ACA (15CS72) MODULE-1

- Objective
- Introduction

- The state of computing
  - Evolution of computer system
  - Elements of Modern Computers
  - Flynn's Classical Taxonomy
  - System attributes
- Multiprocessor and multicomputer,
  - Shared memory multiprocessors
  - Distributed Memory Multiprocessors
  - A taxonomy of MIMD Computers
- Multi vector and SIMD computers
  - Vector Supercomputer
  - SIMD supercomputers
- PRAM and VLSI model
  - Parallel Random Access machines
  - VLSI Complexity Model
- Keywords
- Summary

1.0 Objective

The main aim of this chapter is to learn about the evolution of computer systems, various attributes on which performance of system is measured, classification of computers on their ability to perform multiprocessing and various trends towards parallel processing.

1.1 Introduction

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

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.

Parallel processing is emerging as one of the key technology in area of modern computers. Parallel appears in various forms such as lookahead, vectorization concurrency, simultaneity, data parallelism, interleaving, overlapping, multiplicity, replication, multiprogramming, multithreading and distributed computing at different processing level.

1.2 The state of computing

Modern computers are equipped with powerful hardware technology at the same time loaded with sophisticated software packages. To access the art of computing we firstly review the history of computers then study the attributes used for analysis of performance of computers.

1.2.1 Evolution of computer system

Presently the technology involved in designing of its hardware components of computers and its overall architecture is changing very rapidly for example: processor clock rate increase about 20% a year, its logic capacity improve at about 30% in a year; memory speed at increase about 10% in a year and memory capacity at about 60% increase a year also the disk capacity increase at a 60% a year and so overall cost per bit improves about 25% a year.

But before we go further with design and organization issues of parallel computer architecture it is necessary to understand how computers had evolved. Initially, man used simple mechanical devices – abacus (about 500 BC) , knotted string, and the slide rule for

computation. Early computing was entirely mechanical like : mechanical adder/subtracter (Pascal, 1642) difference engine design (Babbage, 1827) binary mechanical computer (Zuse, 1941) electromechanical decimal machine (Aiken, 1944). Some of these machines used the idea of a stored program a famous example of it is the Jacquard Loom and Babbage's Analytical Engine which is also often considered as the first real computer. Mechanical and electromechanical machines have limited speed and reliability because of the many moving parts. Modern machines use electronics for most information transmission.

Computing is normally thought of as being divided into generations. Each successive generation is marked by sharp changes in hardware and software technologies. With some exceptions, most of the advances introduced in one generation are carried through to later generations. We are currently in the fifth generation.

Ist generation of computers (1945-54)

The first generation computers where based on vacuum tube technology. The first large electronic computer was **ENIAC** (Electronic Numerical Integrator and Calculator), which used high speed vacuum tube technology and were designed primarily to calculate the trajectories of missiles. They used separate memory block for program and data. Later in 1946 John Von Neumann introduced the concept of stored program, in which data and program where stored in same memory block. Based on this concept **EDVAC** (Electronic Discrete Variable Automatic Computer) was built in 1951. On this concept IAS (Institute of advance studies, Princeton) computer was built whose main characteristic was CPU consist of two units (Program flow control and execution unit).

In general key features of this generation of computers where

1) The switching device used where vacuum tube having switching time between 0.1 to 1 milliseconds.

2) One of major concern for computer manufacturer of this era was that each of the computer designs had a unique design. As each computer has unique design one cannot upgrade or replace one component with other computer. Programs that were written for one machine could not execute on another machine, even though other computer was also designed from the same company. This created a major concern for designers as there were no upward-compatible machines or computer architectures with multiple, differing

implementations. And designers always tried to manufacture a new machine that should be upward compatible with the older machines.

3) Concept of specialized registers where introduced for example index registers were introduced in the Ferranti Mark I, concept of register that save the return-address instruction was introduced in UNIVAC I, also concept of immediate operands in IBM 704 and the detection of invalid operations in IBM 650 were introduced.

4) Punch card or paper tape were the devices used at that time for storing the program. By the end of the 1950s IBM 650 became one of popular computers of that time and it used the drum memory on which programs were loaded from punch card or paper tape. Some high-end machines also introduced the concept of core memory which was able to provide higher speeds. Also hard disks started becoming popular.

5) In the early 1950s as said earlier were design specific hence most of them were designed for some particular numerical processing tasks. Even many of them used decimal numbers as their base number system for designing instruction set. In such machine there were actually ten vacuum tubes per digit in each register.

6) Software used was machine level language and assembly language.

7) Mostly designed for scientific calculation and later some systems were developed for simple business systems.

8) Architecture features

Vacuum tubes and relay memories

CPU driven by a program counter (PC) and accumulator

Machines had only fixed-point arithmetic

9) Software and Applications

Machine and assembly language

Single user at a time

No subroutine linkage mechanisms

Programmed I/O required continuous use of CPU

10) examples: ENIAC, Princeton IAS, IBM 701

IInd generation of computers (1954 – 64)

The transistors were invented by Bardeen, Brattain and Shockely in 1947 at Bell Labs and by the 1950s these transistors made an electronic revolution as the transistor is

smaller, cheaper and dissipate less heat as compared to vacuum tube. Now the transistors were used instead of a vacuum tube to construct computers. Another major invention was invention of magnetic cores for storage. These cores where used to large random access memories. These generation computers has better processing speed, larger memory capacity, smaller size as compared to pervious generation computer.

The key features of this generation computers were

1) The IInd generation computer were designed using Germanium transistor, this technology was much more reliable than vacuum tube technology.

2) Use of transistor technology reduced the switching time 1 to 10 microseconds thus provide overall speed up.

2) Magnetic cores were used main memory with capacity of 100 KB. Tapes and disk peripheral memory were used as secondary memory.

3) Introduction to computer concept of instruction sets so that same program can be executed on different systems.

4) High level languages, FORTRAN, COBOL, Algol, BATCH operating system.

5) Computers were now used for extensive business applications, engineering design, optimation using Linear programming, Scientific research

6) Binary number system very used.

7) Technology and Architecture

Discrete transistors and core memories

I/O processors, multiplexed memory access

Floating-point arithmetic available

Register Transfer Language (RTL) developed

8) Software and Applications

High-level languages (HLL): FORTRAN, COBOL, ALGOL with compilers and subroutine libraries

Batch operating system was used although mostly single user at a time

9) Example : CDC 1604, UNIVAC LARC, IBM 7090

IIIrd Generation computers*(1965 to 1974)*

In 1950 and 1960 the discrete components (transistors, registers capacitors) were manufactured packaged in a separate containers. To design a computer these discrete

unit were soldered or wired together on a circuit boards. Another revolution in computer designing came when in the 1960s, the Apollo guidance computer and Minuteman missile were able to develop an integrated circuit (commonly called ICs). These ICs made the circuit designing more economical and practical. The IC based computers are called third generation computers. As integrated circuits, consists of transistors, resistors, capacitors on single chip eliminating wired interconnection, the space required for the computer was greatly reduced. By the mid-1970s, the use of ICs in computers became very common. Price of transistors reduced very greatly. Now it was possible to put all components required for designing a CPU on a single printed circuit board. This advancement of technology resulted in development of minicomputers, usually with 16-bit words size these system have a memory of range of 4k to 64K.This began a new era of microelectronics where it could be possible design small identical chips (a thin wafer of silicon's). Each chip has many gates plus number of input output pins.

Key features of IIIrd Generation computers:

1) The use of silicon based ICs, led to major improvement of computer system. Switching speed of transistor went by a factor of 10 and size was reduced by a factor of 10, reliability increased by a factor of 10, power dissipation reduced by a factor of 10. This cumulative effect of this was the emergence of extremely powerful CPUS with the capacity of carrying out 1 million instruction per second.

2) The size of main memory reached about 4MB by improving the design of magnetic core memories also in hard disk of 100 MB become feasible.

3) On line system become feasible. In particular dynamic production control systems, airline reservation systems, interactive query systems, and real time closed lop process control systems were implemented.

4) Concept of Integrated database management systems were emerged.

5) 32 bit instruction formats

6) Time shared concept of operating system.

7) Technology and Architecture features

Integrated circuits (SSI/MSI)

Microprogramming

Pipelining, cache memories, lookahead processing

8) Software and Applications

Multiprogramming and time-sharing operating systems

Multi-user applications

9) Examples : IBM 360/370, CDC 6600, TI ASC, DEC PDP-82

IVth Generation computer (*(1975 to 1990)*

The microprocessor was invented as a single VLSI (Very large Scale Integrated circuit) chip CPU. Main Memory chips of 1MB plus memory addresses were introduced as single VLSI chip. The caches were invented and placed within the main memory and microprocessor. These VLSIs and VVSLIs greatly reduced the space required in a computer and increased significantly the computational speed.

1) Technology and Architecture feature

LSI/VLSI circuits,

semiconductor memory

Multiprocessors,

vector supercomputers,

multicomputers

Shared or distributed memory

Vector processors

Software and Applications

Multprocessor operating systems,

languages,

compilers,

parallel software tools

Examples : VAX 9000, Cray X-MP, IBM 3090, BBN TC2000

Fifth Generation computers(1990 onwards)

In the mid-to-late 1980s, in order to further improve the performance of the system the designers start using a technique known as “instruction pipelining”. The idea is to break the program into small instructions and the processor works on these instructions in different stages of completion. For example, the processor while calculating the result of the current instruction also retrieves the operands for the next instruction. Based on this concept later superscalar processor were designed, here to execute multiple instructions

in parallel we have multiple execution unit i.e., separate arithmetic-logic units (ALUs). Now instead executing single instruction at a time, the system divide program into several independent instructions and now CPU will look for several similar instructions that are not dependent on each other, and execute them in parallel. The example of this design are VLIW and EPIC.

1) Technology and Architecture features

ULSI/VHSIC processors, memory, and switches

High-density packaging

Scalable architecture

Vector processors

2) Software and Applications

Massively parallel processing

Grand challenge applications

Heterogenous processing

3) Examples : Fujitsu VPP500, Cray MPP, TMC CM-5, Intel Paragon

Elements of Modern Computers

The hardware, software, and programming elements of modern computer systems can be characterized by looking at a variety of factors in context of parallel computing these factors are:

- Computing problems

  Algorithms and data structures
- Hardware resources

  Operating systems
- System software support

  Compiler support
-
-
-

Computing Problems

- Numerical computing complex mathematical formulations tedious integer or floating -point computation

- Transaction processing accurate transactions large database management information retrieval
- Logical Reasoning logic inferences symbolic manipulations

Algorithms and Data Structures

- Traditional algorithms and data structures are designed for sequential machines.
- New, specialized algorithms and data structures are needed to exploit the capabilities of parallel architectures.
- These often require interdisciplinary interactions among theoreticians, experimentalists, and programmers.

Hardware Resources

- The architecture of a system is shaped only partly by the hardware resources.
- The operating system and applications also significantly influence the overall architecture.
- Not only must the processor and memory architectures be considered, but also the architecture of the device interfaces (which often include their advanced processors).

Operating System

- Operating systems manage the allocation and deallocation of resources during user program execution.
- UNIX, Mach, and OSF/1 provide support for multiprocessors and multicomputers
- multithreaded kernel functions virtual memory management file subsystems network communication services
- An OS plays a significant role in mapping hardware resources to algorithmic and data structures.

System Software Support

- Compilers, assemblers, and loaders are traditional tools for developing programs in high-level languages. With the operating system, these tools determine the bind of resources to applications, and the effectiveness of this determines the efficiency of hardware utilization and the system's programmability.
- Most programmers still employ a sequential mind set, abetted by a lack of popular parallel software support.

System Software Support

- Parallel software can be developed using entirely new languages designed specifically with parallel support as its goal, or by using extensions to existing sequential languages.
- New languages have obvious advantages (like new constructs specifically for parallelism), but require additional programmer education and system software.
- The most common approach is to extend an existing language.

Compiler Support

- Preprocessors use existing sequential compilers and specialized libraries to implement parallel constructs
- Precompilers perform some program flow analysis, dependence checking, and limited parallel optimzations
- Parallelizing Compilers requires full detection of parallelism in source code, and transformation of sequential code into parallel constructs
- Compiler directives are often inserted into source code to aid compiler parallelizing efforts

1.2.3 Flynn's Classical Taxonomy

Among mentioned above the one widely used since 1966, is Flynn's Taxonomy. This taxonomy distinguishes multi-processor computer architectures according two independent dimensions of *Instruction stream* and *Data stream*. An instruction stream is sequence of instructions executed by machine. And a data stream is a sequence of data including input, partial or temporary results used by instruction stream. Each of these dimensions can have only one of two possible states: *Single* or *Multiple*. Flynn's classification depends on the distinction between the performance of control unit and the data processing unit rather than its operational and structural interconnections. Following are the four category of Flynn classification and characteristic feature of each of them.

