Chapter 1

INTRODUCTION

1.1 PARALLEL COMPUTING

Over a period, beginning from ancient times, man has been consciously or otherwise building complex systems. In these endeavors, he has often resorted to the employment of several people (workers) to cooperate and carry out a given task in a reasonable time. The idea of employing a multitude of workers is to reduce the completion time of the work. It is common knowledge that for a given job, a certain number of workers is optimal and use of more workers beyond this number gives rise to inefficiencies, leading to very small increase in the gain. This situation is analogous to the one in parallel computing, but with the proviso that, man has the ability to create workers (processors) of different speeds and capabilities.

Two important influences on the speed of computing machines are

1. Advances in technology, that increases raw electronic speed of the underlying circuitry and
2. Architectural and programming techniques, that harnesses the amount of parallel activity (parallel processing)

1.1.1 ADVANCES IN TECHNOLOGY

Over the years, technological progress has made possible for today's machines to operate at higher clock rate, by an order of seven, than those built in the 1950s. First generation digital computers used vacuum tube diodes and triodes (equivalent of transistor but consuming 10-20 W of power). Shockley's invention of transistor device at Bell labs in 1947-48 got him the Nobel Prize in 1957. It leads to semiconductor technological revolution in electronics. Transistors were used in radios in the 1950s but computers were late in utilizing them. It was said that T. J. Wattson, the then IBM chief, presented pocket radios to his design engineers so that they could be convinced about adapting this semiconductor device in the computers. During late 50s and early 60s, computers were manufactured using transistor devices. They operated at a frequency of a few 100 kHz. Technology used was the bipolar junction devices. It had a major bearing on power consumption (less) and clock speed (improved). In the 60s, with the advent of small scale integration(SSI), SSI circuits were fabricated that had 10's of transistors and were used commercially resulting in improved packaging and clock speed. This was done by Fairchild

Corporation, a company founded by the few people who had left the company formed by Shockley to manufacture transistors. Later Gordon Moore left Fairchild to form Intel that produced first on chip microprocessor. That was a 4 bit processor 4004 designed to implement a calculator for a Japanese company. Microprocessors revolutionized the society, kept the pace for semiconductor technology development and its applications. The 8085 microprocessor had a clock speed of only 4 MHz(1973) and the main frames of that time had speed of few MHz in 70's. It was not uncommon to have a clock with a period of a few nanoseconds in supercomputers in the late 70's using bipolar Emitter Coupled Logic (ECL) technology that consumed and dissipated a large amount of power requiring special cooling (Liquid nitrogen) around circuit boards. A machine running at a clock period of 1 ns requires everything to be packaged in a sphere of 30 cm radius (light travels 30 cms in one nanosecond) which brings in its own problems of delays on the connecting wires. The CRAY 1 systems [Russell 1978] used a clock frequency of 80 MHz requiring the machine to be packaged into a special cylindrical shape to minimize the signal travel distances. CRAY YMP operates at a clock 250 MHz. Today it is not uncommon to have a clock speed of few GHz. Due to VLSI technology, besides being highly dense, circuits with very small dimensions have become more reliable and faster. Both the density and the speed of the semiconductor technology have doubled every 2-3 years (Moore's law).

SEMICONDUCTOR TECHNOLOGY

Semiconductor technology is a breeding ground for all the developments in the computers and the communications since last 35 years.

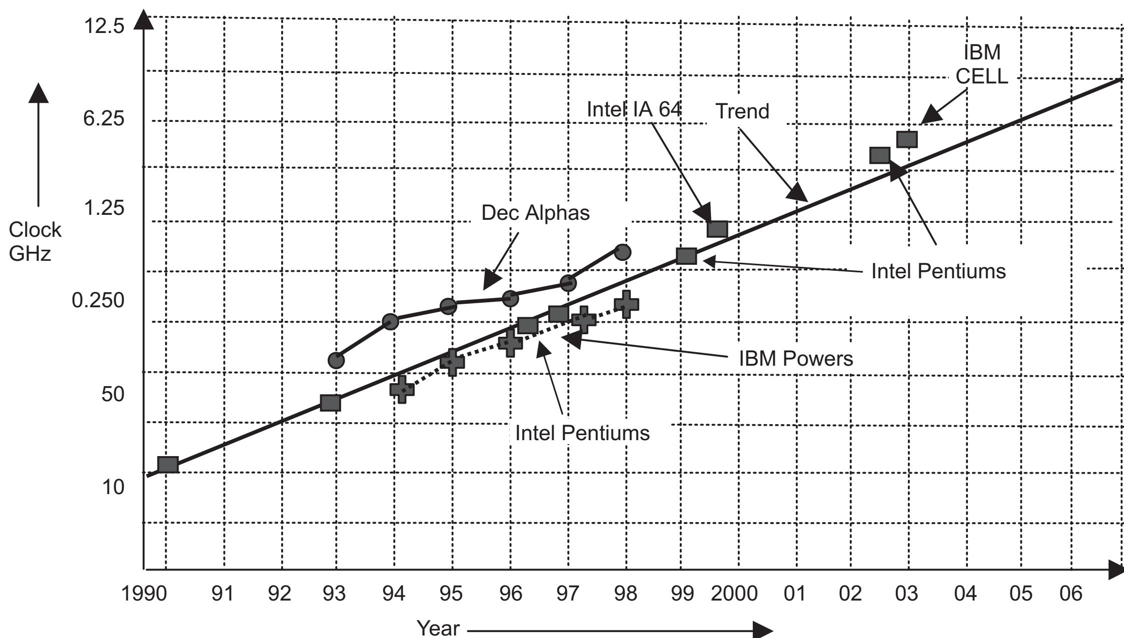


Fig. 1.1 Processor Clock Growth Trend

The 35 year long trends in microelectronics have led to increase in *both* the speed and the packing density as shown in Fig. 1.1 and Fig. 1.2.

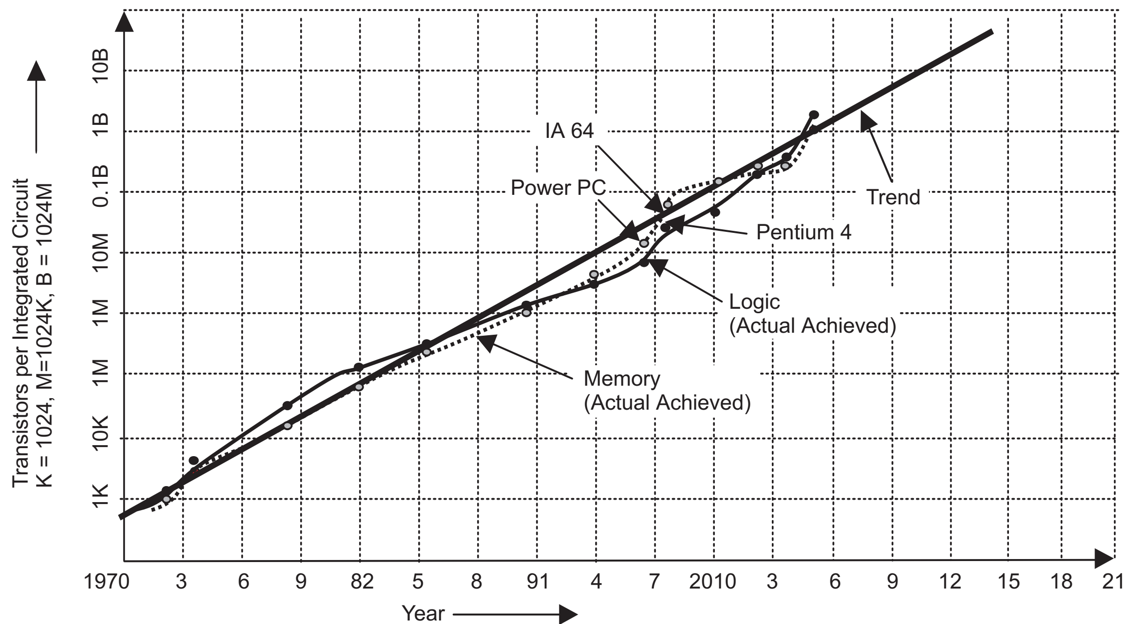


Fig. 1.2 Semiconductor Technology Trend

The semiconductor fabrication technology trend is shown in Fig. 1.3. In 2000, we have quietly entered into nanotechnology age (size of devices fabricated in nanometers). The initial size of a transistor was a few 10's of micron (micrometer) and today we have sub-micron sizes (that is few 10's of nanometer), thereby progressing from microelectronics and microtechnology to nanoelectronics and nanotechnology. This increase in device density requires means for allowing power dissipation, which increases as a function of frequency. However, this upward trend will come to an end as we approach the barrier of heat removal ability. We need an ability to remove the heat from the chip at a faster rate so as to maintain its temperature within working limits. Today the power considerations and the operational speeds are the conflicting parameters and one has to trade off between these. We have entered into a time where tradeoffs are required between speed and density. This is quite in contrast to the traditional simultaneous benefits in speed and density. Imagine if the technological advancements continued on the speed enhancements, as stated by Moore's law till 2015, the chip temperature will exceed the surface temperature of the Sun.

