INTRODUCTION TO PARALLEL PROGRAMMING

CHAPTER 1: HARDWARE DEVELOPMENTS

1.1 History

The notion of a mechanical calculator is due to Charles Babbage. In 1821 he proposed his *Difference Engine* consisting of a set of linked adding mechanisms. Its primary purpose was to manipulate polynomials (by computing finite differences). He built a prototype which could compute second order differences. This machine could compute values of quadratic polynomials. Although he designed one capable of dealing with a sixth degree polynomial, he never built one. His *Analytical Engine* is the forerunner of the architecture of modern computers. It had a storage unit, arithmetic "mill" or unit and punched card input/output. The building technology was based on gears and linkages. According to his estimation, an addition could be done in a second, a multiplication in a minute. Babbage died in 1871. His designs were never implemented since his ideas were far too advanced for his time.

First Generation Machines (1945-58)

These machines used thermionic valves. Two prototypes were built at the University of Pennsylvania (EDVAC and ENIAC). These were huge machines, 80 feet long, 8 feet high with 17,000 vacuum tubes. A major characteristic of a value is known as the *gate delay time*(GDT). The GDT is the amount of time taken by a signal to travel from the input of one logic gate to the input of the next. The typical GDT for such a machine was a microsecond (10^{-6} seconds). The major disadvantages were that they required extraordinary amount of room, dissipation of large amount of heat, and frequent break down! A typical commercial machine is the IBM 704.

Second Generation Machines (1959-65)

In the second generation machines, the vacuum tube gave way to the *transistor*. The transistor was developed by American Physicists Bardeen, Brattain and Shockley of AT&T Bell Labs. They shared the Nobel prize in 1956 for their invention. A transistor is much smaller than a vacuum tube and so can house more active components. They were very reliable, produced very little heat. Computers based on the transistor were very compact. The GDT for such machines had been reduced to 0.3 microseconds. A typical machine of this generation is IBM 7090. The minicomputer (DEC) also belongs to this generation.

Third Generation Machines (1965-1975)

A new component technology was born in 1965 - solid state integrated circuits (invented by Jack Kelby at Texas Instruments). An IC is the size of an ice cube that contains hundreds of miniature transistors). The chief characteristic of these circuits is the silicon chip where several components were combined in a single wafer. In the beginning there were only few gates were placed on such a chip with a GDT of 10 nanoseconds (10^{-9} seconds). By 1975 this was improved to 1 nanosecond by increasing the number of gates. Typical machines of this generation are the hugely successful IBM 360 series, Burroughs 6500 and the UNIVAC 1108.

Fourth Generation Computers (1980 -)

This age belongs to the *microchip,* or more commonly known as VLSI - very large scale integrated circuit. A microchip is the size of a postage stamp containing hundreds of thousands of electronic components. This is the age of the micro or "personal" computer which is still evolving.

The GDT had thus improved from 1 microsecond to 1 nanosecond (1000 fold decrease) in 30 years. Typical machine performances from these generations are:

1st generation: 100 arithmetic operations per second (EDSAC)

2nd generation: 100,000 operations per second (IBM 7090)

3rd generation: 10 million operations per second (*10 mflops,* IBM 360)

Clock rate, CPI, MIPS and FLOPS:

CPU clock speed, or clock rate, is measured in Hertz — generally in gigahertz, or GHz. A

CPU's clock speed rate is a measure of how many clock cycles a CPU can perform per second. For example, a CPU with a clock rate of 1.8 GHz can perform 1,800,000,000 clock cycles per second.

This seems simple on its face. The more clock cycles a CPU can perform, the more things it can get done, right? Well, yes and no.

On the one hand, clock speeds are useful when comparing similar CPUs in the same family. For example, let's say you're comparing two Intel Haswell Core i5 CPUs, which only differ in their clock rate. One runs at 3.4 GHz, and one runs at 2.6 GHz. In this case, the 3.4 GHz processor will perform 30% faster when they're both running at their top speed. This is true because the processors are otherwise the same. But you can't compare the Haswell Core i5's

CPU clock rate against another type of CPU, such as an AMD CPU, ARM CPU, or even an

older

Intel CPU.

This may not seem obvious at first, but it's actually for a very simple reason. Modern

CPUs are becoming much more efficient. That is, they can get more work done per clock cycle. i.e., newer CPU's require less time (or fewer clock cycles) to process an instruction.

