

Example: N=8 => n=3

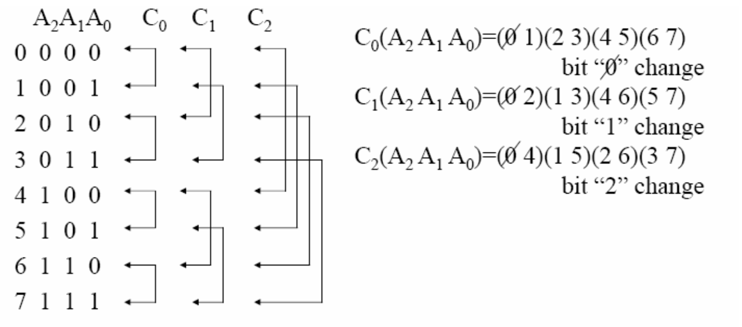

$A_2A_1A_0$ C_0 C_1 C_2
0 0 0 0
1 0 0 1
2 0 1 0
3 0 1 1
4 1 0 0
5 1 0 1
6 1 1 0
7 1 1 1
$C_0(A_2A_1A_0)=(\emptyset\ 1)(2\ 3)(4\ 5)(6\ 7)$
bit "∅" change
$C_1(A_2A_1A_0)=(\emptyset\ 2)(1\ 3)(4\ 6)(5\ 7)$
bit "1" change
$C_2(A_2A_1A_0)=(\emptyset\ 4)(1\ 5)(2\ 6)(3\ 7)$
bit "2" change

Figure 16.12 A multistage Cube network for N = 8

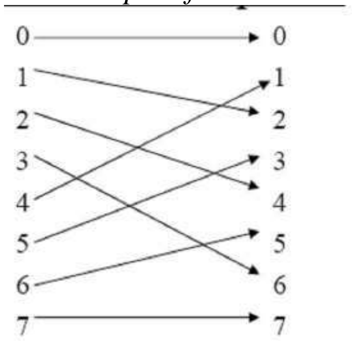
Two functions straight and exchange switch boxes are used in constructing multistage cube networks. The stages are numbered as 0 at input end and increased to n-1 at the output stage.

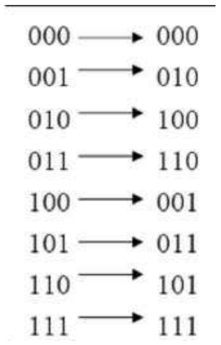
16.2.4 Shuffle-Exchange Omega Networks

A shuffle-exchange network consists of n=2k nodes, and two kinds of connections.
1.Routing
• Have wide applications in implementing parallel algorithms such as FFT, sorting, AT, and polynomial evaluations.
• Two routing functions: shuffle "S" and Exchange "E".
Shuffle:
$S(A)=S(A_{n-1}\ldots A_1A_0)=A_{.n-2}\ldots A_1A_0A_{n-1}$, $0<A<1$
Which is effectively like shuffling the bottom half of a card deck into the top half.

Exchange:
$E(A_{n-1}\ldots A_1A_0)= (A_{n-1}\ldots A_1A_0')=C_0$. The complement of LSM means data exchange between 2 PEs with adjacent addresses.

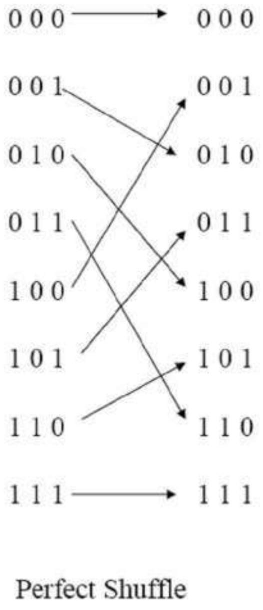
- *Exchange* connections links nodes whose numbers differ in their lowest bit.
- *Perfect shuffle* connections link node i with node 2i mod(n-1), except for node n-1 which is connected to itself.
- *Example of n = 8*





An N by N omega network, consists of n identical stages, where each stage is a perfect shuffle interconnection followed by a column of N/2 four-function interchange boxes under individual box control. The shuffle connects output $P_{n-1}\ldots P_1 P_0$ of stage i to input $P_{n-2}\ldots P_1P_0P_{n-1}$ of stage i-1. Each interchange box in an omega network is controlled by the n-bit destination tags associated with the data on its input lines.

The perfect shuffle cuts the deck into 2 halves from the center and remixes them evenly. The inverse perfect shuffle does the opposite to restore the original ordering.



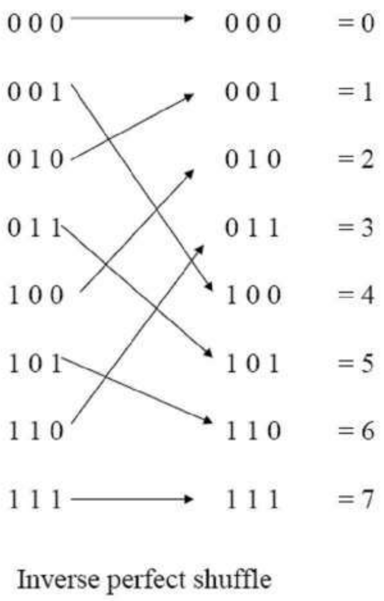


Figure 16.13 (a) Perfect Shuffle and Inverse Perfect Shuffle for n = 8

8-Node Shuffle-Exchange Network

Below is an 8-node shuffle-exchange network, in which shuffle links are shown with solid lines, and exchange links with dashed lines.

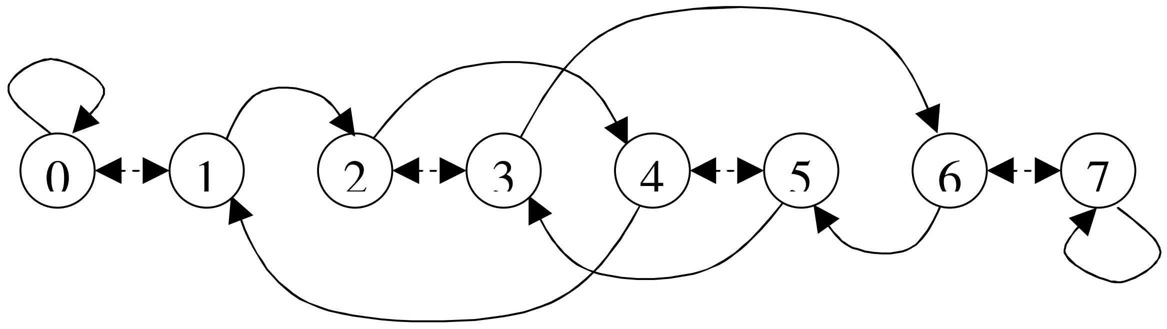


Figure 16.13 (b) Shuffle Exchange recirculating network for N = 8

Shuffle-Exchange Networks

- What is the origin of the name "shuffle-exchange"?
- Consider a deck of 8 cards numbered 0, 1, 2,…,7. The deck is divided into two halves and shuffled perfectly, giving the order:
  0, 4, 1, 5, 2, 6, 3, 7
- The final position of a card i can be found by following the shuffle link of node i in a shuffle-exchange network.

Shuffle-Exchange Networks

- Let ak-1, ak-2,…, a1, a0 be the address of a node in a shuffle-exchange network in binary.
- A datum at this node will be at node number
  ak-2,…, a1, a0, ak-1
  after a shuffle operation.
- This corresponds to a cyclic leftward shift in the binary address.
- After k shuffle operations we get back to the node we started with, and nodes through which we pass are called a *necklace*.

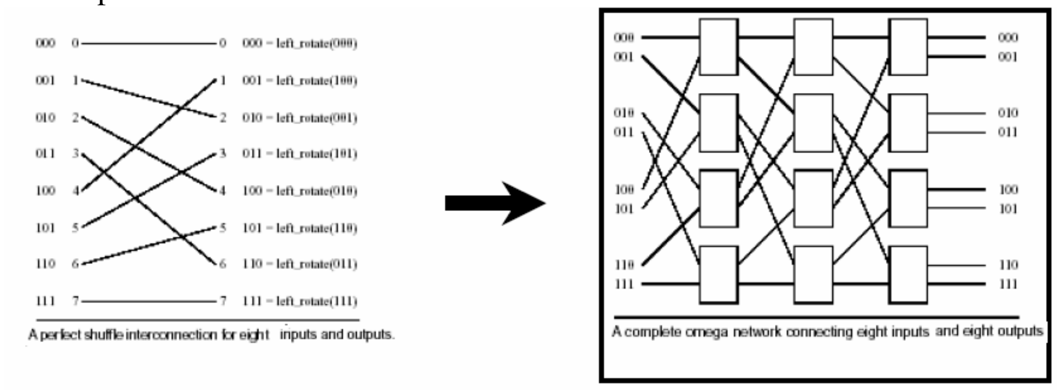

000 0 — 0 000 = left_rotate(000)
001 1 — 1 001 = left_rotate(100)
010 2 — 2 010 = left_rotate(001)
011 3 — 3 011 = left_rotate(101)
100 4 — 4 100 = left_rotate(010)
101 5 — 5 101 = left_rotate(110)
110 6 — 6 110 = left_rotate(011)
111 7 — 7 111 = left_rotate(111)
A perfect shuffle interconnection for eight inputs and outputs.
000
001
010
011
100
101
110
111
A complete omega network connecting eight inputs and eight outputs

Figure 16.14 The Multi stage Omega Network

16.3 Let us Sum Up

Various SIMD interconnection networks has been explained in detail and the different classification of networks as static and dynamic networks. An Illiac network, Cube network and Omega network has been explained.

16.4 Lesson-end Activities

1. Explain Static Networks Vs Dynamic Networks.
2. Discuss Mesh-Connected Illiac Networks and Cube Interconnection Networks
3. Discuss Shuffle-Exchange Omega Networks

16.5 Points for Discussions

- Static Versus Dynamic
- Single Stage versus Multi Stage
- Mesh Connected Illiac Network
- Cube Network
- Omega Network

16.6 References

- C. P. Kruskal and M. Snir, A unified theory of interconnection network structure, Theorey.Comput. Sci., 48 (1986), pp. 75–94.
- H. J. Siegel, The universality of various types of SIMD machine interconnection networks, in Proceedings of the 4th Annual Symposium on Computer Architecture, Silver Spring, MD,1977, pp. 70–79.
- Quinn, Michael. Parallel Programming in with MPI and OpenMP. McGraw Hill. ISBN 070281656-2
- Dally, William. Towles, Brian. Principles and Practices of Interconnection Networks. Morgan Kaufmann. ISBN 0-12-200751-4

Lesson 17 : Microprocessor Architecture and Programming, Functional structures

Contents:

17.0 Aims and Objectives

The main aim of this lesson is to learn the architectural models of multiprocessor defined as loosely coupled and tightly coupled multiprocessor. A number of architectural features are described below for a processor to be effective in multiprogramming environment.

17.1 Introduction

A multiprocessor is expected to reach faster speed than the fastest uni-processor. Multiprocessor characteristics are Interconnection Structures, Interprocessor Arbitration, Interprocessor Communication and Synchronization, Cache Coherence. Multiprocessing sometimes refers to the execution of multiple concurrent software processes in a system as opposed to a single process at any one instant. The 2 architectural models of Multiprocessor are
Tightly Coupled Multiprocessor are defined as Tasks and/or processors communicate in a highly synchronized fashion, Communicates through a common shared memory, Shared memory system
Loosely Coupled System is defined as Tasks or processors do not communicate in a synchronized fashion, Communicates by message passing packets, Overhead for data exchange is high, Distributed memory system.

17.2 Micro Processor Architecture and Programming

A number of processors (two or more) are connected in a manner that allows them to share the simultaneous execution of a single task. In addition, a multiprocessor consisting of a number of single uni-processors is expected to be more cost effective than building a high-performance single processor. An additional advantage of a multiprocessor consisting of *n* processors is that if a single processor fails, the remaining fault-free *n*-1 processors should be able to provide continued service albeit with degraded performance.

17.2.1 Functional Structures

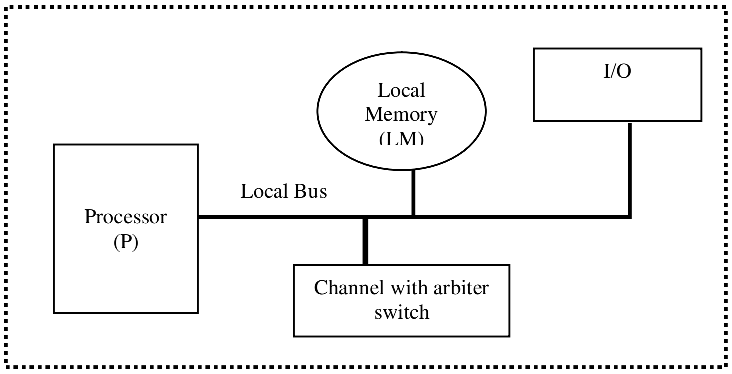
Multiproceesors are characterized by 2 attributes: First a multiprocessor is a single computer that includes multiple processor and second processors may communicate and cooperate at different levels in solving a given problem. The communication may occur by sending messages from one processor to the other by sharing a common memory.

2 Different Architectural Models of Multiprocessor are
Tightly Coupled System
- Tasks and/or processors communicate in a highly synchronized fashion
- Communicates through a common shared memory
- Shared memory system

Loosely Coupled System
- Tasks or processors do not communicate in a synchronized fashion
- Communicates by message passing packets
- Overhead for data exchange is high
- Distributed memory system

17.2.2 Loosely Coupled Multiprocessors

- In loosely coupled Multiprocessors each processor has a set of input-output devices and a large local memory from where instructions and data are accessed.
- Computer Module is a combination of
    - Processor
    - Local Memory
    - I/O Interface
- Processes which executes on different computer modules communicate by exchanging messages through a Message Transfer System (MTS).
- The Degree of coupling is very Loose. Hence it is often referred as distributed system.
- Loosely coupled system is usually efficient when the interactions between tasks are minimal.