1. Single instruction stream, single data stream (SISD)

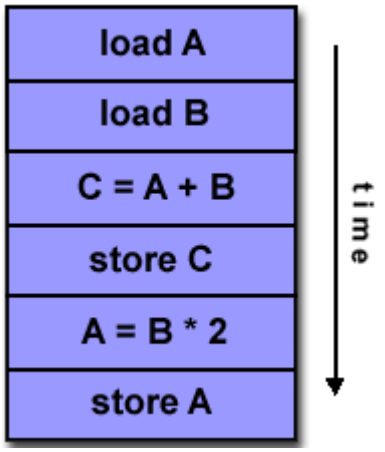


Figure 1.1 Execution of instruction in SISD processors

The figure 1.1 is represents a organization of simple SISD computer having one control unit, one processor unit and single memory unit.

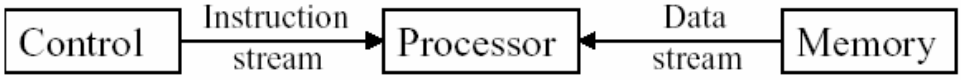


Figure 1.2 SISD processor organization

- They are also called scalar processor i.e., one instruction at a time and each instruction have only one set of operands.
- 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
- Instructions are executed sequentially.
- This is the oldest and until recently, the most prevalent form of computer
- Examples: most PCs, single CPU workstations and mainframes

b) Single instruction stream, multiple data stream (SIMD) processors

- A type of parallel computer
- Single instruction: All processing units execute the same instruction issued by the control unit at any given clock cycle as shown in figure 13.5 where there are multiple processor executing instruction given by one control unit.

- Multiple data: Each processing unit can operate on a different data element as shown if figure below the processor are connected to shared memory or interconnection network providing multiple data to processing unit

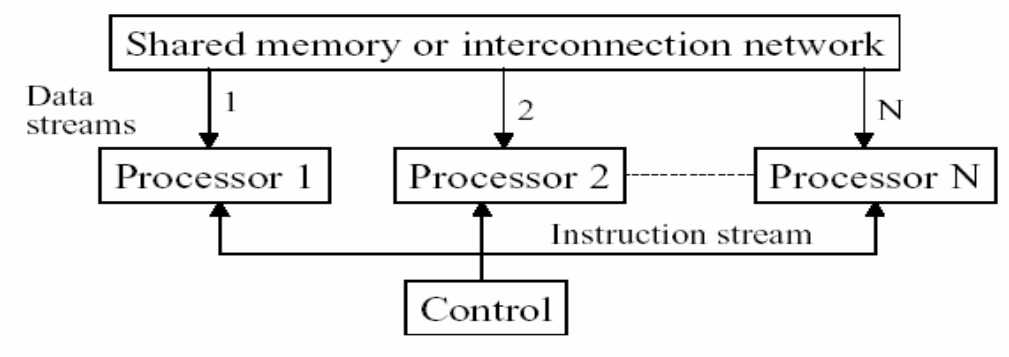


Figure 1.3 SIMD processor organization

- 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.
- Thus single instruction is executed by different processing unit on different set of data as shown in figure 1.3.
- Best suited for specialized problems characterized by a high degree of regularity, such as image processing and vector computation.
- Synchronous (lockstep) and deterministic execution
- Two varieties: Processor Arrays e.g., Connection Machine CM-2, Maspar MP-1, MP-2 and Vector Pipelines processor e.g., IBM 9000, Cray C90, Fujitsu VP, NEC SX-2, Hitachi S820

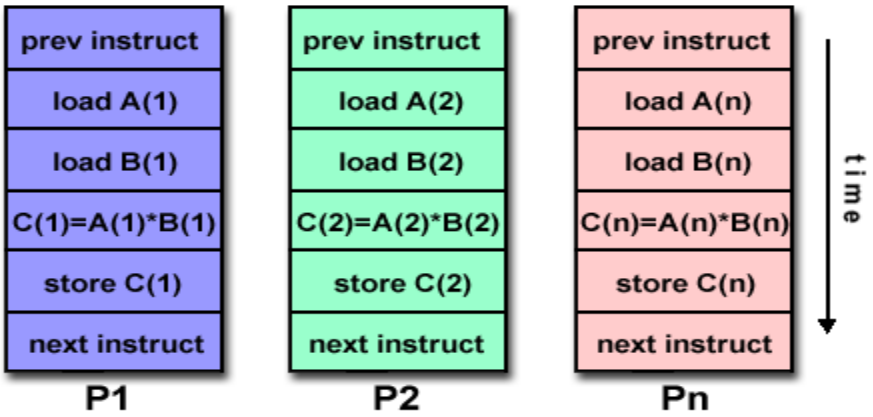


Figure 1.4 Execution of instructions in SIMD processors

c) Multiple instruction stream, single data stream (MISD)

- A single data stream is fed into multiple processing units.
- Each processing unit operates on the data independently via independent instruction streams as shown in figure 1.5 a single data stream is forwarded to different processing unit which are connected to different control unit and execute instruction given to it by control unit to which it is attached.

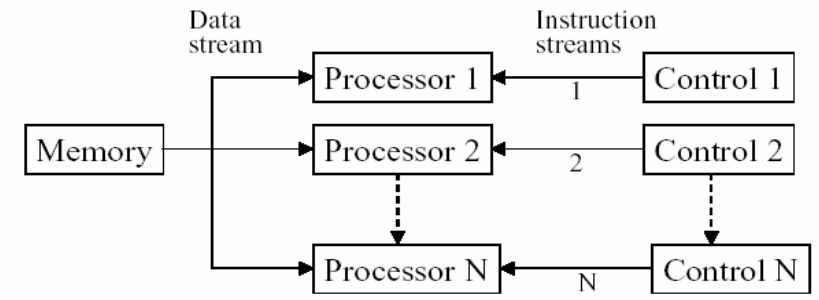


Figure 1.5 MISD processor organization

- Thus in these computers same data flow through a linear array of processors executing different instruction streams as shown in figure 1.6.
- This architecture is also known as systolic arrays for pipelined execution of specific instructions.
- 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:

1. multiple frequency filters operating on a single signal stream
2. multiple cryptography algorithms attempting to crack a single coded message.

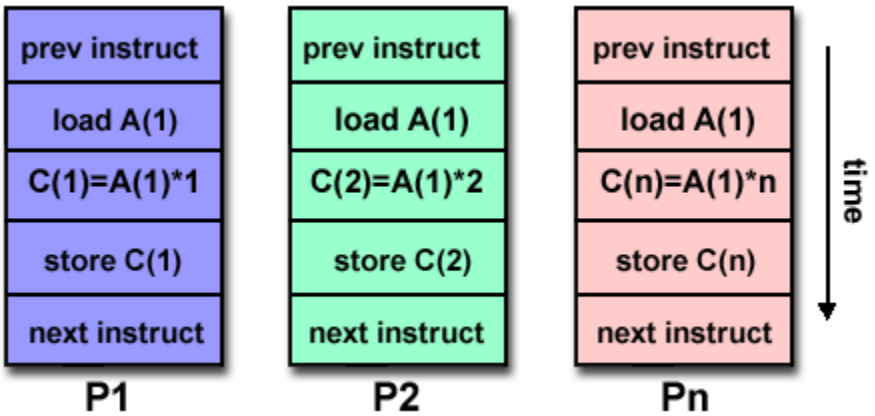


Figure 1.6 Execution of instructions in MISD processors

d) Multiple instruction stream, multiple data stream (MIMD)

- Multiple Instruction: every processor may be executing a different instruction stream
- Multiple Data: every processor may be working with a different data stream as shown in figure 1.7 multiple data stream is provided by shared memory.
- Can be categorized as loosely coupled or tightly coupled depending on sharing of data and control
- Execution can be synchronous or asynchronous, deterministic or non-deterministic

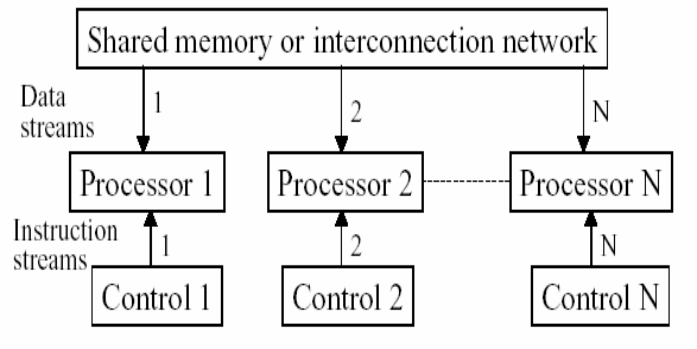


Figure 1.7 MIMD processor organizations

- As shown in figure 1.8 there are different processor each processing different task.
- Examples: most current supercomputers, networked parallel computer "grids" and multi-processor SMP computers - including some types of PCs.

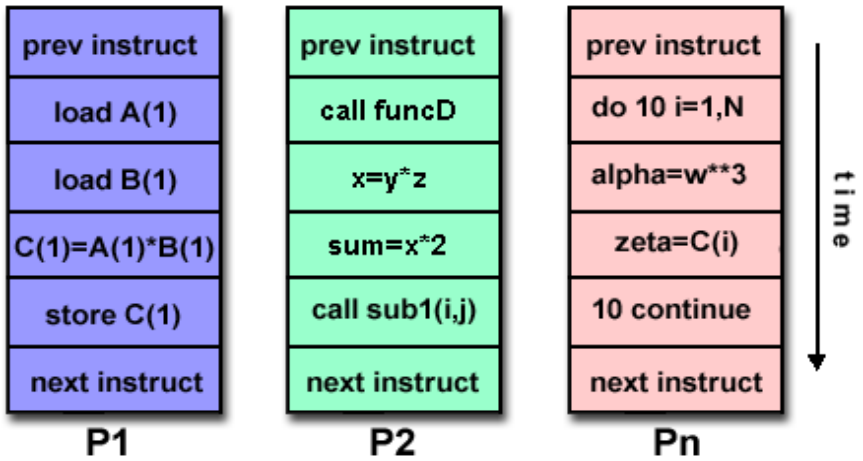


Figure 1.8 execution of instructions MIMD processors

Here the some popular computer architecture and there types

SISD IBM 701, IBM 1620, IBM 7090, PDP VAX11/ 780

SISD (With multiple functional units) IBM360/91 (3); IBM 370/168 UP

SIMD (Word Slice Processing) Illiac – IV ; PEPE

SIMD (Bit Slice processing) STARAN; MPP; DAP

MIMD (Loosely Coupled) IBM 370/168 MP; Univac 1100/80

MIMD(Tightly Coupled) Burroughs- D – 825

1.2.4 PERFORMANCE ATTRIBUTES

Performance of a system depends on

- hardware technology
- architectural features
- efficient resource management
- algorithm design
- data structures
- language efficiency
- programmer skill
- compiler technology

When we talk about performance of computer system we would describe how quickly a given system can execute a program or programs. Thus we are interested in knowing the turnaround time. Turnaround time depends on:

- disk and memory accesses
- input and output
- compilation time
- operating system overhead
- CPU time

An ideal performance of a computer system means a perfect match between the machine capability and program behavior. The machine capability can be improved by using better hardware technology and efficient resource management. But as far as program behavior is concerned it depends on code used, compiler used and other run time conditions. Also a machine performance may vary from program to program. Because there are too many programs and it is impractical to test a CPU's speed on all of them,

benchmarks were developed. Computer architects have come up with a variety of metrics to describe the computer performance.

Clock rate and CPI / IPC : Since I/O and system overhead frequently overlaps processing by other programs, it is fair to consider only the CPU time used by a program, and the user CPU time is the most important factor. CPU is driven by a clock with a constant cycle time (usually measured in nanoseconds, which controls the rate of internal operations in the CPU. The clock mostly has the constant cycle time (t in nanoseconds). The inverse of the cycle time is the clock rate (*f* = 1/⏐, measured in megahertz). A shorter clock cycle time, or equivalently a larger number of cycles per second, implies more operations can be performed per unit time. The size of the program is determined by the instruction count (Ic). The size of a program is determined by its instruction count, I*c*, the number of machine instructions to be executed by the program. Different machine instructions require different numbers of clock cycles to execute. CPI (cycles per instruction) is thus an important parameter.

Average CPI

It is easy to determine the average number of cycles per instruction for a particular processor if we know the frequency of occurrence of each instruction type.

Of course, any estimate is valid only for a specific set of programs (which defines the instruction mix), and then only if there are sufficiently large number of instructions.

In general, the term CPI is used with respect to a particular instruction set and a given program mix. The time required to execute a program containing I*c* instructions is just T = I*c* * CPI * ⏐.

Each instruction must be fetched from memory, decoded, then operands fetched from memory, the instruction executed, and the results stored.

The time required to access memory is called the memory cycle time, which is usually k times the processor cycle time ⏐. The value of k depends on the memory technology and the processor-memory interconnection scheme. The processor cycles required for each instruction (CPI) can be attributed to cycles needed for instruction decode and execution (*p*), and cycles needed for memory references (*m* k*).