In the next ten years, the impact of nano-electronics research is expected in enabling improvement in the evolution and the advancement of the CMOS technology. This requires new materials and processes to reduce the gate and the channel leakage problems associated with the device's size reduction. Different MOSFET structures may be required to sustain the size reduction beyond the 22-nm technology.

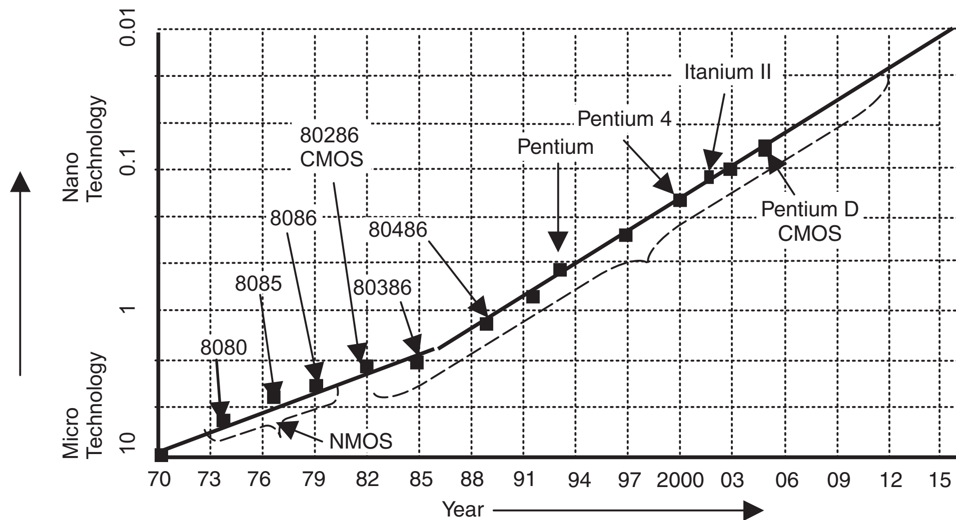


Fig. 1.3 Intel Processor Technology

WHY DO WE NEED PARALLEL COMPUTERS?

The need for parallel computers stems from the need to build faster machines. Do we really need very fast machines? This question can best be answered by an analogy from the area of automobiles. Initially, man was happy moving on his own or at most using animal carts. But then, his world of activity was limited and he was used to that kind of world. When the automobiles were invented and introduced, the sphere of his activity and the interaction with others grew. Today due to advances in air travel and telecommunications, man can interact with others at almost any place on the earth, which men could not have thought possible in earlier centuries. Even with today's fast jets, the need for faster travel is felt. There are ongoing projects to enable travel to any corner of the earth possible in an hour or so.

What we see in these trends is that when a technology is available, its use initially is restricted but when it matures, the cost reduces, which in turn makes the application spectrum grow-giving further boost to the growth of the technology and consequent price reductions. Similar scenario exists in computer industry. As an analogy, if we assume that aviation industry has advanced technology like semiconductor technology, it would mean travel time from New York to Mumbai takes only 8 milliseconds and costs 1 paisa.

The increased computing power and the decreased cost, changes one's thinking in terms of the machine's use and envisages new applications. Problems requiring large scale computing (in terms of the memory and the computing speed) are numerous. Most notable among them are large sized problems in optimization, planning, scheduling, network flows, field problems, artificial intelligence requiring logical deductions and search. These problems become non-computable when the size of the problem becomes too large to fit into the computing power (the memory and the speed) of the available machines.

Applications that require high-speed computers for the solution of some of the above class of problems are:

1. Design of VLSI circuits.

2. CAD/CAM applications in all spheres of engineering activity.
3. Solving the field problems. These are modeled using partial differential equations and require operations on large sized matrices. These problems appear in all the areas of engineering, notable among them are:
   (*a*) Structural dynamics in aerospace and civil engineering
   (*b*) Material and nuclear research
   (*c*) Particle system problems in Physics
4. Weather forecasting, meteorological and oceanographic processing.
5. Intelligent systems.
6. Modeling and simulation in economics, planning and many other areas.
7. Remote sensing requires processing of large data gathered from satellites.
8. Problems in nuclear energy.
9. Chemical and nuclear reactions.
10. Biological, human genome.
11. Geological, seismic activity.

In the absence of adequate computing power, we still solve these problems, but with a scaled down version. For example, the weather forecasting can be made more accurate by modeling with ever decreasing grid size. However, the grid size selection is based on the power available, so we accept the predictions with lesser accuracy.

MASSIVE PARALLEL PROCESSING: A DIFFICULT ROAD

Due to the limit on speed of processing by sequential computers and the ease of availability of VLSI technology and its cost effectiveness, it has become possible to think in terms of employing a large number of processors to carry out a given computation. Should the high-speed computers be built using few, very fast processors or large number of low cost processors. This is analogous to asking: Do we use few elephants or million ants to do a job? We would like to caution our readers that putting million cheap processors (million ants approach) may not give the required speed advantage in general. Commercial practice till late 1990s show that for the general purpose computing intensive applications, the few elephants approach has not been challenged by the million ants approach because better and better elephants are being created. But today, it seems that we have reached the limit of harnessing the computing power from a single processor. So, any increase in computing power must come from employing the processors in large numbers. The chaos inherent in the development is shown in Fig. 1.4. IBM Blue Gene/L operational at LLNL California has 32K processors and gives 71 TeraFlops (1 TeraFlop = 1000 GFlops). The obstacle to use of a large number of processors stems from the following:

1. The presence of sizable sequential codes in the otherwise parallel programs, and
2. The time spent in talking (communicating), even for a millionth of a second among themselves, is necessarily large on many occasions. 'Thus doing more talk (communication) than work (computation)'.

Fig. 1.4 Parallel Computing Chaos: Hardware Easy to Build: Software is a Problem

The first scenario is analogous to bridge crossing by million and few. The time required in the above scenario for few elephants and million ants is obvious. The operations on a shared data are examples of such cases (bridge crossing) in actual computations. The second scenario is analogous in having too much talk and too little work when a large crowd works on a common goal in an apparently unorganized manner. It should be noted that the individual computational problems could be studied and those with good communication and computing regularities could be effectively solved using massive parallelism.

GRANULARITY OF PARALLELISM

Parallel processing emphasizes the use of several processing elements (processors) with a prime objective of gaining the speed in carrying out a time consuming computing job. A multitasking operating system executes jobs concurrently, but the objective is to affect the continued progress of all the tasks by sharing the resources in an orderly manner. The parallel processing emphasizes the exploitation of concurrency available in a problem for carrying out the computation by employing more than one processor to achieve better speed and/or throughput. The concurrency in the computing process could be looked upon for parallel processing at various levels (granularity of parallelism) in the system. The granularity of parallelism may be thought of as the amount of work (computation) constituting the unit of allocation to the processing elements. A computing system may have several granularities coexisting. The following granularities of parallelism can be easily identified in the existing systems.

1. Program level parallelism.
2. Process or task level parallelism.
3. Parallelism at a level of group of statements.
4. Statement level parallelism.

5. Parallelism within a statement.
6. Instruction level parallelism.
7. Parallelism within an instruction.
8. Logic and circuit level parallelism.

The above granularities are listed in the increasing degree of fineness. The level 1 is the most coarse level (coarse grain parallelism), while level 8 is the finest level (fine grain parallelism). The granularities at the higher levels (levels 1 and 2 but rarely 3) can be implemented easily on a conventional multiprocessor system. Most multitasking operating systems allow creation and scheduling of processes on the available resources including CPUs. Since a process represents a sizable code in terms of execution time, the overheads in exploiting the parallelism at these granularities are not excessive. If the same principle is applied to next few levels, the increased scheduling overheads may not warrant a parallel execution. This is so because, the unit of work of a multiprocessor is currently modeled at the level of a process or tasks and is reasonably supported on the current architectures. The last three levels (fine grain parallelism) are best handled by the hardware. Several machines have been built to provide the fine grain parallelism in varying degrees. A machine having instruction level parallelism executes several instructions simultaneously. Examples are pipeline instruction processors, synchronous array processors and many others. Circuit level parallelism exists in most machines in the form of processing multiple bits/bytes simultaneously. Today we have no alternative but to employ large number of processors to gain computing power. This has given rise to some standards on how the multiprocessor systems should execute and interact when running a single program on number of processors so that software remains portable and efficient.