For example: Consider two processors(P1 and P2). P1 has a 3 GHz clock and requires 5 cycles/instruction (CPI). P2 has a 4 GHZ clock and requires 8 cycles/instruction.
Performance of P1 = 3 x 10^9 / 5 = 600 x 10^6 instructions/second or 600 MIPS (Million Instructions Per Second).
[Typically, floating point instructions are considered and the terminology is FLOPS – FLoating point Operations Per Second]
Similarly Performance of P2 is 4 x 10^9 / 8 = 500 MIPS.
P1 is thus faster than P2 even though P2 has a higher clock rate than P1.

1.The von Neumann Architecture

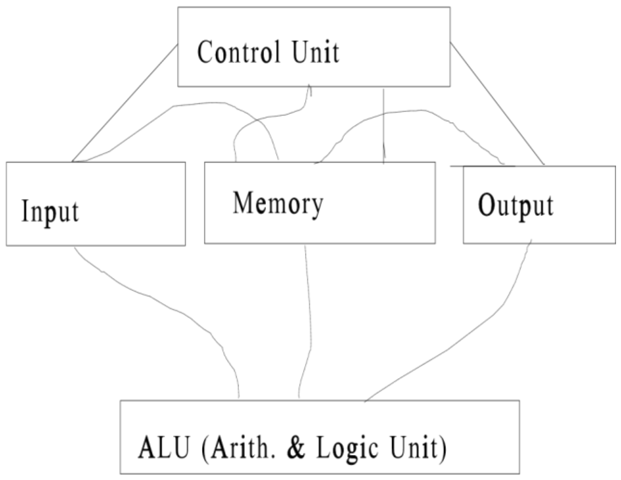
All the computers designed in these generations, including the Babbage Analytical Engine, were based on essentially one model of architecture: the *von Neumann* model. This consisted essentially of one each of the following:

I/O : input/output device

MEMORY location for storage of data and instructions

CU : control unit for instruction interpretation

ALU : arithmetic and logical unit for processing data.



(Dotted line : Data, Solid Line: Control)

In this model, under the control of the control unit, the following operations are repeated over and over until it reaches a "stop" instruction:

1. Read an instruction from memory;
2. Read any data required by instructions from memory;
3. Perform the operation(s) on the data;
4. Store results in memory;
5. Go to 1.

The hardware of modern day computers acts in much more complex manner. However, for purposes of understanding, this simple model suffices. The important fact to note is that processing speed of the machine is limited by the rate at which instructions and data can be transferred from memory to processing unit. This narrow connection between instructions and data held in memory and the single processing unit forms the so called *von Neumann's bottleneck.*

In spite of the bottleneck, each succeeding generation of computer evolution, the processing speed of the machines, measured by the number of operations per second had improved by an order of 10^5. There has been a real need for faster and faster machines to solve larger and more demanding problems. This has been especially true of scientific and engineering communities. Typical examples of such problems are aerodynamic problems, database retrieval, computer aided manufacture.

1.3 Performance measurement

It is obvious that the bottleneck of the von Neumann architecture can be avoided by simply using many processing units and many memories, and possibly increasing control units. The advantage would be that many data instruction streams would be active simultaneously - in parallel. The memories and processors would have to be connected together by some sort of *connection network*. However, it is not at all clear how to arrange this multiple processor configuration so that it increases the overall computational speed. *Speedup* is a benchmark to evaluate parallel computers and is defined as the ratio of “serial computation time” and “parallel computation time”.

Consider a parallel computer with *N* processors, each computing at a peak rate of *T* operations per second. It would compute (*N*T - V)* operations where *V* is the overhead cost associated with the set up and the particular application. The *Speedup* to compute a program with “t” instructions is given by

$$\frac{t/T}{t/(NT-V)} = N \text{ (approximately assuming } NT >> V)$$

This is called *linear speedup*. However, linear speedup is seldom obtained practice.

Is it possible to design architectures in which several processors would work in harmony on a single problem, and produce close to a linear speedup? Speed was essential since according to Grosch's law (Herb Grosch, 1960)," a manufacturer can sell a new generation machine to the consumer for twice the price, only if it is four times faster." The reasoning here is that buyers are not impressed by marginal improvements since technology was evolving rapidly. Note that according to Keynesian economics, twice the hardware cost twice as much so that speed cannot be obtained by simply "combining" two machines. Seymour Cray defied Grosch's law for a while during 1976 with his Cray-1 which had 80 MIPS, and then with his Cray Y-MP (1988) which had more than 1 *gigaflops* performance. Cray achieved this by making the circuits small (successive Crays have tended to be smaller and smaller) and used gallium arsenide chips. Grosch's law was defeated with the help of large scale integration.

There is however a limit to the kind of speed up achieved by Cray by simply reducing the size of the circuits and using different materials. Ultimately, to realize linear speedup one has to take recourse to parallel processing machines. Even though the economic incentive to manufacture such machines exists, machines with non von Neumann architecture were not built for a long time because of another law - *Amdahl's law*.