(a) Figure 17.1 A Computer Module

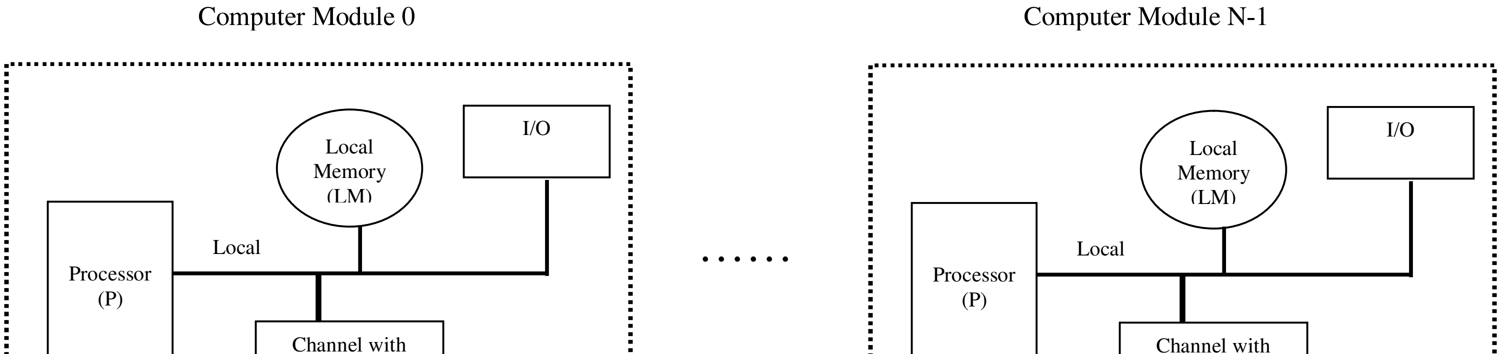


Figure 17.2 Non-Hierarchical Loosely coupled Multiprocessor System

It consists of a

    - processor
    - local memory
    - Local Input-output devices
    - Interface to other computer Modules

- The interface may contain a channel and arbiter switch (CAS)
- If requests from two or more different modules collide in accessing a physical segment of the MTs, the arbiter is responsible for choosing one of the simultaneous requests according to a given service discipline.
- It is also responsible for delaying requests until the servicing of the selected request is completed.
- The channel within the CAS may have a high speed communication memory which is used for buffering block transfers of messages.
- The message transfer system is a time shared bus or a shared memory system.

17.2.3 Example of Loosely Coupled Multiprocessors : The Cm* Architecture

The example of LCS is the Cm* architecture. Each computer module of the Cm* includes a local switch called the Slocal. The Slocal intercepts and routes the processor's requests to the memory and the I/O devices outside of the computer module via a map bus. It also accepts references from other computer modules to its local memory and I/O devices. The address translation uses the four higher bits of the processors address along with the current address by the X-bit of the LSI -11 processor status word (PSW), to access a mapping tablewhich determines whether the reference is local or nonlocal.
A virtual address of the nonlocal reference is formed by concatenating the nonlocal address field given by the mapping table and the source processors identification. The virtual address is fetched by the Kmap via the map bus in response to a service request for nonlocal access. A number of computer modules may be connected to a map bus so that they share the use of a single KMap. The Kmap is a processor that is responsible for mapping addresses and routing data between Slocals.

A cluster is regared as the lowest level, is made up of computer modules, a KMap and the map bus. Clustering can enhance the cooperative ability of the processors in a cluster to operate on shared data with low communication overhead.
The three processors in the Kmap are the Kbus, the Linc and the Pmap. The KBus is the bus controller which arbitrates requests to the map bus. The Pmap is the mapping processor which responds to requests from the Kbus and Linc. It also performs most of the request processing. The Pmap communicates with the computer modules in its cluster via the map bus which is a packet switched bus.
Three sets of queues provide interfaces between the Kbus, Linc and the Pmap. Since PMap is much faster than the memory in the computer modules, it is multiprogrammed to handle upto eight concurrent requests. Each of the eight partition is called a context and exists in the Pmap. Typically, each context processes one transaction. If one context needs to wait for a message packet to return with the reply to some request, the PMap switches to another context that is ready to run so that some other transaction can proceed concurrently.

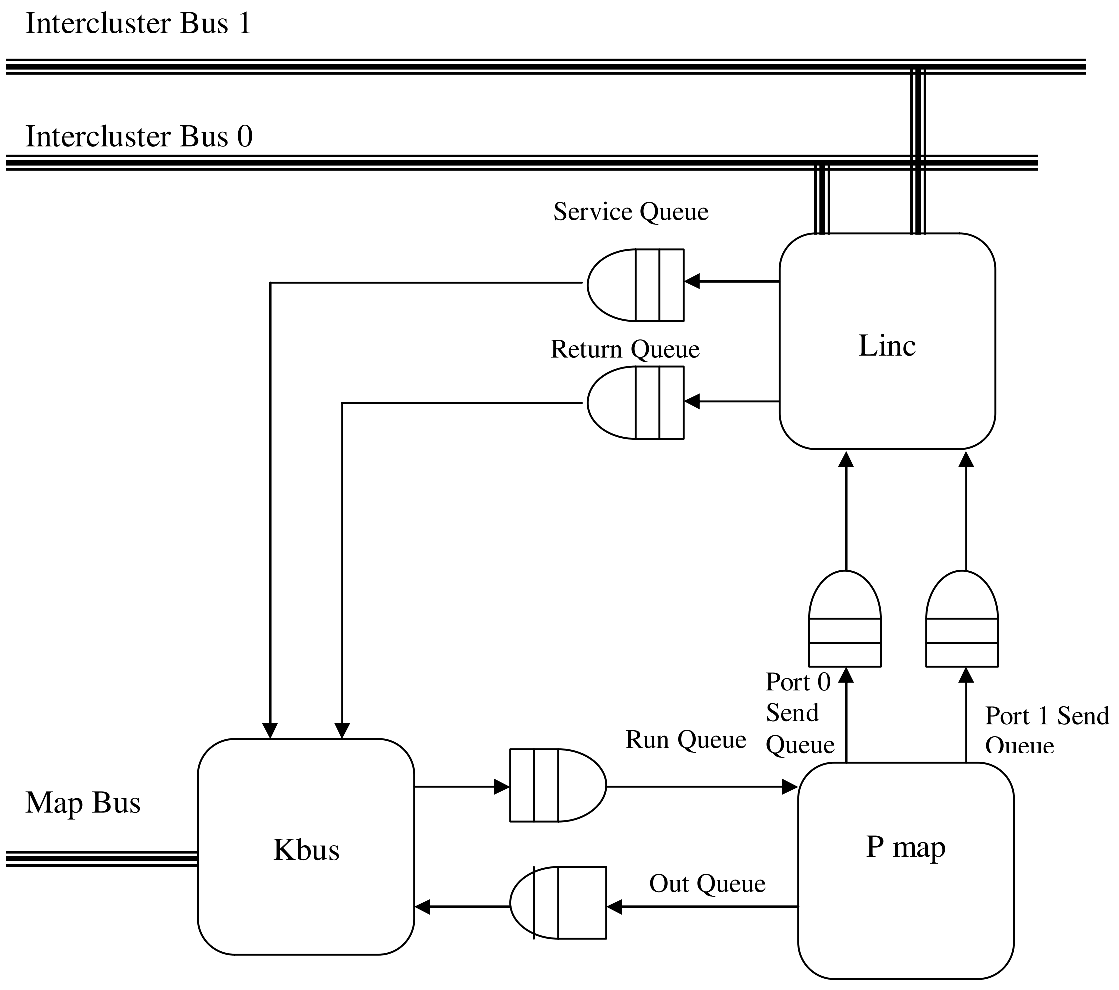


Figure 17.3 The components of the kmap in Cm*

17.2.4 Tightly Coupled Multiprocessors(TCS)

The throughput of the hierarchical loosely coupled multiprocessor may be too slow for some applications that require fast response times. If high speed or real time processing is required the TCS may be used.
Two typical models are discussed
In the first model, it consists of

- p processors
- l memory modules
- d input-output channels

The above units are connected through a set of three interconnection networks namely the

- processor-memory interconnection network (PMIN)
- I/O processor interconnection network (IOPIN)
- Interrupt Signal Interconnection Network (ISIN)

The PMIN is a switch which can connect every processor to every module. It has pl set of cross points. It is a multistage network. A memory can satisfy only one processor's request in a given memory cycle. Hence if more than one processors attempt to access the same memory module, a conflict occurs and it is resolved by the PMIN.

Another method to resolve the conflict is to associate a reserved storage area with each processor and it is called as ULM unmapped local memory. It helps in reducing the traffic in PMIN.

Since each memory references goes through the PMIN, it encounters delay in the processors memory switch so this can be overcome by using a private cache with each processor.

A multiprocessor organization which uses a private cache with each processor is shown. This multiprocessor organization encounters the cache coherence problem. More than one consistent copy of data may exist in the system.

In the figure there is a module attached to each processor that directs the memory reference to either the ULM or the private cache of that processor. This module is called the memory map and is similar in operation to the Slocal.

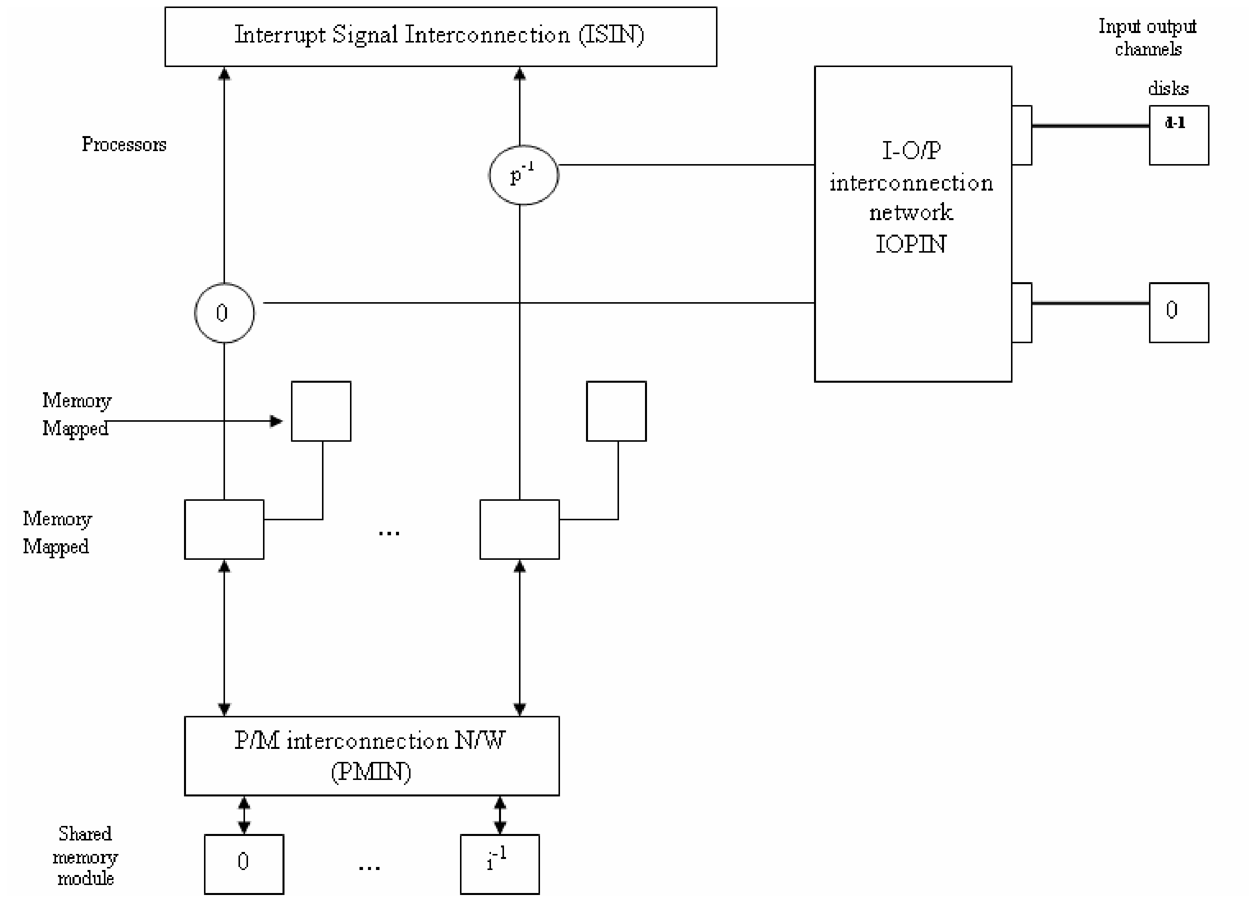

Interrupt Signal Interconnection (ISIN)
Input output channels
disks
d-1
Processors
p-1
I-O/P interconnection network IOPIN
0
0
Memory Mapped
Memory Mapped
...
P/M interconnection N/W (PMIN)
Shared memory module
0
...
i-1

Figure 17.4 Tightly Configured Multiprocessor configurations

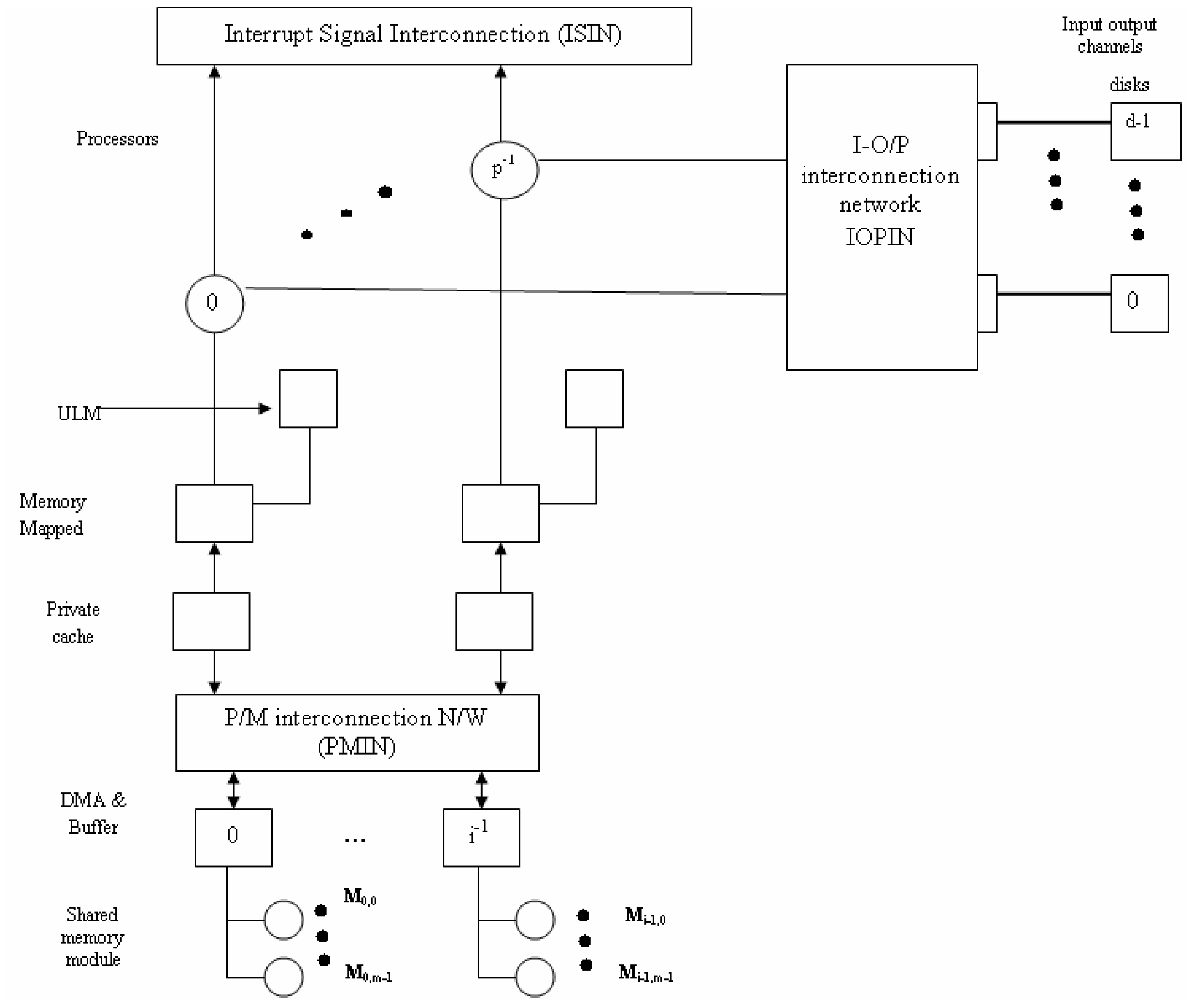


Figure 17.4 (Continued)

17.2.5 Example of Tightly Coupled Multiprocessor : The Cyber – 170 Architecture

This configuration has 2 subsystems.