1.2 PARALLEL ARCHITECTURES

There are numerous architectures that have been used in the design of high-speed computers. The architectures basically fall into two basic classes, viz.,

(*a*) General purpose and

(*b*) Special purpose.

The general purpose computer architectures are designed so as to provide the rated speeds and other computing requirements for a wide class of problems with more or less same performance. The important architectural ideas that are being tried out in designing general purpose high speed computers are based on the following:

1. Pipeline Architectures
2. Asynchronous Multiprocessors
3. Data Flow Computers

The special purpose machines should excel in their performance for which they are designed although they may or may not do so for other applications. Some of the important architectural ideas in the design of special purpose computers are:

4. Synchronous Multiprocessors (array processors)
5. Systolic Arrays

6. Neural Networks (connectionist architectures)

A brief introduction to the architectural ideas mentioned above is presented next.

1.2.1 PIPELINED ARCHITECTURES

Machines based on pipeline architecture have been dominating the market, indicating the success of a simple and novel idea. These machines are cost effective and give the usual sequential view of a computing process. The conventional software written in higher level languages is portable and can be run on these. In fact, the pipeline processing is an architectural technique employed to implement the conventional von Neumann machine with increased speed. It is based on industrial assembly lines found in factories. In an assembly line, a product making process is divided into a number of sequential steps through which a job passes and finally rolls out as a finished product. Moreover, at a time there are several products (jobs) in the assembly line, each one in a different stage of completion. The pipeline model used in building a processor is based on a similar principle. A processor having an instruction cycle of n steps may be designed with an 'n' stage instruction pipeline, in which each stage executes a substep of an instruction. The tasks (instructions) enter from one end of the pipeline and flow through it are considered completed when they flow out from the other end of the pipeline. This model allows n instructions to be simultaneously active inside the pipe providing an ideal 'speedup' of n. Figure 1.5 shows a 4-stage instruction pipeline.

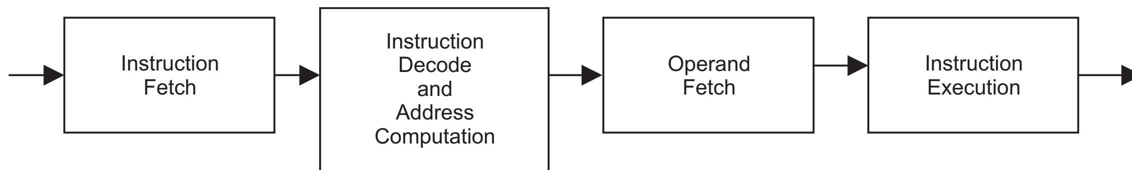


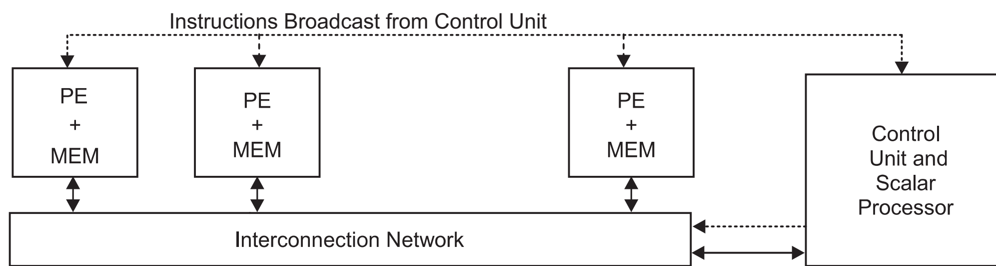
Fig. 1.5 Instruction Processing and Execution Pipe

Pipeline processing is the most successful idea in computer architecture. It is based on a serialization (queue) concept and orderly service to the tasks. It has been successfully used in high performance computers. Pipeline processing is applicable at any level of granularity, although its use in design of instruction processing and arithmetic units (fine granularity) has been most widely exploited.

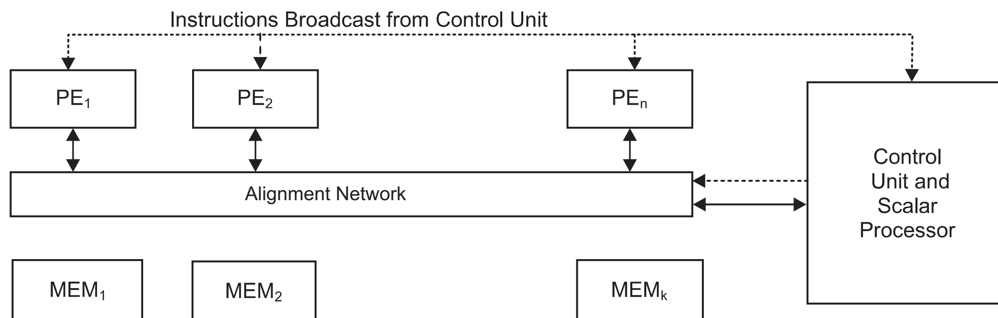
1.2.2 SYNCHRONOUS MULTIPROCESSORS (ARRAY PROCESSORS)

A machine based on synchronous parallel computing model consists of several Processing Elements (PEs), all of which execute the same instruction on different data. The instructions are usually fetched and broadcasted to all the PEs by a common control unit. The PEs execute instructions on the data residing in their own memory. Due to this architectural arrangement of a single instruction stream with multiple data streams (SIMD), these machines are called SIMD array processor. The individual PEs are connected via an interconnection network to carry out data communication between them. There are several ways of connecting these, and the topic of interconnection itself was and is being researched by a large number of research workers. These machines require special programming efforts to achieve the possible speed advantage. The computations are carried out synchronously by the hardware and therefore synchronization is not an explicit problem. Figure 1.6 shows the schematics of the SIMD array

processor architectures. The architecture shown in Fig. 1.6 (*a*) consists of processor memory modules connected by an interconnection network, while the architecture shown in Fig. 1.6 (*b*) has processors and memory modules connected by an alignment network. Note that in this architecture, the number of memory modules and the number of processing elements may be different. In SIMD processing, all the processing units execute the same instruction at any given clock cycle on their local data. Each processing unit can operate on a different data element. This type of machine typically has an instruction dispatcher, a very high-bandwidth internal network and a very large array of very small-capacity processing units. These machines are best suited for specialized problems, characterized by a high degree of regularity, such as image processing. It performs synchronous (lockstep) and deterministic execution as shown in Fig. 1.7.



(a) SIMD Array Processor Using an Interconnection Network



(b) SIMD Array Processor Using Alignment Network

Fig. 1.6 SIMD Array Processor Architecture

Several machines have been built using this model. Most notable are ILLIAC IV [Bernes 1968], Burrough Scientific Processor [Kuck and Stroke 1982]. ICL DAP [1978], MPP [Batcher 1980] and Connection Machines CM1 and CM2 [Hillis 1985] are SIMD processors having one bit processing elements and small local memory. Each processor in these supports only one bit arithmetic and logical operations. To implement word operations, it is required to program the processor using bit sequential algorithms [SIPS 1984]. A product called GAPP was available in early 80s from NCR Corporation to design SIMD parallel processor with large number of PEs.