The performance of a parallel program can be severely affected if it contains serial portions of the code, even if they are relatively small compared to the parallel portion of the program. Let □ be the fraction of a program that is serial, $(1 - \square)$ the part that is parallel. Assume that we have N processors and let $T(j)$ be the computational time to run the program with j processors.

The serial part of the program is to be done by a single processor, so is done in time $\square.T(1)$. If a single processor does the parallel part also, it would be done in $(1 - \square).T(1)$ units of time. If we used N processors, the parallel portion would be done in time $(1 - \square).T(1)/N$. Hence, using N processors, the complete program can be run in time

$$T(N) = T(1)\square + (1-\square).T(1)/N.$$

Using the formula, Speedup $S = T(1)/T(N)$, we find that $S = N/(\square N + (1-\square))$.

This is Amdahl's law. This is particularly depressing! For example if $\square = 0.1$, that is 10% of a program is serial, then with 10 processors, the maximum speedup = 10/(1 + 0.9) = 5(approx.)

1.4 Classification of Parallel architectures

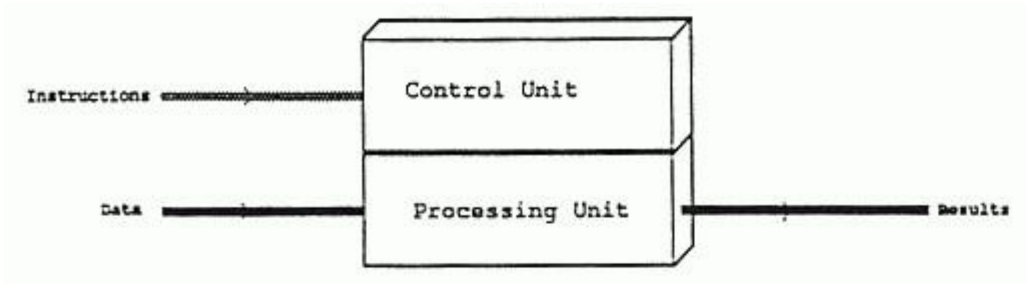
There are various ways of classifying parallel architectures. The major classifications consider the number of instructions and/or data sets processed simultaneously, the internal organization of processors, the internal memory organization, the interprocessor connection network, or the methods used to control the flow of instructions and data.

1.4.1 Flynn's Classification of parallel architectures

M. J. Flynn proposed a classification of parallel computers in 1966. It is a good starting point for classifying parallel architectures based on the parallelism of instruction streams or data streams. A stream simply means a sequence of items (data or instructions). Computers can be roughly divided into four major groups based on the multiplicities of the instruction and data streams as described below:

	Single instruction stream	Multiple instruction streams
Single data stream	SISD	MISD
Multiple data streams	SIMD	MIMD

1. **SISD or Single Instruction set Single Datum set.** This is essentially a serial machine with the von Neumann model architecture. A single data stream is being processed by one instruction stream by one processor.



2. **SIMD or Single instruction set Multiple Datum set.** Here the single master control unit MCU broadcasts identical instruction to each of the processing elements which are executed at the same time. i.e., all processors work in lockstep fashion with different data. The operations are said to be *synchronous*. Each processor has a local memory and possible access to global memory. All the processors are connected by an ICN. This category of computers is also referred to as Array Processors or Vector Processors. Those processors are usually special purpose (e.g.