- The central processing sub system
- The peripheral processing sub system

These subsystems have to access to a common central memory (CM) through a central memory controller, which is essentially a high speed cross bar switch. In addition to the central memory there is an optional secondary memory called the extended core memory (ECM) which is a low speed random access read-write memory. The ECM and CM form a 2 level memory hierarchy. Here the CMC becomes the switching center, which performs the combined functions of ISIN,IOPIN, and PMIN.

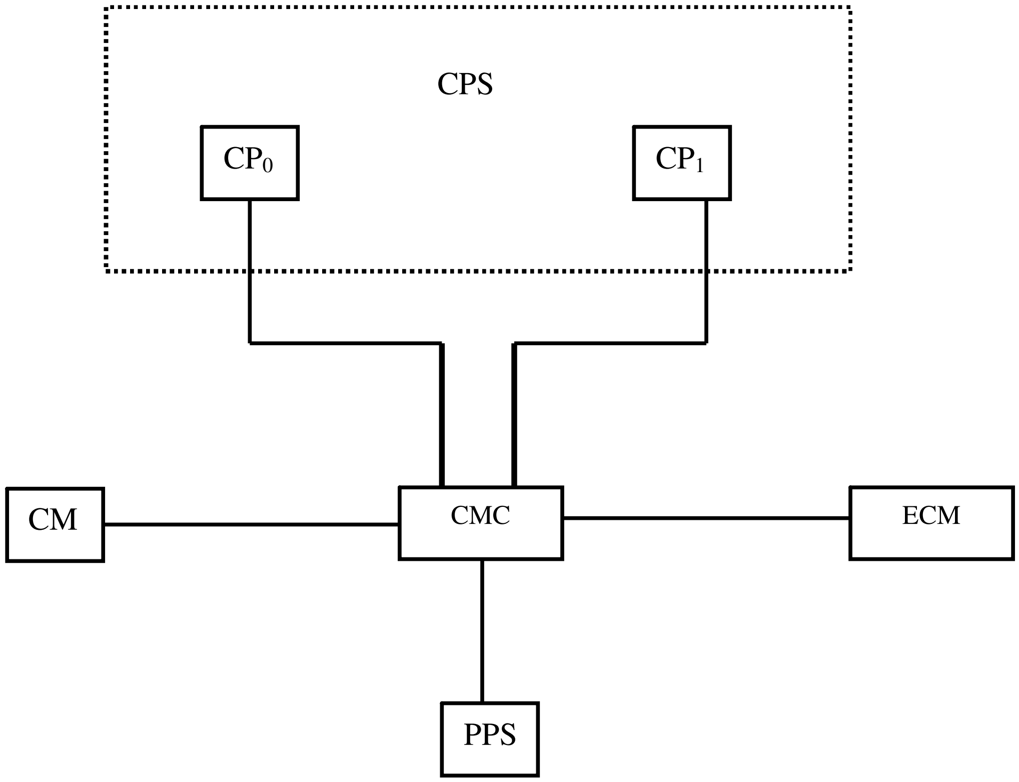


Figure 17.5 A Cyber-170 Multiprocessor configurations with the two processors

CM	: Central Memory
CMC	: Central Memory Controller
CP_i	: ith central processor
CPS	: Central Processing System
PPS	: peripheral processing subsystem

17.2.6 Processor Characteristics for Multiprocessing

A number of desirable architectural features are described below for a processor to be effective in multiprocessing environment.

Processor recoverability

The process and processor are 2 different entities. If the processor fails there should be another processor to take up the routine. Reliability of the processor should be present.

Efficient Context Switching

A general purpose register is a large register file that can be used for multi-programmed processor. For effective utilization it is necessary for the processor to support more than one addressing domain and hence to provide a domain change or context switching operation.

Large Virtual and Physical address space

A processor intended to be used in the construction of a general purpose medium to large scale multiprocessor must support a large physical address space. In addition a large virtual space is also needed.

Effective Synchronization Primitives

The processor design must provide some implementation of invisible actions which serve as the basis for synchronization primitives.

Interprocessor Communication mechanism

The set of processor used in multiprocessor must have an efficient means of interprocessor mechanism.

Instruction Set

The instruction set of the processor should have adequate facilities for implementing high level languages that permit effective concurrency at the procedural level and for efficiently manipulating data structures.

17.3 Let us Sum Up

The architectural model of Multiprocessor system has been discussed such as Tightly and Loosely coupled multiprocessor, its block diagram and examples has been neatly explained. The tightly coupled processor shares a common memory and the loosely coupled processor does not share a common memory. The concept of cache coherence has also be discussed. Various processor characteristics of multiprocessor have been last topic of this lesson.

17.4 Lesson-end Activities

1. Discuss in detail Micro Processor Architecture and Programming
2. Explain the various multiprocessor characteristics.

17.5 Points for Discussions

- Multiprocessor
- Tightly Coupled Multiprocessor
- Loosely Coupled Multiprocessor
- Cache Coherence

17.6 References

- Fundamentals of Computer Organization and Architecture Mostafa Abd-El-Barr & Hesham El-Rewini
- EE3.cma - Computer Architecture Roger Webb,R.Webb@surrey.ac.uk, University of Surrey http://www.ee.surrey.ac.uk/Personal/R.Webb/l3a15 also link from Teaching/Course page

Lesson 18 : Interconnection Networks

Contents:

18.0 Aims and Objectives

The main objective of this lesson is to define the characteristics of multiprocessor systems, (ie) is the ability of each processor to share a set of memory modules, and I/O devices. This sharing capability is provided through a set of interconnection networks.

18.1 Introduction

Interconnection networks helps in sharing resources, one is between the processor and the memory modules and the other between the processor and the I/O Systems. There are different physical forms available for interconnection networks. They are time shared or common bus, Crossbar switch and multistage networks for multiprocessors.

18.2 Interconnection Networks

The organization and performance of a multiple processor system are greatly influenced by the interconnection network used to connect them.

18.2.1 Time shared or Common Buses

The simplest interconnection system for multiprocessors is a communication path connecting all of the functional units. The common path is called as time shared or common bus. The organization is least complex and easier to configure. Such an interconnection is often a totally passive unit having no active components such as switches. Transfer operations are completely controlled by the bus interfaces of the sending and receiving units. Since the bus is shared, a mechanism must provide to resolve contention. The conflict resolution methods include static or fixed priorities, FIFO queues and daisy chaining.

A unit that wishes to initiate transfer must first determine the availability of status of the bus, and then address the destination unit to determine its availability and capability to receive the transfer. A command is also issued to inform the destination unit what operation it is to perform with the data being transferred, after which the data transfer is finally initiated. A receiving unit recognizes its address placed on the bus and responds of the control signals from the sender.

Example of Time shared bus is PDP – 11.

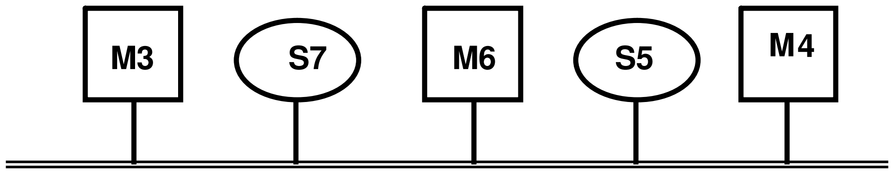


Figure 18.1 A single bus Multiprocessor organization

M3 wishes to communicate with S5

[1] M3 sends signals (address) on the bus that causes S5 to respond

[2] M3 sends data to S5 or S5 sends data to M3 (determined by the command line)

Multiprocessor with unidirectional buses uses both the buses and not much is actually gained.

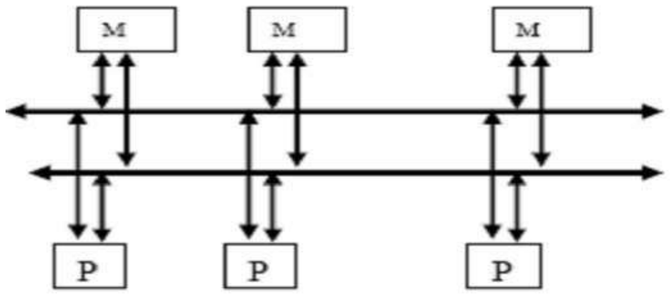


Figure 18.2 Multi-bus multiprocessor organization

This method increases the complexity. The interconnection subsystem becomes an active device.

Characteristics that affects the performance of the bus

- Number of active devices on the bus
- Bus arbitration Algorithm
- Centralization
- Data Bandwidth
- Synchronization of data transmission
- Error detection

Various Bus arbitration Algorithms are

The static Priority Algorithm

When multiple devices concurrently request the use of bus, the device with highest priority is granted the bus. This method is called as daisy chaining and all the services are assigned static priorities according to their locations along a bus grant control line. The device closest to the bus control unit will get highest priority.
Requests are made through a common line, BRQ Bus request. If the central bus control accepts the request it passes a BGT Bus grant signal to the concerned device iff the SACK Bus busy line is free or idle.

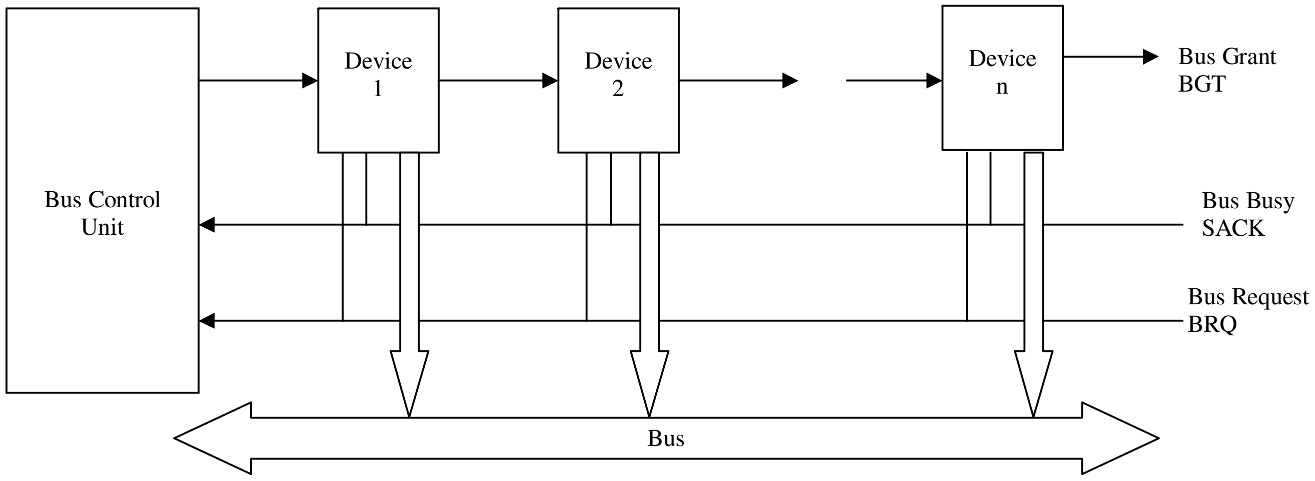


Figure 18.3 Static daisy chain implementation of a system bus

The Fixed Time Slice Algorithm

The available bus band widths divided into fixed time slices and then offered to various devices in around robin fashion. If the allotted device does not use the time slice then the time slice is wasted. This is called as fixed time slicing or time division multiplexing

Dynamic Priority Algorithm

The priorities allocated to the devices are done dynamically and thus every device gets a chance to use the bus and does not suffer longer turn around time.
The two algorithms are

- Least Recently used (LRU)
- Rotating daisy chain (RDC)

The LRU gives the highest priority to the requesting device that has not used the bus for the longest interval.

In a RDC scheme, no central controller exists, and the bus grant line is connected from the last device back to the first in a closed loop. Whichever device is granted access to the bus serves as a bus controller for the following arbitration (an arbitrary device is selected to have initial access to the bus). Each device's priority for a given arbitration is determined by that device's distance along the bus grant line from the device currently serving as a bus controller; the latter device has the lowest priority. Hence the priorities are dynamically with each bus cycle.

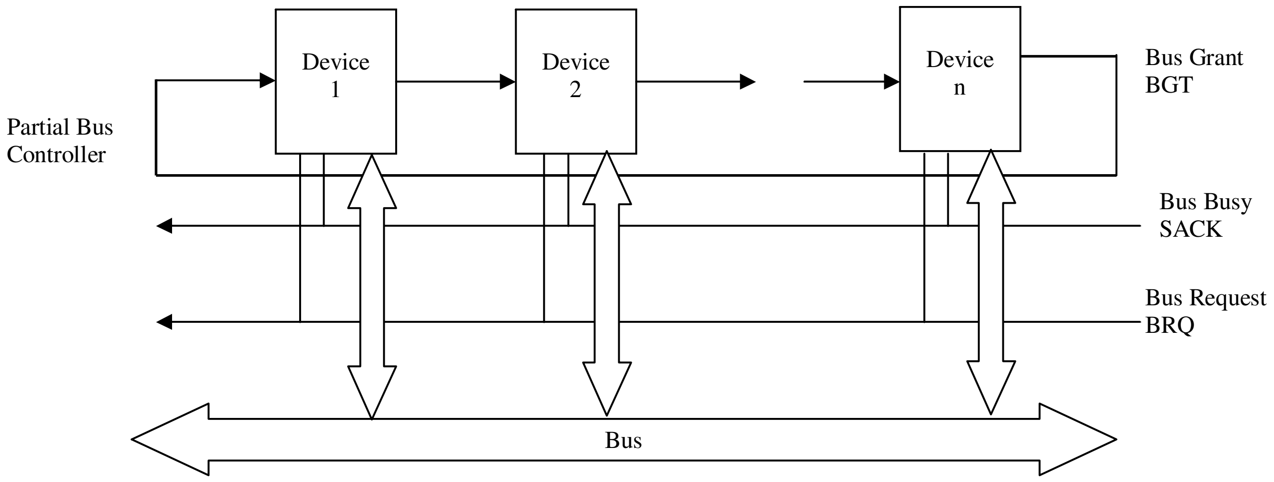


Figure 18.4 Rotating daisy chain implementation of a system bus

The first come First Served Algorithm

In the FCFS, requests are simply honored in the order received. It favors no particular processor or device on the bus. The performance measures will be degraded in case of FCFS since the device waiting for the bus for a longer period of time just because the request is made late.

18.2.2 Crossbar Switch and Mutliport Memories

If the number of buses in a time shared bus is increased, a point is reached at which there is a separate path available for each memory unit. The interconnection networking is called as a nonblocking crossbar.