The total time needed to execute a program can then be rewritten as

T = I*c** (p + m*k)* ⏐.

MIPS: The *millions of instructions per second*, this is calculated by dividing the number of instructions executed in a running program by time required to run the program. The MIPS rate is directly proportional to the clock rate and inversely proportion to the CPI. All four systems attributes (instruction set, compiler, processor, and memory technologies) affect the MIPS rate, which varies also from program to program. MIPS does not proved to be effective as it does not account for the fact that different systems often require different number of instruction to implement the program. It does not inform about how many instructions are required to perform a given task. With the variation in instruction styles, internal organization, and number of processors per system it is almost meaningless for comparing two systems.

MFLOPS (pronounced ``megaflops'') stands for ``millions of floating point operations per second.'' This is often used as a ``bottom-line'' figure. If one know ahead of time how many operations a program needs to perform, one can divide the number of operations by the execution time to come up with a MFLOPS rating. For example, the standard algorithm for multiplying **n*n** matrices requires $\mathbf{2n^3 - n}$ operations (n^2 inner products, with **n** multiplications and **n-1**additions in each product). Suppose you compute the product of two 100 *100 matrices in 0.35 seconds. Then the computer achieves

$(2(100)^3 - 100)/0.35 = 5{,}714{,}000$ ops/sec = 5.714 MFLOPS

The term ``theoretical peak MFLOPS'' refers to how many operations per second would be possible if the machine did nothing but numerical operations. It is obtained by calculating the time it takes to perform one operation and then computing how many of them could be done in one second. For example, if it takes 8 cycles to do one floating point multiplication, the cycle time on the machine is 20 nanoseconds, and arithmetic operations are not overlapped with one another, it takes 160ns for one multiplication, and

(1,000,000,000 nanosecond/1sec)*(1 multiplication / 160 nanosecond) = $6.25*10^6$ multiplication /sec so the theoretical peak performance is 6.25 MFLOPS. Of course, programs are not just long sequences of multiply and add instructions, so a machine rarely comes close to this level of performance on any real program. Most machines will achieve less than 10% of their peak rating, but vector processors or other machines with internal pipelines that have an effective CPI near 1.0 can often achieve 70% or more of their theoretical peak on small programs.

Throughput rate : Another important factor on which system's performance is measured is throughput of the system which is basically how many programs a system can execute per unit time Ws. In multiprogramming the system throughput is often lower than the CPU throughput Wp which is defined as

Wp = f/(Ic * CPI)

Unit of Wp is programs/second.

Ws <Wp as in multiprogramming environment there is always additional overheads like timesharing operating system etc. An Ideal behavior is not achieved in parallel computers because while executing a parallel algorithm, the processing elements cannot devote 100% of their time to the computations of the algorithm. Efficiency is a measure of the fraction of time for which a PE is usefully employed. In an ideal parallel system efficiency is equal to one. In practice, efficiency is between zero and one

s of overhead associated with parallel execution

Speed or Throughput (W/Tn) - the execution rate on an n processor system, measured in FLOPs/unit-time or instructions/unit-time.

Speedup (Sn = T1/Tn) - how much faster in an actual machine, n processors compared to 1 will perform the workload. The ratio T1/T∞is called the *asymptotic speedup*.

Efficiency (En = Sn/n) - fraction of the theoretical maximum speedup achieved by n processors

Degree of Parallelism (DOP) - for a given piece of the workload, the number of processors that can be kept busy sharing that piece of computation equally. Neglecting overhead, we assume that if k processors work together on any workload, the workload gets done k times as fast as a sequential execution.

Scalability - The attributes of a computer system which allow it to be gracefully and linearly scaled up or down in size, to handle smaller or larger workloads, or to obtain proportional decreases or increase in speed on a given application. The applications run on a scalable machine may not scale well. Good scalability requires the algorithm *and* the machine to have the right properties

Thus in general there are five performance factors (I*c*, p, m, k, t) which are influenced by four system attributes:

- instruction-set architecture (affects *Ic* and *p*)

- compiler technology (affects *Ic* and *p* and *m*)
- CPU implementation and control (affects *p *t*) cache and memory hierarchy (affects memory access latency, *k ´t*)
- Total CPU time can be used as a basis in estimating the execution rate of a processor.

Programming Environments

Programmability depends on the programming environment provided to the users.

Conventional computers are used in a sequential programming environment with tools developed for a uniprocessor computer. Parallel computers need parallel tools that allow specification or easy detection of parallelism and operating systems that can perform parallel scheduling of concurrent events, shared memory allocation, and shared peripheral and communication links.

Implicit Parallelism

Use a conventional language (like C, Fortran, Lisp, or Pascal) to write the program.

Use a parallelizing compiler to translate the source code into parallel code.

The compiler must detect parallelism and assign target machine resources.

Success relies heavily on the quality of the compiler.

Explicit Parallelism

Programmer writes explicit parallel code using parallel dialects of common languages.

Compiler has reduced need to detect parallelism, but must still preserve existing parallelism and assign target machine resources.

Needed Software Tools

Parallel extensions of conventional high-level languages.

Integrated environments to provide different levels of program abstraction validation, testing and debugging performance prediction and monitoring visualization support to aid program development, performance measurement graphics display and animation of computational results

1.3 MULTIPROCESSOR AND MULTICOMPUTERS

Two categories of parallel computers are discussed below namely shared common memory or unshared distributed memory.

1.3.1 Shared memory multiprocessors

- Shared memory parallel computers vary widely, but generally have in common the ability for all processors to access all memory as global address space.
- Multiple processors can operate independently but share the same memory resources.
- Changes in a memory location effected by one processor are visible to all other processors.
- Shared memory machines can be divided into two main classes based upon memory access times: ***UMA*** , ***NUMA and COMA***.

Uniform Memory Access (UMA):

- Most commonly represented today by Symmetric Multiprocessor (SMP) machines
- Identical processors
- Equal access and access times to memory
- Sometimes called CC-UMA - Cache Coherent UMA. Cache coherent means if one processor updates a location in shared memory, all the other processors know about the update. Cache coherency is accomplished at the hardware level.

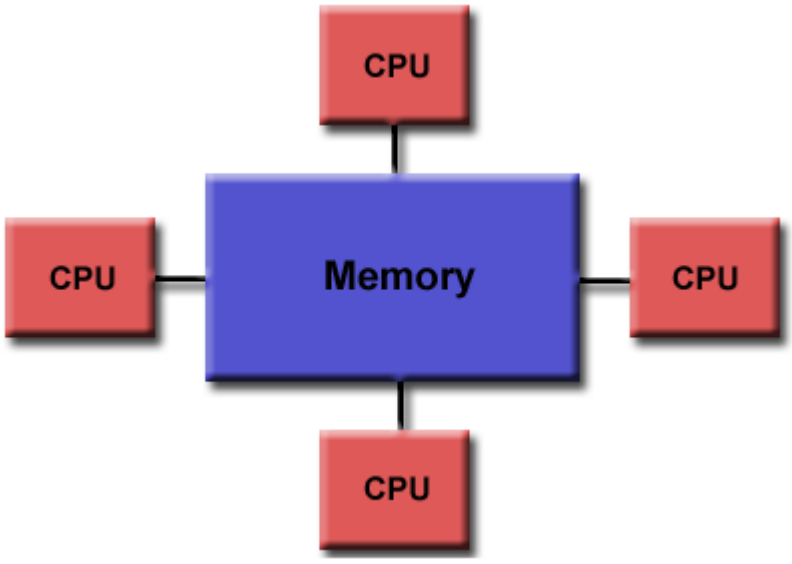


Figure 1.9 Shared Memory (UMA)

Non-Uniform Memory Access (NUMA):

- Often made by physically linking two or more SMPs
- One SMP can directly access memory of another SMP
- Not all processors have equal access time to all memories
- Memory access across link is slower

If cache coherency is maintained, then may also be called CC-NUMA - Cache Coherent NUMA

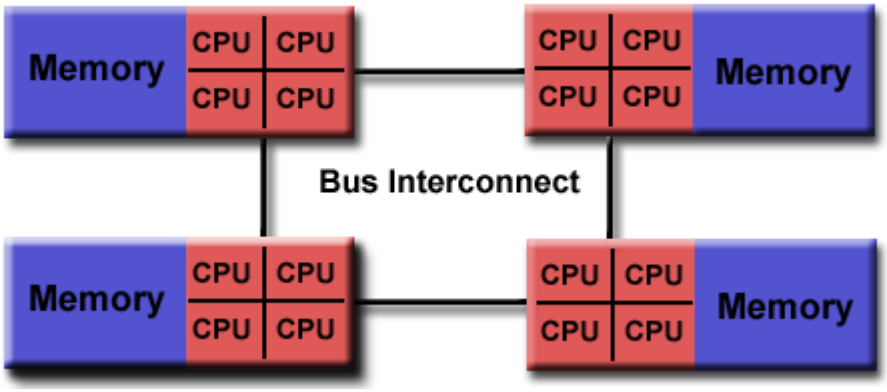


figure 1.10 **Shared Memory (NUMA)**

The COMA model : The COMA model is a special case of NUMA machine in which the distributed main memories are converted to caches. All caches form a global address space and there is no memory hierarchy at each processor node.

Advantages:

- Global address space provides a user-friendly programming perspective to memory
- Data sharing between tasks is both fast and uniform due to the proximity of memory to CPUs

Disadvantages:

- Primary disadvantage is the lack of scalability between memory and CPUs. Adding more CPUs can geometrically increases traffic on the shared memory-

CPU path, and for cache coherent systems, geometrically increase traffic associated with cache/memory management.

- Programmer responsibility for synchronization constructs that insure "correct" access of global memory.
- Expense: it becomes increasingly difficult and expensive to design and produce shared memory machines with ever increasing numbers of processors.

1.3.2 Distributed Memory

- Like shared memory systems, distributed memory systems vary widely but share a common characteristic. Distributed memory systems require a communication network to connect inter-processor memory.

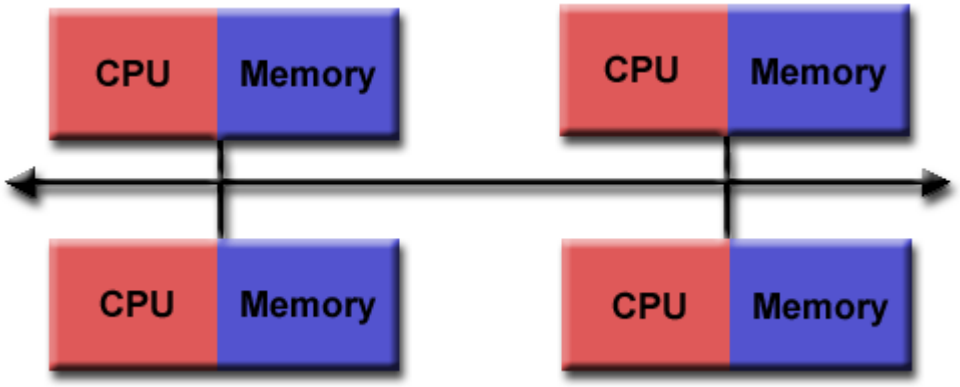


Figure 1.11 distributed memory systems

- Processors have their own local memory. Memory addresses in one processor do not map to another processor, so there is no concept of global address space across all processors.
- Because each processor has its own local memory, it operates independently. Changes it makes to its local memory have no effect on the memory of other processors. Hence, the concept of cache coherency does not apply.
- When a processor needs access to data in another processor, it is usually the task of the programmer to explicitly define how and when data is communicated. Synchronization between tasks is likewise the programmer's responsibility.

- Modern multicomputer use hardware routers to pass message. Based on the interconnection and routers and channel used the multicomputers are divided into generation
    - 1st generation : based on board technology using hypercube architecture and software controlled message switching.
    - 2nd Generation: implemented with mesh connected architecture, hardware message routing and software environment for medium distributed – grained computing.
    - 3rd Generation : fine grained multicomputer like MIT J-Machine.
- The network "fabric" used for data transfer varies widely, though it can be as simple as Ethernet.

Advantages:

- Memory is scalable with number of processors. Increase the number of processors and the size of memory increases proportionately.
- Each processor can rapidly access its own memory without interference and without the overhead incurred with trying to maintain cache coherency.
- Cost effectiveness: can use commodity, off-the-shelf processors and networking.

Disadvantages:

- The programmer is responsible for many of the details associated with data communication between processors.
- It may be difficult to map existing data structures, based on global memory, to this memory organization.
- Non-uniform memory access (NUMA) times

1.4 MULTIVECTOR AND SIMD COMPUTERS

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.

1.4.1Vector 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.

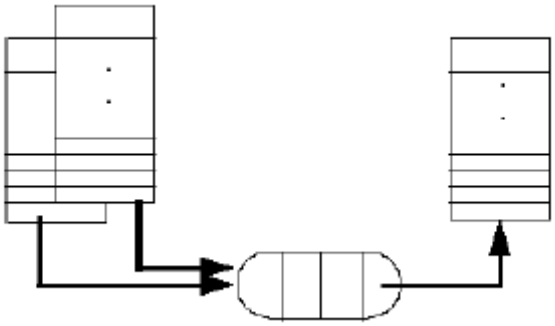


Figure 1.12 Vector Hardware

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.