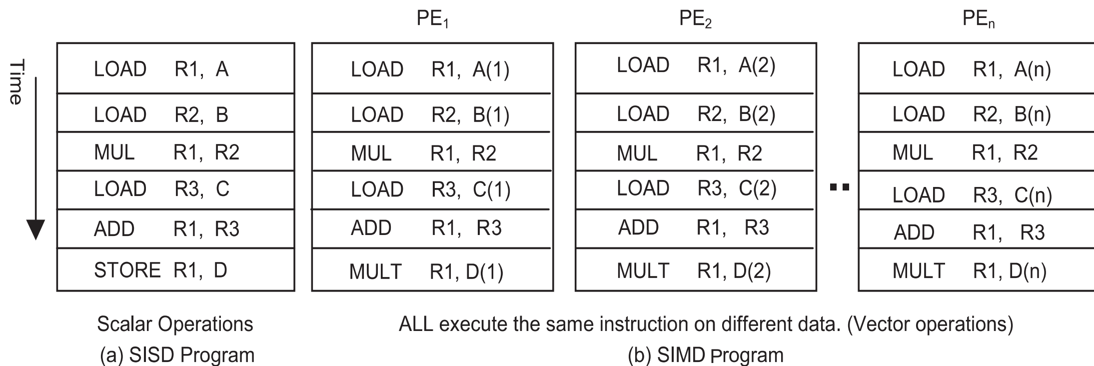


Fig. 1.7 SISD and SIMD Programs

A GAPP chip from NCR Corporation integrates 72 mesh connected PEs. Using many GAPP chips, it was possible to design arrays of PEs of any size. The chip is useful in implementing 'systolic designs' for image processing applications [Davis and Thomas1984, 85]. This kind of bit sequential architecture is one of the ways to build Vector Processors. Connection Machine CM-2, Maspar—MP-1, MP-2 are built using this approach to vector processing. Other approach to vector processing is non-SIMD and use of pipeline vector processors such as IBM 9000, Cray C90, Fujitsu VP, NEC SX-2, Hitachi S820. Today, limited vector processing is available on general purpose processors like Pentium 4, Ultra Spark and IBM Powers, etc. This is illustrated with the help of an example. An existing 64 bit adder in the processor can be used for operation on eight numbers each 8 bit long by disabling the carriers across 8-bit groups, making 8 adders available for adding eight 8 bit numbers in parallel by a single instruction. This is quite useful and inexpensive, since without committing too much resources on the chip we get more adders. It is implemented as MMX instruction extensions in Pentiums. These are useful in rendering and similar application for the display generation. The idea was first implemented by UltraSPARC as VIS extension to its instruction set for multimedia applications. IBM PowerPC AltiVec provides a full set of 32 new 128-bit registers. The registers can also be divided as 16-bit integer values or as 32-bit floating-point values. With AltiVec, it is possible to execute a 4-way parallel floating-point multiply-add as a single instruction that runs on the pipe. SIMD vector instructions can produce great speedups for applications where this type of 'data parallelism' is available and easy to extract. The original target applications for these extensions were primarily in the area of image and video processing on the desktops. But they could be easily used in new applications like audio processing, speech recognition, 3D graphics rendering and many types of scientific programs.

1.2.3 ASYNCHRONOUS (CONVENTIONAL) MULTIPROCESSORS

Multiprocessing based on multiple CPUs and memory banks connected either through a bus (Fig.1.8 (*a*)) or connection network (Fig. 1.8 (*b*) and Fig. 1.8(*c*)) is a commonly employed technique to provide increased throughput and/or response time in a general purpose computing environment. In such systems, each CPU operates independently on the quantum of work given to it. The computation, which could be a part of a larger computing goal or an independent computation thus proceeds in parallel on the CPUs and progresses independently as per the availability of the other resources. Multiprocessors have been highly successful in providing

increased throughput and/or response time in time shared systems. Effective reduction of the execution time of a given job requires that the job be broken into sub-jobs that are to be handled separately by the available physical processors. It works well for tasks running more or less independently, *i.e.*, for tasks having low communication and synchronization requirements. Communication and synchronization is implemented either through the shared memory or by message system. Based on this aspect of communication, following basic multiprocessor architectures are possible:

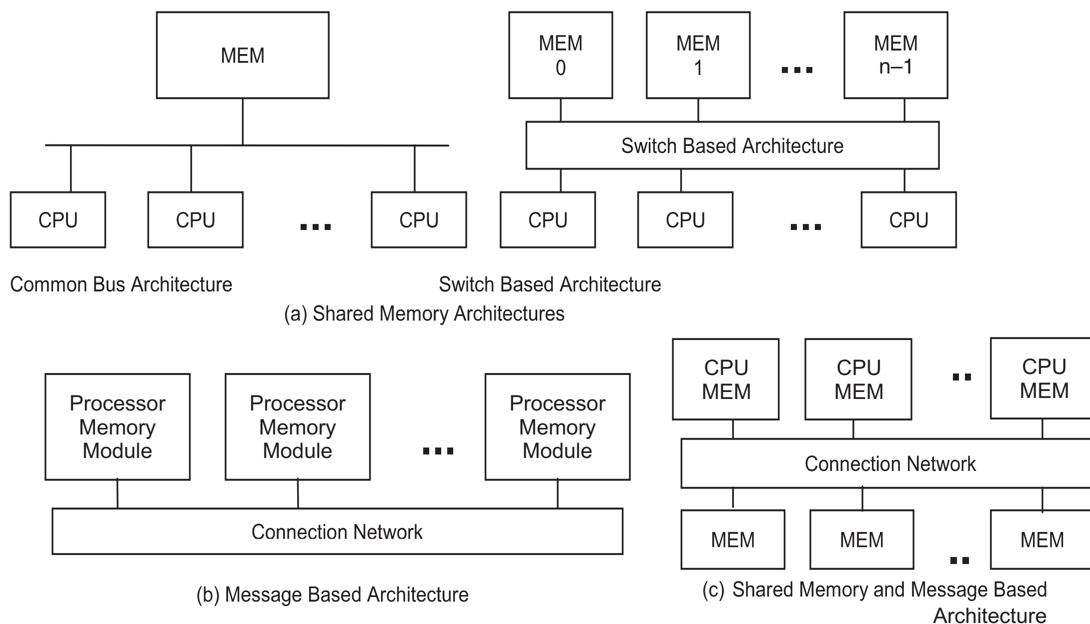


Fig. 1.8 Mutliprocessor Architectures

1. Shared memory multiprocessor
2. Message based multiprocessor
3. Hybrid approach using both shared memory and message based multiprocessor
4. Cluster based computing

The shared memory multiprocessors (SMP) have been common in practice due to their simplicity and the ease of operation on a single bus system. In such systems, there is a limit on the number of processors that can be effectively operated in parallel. This limit is usually of few processors, typically of order 10. Another approach is to use the communication network for the talk among the PEMs (processor memory modules). This approach allows the number of processors to grow without limit, but the connection and the communication cost may dominate and thus saturate the performance gain. Due to this reason, a hybrid approach shown in Fig.1.8(c) may be followed. Many general purpose commercial systems use common bus architecture to access global memory, disk and input/output, while separate memory processor bus or network handles the processor memory traffic. Cluster based computing employs on-site clusters of computers that are network connected using standard network technology and used as message based parallel computers. Currently, the most modern parallel computers

fall into this category of MIMD. Here, every processor may be executing on different instruction stream on its own data stream. Execution can be synchronous or asynchronous, deterministic or non-deterministic. Also, several processors may be running the same program but on different data in parallel. These machines are single program multiple data (SPMD) machines.

Most current supercomputers are based on networked parallel computer "grids" and multi-processor SMP computers are also in the common use. Scalability is the system's (hardware and/or software) ability to demonstrate a proportionate increase in speedup with the addition of more processors. Number of factors affects scalability, including hardware, memory-CPU bandwidths and network bandwidth and different kinds of communications. Table 1.1 lists some of current commercial machines with their memory architectures.

Table 1.1 Shared and Distributed Memory Commercial Architectures

MULTIPROCESSORS	Memory Architecture		
	Closely Coupled Uniform Memory Access (CC-UMA)	Closely Coupled Non-uniform Memory Access (CC-NUMA)	Distributed
Commercial machines	SMPs , Sun Vexx, DEC/Compaq SGI Challenge IBMPOWER3	SGI Origin HP Exemplar Sequent IBM POWER4 (MCM)	Cray T3E IBM SP2 Maspar
Communication	MPI Threads OpenMP Shared memory	MPI Threads OpenMP Shared memory	MPI
Scalability	to 10s of processors	to 100s of processors	to 1000s of processors
Bottlenecks	Memory-CPU bandwidth	Memory-CPU bandwidth Non-uniform access times	Program development and maintenance is hard
Software availability	High	High	Poor

The largest and fastest computers in the world today employ both shared and distributed memory architectures as shown in Fig. 1.9. The shared memory component is usually a cache coherent symmetric multiprocessor(SMP) machine. Processors on a given SMP can address that machine's memory through a global address space. The distributed memory component is the networking of multiple SMPs. SMPs know only about their own memory not the memory on another SMP. Therefore, network communications are required to move data from one SMP to another. Current trends seem to indicate that this type of memory architecture will continue to prevail and increase at the high end of computing for the foreseeable future. Moreover multi-core CPUs with multithreading will easily make it possible to have around 16 to 64 processors on a single chip in near future (Module in Fig. 1.9 has 4 CPUs shown).