Multiport Memory Module
- Each port serves a CPU

Memory Module Control Logic
- Each memory module has control logic
- Resolve memory module conflicts fixed priority among CPUs

Advantages
- Multiple paths -> high transfer rate

Disadvantages

- Memory control logic
- Large number of cables and connections

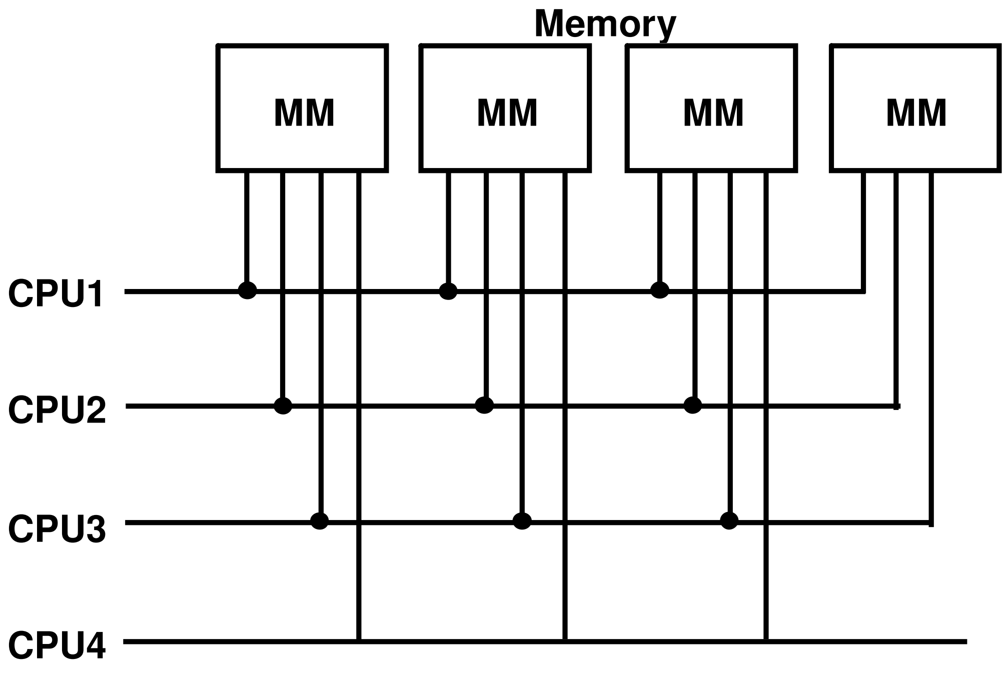


Figure 18.5 Cross Bar switch system organization for Multiprocessors

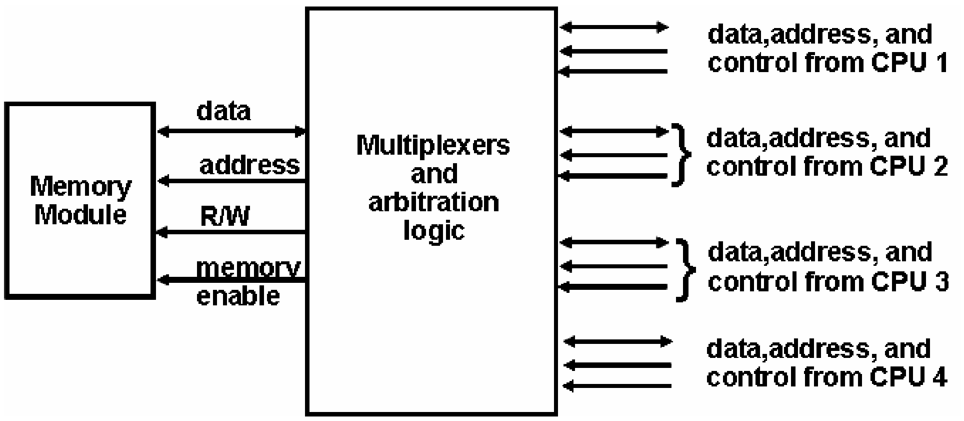

Memory Module
data
address
R/W
memory enable
Multiplexers and arbitration logic
data,address, and control from CPU 1
data,address, and control from CPU 2
data,address, and control from CPU 3
data,address, and control from CPU 4

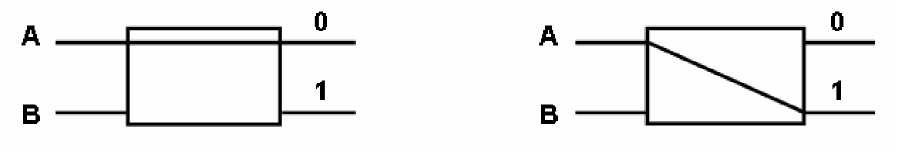
Figure 18.6 Functional structure of a crosspoint in a crossbar network

18.2.3 Multistage Networks for Multiprocessors

In order to design multistage networks, the basic principles involved in the construction and control of simple crossbar has to be understood.

This 2 x 2 switch has the capability of connecting the input A to either the output labelled 0 or to the output labelled 1, depending on the value of some control bit C_A of the input A. If

C_A = 0 the input is connected to the upper output, and if C_A = 1, the connection is made to the lower output. Terminal B of the switches behaves similarly with a control bit C_B. The 2 x 2 module also has the capability to arbitrate between conflicting requests. If both inputs A and B require the same output terminal, then only one of them will be connected and the other will be blocked or rejected.



Control Bit of A $C_A = 0$

Control Bit of A $C_A = 1$

Figure 18.6 A 2 x2 Crossbar Switch

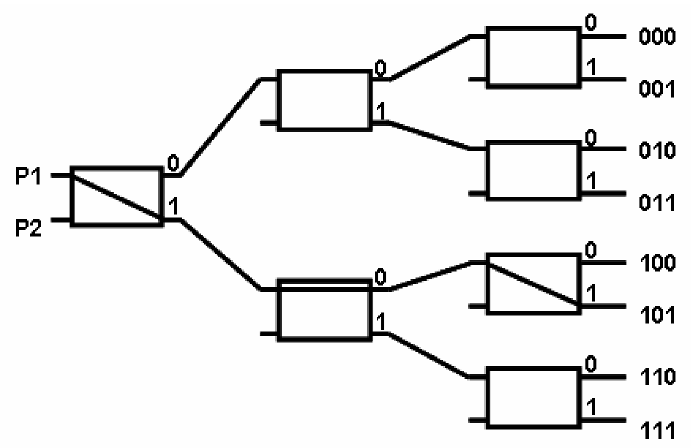


Figure 18.7 1-by-8 demultiplexer implemented with 2 x 2 switch boxes

It is straight forward to construct a 1 x 2^n demultiplexer using the above 2 x 2 module. This is accomplished by constructing a binary tree of these modules. The destinations are marked in binary, If the source A is required to connect to destination $(d_2d_1d_0)_2$ then the root is controlled by bit d_2, the modules in the second stage are controlled by bit d_1, and the modules in the third stage are controlled by bit d_0. It is clear that A can be connected o any one of the eight output terminals. It is also obvious that B can be switched to any one of the eight outputs. The method of constructing the 1 x 2^n demultiplexer tree can be extended to build a multistage network called a banyan tree.

Table 18.1 Comparison of three multiprocessor hardware organizations.

Multiprocessor with Time Shared Bus

1 Lowest overall system cost for hardware and least complex
2 Very easy to physically modify the hardware system configuration by adding or removing functional units
3 Overall system capacity limited by bus transfer rate. Failure of the bus is a catastrophic failure.
4 Expanding the system by addition of functional units may degrade overall system performance
5 The system efficiency attainable is the lowest of all three basic interconnection systems.
6 This organization is usually appropriate for smaller systems only.

Multiprocessor with Crossbar Switch

1 This is the most complex interconnection system. There is a potential for the highest total transfer rate.
2 The functional units are the simplest and cheapest since the control and switching logic is in the switch
3 Because a basic switching matrix is required to assemble any functional units into a working configuration, this organization is usually cost effective for multiprocessors only.
4 System expansion usually improves overall performance.
5 Theoretically, expansion of the system is limited only by the size of the switch matrix, which can often be modularly expanded within initial design or other engineering limitations.
6 The reliability of the switch, and therefore the system can be improved by segmentation and / or redundancy within the switch.

Multiprocessors with Multiport Memory

1 Requires the most expensive memory units since most of the control and switching circuitry is included in the memory unit
2 The characteristics of the functional units permit a relatively low cost uniprocessor to be assembled from them.
3 There is potential for a very high total transfer rate in the overall system.
4 The size and configuration options possible are determined by the number and type of memory ports available; this design decision is made quite early in the overall design and is difficult to modify.
5 A large number of cables and connectors are required.

18.3 Let us Sum Up

The interconnection networks such as time shared, Crossbar Switch and multiport memory has been discussed. In time shared bus various bus arbitration algorithms such as daisy chain, round robin, LRU, FCFS has been explained. Crossbar Switch is the most complex interconnection system. There is a potential for the highest total transfer rate and the multiport memory is also an expensive memory units since the switching circuitry us included in the memory unit and the final table which has clearly differentiated among all three interconnection networks.

18.4 Lesson-end Activities

1. Discuss the various multiprocessor interconnection networks.
2. Compare the various multiprocessor hardware organizations.

18.5 Points for Discussions

Interconnection Networks: Interconnection networks helps in sharing resources, one is between the processor and the memory modules and the other between the processor and the I/O Systems
Time shared Bus: All processors (and memory) are connected to a common bus or busses

- Memory access is fairly uniform, but not very scalable

Crossbar Switch: Uses O(mn) switches to connect m processors and n memories with distinct paths between each processor/memory pair
Multistage Networks: Multistage networks provide more scalable performance than bus but at less cost than crossbar
Bus Arbitration Algorithms: Daisy Chaining, FCFS, LRU, Round Robin Scheduling, Time slice

18.6 References

- System Interconnects, Tarek El-Ghazawi, The George Washington University
- http://www.gup.uni-linz.ac.at/thesis/diploma/christian_schaubschlaeger/html/biblio.html#774418
- Larry Carter, carter@cs.ucsd.edu, www.cs.ucsd.edu/classes/fa01/cs260

UNIT – V

Lesson 19 : Parallel Algorithms, Models of Computation

Contents:

19.0 Aims and Objectives

The main aim is to learn about parallel algorithms and how a problem can be solved in a parallel computer. The abstract machine models are discussed and these models are useful in the design of analysis of parallel algorithms.

19.1 Introduction

In order to simplify the design and analysis of parallel algorithms, parallel computers are represented by various abstract models. These models capture the important features of parallel computer. The Random Access Machine is the preliminary model and the PRAM is one of the popular models for designing parallel algorithms. Various division of PRAM models are EREW, CREW,ERCW, CRCW. The concept of interconnection networks has been discussed since the exchanges of data among processors takes place through the shared memory. The Combinational circuit can be viewed as a device with a set of input lines at one end and a set of output lines at the other end is also a model of parallel computers.

19.2 Parallel Algorithms

The Algorithms designed for sequential computers are known as Sequential Algorithms. The algorithm worked out for a parallel computer is termed as Parallel algorithms.

- Traditionally, software has been written for ***serial*** computation:
    - To be run on a single computer having a single Central Processing Unit (CPU);
    - A problem is broken into a discrete series of instructions.
    - Instructions are executed one after another.
    - Only one instruction may execute at any moment in time.

- In the simplest sense, ***parallel computing*** is the simultaneous use of multiple compute resources to solve a computational problem.
    - To be run using multiple CPUs
    - A problem is broken into discrete parts that can be solved concurrently
    - Each part is further broken down to a series of instructions
    - Instructions from each part execute simultaneously on different CPUs

Parallelism refers to the simultaneous occurrence of events on a computer.
An event typically means one of the following:

- An arithmetical operation
- A logical operation
- Accessing memory
- Performing input or output (I/O)

19.2.1 Models of Computation

Various abstract machine models are discussed, and it helps in designing parallel algorithms.

19.2.2 Random Access machine (RAM)

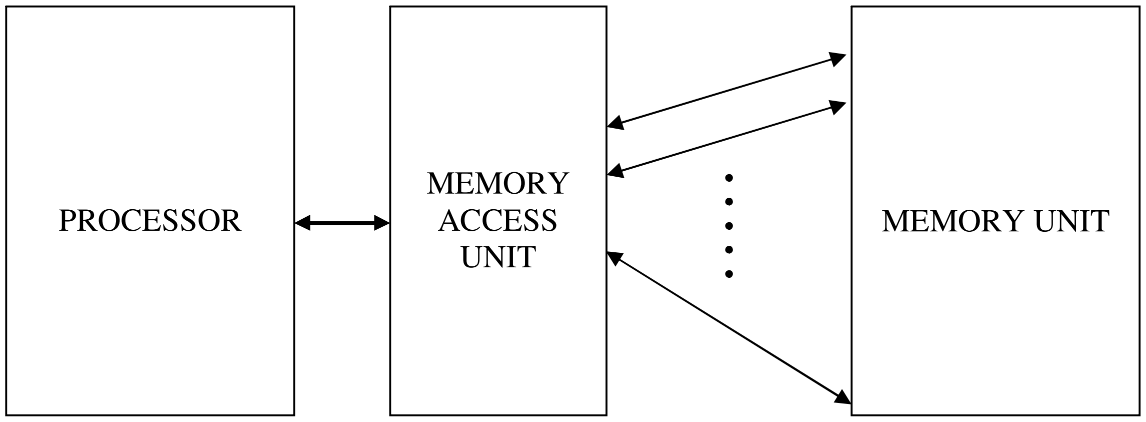


Figure 19.1 RAM Model

The basic functional units if the RAM are :

1. A memory unit with M locations. M can be unbounded.
2. A processor that operates under the control of sequential algorithm. The processor can read data from a memory location, write to a memory location, and can perform basic Arithmetic and logical operations.
3. A memory Access unit (MAU) which creates path from the processor to an arbitrary location in the memory.

RAM model consists of three phases namely

- Read : The processor reads a datum from the memory. This datum is usually stored in one of its local registers.
- Write : The processor writes the content of one register into an arbitrary memory location.

- Execute : The processor performs a basic arithmetic and logic operations on the contents of one or two of its registers.

19.2.3 Parallel Random Access machine (PRAM)

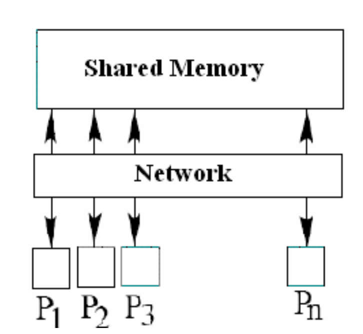


Figure 19.2 Pram Model

- Let $P_1, P_2, \ldots, P_n$ be identical processors
- Each processor is a RAM processor with a private local memory.
- The processors communicate using m shared (or global) memory locations. Each Pi can read or write to each of the m shared memory locations.
- All processors operate synchronously (i.e. using same clock), but can execute a different sequence of instructions.

Most commonly used model for expressing parallel algorithms

- Shared Memory

Each processor may have local memory to store local results

- MIMD

Model is MIMD, but algorithms tend to be SIMD.