1.4.2 SIMD Array Processors

The Synchronous parallel architectures coordinate Concurrent operations in lockstep through global clocks, central control units, or vector unit controllers. A synchronous array of parallel processors is called an array processor. These processors are composed of N identical processing elements (PES) under the supervision of a one control unit (CU) This Control unit is a computer with high speed registers,

local memory and arithmetic logic unit.. An array processor is basically a single instruction and multiple data (SIMD) computers. There are N data streams; one per processor, so different data can be used in each processor. The figure below show a typical SIMD or array processor

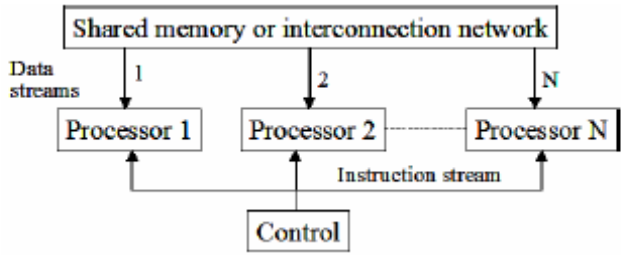


Figure 1.13 Configuration of SIMD Array Processor

These processors consist of a number of memory modules which can be either global or dedicated to each processor. Thus the main memory is the aggregate of the memory modules. These Processing elements and memory unit communicate with each other through an interconnection network. SIMD processors are especially designed for performing vector computations. SIMD has two basic architectural organizations

a. Array processor using random access memory

b. Associative processors using content addressable memory.

All N identical processors operate under the control of a single instruction stream issued by a central control unit. The popular examples of this type of SIMD configuration is ILLIAC IV, CM-2, MP-1. Each PEi is essentially an arithmetic logic unit (ALU) with attached working registers and local memory PEMi for the storage of distributed data. The CU also has its own main memory for the storage of program. The function of CU is to decode the instructions and determine where the decoded instruction should be executed. The PE perform same function (same instruction) *synchronously in a lock step fashion under command of CU. In order to maintain synchronous operations* a global

clock is used. Thus at each step i.e., when global clock pulse changes all processors execute the same instruction, each on a different data (single instruction multiple data). SIMD machines are particularly useful at in solving problems involved with vector calculations where one can easily exploit data parallelism. In such calculations the same set of instruction is applied to all subsets of data. Lets do addition to two vectors each having N element and there are N/2 processing elements in the SIMD. The same addition instruction is issued to all N/2 processors and all processor elements will execute the instructions simultaneously. It takes 2 steps to add two vectors as compared to N steps on a SISD machine. The distributed data can be loaded into PEMs from an external source via the system bus or via system broadcast mode using the control bus.

The array processor can be classified into two category depending how the memory units are organized. It can be

a. Dedicated memory organization
b. Global memory organization

A SIMD computer C is characterized by the following set of parameter

C= <N,F,I,M>

Where N= the number of PE in the system . For example the iliac –IV has N=64 , the BSP has N= 16.

F= a set of data routing function provided by the interconnection network

I= The set of machine instruction for scalar vector, data routing and network manipulation operations

M = The set of the masking scheme where each mask partitions the set of PEs into disjoint subsets of enabled PEs and disabled PEs.

1.5 PRAM AND VLSI MODELS

1.5.1 PRAM model (Parallel Random Access Machine):

PRAM Parallel random access machine; a theoretical model of parallel computation in which an arbitrary but finite number of processors can access any value in an arbitrarily large *shared memory* in a single time step. Processors may execute different instruction streams, but work *synchronously*. This model assumes a shared memory, multiprocessor machine as shown:

1. The machine size n can be arbitrarily large

2. The machine is synchronous at the instruction level. That is, each processor is executing it's own series of instructions, and the entire machine operates at a basic time step (cycle). Within each cycle, each processor executes exactly one operation or does nothing, i.e. it is *idle.* An instruction can be any random access machine instruction, such as: fetch some operands from memory, perform an ALU operation on the data, and store the result back in memory.

3. All processors implicitly synchronize on each cycle and the synchronization overhead is assumed to be zero. Communication is done through reading and writing of shared variables.

4. Memory access can be specified to be UMA, NUMA, EREW, CREW, or CRCW with a defined conflict policy.

The PRAM model can apply to SIMD class machines if all processors execute identical instructions on the same cycle, or to MIMD class machines if the processors are executing different instructions. Load imbalance is the only form of overhead in the PRAM model.

The four most important variations of the PRAM are:

- **EREW** - Exclusive read, exclusive write; any memory location may only be accessed once in any one step. Thus forbids more than one processor from reading or writing the same memory cell simultaneously.
- **CREW** - Concurrent read, exclusive write; any memory location may be read any number of times during a single step, but only written to once, with the write taking place after the reads.
- **ERCW** – This allows exclusive read or concurrent writes to the same memory location.
- **CRCW** - Concurrent read, concurrent write; any memory location may be written to or read from any number of times during a single step. A CRCW PRAM model must define some rule for resolving multiple writes, such as giving priority to the lowest-numbered processor or choosing amongst processors randomly. The PRAM is popular because it is theoretically tractable and because it gives

algorithm designers a common target. However, PRAMs cannot be emulated *optimally* on all *architectures*.

1.5.2 VLSI Model:

Parallel computers rely on the use of VLSI chips to fabricate the major components such as processor arrays memory arrays and large scale switching networks. The rapid advent of very large scale intergrated (VSLI) technology now computer architects are trying to implement parallel algorithms directly in hardware. An AT^2 model is an example for two dimension VLSI chips

1.6 Summary

Architecture has gone through evolutional, rather than revolutional change.

Sustaining features are those that are proven to improve performance. Starting with the von Neumann architecture (strictly sequential), architectures have evolved to include processing lookahead, parallelism, and pipelining. Also a variety of parallel architectures are discussed like SIMD, MIMD, Associative Processor, Array Processor, multicomputers, Mutiprocessor. The performance of system is measured as CPI, MIPS. It depends on the clock rate lets say t. If C is the total number of clock cycles needed to execute a given program, then total CPU time can be estimated as

$T = C * t = C / f$.

Other relationships are easily observed:

$CPI = C / Ic$

$T = Ic * CPI * t$

$T = Ic * CPI / f$

Processor speed is often measured in terms of millions of instructions per second, frequently called the MIPS rate of the processor. The multiprocessor architecture can be broadly classified as tightly coupled multiprocessor and loosely coupled multiprocessor. A tightly coupled Multiprocessor is also called a UMA, for uniform memory access, because each CPU can access memory data at the same (uniform) amount of time. This is the true multiprocessor. A loosely coupled Multiprocessor is called a NUMA. Each of its node computers can access their local memory data at one (relatively fast) speed, and

remote memory data at a much slower speed. PRAM and VSLI are the advance technologies that are used for designing the architecture.

1.7 Keywords

multiprocessor A computer in which processors can execute separate instruction streams, but have access to a single *address space*. Most multiprocessors are *shared memory* machines, constructed by connecting several processors to one or more memory banks through a *bus* or *switch*.

multicomputer A computer in which processors can execute separate instruction streams, have their own private memories and cannot directly access one another's memories. Most multicomputers are *disjoint memory* machines, constructed by joining *nodes* (each containing a microprocessor and some memory) via *links*.

M**IMD** Multiple Instruction, Multiple Data; a category of *Flynn's taxonomy* in which many instruction streams are concurrently applied to multiple data sets. A MIMD *architecture* is one in which *heterogeneous* processes may execute at different rates.

MIPS one Million Instructions Per Second. A performance rating usually referring to integer or non-floating point instructions

vector processor A computer designed to apply arithmetic operations to long vectors or arrays. Most vector processors rely heavily on *pipelining* to achieve high performance

pipelining Overlapping the execution of two or more operations

Program & network properties

- Objective
- Introduction
- Condition of parallelism
  - Data dependence and resource dependence
  - Hardware and software dependence
  - The role of compiler
- Program partitioning and scheduling
  - Grain size and latency
  - Grain packing and scheduling
- Program flow mechanism
- System interconnect architecture.
  - Network properties and routing
  - Static connection network
  - Dynamic connection network
- Summary
- Keywords

2.0 Objective

In this lesson we will study about fundamental properties of programs how parallelism can be introduced in program. We will study about the granularity, partitioning of programs , program flow mechanism and compilation support for parallelism. Interconnection architecture both static and dynamic type will be discussed.

2.1 Introduction

The advantage of multiprocessors lays when parallelism in the program is popularly exploited and implemented using multiple processors. Thus in order to implement the parallelism we should understand the various conditions of parallelism.

What are various bottlenecks in implementing parallelism? Thus for full implementation of parallelism there are three significant areas to be understood namely computation models for parallel computing, interprocessor communication in parallel architecture and system integration for incorporating parallel systems. Thus multiprocessor system poses a number of problems that are not encountered in sequential processing such as designing a parallel algorithm for the application, partitioning of the application into tasks, coordinating communication and synchronization, and scheduling of the tasks onto the machine.

2.2 Condition of parallelism

The ability to execute several program segments in parallel requires each segment to be independent of the other segments. We use a dependence graph to describe the relations. The nodes of a dependence graph correspond to the program statement (instructions), and directed edges with different labels are used to represent the ordered relations among the statements. The analysis of dependence graphs shows where opportunity exists for parallelization and vectorization.

2.2.1 Data and resource Dependence

Data dependence: The ordering relationship between statements is indicated by the data dependence. Five type of data dependence are defined below:

1. Flow dependence: A statement S2 is flow dependent on S1 if an execution path exists from s1 to S2 and if at least one output (variables assigned) of S1feeds in as input (operands to be used) to S2 also called RAW hazard and denoted as $S_1 \rightarrow S_2$

2. Antidependence: Statement S2 is antidependent on the statement S1 if S2 follows S1 in the program order and if the output of S2 overlaps the input to S1 also called RAW hazard and denoted as $S_1 \mapsto S_2$

3. Output dependence : two statements are output dependent if they produce (write) the same output variable. Also called WAW hazard and denoted as $S_1 \multimap S_2$

4. I/O dependence: Read and write are I/O statements. I/O dependence occurs not because the same variable is involved but because the same file referenced by both I/O statement.

5. Unknown dependence: The dependence relation between two statements cannot be determined in the following situations:

- The subscript of a variable is itself subscribed(indirect addressing)
- The subscript does not contain the loop index variable.
- A variable appears more than once with subscripts having different coefficients of the loop variable.
- The subscript is non linear in the loop index variable.

Parallel execution of program segments which do not have total data independence can produce non-deterministic results.

Consider the following fragment of any program:

S1 Load R1, A

S2 Add R2, R1

S3 Move R1, R3

S4 Store B, R1

• here the Forward dependency S1to S2, S3 to S4, S2 to S2

• Anti-dependency from S2to S3

• Output dependency S1 toS3

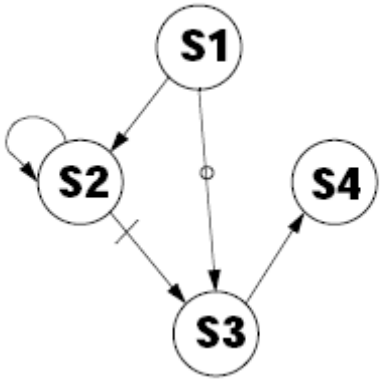


Figure 2.1 Dependence graph

Control Dependence: This refers to the situation where the order of the execution of statements cannot be determined before run time. For example all condition statement, where the flow of statement depends on the output. Different paths taken after a conditional branch may depend on the data hence we need to eliminate this data dependence among the instructions. This dependence also exists between operations

performed in successive iterations of looping procedure. Control dependence often prohibits parallelism from being exploited.

Control-independent example:

```
for (i=0;i<n;i++) {
a[i] = c[i];
if (a[i] < 0) a[i] = 1;
}
```

Control-dependent example:

```
for (i=1;i<n;i++) {
if (a[i-1] < 0) a[i] =
1; }
```

Control dependence also avoids parallelism to being exploited. Compilers are used to eliminate this control dependence and exploit the parallelism.

Resource dependence:

Data and control dependencies are based on the independence of the work to be done.

Resource independence is concerned with conflicts in using shared resources, such as registers, integer and floating point ALUs, etc. ALU conflicts are called ALU dependence. Memory (storage) conflicts are called storage dependence.

Bernstein's Conditions - 1

Bernstein's conditions are a set of conditions which must exist if two processes can execute in parallel.