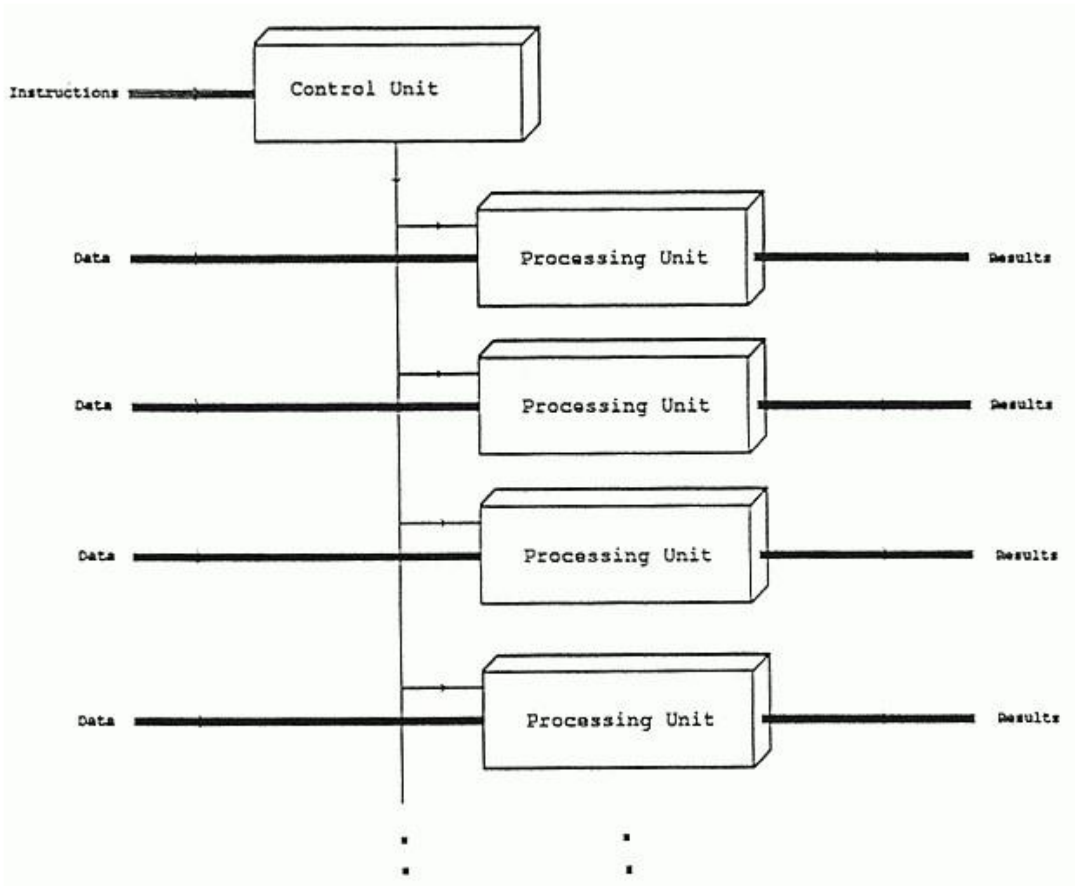
for image processing), since full generality is not required.

If "N" operands are to be processed (assuming a floating point computation requiring "T" seconds) in a SIMD architecture with "M" processors

$$\text{Speed up} = \frac{NT}{\lceil N/M \rceil T}$$

$$= M \text{ (approximately assuming } N >> M)$$

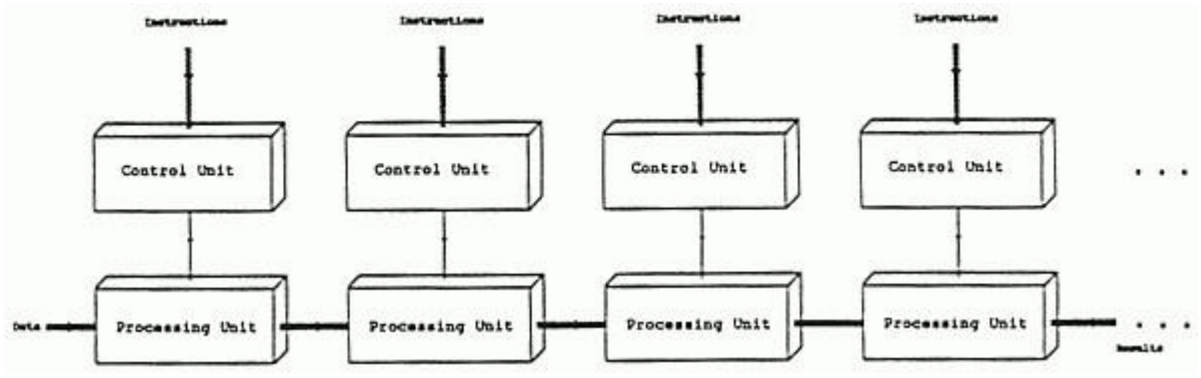
In theory, the Speed up with M processors achieves a linear speed up of M.