Each PRAM step consists of three phases, executed in the following order:

- A **read phase** in which each processor may read a value from shared memory
- A **compute phase** in which each processor may perform basic arithmetic/logical operations on their local data.
- A **write phase** where each processor may write a value to shared memory.
- Note that this prevents reads and writes from being simultaneous.
- Above requires a PRAM step to be sufficiently long to allow processors to do different arithmetic/logic operations simultaneously.

Strengths of PRAM Model

- PRAM model removes algorithmic details concerning synchronization and communication, allowing designers to focus on problem features
- A PRAM algorithm includes an explicit understanding of the operations to be performed at each time unit and an explicit allocation of processors to jobs at each time unit.
- PRAM design paradigms have turned out to be robust and have been mapped efficiently onto many other parallel models and even network models.

Each active processor executes same instruction

- Synchronous
- Memory Access

Shared memory computers can be classified as follows depending on whether two or more processors can gain access to the same memory simultaneously.

- **Exclusive Read, Exclusive Write (EREW)**
    - Access to memory locations is exclusive, i.e., no 2 processors are allowed to simultaneously read from or write into the same location.
- **Concurrent Read, Exclusive Write (CREW)**
    - Multiple processors are allowed to read from the same location, but write is still exclusive, i.e., no 2 processors are allowed to write into the same location simultaneously.
- **Exclusive Read, Concurrent Write (ERCW)**
    - Multiple processors are allowed to write into the same location, but read access remains exclusive.
- **Concurrent Read, Concurrent Write (CRCW)**
    - Both multiple read and write privileges are allowed.
    - Handling concurrent writes
        - Common CRCW PRAM allows concurrent writes only when all processors are writing the same value
        - Arbitrary CRCW PRAM allows an arbitrary processor to succeed at writing to the memory location
        - Priority CRCW PRAM allows the processor with minimum index to succeed

19.2.4 Interconnection Networks

- Parallel computers with many processors do not use shared memory hardware.
- Instead each processor has its own local memory and data communication takes place via message passing over an interconnection network.
- The characteristics of the interconnection network are important in determining the performance of a multicomputer.
- If network is too slow for an application, processors may have to wait for data to arrive.

19.2.5 Combinatorial Circuits

It is another model of parallel computers. Combinational circuit refers to a family of models of computation. A combinational circuit can be viewed as a device that has a set of input lines at one end and set of output lines at the other. And each component, having received its inputs, does a simple arithmetic and logical operations in one time and produces the results as output.

A combinational circuit consists of a number of interconnected components arranged in columns called stages.

Each component is a simple processor with a constant fan-in and fan-out

- Fan-in: Number of input lines carrying data from outside world or from a previous stage.

- Fan-out: Number of output lines carrying data to the outside world or to the next stage.

Component Characteristics

- Only active after input arrives
- Computes a value to be output in O(1) time, usually using only simple arithmetic or logic operations.
- Component is hardwired to execute its computation.

Component Circuit Characteristics

- Has no program
- Has no feedback
- Depth: The number of stages in a circuit . Gives worst case running time for problem
- Width: Maximal number of components per stage.
- Size: The total number of components

Two Way Combinational Circuits

- copying data to the outside world or to the next stage.
- Sometimes used as a two-way devices
- Input and output switch roles
  - data travels from left-to-right at one time and from right-to-left at a later time.
- Useful particularly for communications devices.
- Subsequently, the circuits are assumed to be two-way devices.
  - Needed to support MAU (memory access unit) for RAM and PRAM

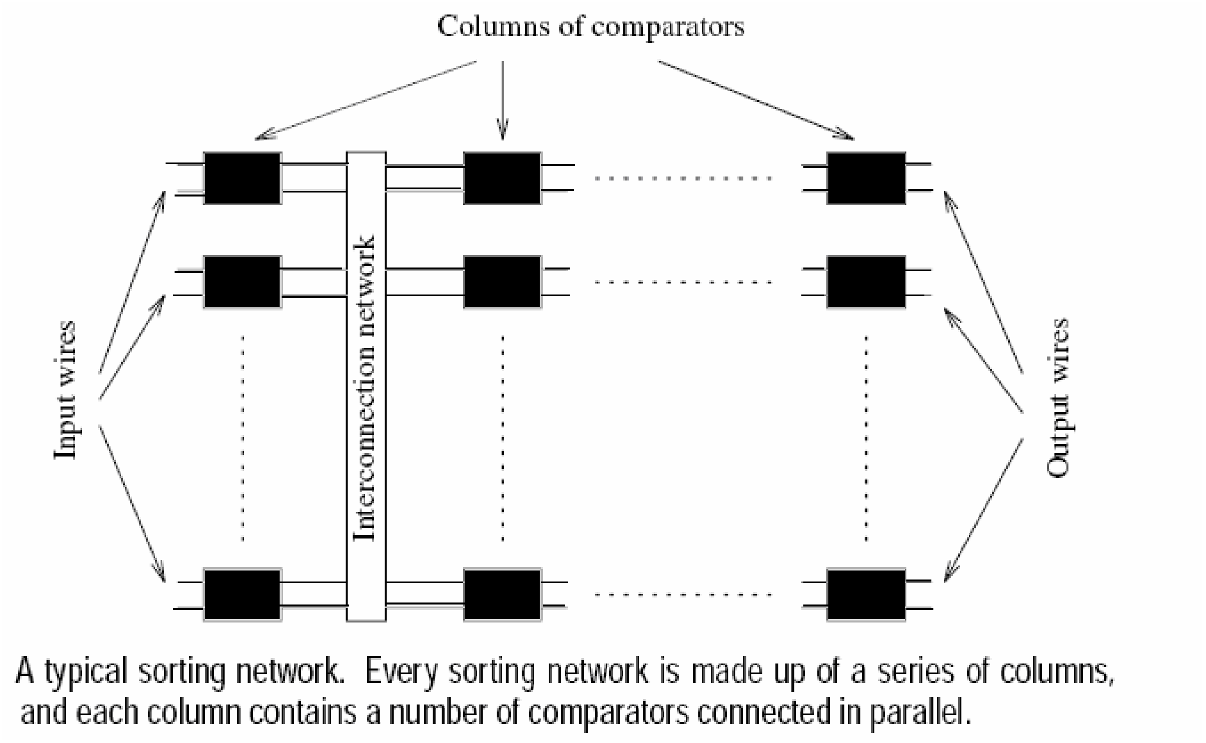

Columns of comparators
Input wires
Interconnection network
Output wires
A typical sorting network. Every sorting network is made up of a series of columns, and each column contains a number of comparators connected in parallel.

Figure 19.3 Combinatorial Circuit

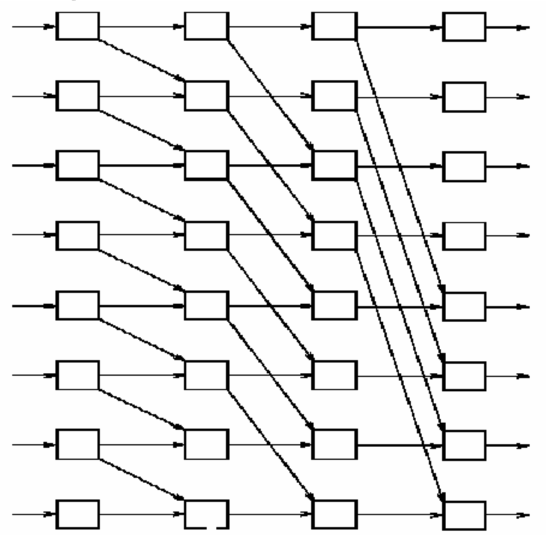


Figure 19.4 Butterfly Circuit

The butterfly circuit has n inputs and n outputs. The depth of the circuit is (1 + log n) and the width of the circuit is n. The size in this case is (n + n log n) where n = 8

19.3 Let us Sum Up

Various models of computations have been discussed with its schematic diagram. The PRAM is one of the popular models for designing parallel algorithms, its phases of algorithms and it's Model of subdivision based on memory access called as EREW, CREW, ERCW, and CRCW has been discussed. The concept of combinatorial circuit its important parameters such as width, size and depth has been given as an example with respect to butterfly circuit.

19.4 Lesson-end Activities

1. Write short notes on parallel algorithms.
2. Explain the abstract machine models of computation.

19.5 Points for discussions

- Random Access Machine
- Parallel Random Access Machine
- Definitions of EREW, CREW, ERCW, CRCW
- Combinatorial Circuits

19.6 References

- PRAM Models, Advanced Algorithms & Data Structures, Lecture Theme 13, Prof. Dr. Th. Ottmann
- PRAM and Basic Algorithms by Shietung Peng
- Introduction to Parallel Processing: Algorithms and Architectures by Behrooz Parhami

Lesson 20 : Analysis of Parallel Algorithms, Prefix Computation

Contents:

20.0 Aims and Objectives

The main aim of this lesson is to analyse the parallel algorithms based on the principle criteria Running time, Number of Processors and Cost.

20.1 Introduction

Any sequential algorithm is evaluated based on 2 parameters called as the running time complexity and space complexity. The parallel algorithms are evaluated based on running time, number of processors and cost factors. The concept of prefix computation has been discussed which is a very useful sub operation in many parallel algorithms.

20.2 Analysis of Parallel Algorithms

Any sequential algorithm is evaluated in terms of 2 parameters

- Running time complexity
- Space complexity

Parallel algorithms are evaluated based on the following

- Running Time
- Number of Processors
- Cost

20.2.1 Running Time

The running time of an algorithm is a function of the input given to the algorithm. The algorithm may perform well for certain inputs and relatively bad for certain others. The worst case running time is defined as the maximum running time of the algorithm taken over all the

inputs. The average case running time is the average running time of the algorithm over all the inputs.
The running time depends not only on the algorithm but also on the machine it is being executed.

Notion of Order

Consider 2 functions f(n) and g(n) defined from the set of positive integer to the set of positive reals.

- The function g(n) is said to be order of at least f(n) denoted by $\Omega(f(n))$if there exist positive constantsc,n_0 such that g(n) >= cf(n) for all n >= n_0
- The function g(n) is said to be order at most f(n) denoted by O (f(n))if there exist positive constantsc,n_0 such that g(n) <= cf(n) for all n >= n_0
- The function g(n) is said to be of the same order as f(n) if $g(n) \in O(f(n))$ and $g(n) \in \Omega(f(n))$.

Lower and Upper Bound

The lower bound on a problem is indicative of the minimum number of steps required to solve the problem in the worst case.. Every problem is associated with a lower bound and upper bound.

Speedup

Speed-up is the ratio of the time taken to run the best sequential algorithm on one processor of the parallel machine divided by the time to run on N processors of the parallel machine.
S(N) = Tseq/Tpar(N)
Efficiency is the speed-up per processor.
$\in (N) = S(N)/N=(1/N)(T_{seq}/T_{par}(N))$
Overhead is defined as
$f(N) = 1/ \in (N) -1$

20.2.2 Number of Processors

The next important criteria for evaluating parallel algorithm are the number of processors required to solve the problem. Given a problem of input size n, the number of processors required by an algorithm, is a function of n denoted by p(n).

20.2.3 Cost

The cost of a parallel algorithm is defined as the product of the running time of the parallel algorithm and the number of processors used.

Cost =running time x number of processors

The **efficiency** of a parallel algorithm is defined as the ratio of worst case running time of the fastest sequential algorithm and the cost of the parallel algorithm.

20.2.4 Prefix Computation

Prefix computation is a very useful sub operation in many parallel algorithms. The prefix computation problem is defined as follows.

A set λ is given, with an operation o on the set such that

1. o is a binary operation
2. λ is a closed under o
3. o is associative

Let X = { $x_0, x_1, \ldots, x_{n-1}$ } where xi $\in\lambda$ (0<=i<=n-1). Define

$$S_0 = x_0$$
$$S_1 = .x_0 \text{ o } x_1$$
$$.$$
$$.$$
$$.$$
$$S_{n-1} = x_0 \text{ o } x_1 \text{ o } \ldots\ldots \text{ o } x_{n-1}$$

Obtaining the set S= {$s_0, s_1, \ldots, s_{n-1}$} given the set X is known as the *prefix computation.* The indices of the element used to compute is form a string 012…I, which is a prefix of the string 012…n-1, and hence the name prefix computation for this problem.

20.2.5 Prefix Computation on the PRAM

Assume that the set $\lambda$is the set of natural numbers and the operation o is the + operation. Let X = {$x_0, x_1, \ldots, x_{n-1}$}be the input. The partial sums s_i(0<=i<=n-1) where

si = $x_0 + x_1 + \ldots\ldots + x_i$ can be found easily on a RAM in o(n) time

The PRAM algorithm given here requires n processors $P_0, P_1, \ldots, P_{n-1,}$ where n is a power of 2. There are n memory locations m0, m1, … m n-1. Initially the memory location m_i(0<=i<=n-1) contains the input x_i. When the algorithm terminates the location m_i (0<=i<=n-1) has the partial sum output s_i. The algorithm is given below.

It contains of (log n) iterations; during each step, the binary operation + is performed by pairs of processors whose indices are separated by a distance twice that in the previous iteration.

Procedure Prefix Computation

```
    for j : = 0 to (log n) – 1 do
        for i : = $2^j$ to n-1 do in parallel
            $s_i := s_{(i-2^j)} + s_i$
        end for
end for
```

Analysis

There are log n iterations in this algorithm. In each iteration one addition is done in parallel and hence takes O(1) time. The time complexity is therefore O(log n) Since the algorithm uses n processors, the cost of the algorithm is O (n log n), which is clearly not cost optimal. This algorithm has the optimal time complexity among the non-CRCW machines. This is because prefix computation of s_{n-1} requires O (n) additions which has a lower bound of O (log n) on any non-CRCW parallel machine.

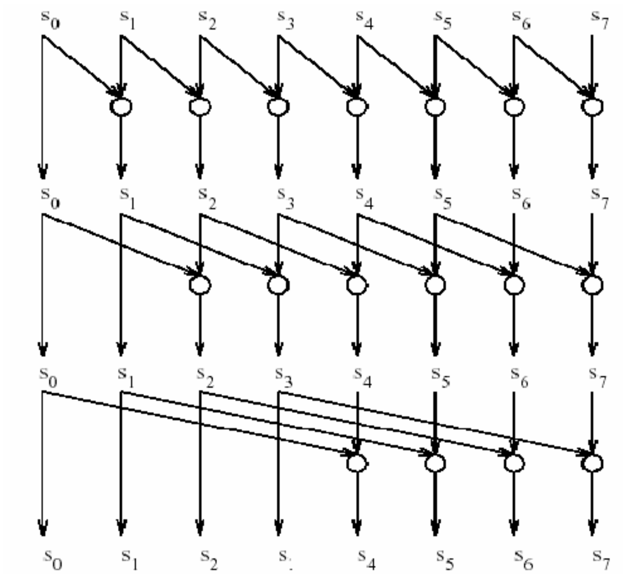

s_0 s_1 s_2 s_3 s_4 s_5 s_6 s_7
s_0 s_1 s_2 s_3 s_4 s_5 s_6 s_7
s_0 s_1 s_2 s_3 s_4 s_5 s_6 s_7
s_0 s_1 s_2 s_3 s_4 s_5 s_6 s_7

Figure 19. 5 Prefix Computation on the PRAM

A Cost Optimal Algorithm for Prefix Computation

This algorithm also runs in o(log n) time but makes use of fewer processors to achieve this. Let $X = \{ x_0, x_1, \ldots, x_{n-1} \}$be the input to the prefix computation. Let k= log n and m=n/k. The algorithm uses m processors P_0, P_1, …. , P_{m-1}. The input sequence X is split into m subsequences, each of size k, namely

$$Y_0 = x_0, x_1, \ldots.., x_{k-1}$$
$$Y_1 = x_k, x_{k+1}, \ldots.., x_{2k-1}$$

.