Notation

Ii is the set of all input variables for a process *Pi* . *Ii* is also called the read set or domain of *Pi. Oi* is the set of all output variables for a process *Pi* .Oi is also called write set

If P*1* and P*2* can execute in parallel (which is written as P*1* || P*2*), then:

$$\mathbf{I}_1 \cap \mathbf{O}_2 = \varnothing$$

$$\mathbf{I}_2 \cap \mathbf{O}_1 = \varnothing$$

$$\mathbf{O}_1 \cap \mathbf{O}_2 = \varnothing$$

Bernstein's Conditions - 2

In terms of data dependencies, Bernstein's conditions imply that two processes can execute in parallel if they are flow-independent, antiindependent, and output-independent. The parallelism relation || is commutative (P*i* || P*j* implies P*j* || P*i*), but not transitive (P*i* || P*j* and P*j* || P*k* does not imply P*i* || P*k*) . Therefore, || is not an equivalence relation. Intersection of the input sets is allowed.

2.2.2 Hardware and software parallelism

Hardware parallelism is defined by machine architecture and hardware multiplicity i.e., functional parallelism times the processor parallelism .It can be characterized by the number of instructions that can be issued per machine cycle. If a processor issues *k* instructions per machine cycle, it is called a *k-issue* processor. Conventional processors are *one-issue* machines. This provide the user the information about **peak attainable performance**. Examples. Intel i960CA is a three-issue processor (arithmetic, memory access, branch). IBM RS -6000 is a four-issue processor (arithmetic, floating-point, memory access, branch).A machine with *n k*-issue processors should be able to handle a maximum of *nk* threads simultaneously.

Software Parallelism

Software parallelism is defined by the control and data dependence of programs, and is revealed in the program's flow graph i.e., it is defined by dependencies with in the code and is a function of algorithm, programming style, and compiler optimization.

2.2.3 The Role of Compilers

Compilers used to exploit hardware features to improve performance. Interaction between compiler and architecture design is a necessity in modern computer development. It is not necessarily the case that more software parallelism will improve performance in conventional scalar processors. The hardware and compiler should be designed at the same time.

2.3Program Partitioning & Scheduling

2.3.1 Grain size and latency

The size of the parts or pieces of a program that can be considered for parallel execution can vary. The sizes are roughly classified using the term "granule size," or simply "granularity." The simplest measure, for example, is the number of instructions in a

program part. Grain sizes are usually described as fine, medium or coarse, depending on the level of parallelism involved.

Latency

Latency is the time required for communication between different subsystems in a computer. Memory latency, for example, is the time required by a processor to access memory. Synchronization latency is the time required for two processes to synchronize their execution. Computational granularity and communication latency are closely related. Latency and grain size are interrelated and some general observation are

- As grain size decreases, potential parallelism increases, and overhead also increases.
- Overhead is the cost of parallelizing a task. The principle overhead is communication latency.
- As grain size is reduced, there are fewer operations between communication, and hence the impact of latency increases.
- Surface to volume: inter to intra-node comm.

Levels of Parallelism

Instruction Level Parallelism

This fine-grained, or smallest granularity level typically involves less than 20 instructions per grain. The number of candidates for parallel execution varies from 2 to thousands, with about five instructions or statements (on the average) being the average level of parallelism.

Advantages:

There are usually many candidates for parallel execution

Compilers can usually do a reasonable job of finding this parallelism

Loop-level Parallelism

Typical loop has less than 500 instructions. If a loop operation is independent between iterations, it can be handled by a pipeline, or by a SIMD machine. Most optimized program construct to execute on a parallel or vector machine. Some loops (e.g. recursive) are difficult to handle. Loop-level parallelism is still considered fine grain computation.

Procedure-level Parallelism

Medium-sized grain; usually less than 2000 instructions. Detection of parallelism is more difficult than with smaller grains; interprocedural dependence analysis is difficult and history-sensitive. Communication requirement less than instruction level SPMD (single procedure multiple data) is a special case Multitasking belongs to this level.

Subprogram-level Parallelism

Job step level; grain typically has thousands of instructions; medium- or coarse-grain level. Job steps can overlap across different jobs. Multiprograming conducted at this level No compilers available to exploit medium- or coarse-grain parallelism at present.

Job or Program-Level Parallelism

Corresponds to execution of essentially independent jobs or programs on a parallel computer. This is practical for a machine with a small number of powerful processors, but impractical for a machine with a large number of simple processors (since each processor would take too long to process a single job).

Communication Latency

Balancing granularity and latency can yield better performance. Various latencies attributed to machine architecture, technology, and communication patterns used. Latency imposes a limiting factor on machine scalability. Ex. Memory latency increases as memory capacity increases, limiting the amount of memory that can be used with a given tolerance for communication latency.

Interprocessor Communication Latency

- Needs to be minimized by system designer
- Affected by signal delays and communication patterns Ex. n communicating tasks may require n (n - 1)/2 communication links, and the complexity grows quadratically, effectively limiting the number of processors in the system.

Communication Patterns

- Determined by algorithms used and architectural support provided
- Patterns include permutations broadcast multicast conference
- Tradeoffs often exist between granularity of parallelism and communication demand.

2.3.2 Grain Packing and Scheduling

Two questions:

How can I partition a program into parallel “pieces” to yield the shortest execution time?
What is the optimal size of parallel grains?
There is an obvious tradeoff between the time spent scheduling and synchronizing parallel grains and the speedup obtained by parallel execution.
One approach to the problem is called “grain packing.”

Program Graphs and Packing

A program graph is similar to a dependence graph Nodes = { (n,s) }, where n = node name, s = size (larger s = larger grain size).
Edges = { (v,d) }, where v = variable being “communicated,” and d = communication delay.
Packing two (or more) nodes produces a node with a larger grain size and possibly more edges to other nodes. Packing is done to eliminate unnecessary communication delays or reduce overall scheduling overhead.

Scheduling

A schedule is a mapping of nodes to processors and start times such that communication delay requirements are observed, and no two nodes are executing on the same processor at the same time. Some general scheduling goals

- Schedule all fine-grain activities in a node to the same processor to minimize communication delays.
- Select grain sizes for packing to achieve better schedules for a particular parallel machine.

Node Duplication

Grain packing may potentially eliminate interprocessor communication, but it may not always produce a shorter schedule. By duplicating nodes (that is, executing some instructions on multiple processors), we may eliminate some interprocessor communication, and thus produce a shorter schedule.

Program partitioning and scheduling

Scheduling and allocation is a highly important issue since an inappropriate scheduling of tasks can fail to exploit the true potential of the system and can offset the gain from parallelization. In this paper we focus on the scheduling aspect. The objective of scheduling is to minimize the completion time of a parallel application by properly

allocating the tasks to the processors. In a broad sense, the scheduling problem exists in two forms: *static* and *dynamic*. In static scheduling, which is usually done at compile time, the characteristics of a parallel program (such as task processing times, communication, data dependencies, and synchronization requirements) are known before program execution

A parallel program, therefore, can be represented by a node- and edge-weighted directed acyclic graph (DAG), in which the node weights represent task processing times and the edge weights represent data dependencies as well as the communication times between tasks. In dynamic scheduling only, a few assumptions about the parallel program can be made before execution, and thus, scheduling decisions have to be made on-the-fly. The goal of a dynamic scheduling algorithm as such includes not only the minimization of the program completion time but also the minimization of the scheduling overhead which constitutes a significant portion of the cost paid for running the scheduler. In general dynamic scheduling is an NP hard problem.

2.4 Program flow mechanism

Conventional machines used control flow mechanism in which order of program execution explicitly stated in user programs. Dataflow machines which instructions can be executed by determining operand availability.

Reduction machines trigger an instruction's execution based on the demand for its results.

Control Flow vs. Data Flow In Control flow computers the next instruction is executed when the last instruction as stored in the program has been executed where as in Data flow computers an instruction executed when the data (operands) required for executing that instruction is available

Control flow machines used shared memory for instructions and data. Since variables are updated by many instructions, there may be side effects on other instructions. These side effects frequently prevent parallel processing. Single processor systems are inherently sequential.

Instructions in dataflow machines are unordered and can be executed as soon as their operands are available; data is held in the instructions themselves. *Data tokens* are passed from an instruction to its dependents to trigger execution.

Data Flow Features

No need for shared memory program counter control sequencer Special mechanisms are required to detect data availability match data tokens with instructions needing them enable chain reaction of asynchronous instruction execution

A Dataflow Architecture – 1 The Arvind machine (MIT) has N PEs and an N -by –N interconnection network. Each PE has a token-matching mechanism that dispatches only instructions with data tokens available. Each datum is tagged with

- address of instruction to which it belongs
- context in which the instruction is being executed

Tagged tokens enter PE through local path (pipelined), and can also be communicated to other PEs through the routing network. Instruction address(es) effectively replace the program counter in a control flow machine. Context identifier effectively replaces the frame base register in a control flow machine. Since the dataflow machine matches the data tags from one instruction with successors, synchronized instruction execution is implicit.

An I-structure in each PE is provided to eliminate excessive copying of data structures. Each word of the I-structure has a two-bit tag indicating whether the value is empty, full, or has pending read requests.

This is a retreat from the pure dataflow approach. Special compiler technology needed for dataflow machines.

Demand-Driven Mechanisms

Data-driven machines select instructions for execution based on the availability of their operands; this is essentially a bottom-up approach.

Demand-driven machines take a top-down approach, attempting to execute the instruction (a *demander*) that yields the final result. This triggers the execution of instructions that yield its operands, and so forth. The demand-driven approach matches naturally with functional programming languages (e.g. LISP and SCHEME).

Pattern driven computers : An instruction is executed when we obtain a particular data patterns as output. There are two types of pattern driven computers

String-reduction model: each demander gets a separate copy of the expression string to evaluate each reduction step has an operator and embedded reference to demand the corresponding operands each operator is suspended while arguments are evaluated

Graph-reduction model: expression graph reduced by evaluation of branches or subgraphs, possibly in parallel, with demanders given pointers to results of reductions. based on sharing of pointers to arguments; traversal and reversal of pointers continues until constant arguments are encountered.

2.5 System interconnect architecture.

Various types of interconnection networks have been suggested for SIMD computers. These are basically classified have been classified on network topologies into two categories namely

Static Networks

Dynamic Networks

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.

The topological structure of an SIMD array processor is mainly characterized by the data routing network used in the interconnecting the processing elements. To execute the communication the routing function f is executed and via the interconnection network the PEi copies the content of its Ri register into the Rf(i) register of PEf(i). The f(i) the processor identified by the mapping function f. The data routing operation occurs in all active PEs simultaneously.

2.5.1 Network properties and routing

The goals of an interconnection network are to provide low-latency high data transfer rate wide communication bandwidth. Analysis includes latency bisection bandwidth data-routing functions scalability of parallel architecture

These Network usually represented by a graph with a finite number of nodes linked by directed or undirected edges.

Number of nodes in graph = network size .

Number of edges (links or channels) incident on a node = node degree d (also note in and out degrees when edges are directed).

Node degree reflects number of I/O ports associated with a node, and should ideally be small and constant.

Network is symmetric if the topology is the same looking from any node; these are easier to implement or to program.

Diameter : The maximum distance between any two processors in the network or in other words we can say **Diameter**, is the maximum number of (routing) processors through which a message must pass on its way from source to reach destination. Thus diameter measures the maximum delay for transmitting a message from one processor to another as it determines communication time hence smaller the diameter better will be the network topology.

Connectivity: How many paths are possible between any two processors i.e., the multiplicity of paths between two processors. Higher connectivity is desirable as it minimizes contention.

Arch connectivity of the network: the minimum number of arcs that must be removed for the network to break it into two disconnected networks. The arch connectivity of various network are as follows

- 1 for linear arrays and binary trees
- 2 for rings and 2-d meshes
- 4 for 2-d torus
- d for d-dimensional hypercubes

Larger the arch connectivity lesser the conjunctions and better will be network topology.

Channel width : The channel width is the number of bits that can communicated simultaneously by a interconnection bus connecting two processors

Bisection Width and Bandwidth: In order divide the network into equal halves we require the remove some communication links. The minimum number of such communication links that have to be removed are called the Bisection Width. **Bisection width basically provide us the information about** the largest number of messages which can be sent simultaneously (without needing to use the same wire or routing processor at the same time and so delaying one another), no matter which processors are sending to which other processors. Thus larger the bisection width is the better the network topology is considered. Bisection Bandwidth is the minimum volume of communication allowed

between two halves of the network with equal numbers of processors This is important for the networks with weighted arcs where the weights correspond to the *link width i.e.,* (how much data it can transfer). The Larger bisection width the better network topology is considered.

Cost the cost of networking can be estimated on variety of criteria where we consider the the number of communication links or wires used to design the network as the basis of cost estimation. Smaller the better the cost

Data Routing Functions: A data routing network is used for inter –PE data exchange. It can be static as in case of hypercube routing network or dynamic such as multistage network. Various type of data routing functions are Shifting, Rotating, Permutation (one to one), Broadcast (one to all), Multicast (many to many), Personalized broadcast (one to many), Shuffle, Exchange Etc.