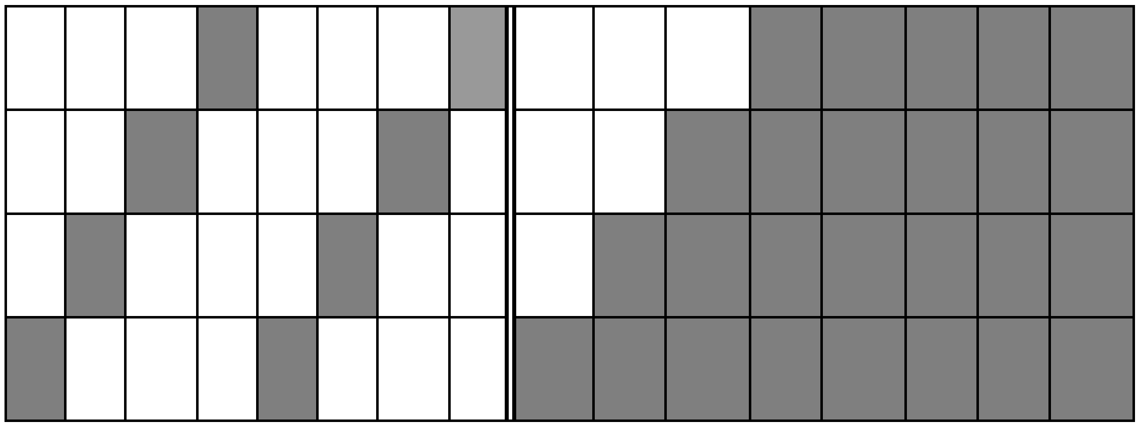
3. **MISD or Multiple Instruction set Single Datum set.** Not many parallel processors fit this category. However, special-purpose machines are certainly conceivable that would fit into this niche: multiple frequency filters operating on a single signal stream, or multiple cryptography algorithms attempting to crack a single coded message. Both of these are examples of this type of processing where multiple, independent instruction streams are applied simultaneously to a single data stream. i.e., A single data stream passes through a pipeline, and processed by multiple (micro-) instructions in different segments of the pipeline. This category of computers is referred to as Pipelined Processors.

Pipelining is often done at assembler level is to start each operation before the previous one is completed. Generally, in such a system, a sequence of data values passes through the

pipeline and is changed by each processor. The input for a processor arrives from the left and the completed result is passed to the processor on the right.



Example: The addition of two floating point numbers *X* and *Y* is done in four stages after converting them to their binary equivalent. Let $X = A.2^{□}$, $Y = B.2^{□}$, where *A* is the mantissa and □ the exponent for *X*, and similarly for *Y*. The four stages are: (a) Compare □, □ to find *(□ - □)*, (b) Use *(□ - □)* to shift A with respect to B to line up the binary points, (c) add *A* and *B*, the mantissas, and (d) normalize the sum (to ensure that the first digit after the binary point is non-zero). With a single processor, all these stages would be done one after the other, as per the von Neumann architecture. If each stage takes *T* seconds, the whole operation with four stages would take *4T* seconds. More generally, with an operation with *S* stages, it would take *ST* seconds. For *N* pairs of numbers, the total time would be *STN* seconds. Hence, the speed in this case is *1/STN* flops. Note that we get a sum after every *ST* seconds.



Serial Processing Pipeline processing

With pipelining, the first pair of operands are added in time *ST* seconds, after which we have a new sum after every *T* seconds. Thus the total time to process *N* operand pairs in pipeline is *TS + (N - 1)T =(S + N - 1)T* seconds.

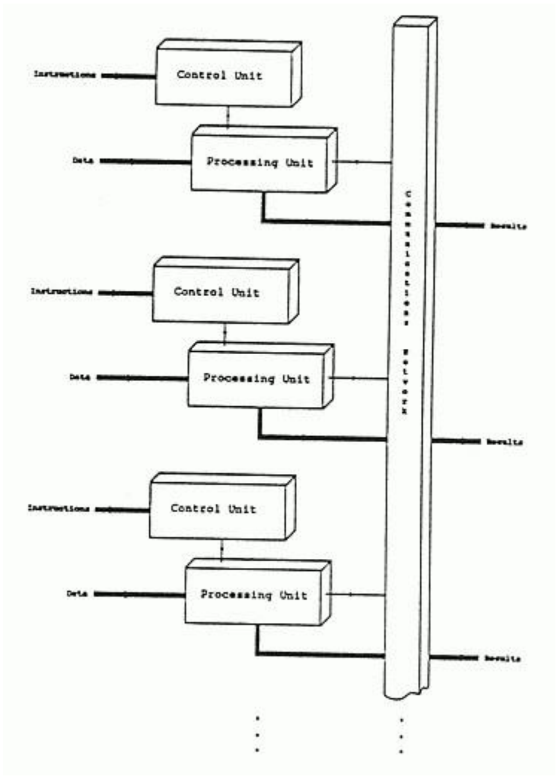
Number of floating point operations per second = *1/(S+N-1)T*

$$\text{Speed up} = \frac{SNT}{(S + N - 1)T}$$

$$= S \text{ (approximately assuming } N >> S)$$

In theory, the Speed up with S processing segments achieves a linear speed up of S.