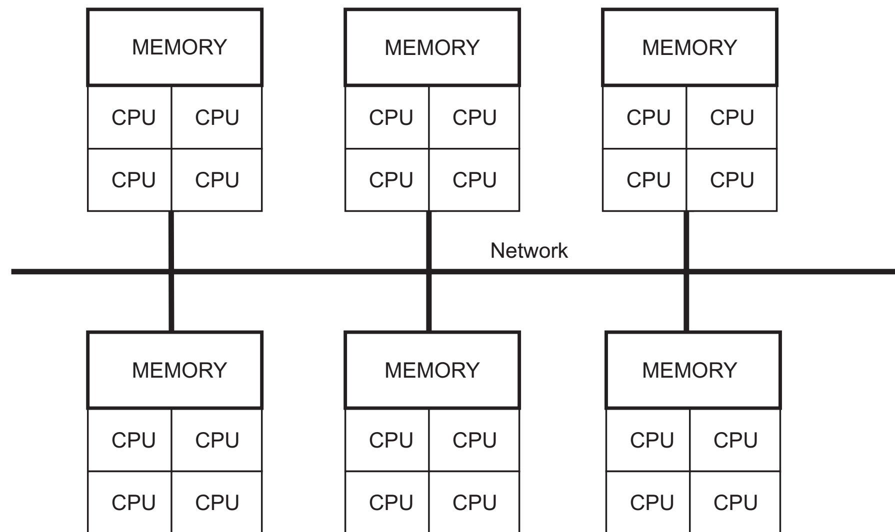


Fig. 1.9 Shared and Distributed Memory Architecture

1.2.4 DATA FLOW COMPUTERS

Jack Dennis [Dennis 1975,1980] has suggested a new fine grain parallel processing approach based on data flow computing model. In the data flow model, a number of data flow operators, each capable of doing an operation are employed. A program for such a machine is a connection graph of the operators. The operators form the nodes of the graph while the arcs represent the data movement between the nodes. An arc is labeled with a token to indicate that it contains the data. A token is generated on the output of a node when it computes the function based on the data on its input arcs. This is called as "firing" of the node. A node can fire only when all of its input arcs have tokens and there is no token on the output arc. When a node fires, it removes the input tokens to signify that the data have been consumed. Usually, computation starts with the arrival of the data on the input nodes of the graph.

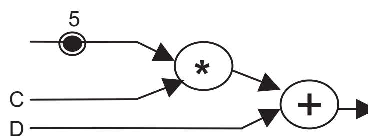


Fig. 1.10 Data Flow Graph

Figure 1.10 shows the data flow graph for the computation: A := 5 * C + D. In the above graph, the operator '*' can fire if both its inputs are available. The input 5 is available as shown by the token (a filled circle shown) but the input C is not yet available. When it arrives (may be from memory), the multiplication node fires. In doing so, it removes the tokens from the input arcs and puts one token on the output arc. In the graph shown, the addition starts after the multiplication is completed, provided that the data on the input D is available. The data flow graphs depict the data dependencies in the computations and therefore the computations progress as per the data availability. Since the sequence of computations are ordered by the

data flow, the machines based on these ideas are called data flow machines in contrast to the conventional control flow machines. Many conventional machines employing multiple functional units employ the data flow model for scheduling the functional unit pipelines. A functional unit (FU) has reservation station that is like input arch of a data flow operator(FU), and when both the operands are available (both tags 0), the FU initiates operation and generates output value (token) which is placed at desired destinations as given by Tomasulo's algorithm in IBM 360/67. Jack Dennis rediscovered and abstracted this idea to produce data flow computer architecture model. There was wide interest in the architecture and several workers have built experimental machines. Notable among these are Manchester machine [1984] and MIT machine [1980]. The data flow computers provide fine granularity of parallel processing, since the data flow operators are typically elementary arithmetic and logic operators. In spite of fine grain parallelism being exploited, a practical conceivable data flow machine is hindered in its performance by sequentiality in the control functions that are centralized.

It is not be premature today to say that these efforts may not succeed to provide the effective solution for using very large number of computing elements in parallel. While with its asynchronous data driven control, it has a promise for exploitation of the parallelism available in the problem, but its machine implementations are no better than conventional pipelined machines employing multiple functional units using Tomasulo's algorithm [Tomasulo 1967].

1.2.5 SYSTOLIC ARCHITECTURES

The two important architectural ideas that have been of contemporary interest are systolic architectures [Kung 1982] and neural networks. The advent of VLSI has made it possible to develop special architectures suitable for direct implementation in the VLSI. Systolic architectures are basically pipelines operating in one or more dimensions. The name systolic is derived from the analogy of the operation of the blood circulation system through the heart. Conventional architectures operate on the data using load and store operations from the memory. Processing usually involves several operations. Each operation accesses the memory for data, processes it and then stores the result. This requires too many memory references as shown in Fig. 1.11 (*b*). Alternate processing method is shown in Fig. 1.11 (*a*) in which the data to be processed flows through various operation stages and then finally is put in the memory. In such architectures, data to be processed is taken from the memory and enters the processing for operation f_1 as shown in Fig. 1.11 (*a*). The data processed by f_1 is given to f_2 and so on. In the end, the processed data from f_n is stored in the memory.

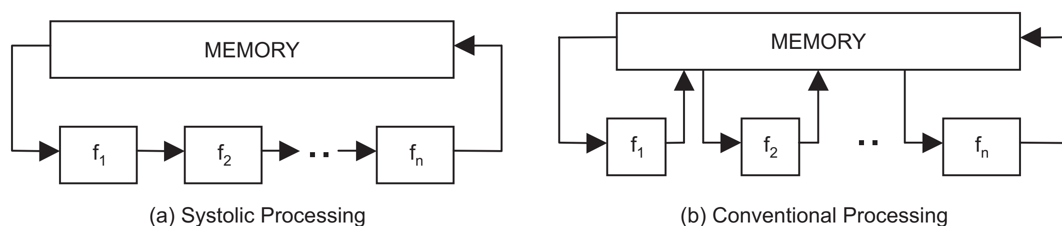


Fig. 1.11 Simplified View of Systolic Architecture

The processing method discussed above, has the similarity to the human systolic system in which the heart pumps the pure blood out from one side and the impure one returns into the

heart from the other side. In the process, it branches out into various subarteries and veins, hence the name Systolic Architecture. It is possible to develop special architectures for a given application and fabricate the same in VLSI. These architectures give very high computing throughputs due to regular data flow and pipeline operation and are useful in designing the special processors for graphics, signal and image processing and many other applications. Major research efforts in the systolic architectures have been in the synthesis, the analysis and the fault tolerance of systolic arrays from the problem description. Figure 1.11 (a) shows only one-dimensional array. Nevertheless it is possible to design complex multidimensional arrays with complex flow patterns in various dimensions.

1.2.6 NEURAL NETWORKS

Computing based on connectionist model is not new and has been under study [Grossberg 1986, 1988, Kohonen 1978, Hopfield 1986]. The idea in these models is derived from the fact that the human brain, although very slow with routine numeric computations, does a remarkable job in complicated tasks like observation (vision), speech and cognition. The research in these application areas have not yet produced computational models to give the desired performance. In fact, these are recognized as highly complex tasks. The neural network model uses the neural system to provide the analog of the brain by using artificial neurons and their dense connections having weights to store and process the knowledge and carry out the cognitive tasks. Some of the models have been experimented by Bhattacharya and Bhujade [1989] in character recognition problems. The neural networks provide a non-algorithmic approach to cognitive tasks in which the connections represented by weights are modified using models from adaptive systems theory. These theories have been used in various diverse disciplines. Recently, the interest has gone beyond the academic circles with several companies offering special boards and softwares to provide users with experimental neural nets that can be used for developing applications.

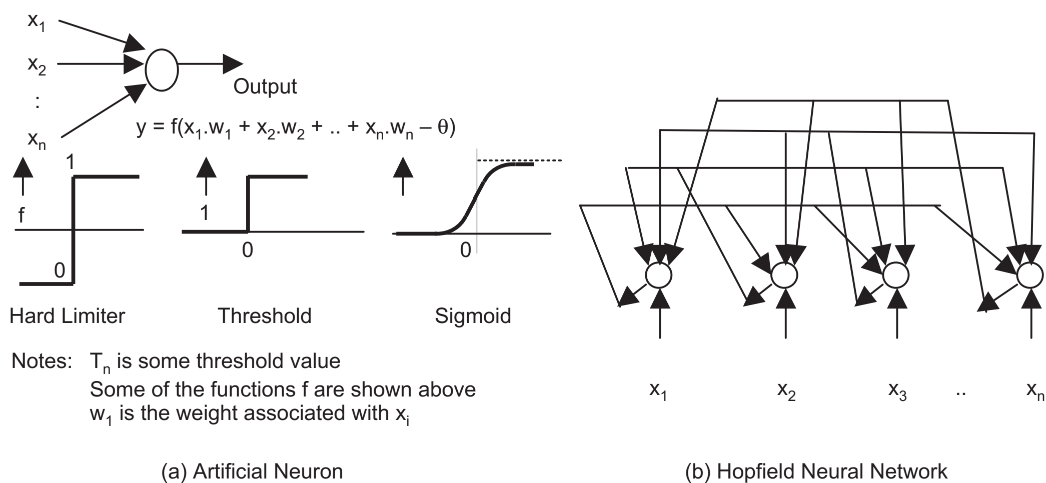


Fig. 1.12 Neutral Network

A neural network is a set of nodes and connections, in which each node may be connected to all other nodes. A connection has a weight associated with it. All nodes compute their output

as the weighted sum of their inputs, and the output of the node is fired if the weighted sum exceeds some threshold. A node could be in an excited state (fired) or in a non-excited state(not fired). The basic idea here is that the weights store the knowledge required to do the cognitive task. Typically, sample example patterns are applied to the net and the net's weights are modified through some algorithm (learning algorithm). When the patterns to be recognized are applied, the net classifies them using the firing rule of the net. Figure 1.12 shows one of the several possible neural network models.