Permutations

Given n objects, there are n ! ways in which they can be reordered (one of which is no reordering). A permutation can be specified by giving the rule for reordering a group of objects. Permutations can be implemented using crossbar switches, multistage networks, shifting, and broadcast operations. The time required to perform permutations of the connections between nodes often dominates the network performance when n is large.

Perfect Shuffle and Exchange

Stone suggested the special permutation that entries according to the mapping of the k-bit binary number a b … k to b c … k a (that is, shifting 1 bit to the left and wrapping it around to the least significant bit position). The inverse perfect shuffle reverses the effect of the perfect shuffle.

Hypercube Routing Functions

If the vertices of a n-dimensional cube are labeled with n-bit numbers so that only one bit differs between each pair of adjacent vertices, then n routing functions are defined by the bits in the node (vertex) address. For example, with a 3-dimensional cube, we can easily identify routing functions that exchange data between nodes with addresses that differ in the least significant, most significant, or middle bit.

Factors Affecting Performance

Functionality – how the network supports data routing, interrupt handling, synchronization, request/message combining, and coherence

Network latency – worst-case time for a unit message to be transferred

Bandwidth – maximum data rate

Hardware complexity – implementation costs for wire, logic, switches, connectors, etc.

Scalability – how easily does the scheme adapt to an increasing number of processors, memories, etc.?

2.5.2 Static connection Networks

In static network the interconnection network is fixed and permanent interconnection path between two processing elements and data communication has to follow a fixed route to reach the destination processing element. Thus it Consist of a number of point-to-point links. Topologies in the static networks can be classified according to the dimension required for layout i.e., it can be 1-D, 2-D, 3-D or hypercube.

One dimensional topologies include Linear array as shown in figure 2.2 (a) used in some pipeline architecture.

Various 2-D topologies are

- The ring (figure 2.2(b))
- Star (figure 2.2(c))
- Tree (figure 2.2(d))
- Mesh (figure 2.2(e))
- Systolic Array (figure 2.2(f))

3-D topologies include

- Completely connected chordal ring (figure 2.2(g))
- Chordal ring (figure 2.2(h))
- 3 cube (figure 2.2(i))

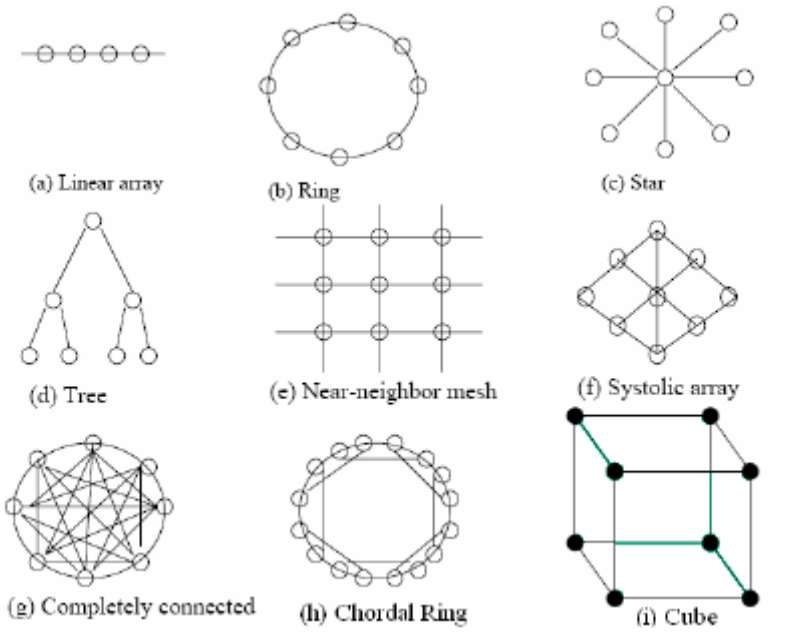


Figure 2.2 Static interconnection network topologies.

Torus architecture is also one of popular network topology it is extension of the mesh by having wraparound connections Figure below is a 2D Torus This architecture of torus is a symmetric topology unlike mesh which is not. The wraparound connections reduce the torus diameter and at the same time restore the symmetry. It can be

- o 1-D torus
- 2-D torus
- 3-D torus

The torus topology is used in Cray T3E

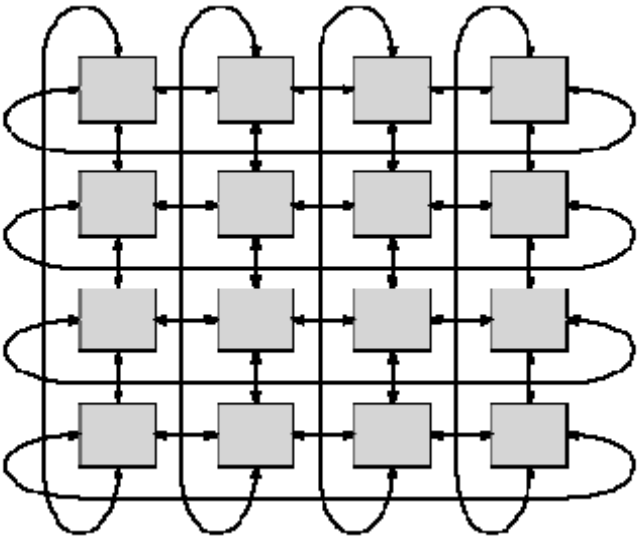


Figure 2.3 Torus technology

We can have further higher dimension circuits for example 3-cube connected cycle. A D-dimension W-wide hypercube contains W nodes in each dimension and there is a connection to a node in each dimension. The mesh and the cube architecture are actually 2-D and 3-D hypercube respectively. The below figure we have hypercube with dimension 4.

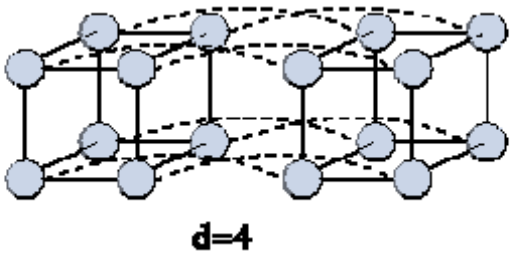


Figure 2.4 4-D hypercube.

2.5.3 Dynamic connection Networks

The dynamic networks are those networks where the route through which data move from one PE to another is established at the time communication has to be performed. Usually all processing elements are equidistant and an interconnection path is established when two processing element want to communicate by use of switches. Such systems are more difficult to expand as compared to static network. Examples: Bus-based, Crossbar, Multistage Networks. Here the Routing is done by comparing the bit-level representation

of source and destination addresses. If there is a match goes to next stage via pass-through else in case of it mismatch goes via cross-over using the switch.

There are two classes of dynamic networks namely

- single stage network
- multi stage

2.5.3.1 Single Stage Networks

A single stage switching network with N input selectors (IS) and N output selectors (OS). Here at each network stage there is a 1- to-D demultiplexer corresponding to each IS such that 1<D<N and each OS is an M-to-1 multiplexer such that 1<M<=N. Cross bar network is a single stage network with D=M=N. In order to establish a desired connecting path different path control signals will be applied to all IS and OS selectors. The single stage network is also called as recirculating network as in this network connection the single data items may have to recirculate several time through the single stage before reaching their final destinations. The number of recirculation depends on the connectivity in the single stage network. In general higher the hardware connectivity the lesser is the number of recirculation. In cross bar network only one circulation is needed to establish the connection path. The cost of completed connected cross bar network is O(N2) which is very high as compared to other most recirculating networks which have cost O(N log N) or lower hence are more cost effective for large value of N.

2.5.3.2 Multistage Networks

Many stages of interconnected switches form a multistage SIMD network. It is basicaaly consist of three characteristic features

- The switch box,
- The network topology
- The control structure

Many stages of interconnected switches form a multistage SIMD networks. Eachbox is essentially an interchange device with two inputs and two outputs. The four possible states of a switch box are which are shown in figure 3.6

- Straight
- Exchange
- Upper Broadcast

- Lower broadcast.

A two function switch can assume only two possible state namely state or exchange states. However a four function switch box can be any of four possible states. A multistage network is capable of connecting any input terminal to any output terminal. Multi-stage networks are basically constructed by 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.

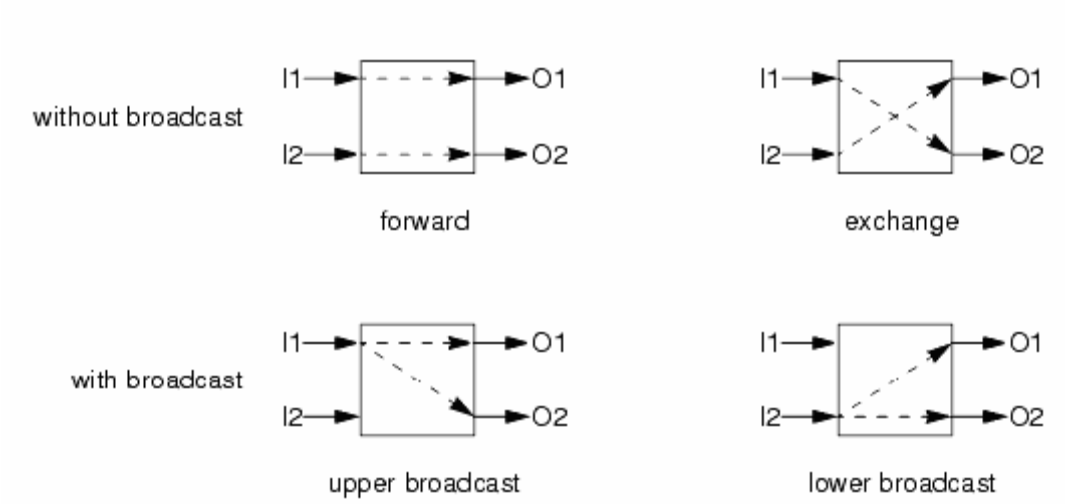


Figure 2.5 A two-by-two switching box and its four interconnection states

A multistage network is capable of connecting an arbitrary input terminal to an arbitrary output terminal. Generally it is consist of n stages where $N = 2^n$ is the number of input and output lines. And each stage use N/2 switch boxes. The interconnection patterns from one stage to another stage is determined by network topology. Each stage is connected to the next stage by at least N paths. The total wait time is proportional to the number stages i.e., n and the total cost depends on the total number of switches used and that is $N\log_2 N$. The control structure can be individual stage control i.e., the same control signal is used to set all switch boxes in the same stages thus we need n control signal. The second control structure is individual box control where a separate control signal is used to set the state of each switch box. This provide flexibility at the same time require n2/2 control signal which increases the complexity of the control circuit. In between path is use of partial stage control.

Examples of Multistage Networks

Banyan

Baseline

Cube

Delta

Flip

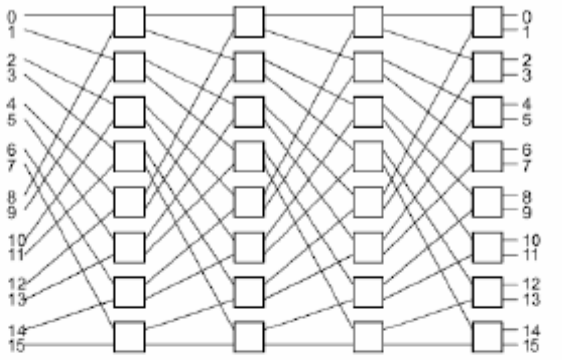
Indirect cube

Omega

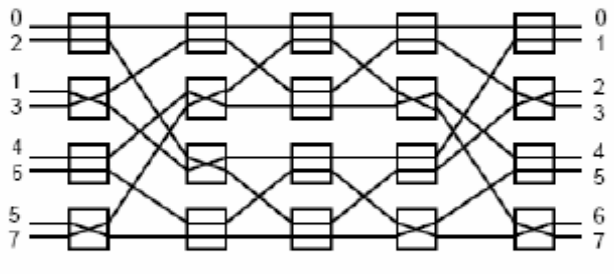
Multistage network can be of two types

- One side networks : also called full switch having input output port on the same side
- Two sided multistage network : which have an input side and an output side. It can be further divided into three class
    - Blocking: In Blocking networks, simultaneous connections of more than one terminal pair may result conflicts in the use of network communication links. Examples of blocking network are the Data Manipulator, Flip, N cube, omega, baseline. All multistage networks that are based on shuffle-exchange elements, are based on the concept of blocking network because not all possible here to make the input-output connections at the same time as one path might block another. The figure 2.6 (a) show an omega network.
    - Rearrangeable : 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. An example of this network topology is Benes Network (see figure 2.6 (b) showing a 8** Benes network)which support synchronous data permutation and a synchronous interprocessor communication.
    - Non blocking : A non –blocking network is the network which can handle all possible connections without blocking. There two possible cases first one is the Clos network (see figure 2.6(c)) where a one to one connection

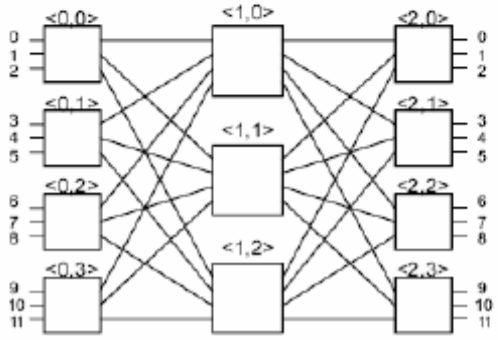
is made between input and output. Another case of one to many connections can be obtained by using crossbars instead of the shuffle-exchange elements. The cross bar switch network can connect every input port to a free output port without blocking.