.

.

$$Y_{m-1} = x_{n-k}, x_{n-k+1}, \ldots.., x_{n-1}$$

The algorithm given below proceeds in 3 steps.

Step1 : The processor P_i (0<=i<=m-1) works on the sequence Yi, and computes the prefix sums of the sequence Yi, using a sequential algorithm. Thus the processor Pi computes $S_{ik}, S_{ik+1}, \ldots.,$ $S_{(i+1)k-1}$, where

$$s_{ik+j} = x_{ik}+x_{ik+1}+ \ldots.., x_{ik+j}$$

for j = 0, 1, ….., k-1.

Step2 : The processors P_0, P_1, ..., P_{m-1} perform a parallel prefix computation on the sequence {s_{k-1}, s_{2k-1}, …, s_{n-1}}. This parallel prefix computation is done using the algorithm described earlier. When this step is completed, sik-1 will be replaced by $s_{k-1} + s_{2k-1},\ldots,s_{ik-1}$.

Step3 : Each processor P_i (0<=i<=m-1)computes $s_{ik+j} = s_{ik-1}+s_{ik+j}$, for j=0,1,…,k-2.

Example for Cost Prefix

Sequence – 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
Use n / [lg n] PEs with lg(n) items each
0,1,2,3 4,5,6,7 8,9,10,11 12,13,14,15
STEP 1: Each PE performs sequential prefix sum
0,1,3,6 4,9,15,22 8,17,27,38 12,19,39,54
STEP 2: Perform parallel prefix sum on last nr. in PEs
0,1,3,6 4,9,15,28 8,17,27,66 12,19,39,120
Now prefix value is correct for last number in each PE
STEP 3: Add last number of each sequence to incorrect sums in next sequence (in parallel)
0,1,3,6 10,15,21,28 36,45,55,66 78,91,105,120

AlgorithmAnalysis

Analysis:
Step 1 takes O(k) = O(lg n) time.
Step 2 takes
- O(lg m) = O(lg n/k)
- O(lg n- lg k) = O(lg n - lg lg n)
- O(lg n)

Step 3 takes O(k) = O(lg n) time
- The overall time for this algorithm is O(n).
- The overall cost is O((lg n) O n/(lg n)) = O(n)

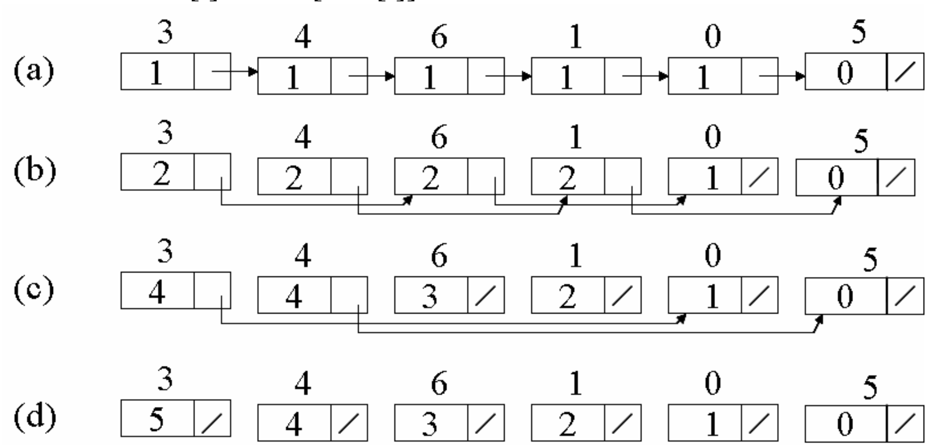
20.2.6 Prefix Computation on the Linked List

The algorithm for prefix computation on a linked list technique is called as pointer jumping. The data are stored in the pointer based structures called as linked lists, trees etc.

- Given a single linked list *L* with *n* objects, compute, for each object in *L*, its distance from the end of the list.
- Formally: suppose *next* is the pointer field
  - $d[i]=$ 0 if $next[i]=nil$
  - d[$next[i]$]+1 if $next[i]\neq nil$
- Serial algorithm: $\Theta(n)$.

```
LIST-RANK(L)   (in O(lg n) time)
        for each processor i, in parallel
        do if next[i]=nil
```

```
            then $d[i] \leftarrow 0$
            else $d[i] \leftarrow 1$
    while there exists an object $i$ such that $next[i] \neq nil$
        do for each processor $i$, in parallel
            do if $next[i] \neq nil$
                then $d[i] \leftarrow d[i] + d[next[i]]$
                     $next[i] \leftarrow next[next[i]]$
```



- Loop invariant: for each *i*, the sum of *d* values in the sublist headed by *i* is the correct distance from *i* to the end of the original list L.
- Parallel memory must be synchronized: the reads on the right must occur before the writes on the left. Moreover, read *d*[*i*] and then read *d*[*next*[*i*]].
- An EREW algorithm: every read and write is exclusive. For an object *i*, its processor reads *d*[*i*], and then its precedent processor reads its *d*[*i*]. Writes are all in distinct locations.

- Loop invariant: for each i, the sum of d values in the sublist headed by i is the correct distance from i to the end of the original list L.
- Parallel memory must be synchronized: the reads on the right must occur before the writes on the left. Moreover, read d[i] and then read d[next[i]].
- An EREW algorithm: every read and write is exclusive. For an object i, its processor reads d[i], and then its precedent processor reads its d[i]. Writes are all in distinct locations.

20.3 Let us Sum Up

The three principle criteria that were used in evaluation parallel algorithms are running time, number of processors and cost has been discussed. Prefix computation a very useful sub operation in many parallel algorithms has been defined and procedure for prefix computation has been analysed and its complexity calculation has been done. The algorithm for prefix computation called pointer jumping helped in solving problems whose data are stored as pointer based.

20.4 Lesson-end Activities

1. Analyse Parallel Algorithms in terms of Running Time, Number of Processors and Cost.
2. Explain prefix computation in detail.

20.5 Points for discussions

Pointer Jumping

The algorithm for prefix computation on a linked list technique is called as pointer jumping. The data are stored in the pointer based structures called as linked lists, trees etc.

20.6 References

- Parallel Computer architectures and Programming, V. Rajaraman & C. Siva Ram Murthy

Lesson 21 : Sorting

Contents:

21.0 Aims and Objectives

The main of this lesson is to learn the concept of sorting and how it is implemented in parallel processor and the design of various algorithms

21.1 Introduction

An given unsorted sequence has to be processed to an sorted sequence. 2 different combinational circuits have been used for performing sorting. They are Bitonic sorting network in which there exists an index i, where 0<=i<=n-1, such that a_0 through a_i is monotonically increasing and a_i through a_{n-1} is monotonically decreasing. The next algorithm discussed was merging in which the unsorted sequences are divided in to two halves and perform sorting on individual half recursively and then merging together to obtain the sorted sequence.

Various sorting algorithms on PRAM models has been discussed. In sorting on linear array, the Odd-even transposition method has been adopted and sorting on hypercube based on bitonic method has been discussed.

21.2 Sorting

The problem of sorting is important as sorting data is at the heart of many computations. Various sorting algorithms has been described.

21.2.1 Combinational Circuits for Sorting

The input to the combinational circuits is n unsorted numbers and the output is the sorted permutation of the n numbers.

The basic processing unit in the combinational circuit is the comparator. A comparator has 2 inputs x and y, and has two output x' and y'.

There are 2 kinds of comparators.
In an increasing comparator, x' = min(x,y) and y' = max(x,y).
In an decreasing comparator, x' = max(x,y) and y' = min(x,y).

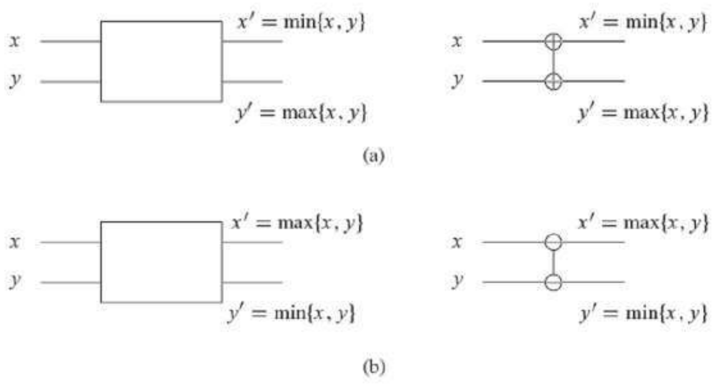


Figure : 21.2 (a) Increasing and (b) Decreasing Comparator

Bitonic Sorting Network

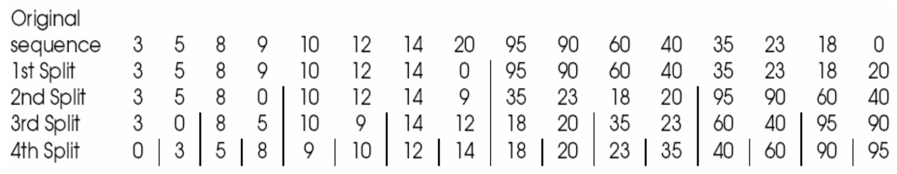
Definition 1 (Bitonic Sequence) A bitonic sequence is a sequence of values, $<a_0,a_1,\ldots a_{n-1}>$with the property that (1) there exists an index i, where 0<=i<=n-1, such that a_0 through a_i is monotonically increasing and a_i through a_{n-1} is monotonically decreasing, or (2) there exists a cyclic shift of indices so that the first condition is satisfied.

Let s = $<a_0,a_1,\ldots,a_{n-1}>$ be a bitonic sequence such that $a_0 \leq a_1 \leq \cdot \cdot \cdot \leq a_{n/2-1}$ and $a_{n/2} \geq$ an/2+1 $\geq \cdot \cdot \cdot \geq$ an-1.
Consider the following subsequences of s:

$$s1 = <\min\{a_0,a_{n/2}\},\min\{a_1,a_{n/2+1}\},\ldots,\min\{a_{n/2-1},a_{n-1}\}>$$
$$s2 = <\max\{a_0,a_{n/2}\},\max\{a_1,a_{n/2+1}\},\ldots,\max\{a_{n/2-1},a_{n-1}\}>$$

Note that s1 and s2 are both bitonic and each element of s1 is less than every element in s2.

The procedure can be applied recursively on s1 and s2 to get the sorted sequence.



Original sequence	3	5	8	9	10	12	14	20	95	90	60	40	35	23	18	0
1st Split	3	5	8	9	10	12	14	0	95	90	60	40	35	23	18	20
2nd Split	3	5	8	0	10	12	14	9	35	23	18	20	95	90	60	40
3rd Split	3	0	8	5	10	9	14	12	18	20	35	23	60	40	95	90
4th Split	0	3	5	8	9	10	12	14	18	20	23	35	40	60	90	95

Figure 21.2 Merging a 16-element bitonic sequence through a series of log 16 bitonic splits.

We can easily build a sorting network to implement this bitonic merge algorithm.
Such a network is called a bitonic merging network.

The network contains log n columns. Each column contains n/2 comparators and performs one step of the bitonic merge.
We denote a bitonic merging network with n inputs by ⊕BM[n].
Replacing the ⊕ comparators by Θ comparators results in a decreasing output sequence; such a network is denoted by ΘBM[n].

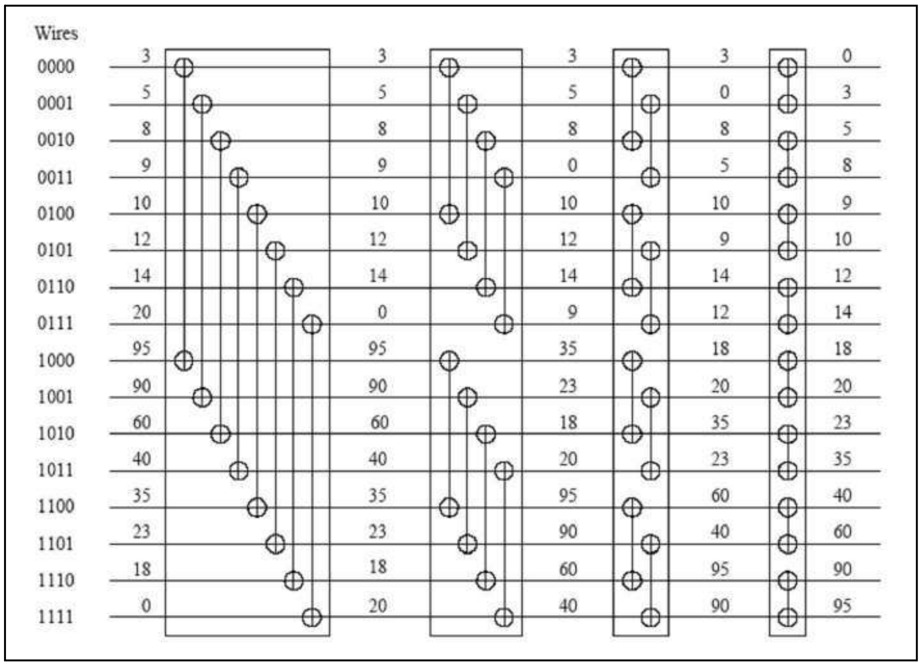
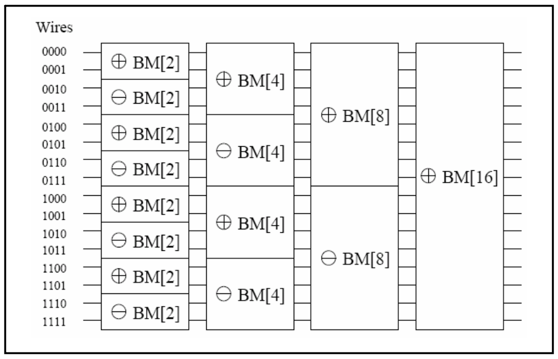


Figure 21.3 Bitonic Merge Network

A bitonic merging network for *n* = 16. The input wires are numbered 0,1,… , *n* - 1, and the binary representation of these numbers is shown. Each column of comparators is drawn separately; the entire figure represents a ⊕BM[16] bitonic merging network. The network takes a bitonic sequence and outputs it in sorted order.

We must first build a single bitonic sequence from the given sequence.
A sequence of length 2 is a bitonic sequence.
A bitonic sequence of length 4 can be built by sorting the first two elements using ⊕BM[2] and next two, using ΘBM[2].
This process can be repeated to generate larger bitonic sequences.