1.3 ARCHITECTURAL CLASSIFICATION SCHEMES

The architectural classification aims at characterizing machines so that their architectural contents are reflected in the characterization. Due to the diversity of the ideas implemented by the various machines, a particular characterization may not be adequate in reflecting the machine's architecture and capability. Several different ideas are presented in this section to describe the architectural characterization for a given machine.

1.3.1 FLYNN'S CLASSIFICATION

Flynn [1966] classified architectures in terms of streams of data and instructions. The essence of the idea is that computing activity in every machine is based on:

(*a*) Stream of instructions, *i.e.*, sequence of instructions executed by the machine.

(*b*) Stream of data, *i.e.*, sequence of data including input, temporary or partial results referenced by instructions.

Further a machine may have a single or multiple streams of each kind. Based on these, computer architectures are characterized by the multiplicity of the hardware to serve instruction and data streams as follows:

1. SISD: Single Instruction and Single Data Stream
2. SIMD: Single Instruction and Multiple Data Stream
3. MIMD: Multiple Instruction and Multiple Data Stream
4. MISD: Multiple Instruction and Single Data Stream

Figure 1.13 shows the architectures based on the above four possibilities. The SISD architecture corresponds to the conventional sequential computers. The SIMD architecture represents the synchronous multiprocessors in which a common instruction stream coming from a control unit is applied to all the PEs operating in its own data stream. The examples are ILLIAC IV [Bernes 1968], ICL DAP [1979] and many others. The conventional multiprocessors are based on the MIMD architectures since the individual processors operate on the programs having their own instructions and data. MISD architecture is used in specialized applications, where same data stream is required to be transformed simultaneously in a number of ways. There are no visible examples of MISD architecture.

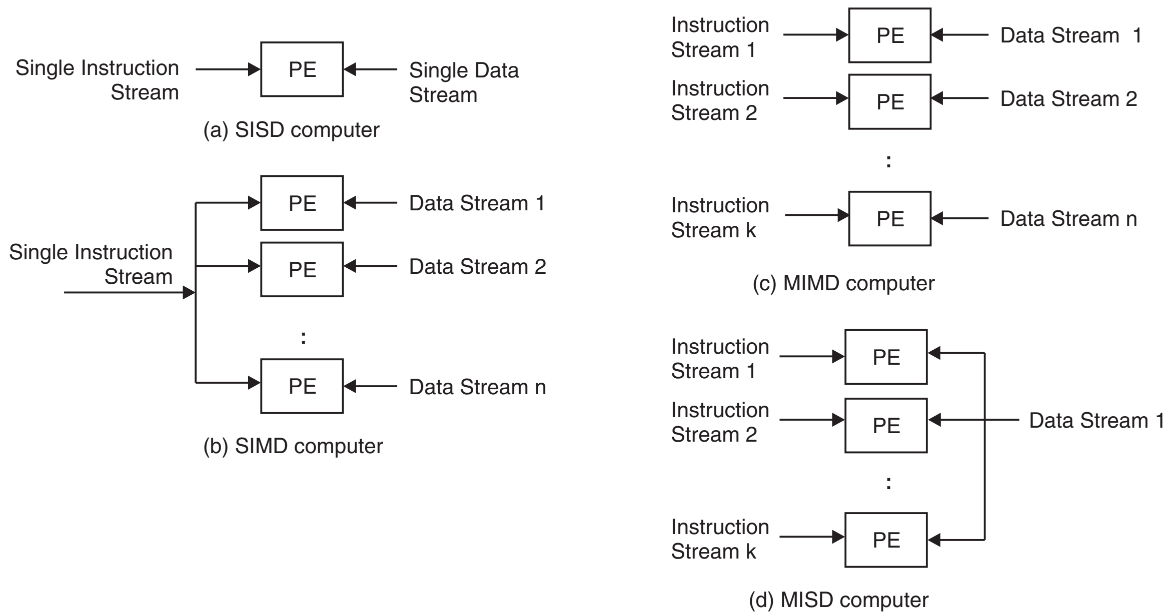


Fig. 1.13 Flynn's Classification of Architectures

The above classification scheme is too broad. It puts everything except multiprocessors in one class. Johnson [Johnson 1988] expanded the classification of MIMD machines as shown in Fig. 1.14 to account for current architectures.

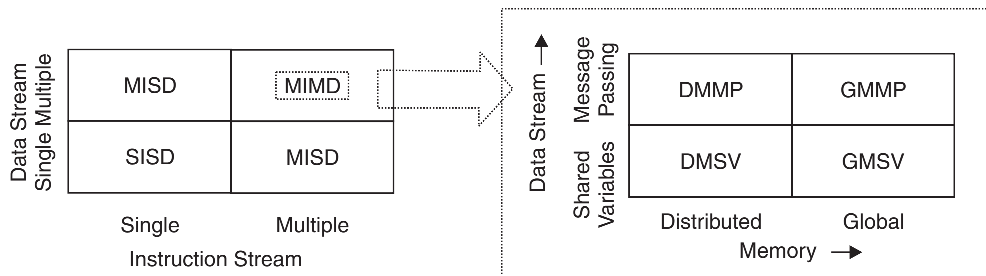


Fig. 1.14 Johnson's Expanded Classification of MIMD Machines

Classification scheme does not reflect the concurrency available through the pipeline processing and thus puts vector computers in SISD class.

1.3.2 SHORE'S CLASSIFICATION

Unlike Flynn, Shore [1973] classified the computers on the basis of organization of the constituent elements in the computer. Six different kinds of machines were recognized and distinguished by numerical designators as discussed next.

Machine 1

These are conventional Von Neumann architectures with following units in single quantities

(*a*) Control Unit (CU)

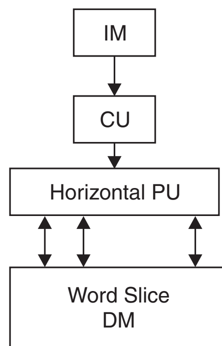
(*b*) Processing Unit (PU)

(*c*) Instruction Memory (IM)

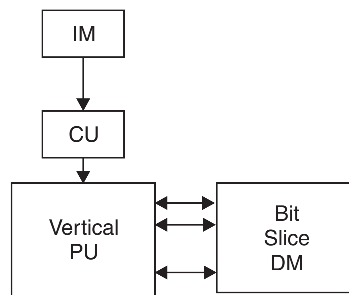
(*d*) Data Memory (DM)

A single DM read produces all the bits of any word for processing in parallel by the PU. The PU may contain multiple functional units which may or may not be pipelined. Therefore, this group again includes both the scalar computers (*e.g.*, IBM 360/91, CDC 7600, etc.) and the pipelined vector computers (*e.g.*, Cray YMX, Cyber 205). Figure 1.15 (a) shows the organization of such a machine. Note that, processing is characterized as horizontal (number of bits in parallel as a word).

Machine 2 organization is similar to that of Machine 1, except that DM fetches a bit slice from all the words in the memory and PU are organized to perform the operations in a bit serial manner on all the words (Fig.1.15 (*b*)). If the memory is regarded as a two dimensional array of bits with one word stored per row, then the Machine 2 reads vertical slice of bits and processes the same, whereas the machine 1 reads and processes a horizontal slice of bits. Examples of machine 2 are ICL DAP [1979] and MPP [Batcher 1980] and CM 2 [Hillis 1985].