(a) Omega Network



(b) Benes Network



(c) Clos Network

Figure 2.6 Several Multistage Interconnection Networks

Mesh-Connected Illiac Networks

A single stage recirculating network has been implemented in the ILLiac –IV array with N= 64 PEs. Here in mesh network nodes are arranged as a q-dimensional lattice. The

neighboring nodes are only allowed to communicate the data in one step i.e., each PEi is allowed to send the data to any one of PE(i+1) , PE (i-1), Pe(i+r) and PE(i-r) where r= square root N(in case of Iliac r=8). In a *periodic mesh*, nodes on the edge of the mesh have wrap-around connections to nodes on the other side this is also 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

Thus we observe the output of IS k is connected to inputs of OSj where j = k-1,K+1,k-r,k+r as shown in figure below.

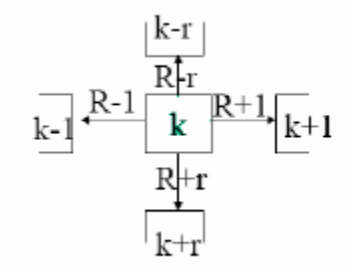


Figure2.7 routing function of mesh Topology

Similarly the OSj gets input from ISk for K= j-1, j+1,j-r,j+r. The topology is formerly described by the four routing functions:

• R+1(i)= (i+1) mod N => (0,1,2…,14,15)

• R-1(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)

The figure given below show how each PEi is connected to its four nearest neighbors in the mesh network. It is same as that used for IILiac –IV except that w had reduced it for N=16 and r=4. The index are calculated as module N.

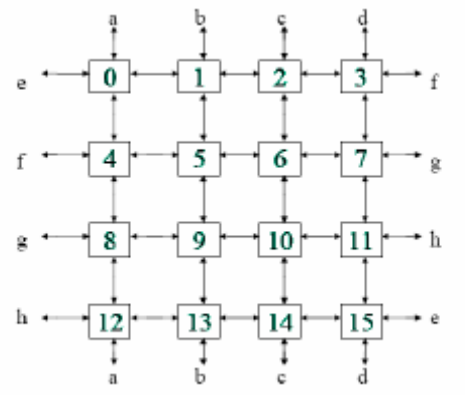


Figure 2.8 Mesh Connections

Thus the permutation cycle according to routing function will be as follows: 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).

Similarly we have vertical permutation also and now by combining the two permutation each with four cycles of order four each the shift distance for example for a network of N = 16 and r = square root(16) = 4, is given as follows:

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)

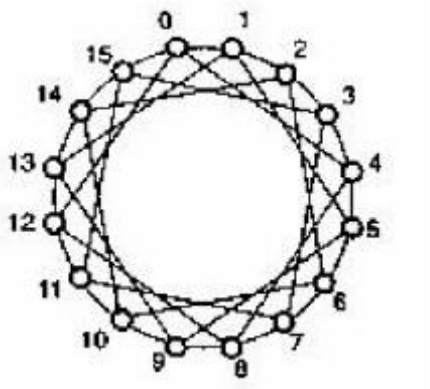


Figure 4.9 Mesh Redrawn

Each PEi is directly connected to its four neighbors in the mesh network. The graph shows that in one step a PE can reach to four PEs, seven PEs in two step and eleven PEs in three steps. In general it takes I steps (recirculations) to route data from PEi to another PEj for a network of size N where I is upper –bound given by

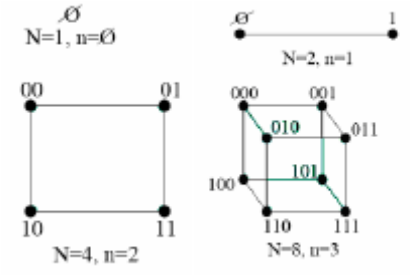
I<= square root(N) -1

Thus in above example for N=16 it will require at most 3 steps to route data from one PE to another PE and for Illiac –IV network with 64 PE need maximum of 7 steps for routing data from one PE to Another.

Cube Interconnection Networks

The cube network can be implemented as either a recirculating network or as a multistage network for SIMD machine. It can be 1-D i.e., a single line with two pE each at end of a line, a square with four PEs at the corner in case of 2-D, a cube for 3-D and hypercube in 4-D. in case of n-dimension hypercube each processor connects to 2n neighbors. This can be also visualized as the unit (hyper) cube embedded in d-dimensional Euclidean space, with one corner at 0 and lying in the positive orthant. The processors can be thought of as lying at the corners of the cube, with their (x1,x2,...,xd) coordinates identical to their processor numbers, and connected to their nearest neighbors on the cube. The popular examples where cube topology is used are : iPSC, nCUBE, SGI O2K.

Vertical lines connect vertices (PEs) whose address differ in the most significant bit position. Vertices at both ends of the diagonal lines differ in the middle bit position. Horizontal lines differ in the least significant bit position. The unit – cube concept can be extended to an n- dimensional unit space called an n cube with n bits per vertex. A cube network for an SIMD machine with N PEs corresponds to an n cube where $n = \log_2 N$. We use binary sequence to represent the vertex (PE) address of the cube. Two processors are neighbors if and only if their binary address differs only in one digit place



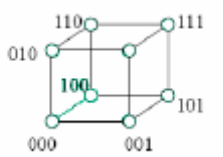
For an n-dimensional cube network of N PEs is specified by the following n routing functions

Ci (An-1 …. A1 A0)= An-1…Ai+1 A'i Ai-1……A0 for i =0,1,2,…,n-1

A n- dimension cube each PE located at the corner is directly connected to n neighbors. The addresses of neighboring PE differ in exactly one bit position. Pease's binary n cube the flip flop network used in staran and programmable switching network proposed for Phoenix are examples of cube networks.

In a recirculating cube network each ISa for 0<=A+< N-1 is connected to n OSs whose addresses are An-1…Ai+1 A'i Ai-1……A0 . When the PE addresses are considered as the corners of an m-dimensional cube this network connects each PE to its m neighbors. The interconnections of the PEs corresponding to the three routing function C0, C1 and C2 are shown separately in below figure.

• Examples



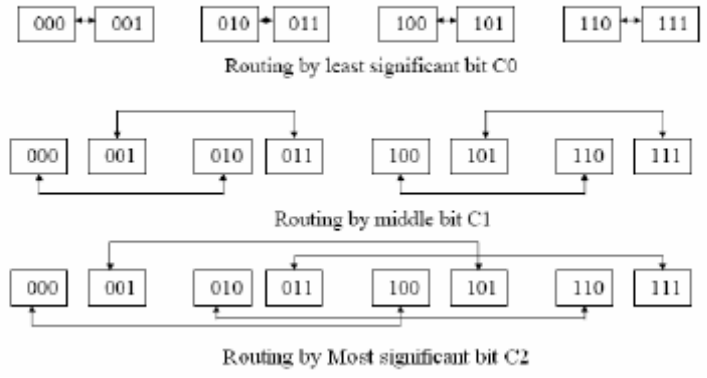


Figure 2.10 The recirculating Network

It takes n<= $\log_2 N$ steps to rotate data from any PE to another.

Example: N=8 => n=3

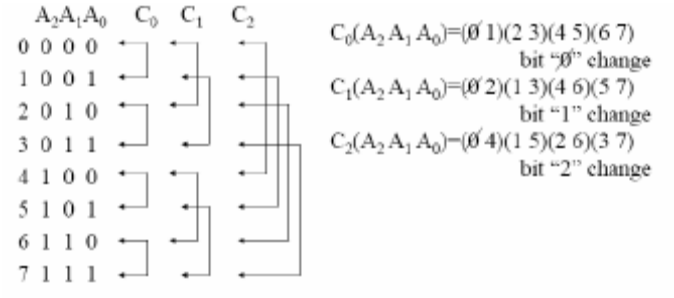


Figure 2.11 Possible routing in multistage Cube network for N = 8

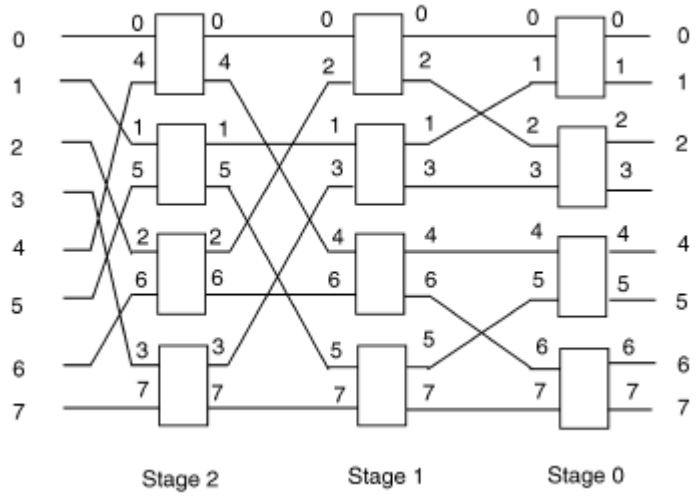


Figure 2.12 **A multistage Cube network for N = 8**

The same set of cube routing functions i.e., C0,C1, C2 can also be implemented by three stage network. Two functions switch box is used which can provide either straight and exchange routing is used for constructing multistage cube networks. The stages are numbered as 0 at input end and increased to n-1 at the output stage i.e., the stage I implements the Ci routing function or we can say at ith stage connect the input line to the output line that differ from it only at the ith bit position.

This connection was used in the early series of Intel Hypercubes, and in the CM-2.

Suppose there are 8 process ring elements so 3 bits are required for there address. and that processor 000 is the root. The children of the root are gotten by toggling the first address bit, and so are 000 and 100 (so 000 doubles as root and left child). The children

of the children are gotten by toggling the next address bit, and so are 000, 010, 100 and 110. Note that each node also plays the role of the left child. Finally, the leaves are gotten by toggling the third bit. Having one child identified with the parent causes no problems as long as algorithms use just one row of the tree at a time. Here is a picture.

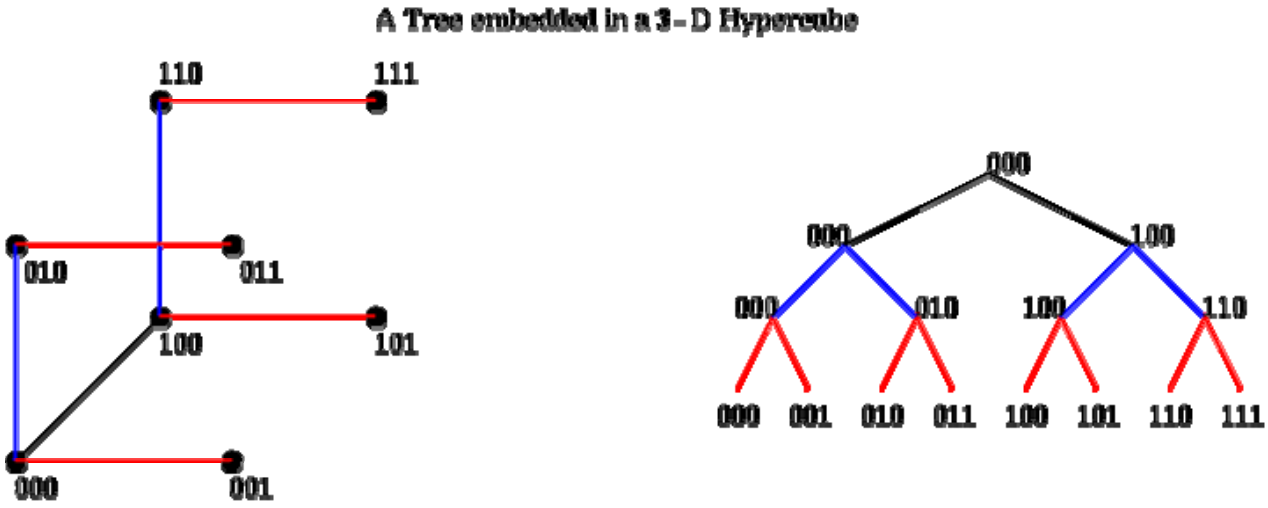


Figure 2.13 A tree embedded in 3-D hypercube

Shuffle-Exchange Omega Networks

A shuffle-exchange network consists of $n=2^k$ nodes and it is based on two routing functions shuffle (S) and exchange (E). Let A= An-1…A1A0be the address of a PE than a shuffle function is given by:

S(A)=S(An-1…A1A0)=A.n-2…A1A0An-1, 0<A<1

The cyclic shifting of the bits in A to the left for one bit osition is performed by the S function. Which is effectively like shuffling the bottom half of a card deck into the top half as shown in figure below.