4. MIMD or Multiple Instruction set Multiple Datum set. Most multiprocessor systems and multiple computer systems can be placed in this category. A MIMD computer has many interconnected processing elements, each of which has their own control unit. The processors work on their own data with their own instructions. Tasks executed by different processors can start or finish at different times. They are not lock-stepped, as in SIMD computers, but run asynchronously with each processor acting independent of each other.



1.4.2 Classification of parallel architectures based on processor-memory interactions

Parallel computers can be classified by their way of memory-processor connectivity. The following two kinds of architectures exist:

- shared-memory architecture
- distributed-memory architecture

Shared-Memory Architecture

The main feature of shared-memory multiprocessors is, that each processor has access to a global address space, which is shared by all processors in the system. Communication and synchronization is implicitly done by shared variables. Problems can arise, if two processors want to write on the same data. To prohibit this situation, the user can use synchronization mechanisms like semaphores and locks.

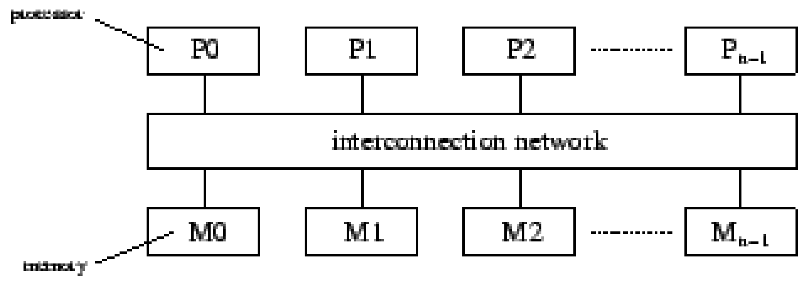


Figure 2.1: Shared-memory connection

Generally, writing programs for shared-memory multiprocessors is said to be easier than for distributed, because the address space is global and can be used in a very convenient way. However, one big drawback for this kind of machines is concerned with the scalability of the system. Adding more processors to the system leads to a more complex hardware structure. Fact is, that most of the systems do not have more than 64 processors, because the centralized memory and the interconnection network are both difficult to scale once built.

Distributed-Memory Architecture

In a distributed-memory multiprocessor each processor has its own address space. The information exchange has to be done over the interconnection network by either message-passing or any software layer.

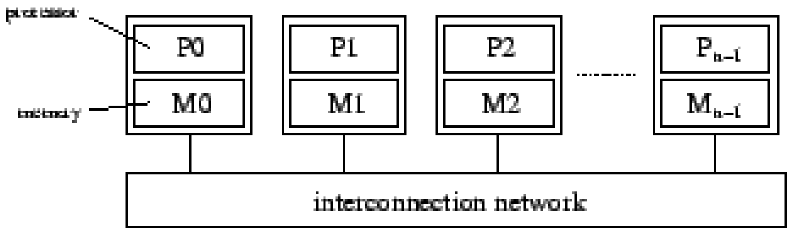


Figure 2.2: Distributed-memory connection

Every processing unit with its memory block can be seen as a node within the multiprocessor. If a processor needs information from another processor, it has to make a request over the communication network. For message-passing systems, different kinds of messages exist. In the last few years, many message-passing software packages have been implemented. However, two (platform independent) systems have made a breakthrough in this area: *PVM* and *MPI*

	Shared Memory	**Distributed Memory**
Other names	Tightly Coupled	Loosely Coupled
Memory access time	Uniform	Non-uniform
Communications	Through Memory	Through Messages
Problems	Memory Contentions	Routing

1.4.3 Classification of parallel architectures based on Interconnection Networks

One of the most important components of a multiprocessor machine is the interconnection network. In order to solve a problem, the processing elements have to cooperate and to exchange their computed data over the network. We can distinguish between two different connection topologies:

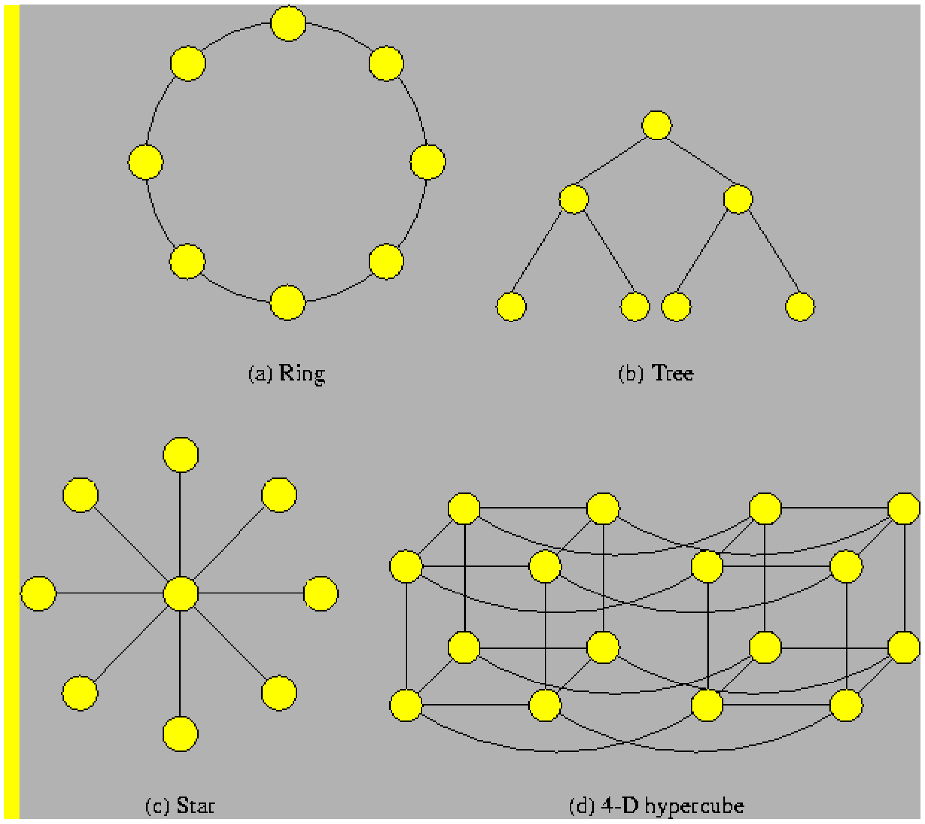
- *static,*

and

- *dynamic.*

Static Connection Topologies

In static networks the nodes are connected to each other directly by wires. Once the connection between nodes has been established, they cannot be changed anymore. The Figure below shows some examples of the wide variety of existing static connection topologies. In a ring, for example, every processing element (PE) is wired with just two neighbors, whereas in a 4-D hypercube every PE is connected to four neighbors. If a node wants to communicate with any node not being its neighbor, all nodes between the two communicating processors have to be passed.



Mesh and Ring

The simplest connection topology is a n-dimensional mesh. In a 1-D mesh all nodes are arranged in a line, where the interior nodes have two and the boundary nodes have one neighbor(s). A twodimensional mesh can be created by having each node in a two-dimensional array connected to all its four nearest neighbors. If the free ends of a mesh circulate back to the opposite sides, the network is then called torus. Connecting the two ends of a 1-D mesh, it forms a ring.

Tree

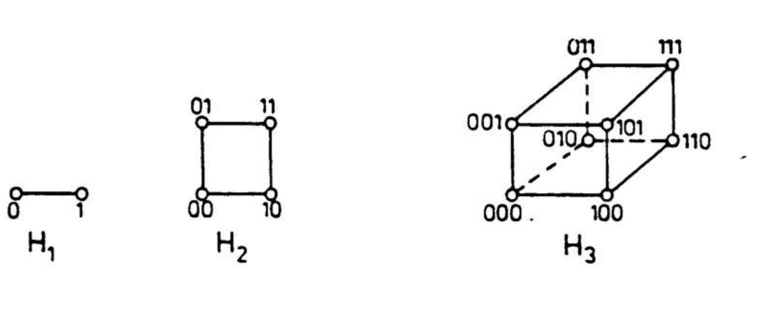
The basic (binary) tree has two sons for the root node. The interior nodes have three connections (two sons, and one father), whereas all the leaves just have one father.

Star

A star has one central node, which is connected to all other nodes. The problem of this topology is obvious. For every communication, a central node has to be passed (bottle neck).

Hypercube

A hypercube configuration in dimension *n* has 2^n vertices and $n.2^{n-1}$ edges. For instance if *n* =1, there are two vertices and 1 edge, in two dimensions, we have a square with four edges and four vertices. A vertex of a hypercube in dimension *n* can be represented by an *n-bit* vector: ($\square_1$ $\square_2$ $\square_3$ ... $\square_n$) where each $\square_i$ = 0 or 1. Two nodes are connected if and only if their bit representation differ in exactly one position.



A typical machine of this architecture is the CM-5 made by Connecting Machines of Massachusetts with 2^{16} = 65,536 processors. (Such machines are often called massively parallel).

Dynamic Connection Topologies

Contrary to the static and unchangeable network, dynamic networks can vary in their topology during runtime by switches. Generally, dynamic networks can be distinguished by their switching strategy, and their number of switching stages (single-stage or multi-stage networks). Single-stage networks have two main representatives of their class: busses and crossbars. Bus systems are quite simple in concept. They connect all the components together using one data path. The speed of a bus can be calculated by multiplying the bandwidth of the bus with its clock frequency. In a crossbar each node has just two connections, leading to a central switch. The switch has a constant delay corresponding to one logic gate independent of the number of nodes. Hence, a crossbar is a grid of logic gates and a number of nodes which are connected via these switches.