(a) Machine 1



(b) Machine 2

Fig. 1.15 Horizontal and Vertical Processing

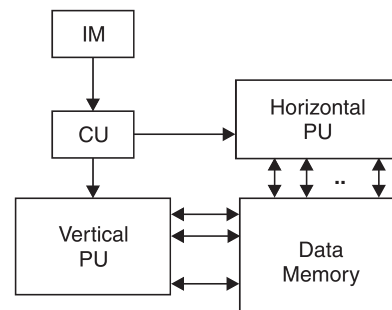


Fig. 1.16 Machine 3 Architecture

Machine 3 is a combination of Machine 1 and Machine 2. It could be characterized with the memory as an array of bits with both horizontal and vertical reading and processing possible. The machine, thus, will have both the vertical and the horizontal processing units. Well known example is the machine OMENN 60 [Higbie 1973]. Figure 1.16 shows the architecture of Machine 3.

Machine 4 architecture as shown in Fig. 1.17 (*a*), is obtained by replicating the PU and DM of the Machine 1. An ensemble of PU and DM is called as Processing Element (PE). The instructions are issued to the PEs by a single control unit. There is no communication between PEs except through the CU. Well known example of this is PEPE [Thurber 1976] machine. Absence of the connection between the PEs limit the applicability of the machine.

Machine 5 is similar to Machine 4, with the addition of the communication between the PEs as shown in Fig. 1.17 (*b*). ILLIAC IV, CM2 and many SIMD processors fall in this category. Machines 1 to 5 maintain separation between data memory and processing units with some data bus or connection unit providing the communication between them. The machine 6 shown in Fig. 1.17 (*c*) includes the logic in the memory itself and is called an **Associative processor**.

Machines based on such architecture spans a range from simple associative memories to complex associative processors.

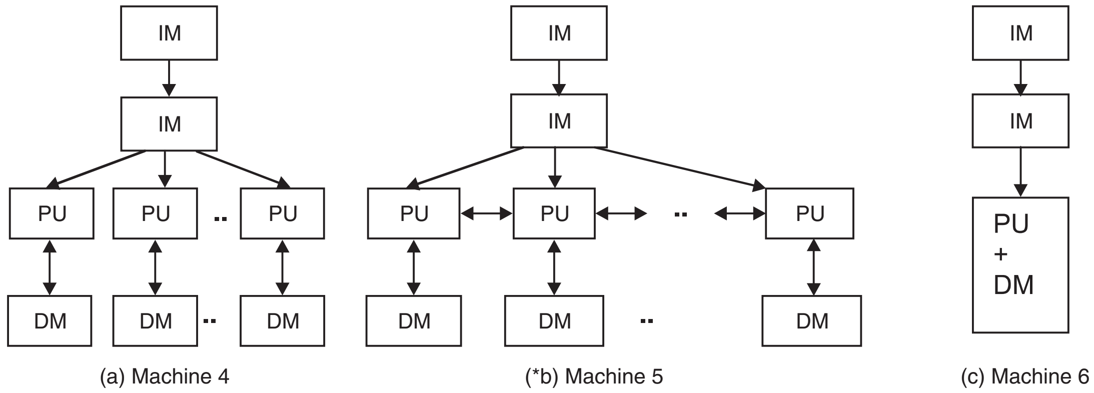


Fig. 1.17 Parallel and Associative Architectures

1.3.3 FENG'S CLASSIFICATION

Feng [1973] also proposed a scheme on the basis of degree of parallelism to classify the computer architectures. Maximum number of bits that can be processed every unit of time by the system is called 'maximum degree of parallelism'. Based on the Feng's scheme, we have sequential and parallel operations at the bit and the word levels to produce the following classification:

Classification		Examples
WSBS	(Word Serial/Bit Serial)	No conceivable implementation
WPBS	(Word Parallel/Bit Serial)	(Staran)
WSBP	(Word Serial/Bit Parallel)	Conventional computers
WPBP	(Word Parallel/Bit Parallel)	ILLIAC IV

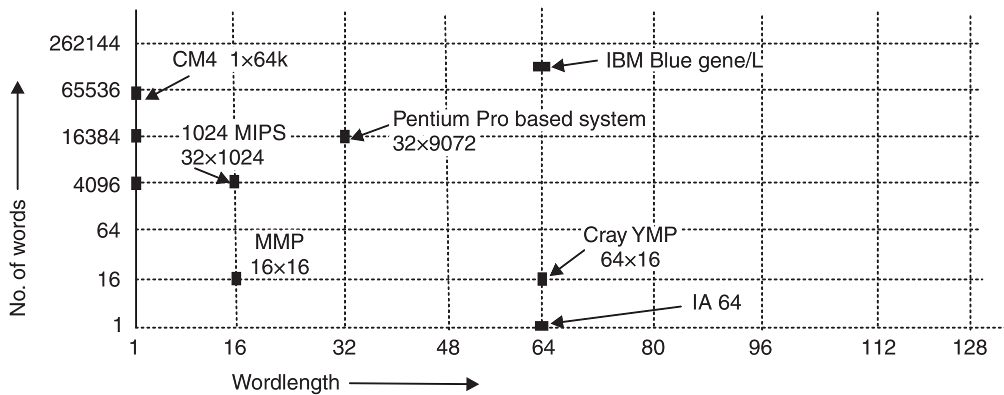


Fig. 1.18 Machines at Bit and Word Coordinates

The product of the number of bits and number of words processed gives the maximum degree of parallelism. Classification of some of the commercially available machines (current and past) is shown in the Fig. 1.18. In this diagram, one axis represents the word length and

the other axis represents the number of words processed in parallel. A machine is placed at a point in the space defined by these coordinates. The area of the rectangle formed by the origin and the point representing the machine gives the maximum degree of parallelism for the machine. For example, Cray 1 and IA-64 have areas of 64 each. It is to be noted again that the scheme fails to project the concurrency in pipeline processors.

1.4 PERFORMANCE OF PARALLEL COMPUTERS

The quantification of the performance characterization of a computer applicable across several applications is not simple. Even for sequential computers, we do not have appropriate grading numbers that truly reflect the power of a particular machine. The Linpack performance is commonly used for evaluating high performance computer (HPC), though other programs as well would serve part of the purpose. For example, 2.0 GHz Xeon and 1.2 GHz AMD Athlon gave 760.63Mflops and 567.73 sustained Mflops respectively on a conjugate gradient routine [Holmgren, Don 2002]. The speedup is one such measure of performance of a parallel computer and is the ratio of time taken by single processor system and that taken by a parallel processing system. Thus speedup is given by:

$$S = \frac{T_1}{T_n}$$

where n = the number of processors
T_1 = single processor execution time,
T_n = n processor execution time,
S = speedup.

Folk Theorem 1.1: $1 \leq S \leq n$

Proof: A parallel processor can act as a sequential processor using one PE. Therefore, a parallel processor is as fast as a sequential processor. Conversely, a sequential processor can simulate a parallel processor. Each time step of a parallel machine, a single processor takes n steps.

We called the theorem as folk theorem. This is so because the arguments in the proof do not cater for real machines. People believe, using the above arguments that the theorem should be true. It is not difficult to demolish the above theorem. David P Helmbold and etal. [1990] modeled speedups greater than n. The efficiency of a parallel computer is defined as the speedup divided by the number of processors. In other words, efficiency 'e' is the speedup per processor or normalized (to number of processors) speedup. Another view of the efficiency is derived from the analog of efficiency in other machines with which we are familiar.

Let the machine run for some period T and solve a problem. A parallel computer contains number of processing elements(PEs), each PE when employed to do work for a unit time shall do some computing work. Work done in a unit time is called power. Here the power, we are talking about is a computing power. When the computing power is employed over a period of time, the work is done by consuming the computing energy. A computing energy of a PE is its use over time. A graph showing the computing power and the time, thus depicts the energy.

A single PE has some power, n PEs have n times the computing power of a single PE. An energy diagram for a parallel computing machine is shown in Fig. 1.19.

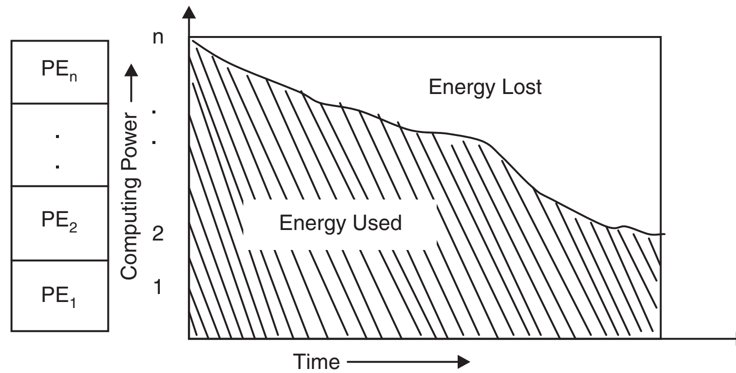


Fig. 1.19 Energy Diagram

Efficiency for any machine (including the computing machine) is given by the following relation:

$$e = \frac{\text{Energy Used}}{\text{Total Energy Supplied}}$$

The two energies referred to above are shown in Fig. 1.19. The loss of energy in many energy conversion machines occurs because of heat losses due to various reasons. Similarly, in the computing machines also, we have losses occurring in the computational energy. This happens mainly because some of the computing power could not be used while doing the work or some of the power is wasted in the losses due to the mismatch in the architecture and the algorithm due to communication, synchronization and other overheads.