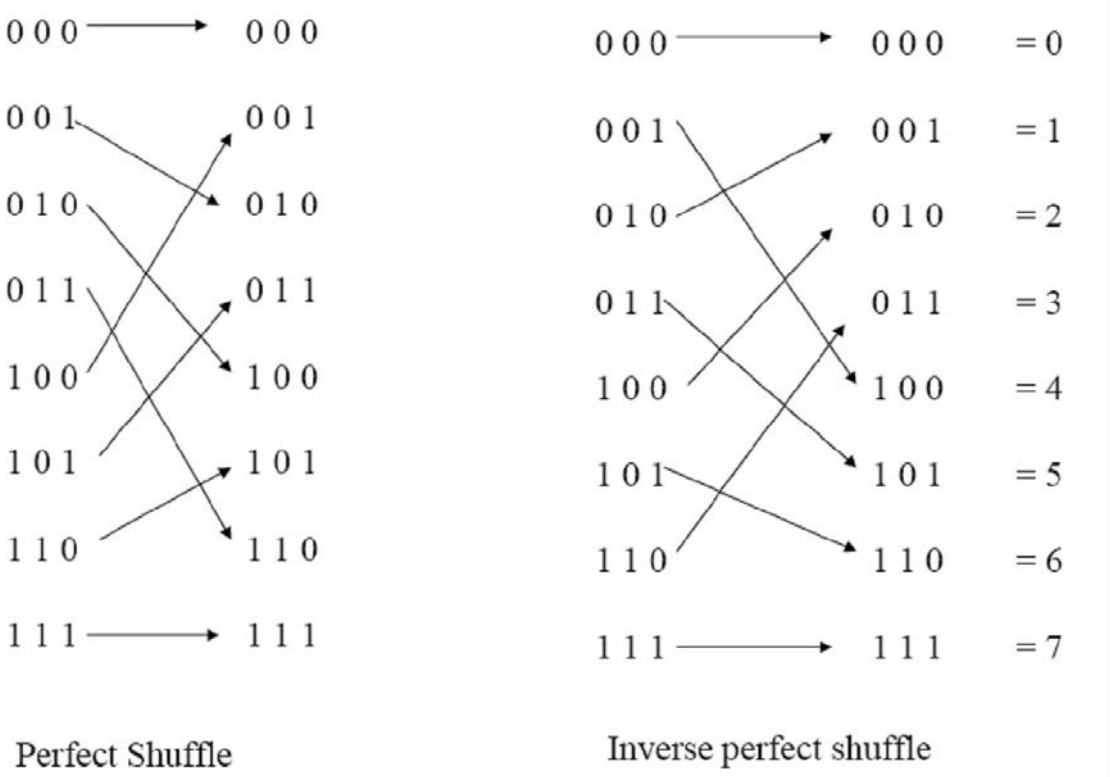


Figure 2.14 Perfect shuffle and inverse perfect shuffle

There are two type of shuffle the perfect shuffle cuts the deck into two halves from the centre and intermix them evenly. *Perfect shuffle provide the routing* connections of node i with node 2i mod(n-1), except for node n-1 which is connected to itself. The inverse perfect shuffle does the opposite to restore the original order it is denoted as exchange routing function E and is defined as :

E(An-1…A1A0)= (An-1…A1A0')

This obtained by complementing the least significant digit means data exchange between two PEs with adjacent addresses. The E(A) is same as the cube routing function as described earlier. *Exchange routing function* connects nodes whose numbers differ in their lowest bit.

The shuffle exchange function can be implemented as either a recirculating network or multistage network. The implementation of shuffle and exchange network through recirculating network is shown below. Use of shuffle and exchange topology for parallel processing was proposed by Stone. It is used for solving many parallel algorithms efficiently. The example where it is used include FFT (fast Fourier transform), sorting, matrix transposition , polynomial evaluations etc.

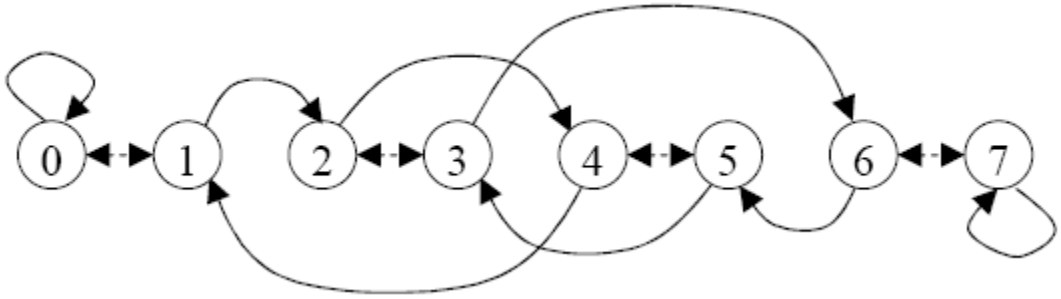
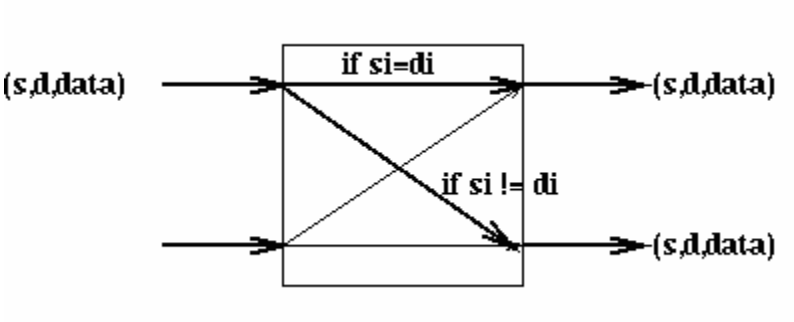


Figure2.15 shuffle and exchange recirculating network for N=8

The shuffle –exchange function have been implemented as multistage Omega network by LAwrie. An N by N omega network, consists of n identical stages. Between two adjacent column there is a perfect shuffle interconnection. Thus after each stage there is a N/2 four-function interchange boxes under independent box control. The four functions are namely straight exchange upper broadcast and lower broadcast. The shuffle connects output P n-l...Pl P0 of stage i to input P n-2...PlP0Pn-l 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.



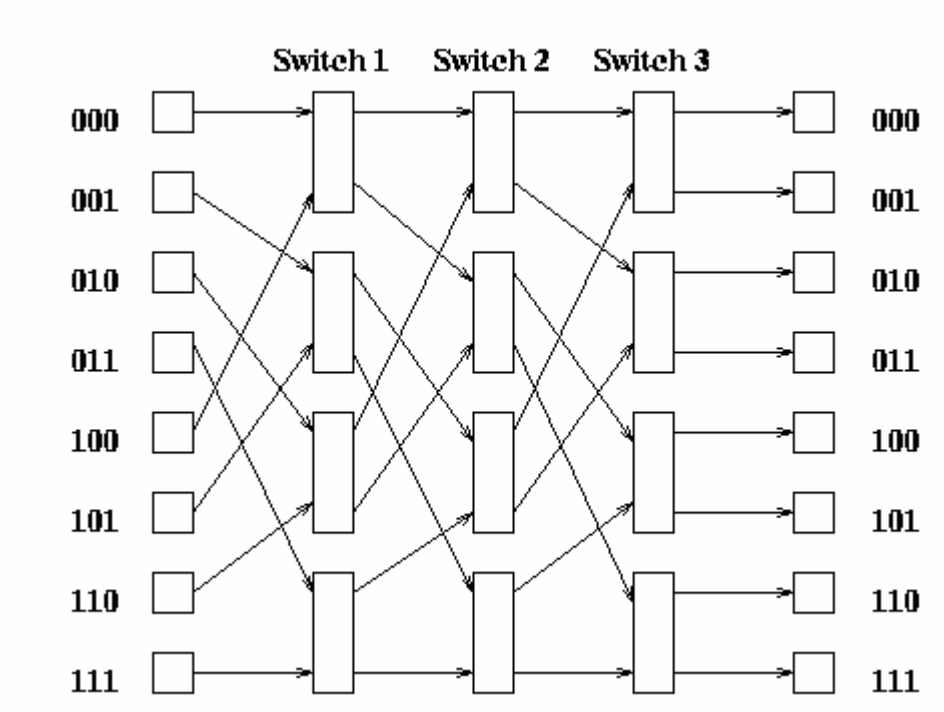


Figure 2.16

The diameter is m=log_2 p, since all message must traverse m stages. The bisection width is p. This network was used in the IBM RP3, BBN Butterfly, and NYU Ultracomputer. If we compare the omega network with cube network we find Omega network can perform one to many connections while n-cube cannot. However as far as bijections connections n-cube and Omega network they perform more or less same.

2.6 Summary

Fine-grain exploited at instruction or loop levels, assisted by the compiler.

Medium-grain (task or job step) requires programmer and compiler support.

Coarse-grain relies heavily on effective OS support.

Shared-variable communication used at fine- and medium grain levels.

Message passing can be used for medium- and coarse grain communication, but fine - grain really need better technique because of heavier communication requirements.

Control flow machines give complete control, but are less efficient than other approaches.

Data flow (eager evaluation) machines have high potential for parallelism and throughput and freedom from side effects, but have high control overhead, lose time waiting for unneeded arguments, and difficulty in manipulating data structures. Reduction (lazy

evaluation) machines have high parallelism potential, easy manipulation of data structures, and only execute required instructions. But they do not share objects with changing local state, and do require time to propagate tokens

Summary of properties of various static network

Network	Diameter	Bisection Width	Arc Connectivity	Cost (No. of links)
Completely-connected	1	$p^2/4$	$p-1$	$p(p-1)/2$
Star	2	1	1	$p-1$
Complete binary tree	$2\log((p+1)/2)$	1	1	$p-1$
Linear array	$p-1$	1	1	$p-1$
2-D mesh, no wraparound	$2(\sqrt{p}-1)$	$\sqrt{p}$	2	$2(p-\sqrt{p})$
2-D wraparound mesh	$2\lfloor\sqrt{p}/2\rfloor$	$2\sqrt{p}$	4	$2p$
Hypercube	$\log p$	$p/2$	$\log p$	$(p\log p)/2$
Wraparound k-ary d-cube	$d\lfloor k/2\rfloor$	$2k^{d-1}$	$2d$	dp

Summary of properties of various dynamic networks

Network Characteristics	Bus System	Multistage Network	Crossbar Switch
Minimum Latency for unit data transfer	Constant	$O(\log_k n)$	Constant
Bandwidth per processor	O(w/n) to O(w)	O(w) to O(nw)	O(w) to O(nw)
Wiring Complexity	O(w)	$O(nw \log_k n)$	$O(n^2w)$
Switching complexity	O(n)	$O(n \log_k n)$	$O(n^2)$
Connectivity and routing capability	Only one to one at a time	Some permutations and broadcast , if network unblocked	All permutations one at a time.

Metrics of dynamic connected nework

Network	Diameter	Bisection Width	Connectivity	Cost (# of links)
Crossbar	1	p	1	p^2
Omega Network	$\log p$	$p/2$	2	$p\log p$
Dynamic Tree	$2\log p$	1	2	$p-1$

2.7 Keywords

Dependence graph : A directed graph whose nodes represent calculations and whose edges represent dependencies among those calculations. If the calculation represented by

node k depends on the calculations represented by nodes i and j, then the dependence graph contains the edges i-k and j-k.

data dependency : a situation existing between two statements if one statement can store into a location that is later accessed by the other statement

granularity The size of operations done by a process between communications events. A fine grained process may perform only a few arithmetic operations between processing one message and the next, whereas a coarse grained process may perform millions

control-flow computers refers to an *architecture* with one or more program counters that determine the order in which instructions are executed.

dataflow A model of parallel computing in which programs are represented as *dependence graphs* and each operation is automatically *blocked* until the values on which it depends are available. The parallel functional and parallel logic programming models are very similar to the dataflow model.

network A physical communication medium. A network may consist of one or more *buses*, a *switch*, or the *links* joining processors in a *multicomputer*.

Static networks: point-to-point direct connections that will not change during program execution

Dynamic networks: switched channels dynamically configured to match user program communication demands include buses, crossbar switches, and multistage networks

routing The act of moving a message from its source to its destination. A routing technique is a way of handling the message as it passes through individual nodes.

Diameter D of a network is the maximum shortest path between any two nodes, measured by the number of links traversed; this should be as small as possible (from a communication point of view).

Channel bisection width b = minimum number of edges cut to split a network into two parts each having the same number of nodes. Since each channel has w bit wires, the wire bisection width B = bw. Bisection width provides good indication of maximum communication bandwidth along the bisection of a network, and all other cross sections should be bounded by the bisection width.

Wire (or channel) length = length (e.g. weight) of edges between nodes.