A schematic representation of a network that converts an input sequence into a bitonic sequence. In this example, ⊕BM[k] and ϴBM[k] denote bitonic merging networks of input size k that use Å and ϴ comparators, respectively. The last merging network (⊕BM[16]) sorts the input. In this example, n = 16.

Analysis

The bitonic sorting network for sorting a sequence of n numbers consists of log n stages. The last stage of the network uses a (+)BM(n) which has a depth of log n. The remaining stages perform a complete sort of n/2 elements. The depth of the sorting network is given by recirrence relation D(n) = d(n/2) + log n solving this we get d(n) = ((log ^{2}) + log n/2 = O(log^2 n)

Combinational Circuit for sorting by Merging

The second combinational circuit for sorting is based on the merge sort algorithm for a sequential computer.

Odd-even merge sort

- First develop a merging circuit. The idea is to split the two sequences to be merged into two subsequence each. merge them separately, then merge two merged sequence into one.
- First, the even and odd indices elements of a sorted array elements are merged by the same odd-even merge algorithm.
- Then the two output sequences are merging into the final sorted sequence. The element only needs to be compared with neighbouring elements from the other sequence, hence can be done in one additional time step.

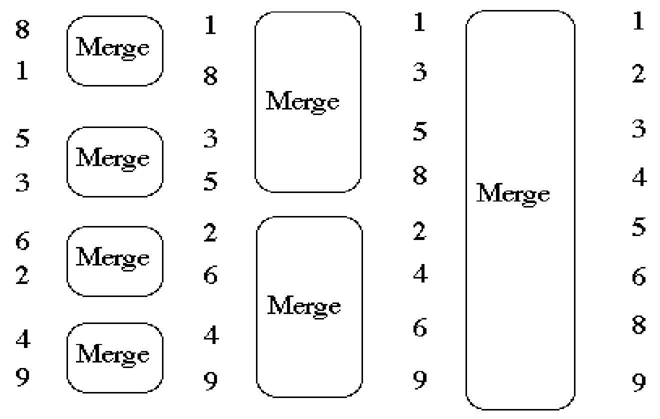


Figure 21.5 Merging Circuit

In general a cuircuit for merging two sequences is obtained as follows.
Let S_1 = <X_1,X_2….X_m> and S_2 = <Y_1,Y_2…Y_m> be the input sequences that need to be merged.

- Using an (m/2,m/2) merging circuit, merge the odd even indexed columns of the two sequences :<X_1,X_3….X_{m-1}> and <Y_1,Y_3….Y_{m-1}> to produce a sorted sequence <U_1,U_2….U_m>.
- Using an (m/2,m/2) merging circuit, merge the even indexed columns of the 2 sequences <X_2,X_2…..X_m> and <Y_2,Y_4…Y_m> to produce a sorted sequence <v1,v2….vm>.
- The output sequence <Z_1, Z_2……Z_{2m}> is obtained as Z_1 = U_1,Z_{2m} = V_m, Z_{2i} = min(U_{i+1},V_i) and Z_{2i+1} = max(U_{i+1},V_i) for I = 1,2,…. m-1.

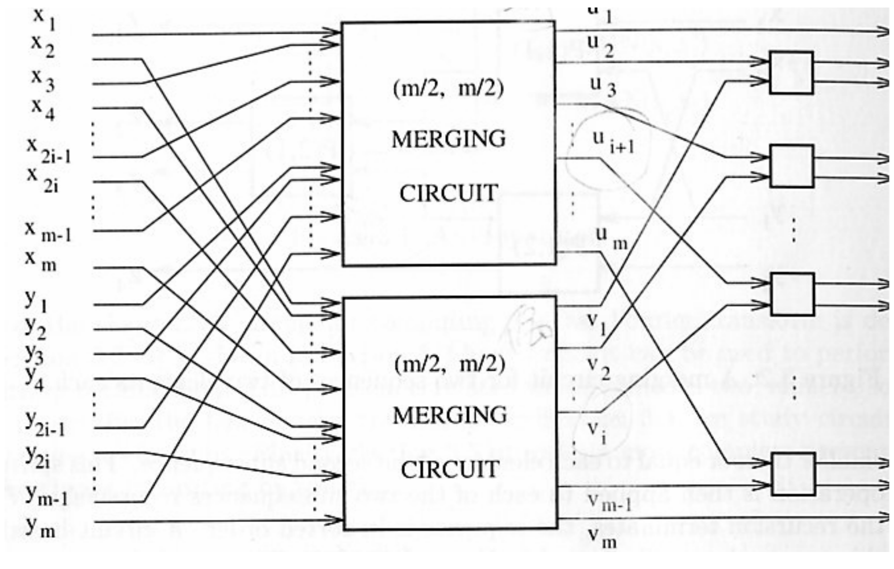


Figure 21.6 Odd – Even Merging Circuit

Analysis

Width : Each comparator has exactly 2 inputs and 2 outputs. The circuit takes 2m inputs and produces 2m outputs. Thus it has width of m.

Depth: The merging circuit consists of 2 smaller circuits for merging sequences of length m/2, followed by one stage of comparators. Thus the depth d(m) of the circuit is

$D(m) = 1 \quad (m = 1)$

$D(m) = d(m/2) + 1 \quad (m > 1)$

Size : The size p(m) of the circuit can be obtained by solving the following recurrence relations.

$P(1) = 1 \quad (m=1)$

$P(m)= 2p(m/2) + m - 1 \quad (m>1)$

Solving we get p(m)0 = 1 + mlog m.

21.2.2 Sorting on PRAM Models

Various sorting algorithms for the PRAM models of a parallel computer. All these algorithms are based on the idea of sorting by enumeration. The sorting is performed in such a way that every element is compared with all other elements and placed in its sorted sequence.

Let S = $<s_1,s_2 \ldots s_n>$ denoted the unsorted input sequence. The position of each elemnt s_i of S in the sorted sequence, known as rank of S_i is determined by computing r_i, the number of elements smaller than it.

The array used is R = $<r_1,r_2 \ldots . r_n>$ to keep track of the rank of the elements in the input sequence. The array R_i is initialized to 0 in all the algorithms. The r_i will contain the number of elements in the input sequence.

CRCW Sorting

A algorithm is described for SUM CRCW model of a parallel computer which has n^2 processors. When more than one processor attempts to write to a common memory location, the sum of all the individual values is written onto the memory location.

n^2 processors are represented in matrix as n x n. $P_{i,j}$ denoted the processor in the ith row and jth column. The processor $P_{i,j}$ tries to write the value 1 in r_i if $s_i > s_j$ or $s_i =$ sj and i > j.

Procedure CRCW Sorting

```
For i := 1 to n do in parallel
   For j := 1 to n do in parallel
      If $s_i > s_j$ or ($s_i = s_j$ and i >j) then
         $P_{i,j}$ writes 1 to $r_i$
      End if
   End for
End for
For i := 1 to n do in parallel
   $P_{i,1}$ puts $s_i$ in ($r_i$ +1) position of S
End for
```

The above algorithm takes 0(1) time and it uses $O(n^2)$ processors which is very large.

CREW Sorting

The sorting algorithm for the CREW model is very similar the previous one. The algorithm takes n processors and has a time complexity of O(n). Let the processors be denoted by $<P_1,P_2....P_n>$. The processor P_i computes the value r_i after n iterations. In the jth iteration of the algorithm, all the processors read sj, the processor pi increments r_i if $s_i > s_j$ or $s_i = s_j$ and $i > j$. Therefore, after n iterations r_i contains the number of elements smaller than s_i in the input sequence.

Procedure CREW Sorting

```
For i := 1 to n do in parallel
   For j := 1 to n do
      If $s_i > s_j$ or ($s_i = s_j$ and i >j) then
         $P_{i,j}$ writes 1 to $r_i$
      End if
   End for
    $P_{i,1}$ puts $s_i$ in ($r_i$ +1) position of S
End for
```

EREW Sorting

Concurrent read conflicts cab be avoided if we ensure that each processor reads a unique element in each iteration. This can be achieved by allowing the processors to read elements in a cyclic manner. In the first iteration the processors $P_1,P_2....P_n$ read s_1,s_2,s_3....and in the second iteration the processors read s2,s3…sn and s1 respectivbelyand so on. This in the jth iteration the processors $P_1,P_2....P_n$ reads S_j , S_{j+1}..... s_{j-1} and update corresponding r values. This has a running complexity of O(n).

Procedure EREW Sorting

```
For i := 1 to n do in parallel
   For j := 0 to n-1 do
       K := (i+j) mod n
      If $s_i > s_k$ or ($s_i = s_k$ and i >k) then
         $P_{i,}$ adds 1 to $r_i$
      End if
   End for
    $P_{i,1}$ puts $s_i$ in ($r_i$ +1) position of S
End for
```

21.2.3 Sorting on Interconnection Networks

In interconnection networks the comparisons are made as, consider 2 adjacent processors Pi and P_j. The processor p_i has the element a_i and the processor p_j has the element a_j. The network processor P_i sends the element a_i to the processor P_j and the processor P_j sends the element a_j to the processor P_i. The processor P_i keeps the element $\min(a_i,a_j)$ and the processor P_j keeps the element $\max(a_i,a_j)$. This is refereed as compare exchange (P_i,P_j).

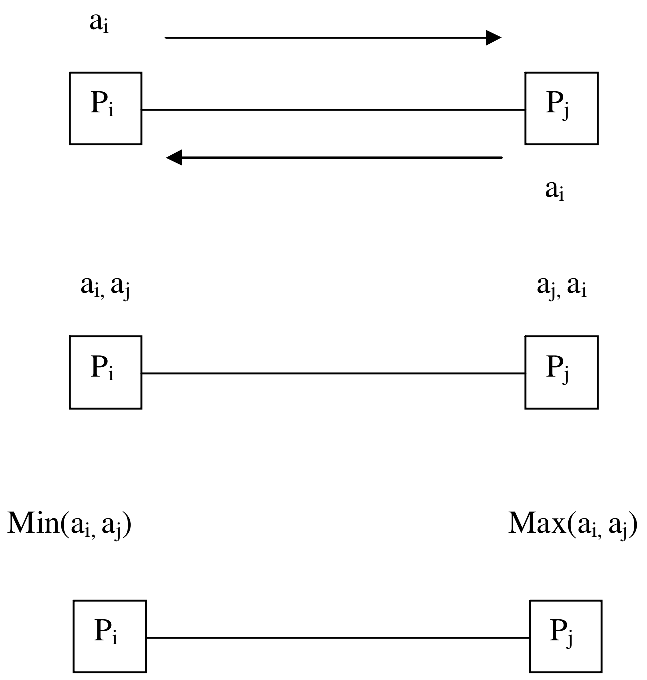


Figure 21.7 A parallel Exchange Operation

Sorting on a Linear Array
A linear array has the processors in a row: each processor can communicate with only the one before it and the one after it. We give a linear array sorting algorithm that takes precisely N steps, where N is the number of processors and the number of data values. N - 1 is a lower bound, because one might need to get a data value from the last cell to the first cell.

Odd-even transposition sort.

At odd steps, compare contents of cell 1 and 2, cells 3 and 4, etc, and swap values if necessary so that the smaller value ends up in the leftmost cell. At even steps do the same but for the pairs 2 and 3, 4 and 5, etc.

Procedure Odd-even Transposition Sort

```
For i:= 1 to n do
   If I is odd then
       For k := 1,3….. 2[n/2] -1 do in parallel
          Compare-exchange ($P_k$,$P_{k+1}$)

   Else
      For k : 2,4…..2[(n-1)/2] do in parallel
         Compare-exchange ($P_k$,$P_{k+1}$)
```

```
        End for
    End if
End for
```

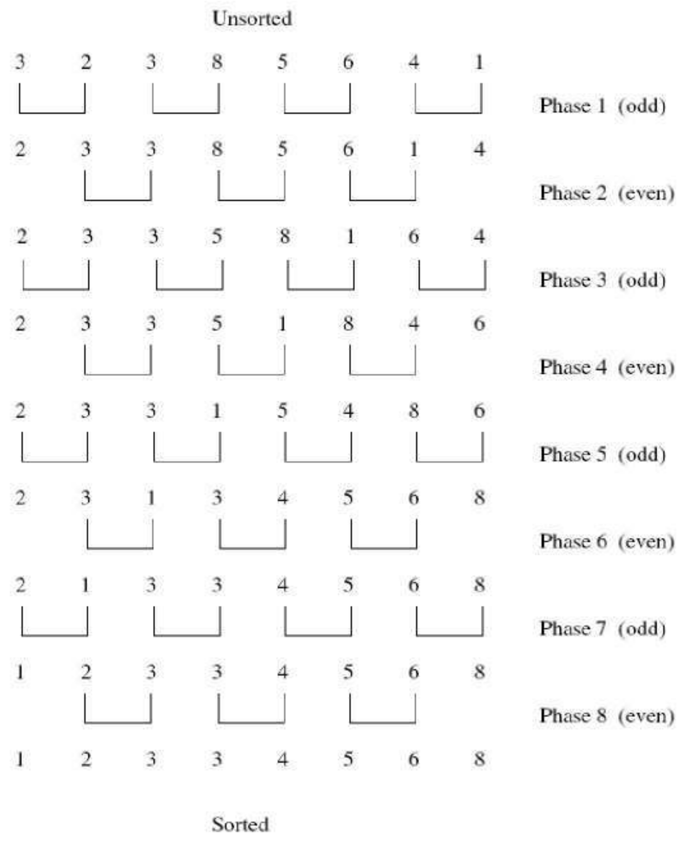


Figure 21.8 Odd-even Transposition Sort

Analysis

There are n parallel phases in the algorithm and during each phase the odd –indexed or even indexed processors perform a compare exchange operations which takes O(1) time. The time complexity of the algorithm is O(n). The number of processors used is n. The cost is O(n^2) which is not optimal.