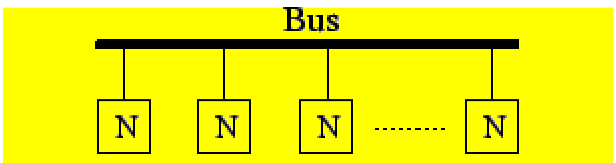
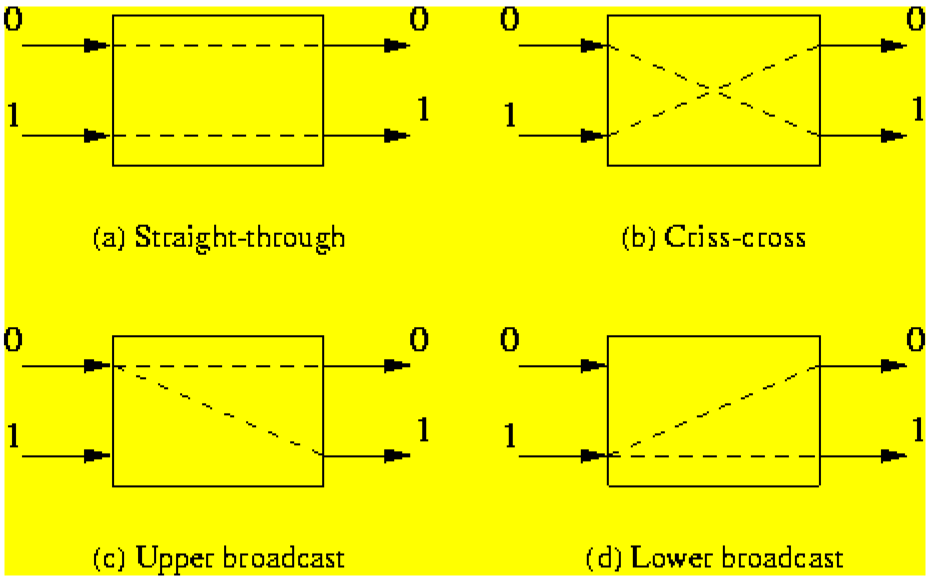


Figure 2.4: Multiprocessor-system with bus-network

Multi-stage networks connect n processors with n memory blocks (so-called modules) with each other. The connection is established over several stages. Each stage consists of a number of switches, where each input must be connected to an output. Normally, a 2 x 2 switch box is used to build up a multi-stage network. Figure below shows four different switching possibilities.

Switch possibilities with a 2x2 box



One of many multi-stage networks is the omega network. It is a member of a group of networks based on the shuffle and exchange permutations.

This group further includes:
Banyan

Baseline

Benes butterfly

Delta

All these networks have different characteristics, but in fact, all are working with 2x2 switch boxes combined to a network. Figure below shows a 8x8 omega network, which can be seen as a basic network structure.

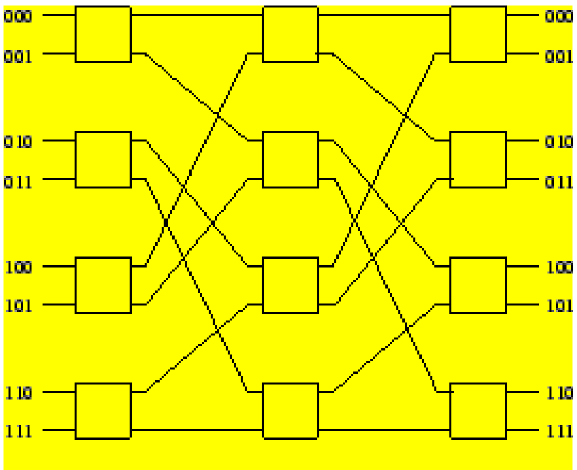


Figure 2.1: 8× 8 omega network [3]

1.4.4 Examples of parallel architectures

1. mars.calstatela.edu and CS1.calstatela.edu

   MIMD; Shared Memory; Static ICN

2. Cluster(oscar.calstatela.edu)

   MIMD; Distributed Memory; Dynamic switching ICN

3. Illiac IV

   SIMD; Distributed Memory; Static (mesh) ICN

4. nCUBE

   MIMD; Distributed Memory; Static (hypercube) ICN

5. CRAY

   MISD-MIMD; Shared & Distributed Memory; Proprietary ICN.