Sorting on a Hypercube

Bitonic sort algorithm can be efficiently mapped onto the hypercube network without much communication overhead. The basic idea is to simulate the functioning of the combinational circuit for bitonic sorting using a hypercube network. Let the input wires (lines) in the combinational circuit are numbered as 0000, 0001……. 1111 in binary from top to bottom. The comparison operation is performed between 2 wires whose indices differ exactly in one bit. In a hypercube, processors whose indices differ in only bit are neighbours. Thus an optimal mapping of input wires to hypercube processors is the one that maps an input wore with index l to a processor with index l where l = 1,2,….n-1 and n is the number of processors. Whenever a

comparator performs a comparison on 2 wires, a compare exchange operation is performed between the corresponding processors in the hypercube.
The time complexity of this algorithm is $O(\log^2 n)$

```
Procedure Bitonic Sort
For i := 0 to d-1 do { $n = 2^d$ ; d is the dimension of hypercube}
  For j := I downto 0 do
      For each $P_k$ such that (i+1)st bit of k $\neq$ jth bit of k do in parallel
          Compare–exchange($P_k$,$P_k^{(j)}$)
                { for any index p, $p^{(1)}$ denotes the index obtained by flipping the 1st bit in the
                binary representation of P}
       End for
  End for
End for
```

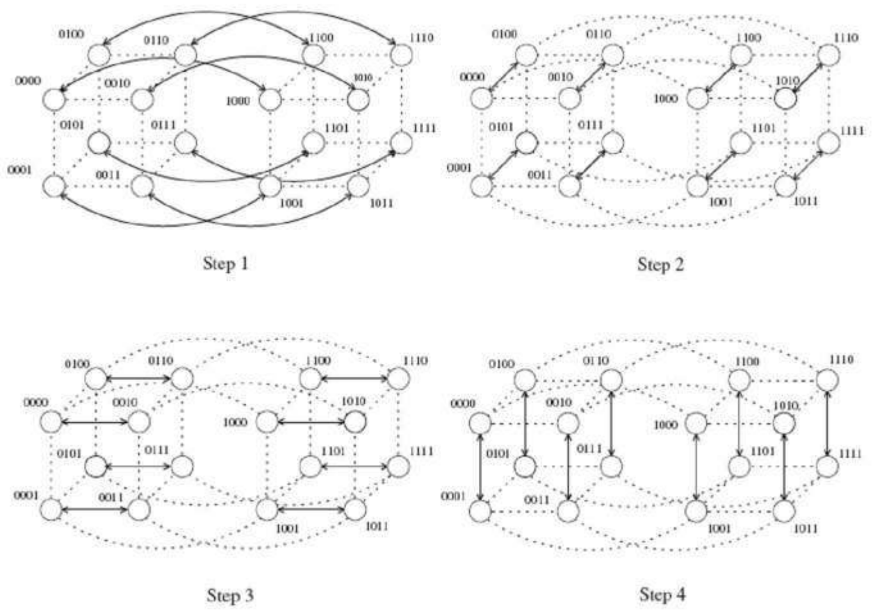
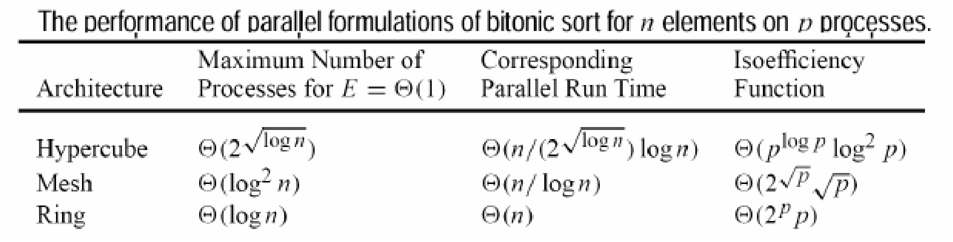


Figure 21.9 Communication during the last stage of bitonic sort

21.3 Let us Sum Up

Various sorting techniques in parallel processors has been discussed. The bitonic sort, odd-even merge sort, CRCW sorting, CREW sorting and EREW sorting. A method called as odd-even transposition sort has been discussed and using the method of bitonic sort sorting on hypercube has been implemented.



The performance of parallel formulations of bitonic sort for n elements on p processes.

Architecture	Maximum Number of Processes for $E = \Theta(1)$	Corresponding Parallel Run Time	Isoefficiency Function
Hypercube	$\Theta(2^{\sqrt{\log n}})$	$\Theta(n/(2^{\sqrt{\log n}}) \log n)$	$\Theta(p^{\log p} \log^2 p)$
Mesh	$\Theta(\log^2 n)$	$\Theta(n/ \log n)$	$\Theta(2^{\sqrt{p}} \sqrt{p})$
Ring	$\Theta(\log n)$	$\Theta(n)$	$\Theta(2^p p)$

21.4 Lesson-end Activities

1. Discuss parallel processor sorting techniques.

21.5 Points for discussions

- Sorting is a concept of arranging unsorted sequence into sorted sequence. Parallel processors and algorithms has been implemented.
- Various types of Sorting are Bitonic sort, odd-even merge sort, odd-even transposition sort, sorting on PRAM models and sorting on hypercube

21.6 References

- Parallel Computer architectures and Programming, V. Rajaraman & C. Siva Ram Murthy
- IFI TE IFI TECHNICAL REPORTS, Institute of Computer Science, Clausthal University of Technology

Lesson 22 Searching & Matrix Operations

Contents:

22.0 Aims and Objectives

The main objective of this lesson is to perform searching operations, matrix multiplication and gauss elimination method using parallel algorithms.

22.1 Introduction

Searching is one of the most fundamental operations encountered in computing. It is used in application where an element belongs to list has to be found out. Here Searching has been implemented through PRAM models and the complexity of the algorithm is calculated. Like searching matrix and gauss elimination are fundamental components of many numerical and non-numerical algorithms.

22.2 Searching

In the simple case, an array, $A = \{a_1, a_2, \ldots, a_n\}$ of $n$elements and an element x, is to be searched for an element, $a_i = x$.

In the worst case a search algorithm requires $O(n)$ time.

If the array is sorted in nondecreasing order then a binary search algorithm can provide the answer in $O(\log n)$ time. This is optimal because it requires this many of bits to identify an element.

22.2.2 Searching on PRAM Models

Let x denote the element to be searched in A

EREW Searching

Consider an N-processor EREW SM SIMD computer.

In this case the value for x must first be broadcast to all of the processors, requiring $O(\log n)$ time.

The sequence A can then be subdivided into N subsequences of length n/N each and each processor P_i is assigned

$$\{A_{(i-1)(n/N)+1}, A_{(i-1)(n/N)+2}, \ldots, A_{i(n/N)}\}.$$

Binary search on these subsequences requires $O(\log(n/N))$ time so that the parallel algorithm requires $O(\log N) + O(\log(n/N))$ time which is $O(\log n)$ and no better than the sequential version.

CREW Searching

Consider an N-processor CREW SM SIMD computer.

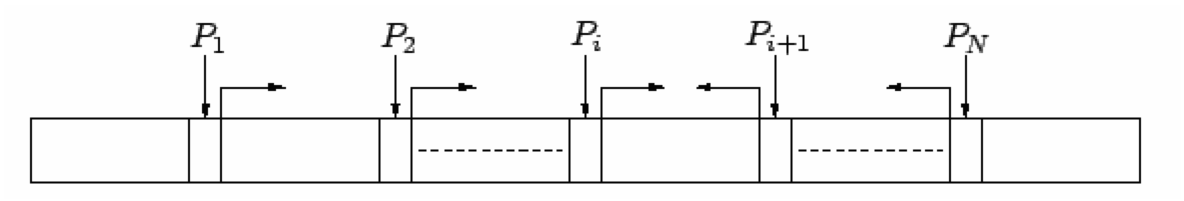
In this case, x does not need to be broadcast since all processors can simply read the value in parallel. However using the naive division of space leads to $O(\log(n/N)) = O(\log n)$ time.

We can modify the binary search itself to achieve a real speedup.

Recall that the binary search algorithm compares first the element that is at the middle of the array, $a_{n/2}$ for odd n and say $a_{\lceil n/2 \rceil}$ for even n, making a decision to keep the upper or lower half of the array (if the middle element is not itself equal to x).

To obtain a speedup we use N processors to do a $N+1$-ary search.

At each stage of the algorithm the array is split into $N+1$ subsequences of equal length and the N processors simultaneously check the boundaries of their assigned subsequence. Each processor determines whether the target element is to the left or the right of the boundary.



The next stage divides the selected subsequences in $N+1$ subsequences and continues.

Since each stage is applied to an array whose length is $1/(N+1)$ the length of the sequence searched during the previous stage less 1, $O(\log_{N+1}(n+1))$ stages are needed.

As intuitive proof (Akl, 1989), let g be the smallest integer such that $n \leq (N+1)^g - 1$, that is $g = \lceil \log(n+1)/\log(N+1) \rceil$. The proof that g stages are need is by induction. The statement is true for $g=0$. Assume it is true for $(N+1)^{g-1} - 1$. Now, to search an array of length $(N+1)^g - 1$, processor P_i, $i = 1, 2, \ldots, N$, compares x to a_j where $j = i(N+1)^{g-1}$. Following the comparison only a subsequence of length $(N+1)^{g-1} - 1$ needs to be search, which completes the proof.

For N processors we've achieved a speedup of

$$\frac{\log_2 n}{\log_{N+1}(n+1)} = \frac{\log_{N+1} n}{\log_{N+1} 2 \log_{N+1}(n+1)}$$

$$= \frac{\log_{n+1} n}{\log_{N+1} 2}$$

$$= O(\log_2 N)$$;for large n .

The efficiency is then $O(\frac{1}{N}\log N)$.

Although this is not cost optimal, it can be shown to be the best possible cost for parallel searching. Note that any algorithm using Nprocessors can compare an input element $x$to at most Nelements of Asimultaneously. The remaining subsequence (on average) will be at least

$$\left\lceil\frac{n-N}{N+1}\right\rceil \geq \frac{n-N}{N+1}.$$

$$\left\lceil\frac{n-N}{N+1}\right\rceil \geq \frac{n-N}{N+1}.$$

CRCW searching

In previous discussion we assumed that all element of A were unique.

For arrays of non-unique elements, the processors may well attempt to return the index of a found element in parallel.

Similar to broadcast, write conflicts over Nprocessors can be resolved in $O(\log N)$ steps. Foor CRCW machines and with $N = n$, this additional constant eliminates any benefits. any speedup.

Procedure Search

```
1. Broadcast x to all the N processors
2. For i = 1 to N do in parallel
Si = {$S_{(i-1)(n/N)+1}$, $S_{(i-1)(n/N)+2}$.....}
Search for x sequentially in $S_i$ and store the result in $k_i$
End for
s3. Fpr I = 1 to N do in parallel
If Ki > 0 then K := $K_i$
End if
End for
```

22.2.2 Matrix Operations

Matrix operations are also the fundamental components of many numerical and non-numerical algorithms.

22.2.3 Matrix Multiplication

The product of an m x n matrix A and n x k matrix of B is M x K matrix of C whose elements are
$Cij = \Sigma\, a_{is} * b_{sj;}\ s = 1\ldots.n$

```
Procedure Matrix Multiplication
For I := 1 to m do
   For j := 1 to k do
     $C_{ij} = 0$
       For s := 1 to n do
          $C_{ij} = C_{ij} + a_{is} * b_{sj}$;
       End for
   End for
End for
```

CREW Matrix Multiplication

The algorithm uses n^2 processors which are arranged in a 2d array of size n x n.
Overall complexity is O(n).

Procedure CREW Matrix Multiplication

```
For I := 1 to n do in parallel
   For j := 1 to n do in parallel
     $C_{i,j} = 0$
       For k := 1 to n do
          $C_{i,j} = C_{i,j} + a_{i,k} * b_{k,j}$;
       End for
   End for
End for
```

EREW Matrix Multiplication

In case of CREW model one advantage is that a memory location can be accessed by any other processor. In EREW model one needs to ensure that every processor reads the value from a memory location which is not being accessed by any other processor.

Procedure EREW Matrix Multiplication

```
For I := 1 to n do in parallel
   For j := 1 to n do in parallel
     $C_{i,j} = 0$
       For k := 1 to n do
           $l_k$ := (i+j+k)mod n+1;
```

```
            $C_{i,j} = C_{i,j} + a_{i,1k} * b_{1k,j}$;
        End for
    End for
End for
```

CRCW Matrix Multiplication
The algorithm uses n 3 processors and runs O(1) time. When more than one processor attempts to write to he same memory location, the sum of the values is written onto the memory location,

Procedure CRCW Matrix Multiplication

```
For I := 1 to n do in parallel
    For j := 1 to n do in parallel
            For s := 1 to n do in parallel
                    $C_{i,j} = 0$
                    $C_{i,j} = C_{i,j} + a_{i,s} * b_{s,,j}$;
        End for
    End for
End for
```

22.2.4 Solving s system of linear equations

The problem of solving a system of linear equations (Ax = b) is central fro many problems in scientific and engineering computing. One of the method for solving linear equation is gauss elimination method.

$$a_{0\,0} x_0 + a_{0\,1} x_1 + \ldots + a_{0\,n-1} x_{n-1} = b_0$$

$$a_{1\,0} x_0 + a_{1\,1} x_1 + \ldots + a_{1\,n-1} x_{n-1} = b_1$$

…….

$$a_{n-1\,0} x_0 + a_{n-1\,1} x_1 + \ldots + a_{n-1\,n-1} x_{n-1} = b_{n-1}$$

- n equations and n variables (x0,x1,...,xn-1)
- Can be expressed in matrix vector notation A x = b
- Reduce Ax=b into Ux=y
    - U is an upper triangular
    - Diagonal elements Uii = 1

$$\begin{aligned} x_0 + u_{0\,1} x_1 + \ldots + u_{0\,n-1} x_{n-1} &= y_0 \\ x_1 + \ldots + u_{1\,n-1} x_{n-1} &= y_1 \\ \ldots\ldots\ldots & \\ x_{n-1} &= y_{n-1} \end{aligned}$$

- Back substitute

- Two phases repeated n times
- Consider the k-th iteration ($0 \leq k < n$)
- Phase 1: Normalize k-th row of A

```
for j = k+1 to n-1    Akj /= Akk
yk   =  bk/Akk
Akk = 1
```

- Phase 2: Eliminate

```
Using k-th row, make k-th column of A zero for row# > k
    for i = k+1 to n-1
    for j = k+1 to n-1  Aij -= Aik*Akj
    bi   -= aik * yk
    Aik = 0
```

- O(n**2**) divides, O(n**3**) subtracts and multiplies
- p = n, row-striped partition

```
for k = 0 to n-1
    k-th phase 1
        performed by Pk
        sequentially, no communication
    k-th phase 2
        Pk broadcasts k-th row to Pk+1, ... , Pn-1
        performed in parallel by Pk+1, ... , Pn-1
```

Upper Triangularization – Pipelined Parallel

- p = n, row-stripes partition

```
for all Pi  (i = 0 .. n-1) do in parallel
    for k = 0 to n-1
        if  (i > k)
                receive row k, send it down
            perform k-th phase 2
        if (i == k)
            perform k-th phase 1
            send row k down
```

Procedure Gauss Elimination

```
For k := 1 to n do
   For j := k+1 to n do
      A[k,j] = a[k,j] / a[k,k]
   End for
  Y[k] := b[k] / a[k,k]
```

```
A[k,k] := 1
For I := k+1 to n do
  For j := k+1 to n do
     A[i,j] := a[i,j] – a[i,k] * a[k,j]
  End for
B[i] := b[i] – a[i,k] * y[k]
A[i,k]:= 0;
  End for
end for
```

22.3 Let us Sum Up

Various algorithms has been carried out for searching operations using PRAM models and the difference in the execution has been discussed. Matrix multiplication has been implemented through CREW, EREW, CRCW model and gauss elimination has been solved using parallel algorithms.

22.4 Lesson-end Activities

1. What are the searching techniques for PRAM models? Explain.
2. Discuss matrix multiplication using parallel algorithms.

22.5 Points for discussions

- Searching on PRAM Models
- Matrix Multiplication on PRAM Models

22.6 References

- http://www-unix.mcs.anl.gov/dbpp/text/node45.html#eqlamatmulc
- Parallel and Distributed Processing, CSE 8380, SMU school of engineering