According to the Folk Theorem 1.1, for n processors' parallel machine, the upper limits on the speedup and efficiency are n and 1 respectively but in practice both of these will usually be much smaller for large n. Following basic losses in solving problems using parallel computers are common:

1. Some of the processors may be idle due to the precedence constraints
2. Time is required for communication of data between processors.
3. Time is spent for synchronization and access of shared data objects.

Amdahl's Law (1967)

Amdahl's law is based on a very simple observation. A program requiring total time T for the sequential execution shall have some part called sequential fraction of computing(SFC) which is inherently sequential (which cannot be made to run in parallel). In terms of total time taken for solving a problem, this fraction called the SFC is an important parameter of a program.

Let f = sequential fraction for a given program.

The Amdahl's law states that the speed up of a parallel computer is limited by:

Energy Lost Energy Used

$$S \leq \frac{1}{f + \frac{1-f}{n}}$$

Proof of the above statement is quite simple. Assuming that total time is T, then the sequential component of this time will be f.T. The parallelizable fraction of time is therefore $(1 - f)$.T. The time $(1 - f)$.T can be reduced by employing n processors to operate in parallel to give the time as $(1 - f)$.T/n. The total time taken by the parallel computer thus, is at least f. T + $(1 - f)$. T/n, while the sequential processor takes time T. The speedup S is limited thus by:

$$s \leq \frac{T}{f.T.\frac{(1-f).T}{n}} \quad i.e., \quad \frac{1}{f + \frac{1-f}{n}}$$

This result is quite interesting. It throws some light on the way parallel computers should be built. A computer architect can use the following two basic approaches while designing a parallel computer:

1. Connect a small number of extremely powerful processors (few elephants approach).
2. Connect a very large number of inexpensive processors (million ants approach).

Consider two parallel computers Me and Ma. The computer Me is built using the approach of few elephants (very powerful processors) such that each processor is capable of executing computations at a speed of M Megaflops. The computer Ma on the other hand is built using ants approach (off-the-shelf inexpensive processors) and each processor of Ma executes r.M Megaflops, where $0 < r < 1$.

Folk Theorem 1.2: If the machine Me attempts a computation whose sequential fraction f is greater than r, then Ma will execute the computations more slowly compared to a single processor of the computer Me.

Proof: Let

W = Total work (computing job)

M = speed (in Mflop) of a PE of machine Me, then

r.M = speed of a PE of Ma. (r is the fraction as discussed earlier).

f.W = sequential work of the job.

T(Ma) = time taken by Ma for the work W

T(Me) = time taken by Me for the work W.

Time taken by any computer is:

$$T = \frac{\text{Amount of work}}{\text{Speed}}$$

T(Ma) = Time for sequential part + Time for parallel part

$$= \frac{f.W}{r.M} + \frac{(1-f)W}{r.M.n}$$

$$= \frac{W}{M}\left(\frac{f}{r} + \frac{(1-f)}{r.n}\right)$$

$$T(Ma) = \frac{W}{M} \cdot \frac{f}{r}, \text{ if } n \text{ is infinitely very large} \qquad ...(1.1)$$

$$T(Me) = \frac{W}{M} \text{ (Assume only one PE of Me)} \qquad ...(1.2)$$

From Eqs. 1.1 and 1.2, it is clear that if $f > r$ then T(Ma) > T(Me). The above theorem is quite interesting. It gives guidelines in building parallel processor using the expensive and the inexpensive technology. The above theorem implies that a sequential component fraction acceptable for the machine Me may not be acceptable for the machine Ma. It does no good to have a computing power of a very large number of processors that goes waste. Processors must maintain some level of efficiency. Let us see how the efficiency '*e*' and sequential fraction '*f*' are related.

The efficiency $e = S/n$ and hence,

$$S = E.n \le \frac{1}{f + \frac{1-f}{n}}$$

$$E \le \frac{1}{f.n + 1 - f}$$

This result is very important. It says that for a constant efficiency, the fraction of sequential component of an algorithm must be inversely proportional to the number of processors. Figure 1.20 shows the graph of sequential component and efficiency with n as a parameter.

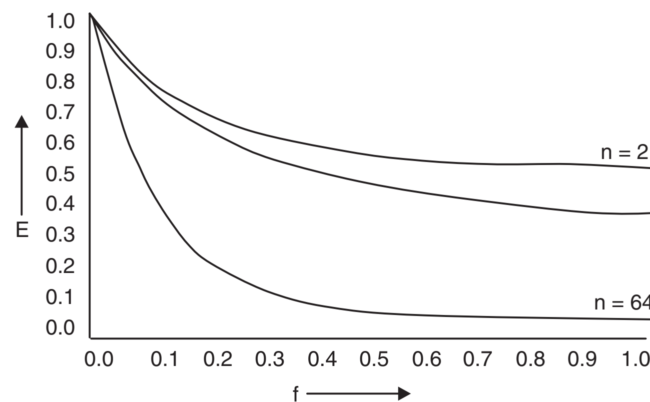


Fig. 1.20 Efficiency and the Sequential Fraction

The idea of using very large number of processors may thus be good only for specific applications for which it is known that the algorithms have a known very small sequential fraction f.

Minsky's Conjecture [Minsky 1970]: For a parallel computer with n processors, the speedup S shall be proportional to $\text{Log}_2\, n$.

This conjecture was stated from the way parallel computers give the overall performance in solving many problems. Proof of why parallel computers behave this way was first given by Flynn [1972]. This proof is limited to only SIMD parallel computers. The essence of Flynn's proof is given next using energy diagram for the SIMD computer. Basic argument of Flynn was based on the execution of the nested IF statement by a SIMD parallel computer. Consider the parallel if statement:

If C1 then E1
else E2.

Here C1 is the vector of Booleans and E1 and E2 are two statements. Every PE of a SIMD machine has one element of the condition vector C. If this condition is true, then it will execute the corresponding then part (E1 statement). PEs having false element of C1 shall execute the else part (E2 statement). For a SIMD computer, execution of the above has to be done sequentially. The first set of PEs (with true elements of C1) executes the then part. Other PEs are masked off from doing the work. They execute NOP. After the first set completes the execution, the second set (PEs having false elements of C1) shall execute the else part and the first set of PEs shall be idle. If there is another IF nested in the E1/E2 statements, one more division amongst the active PEs shall take place. This nesting may go on repetitively. Figure 1.21 shows the energy diagram for the PEs. This diagram assumes the division of true and false values in the condition vectors as half. The successive time slots shows that number of PEs working are getting reduced by a factor of two.

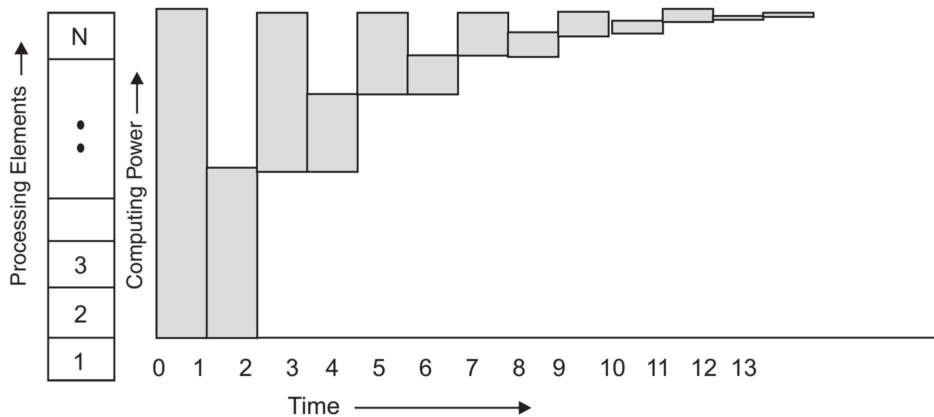


Fig. 1.21 Energy Diagram Showing Loss of Energy

The Minsky's conjecture was very bad for the proponents of the large scale parallel architectures. Flynn and Hennessy gave yet another formulation that is presented in the following Folk Theorem.

Folk Theorem 1.3: Speedup of a n processor parallel system is limited as

$$S \leq n/\log_2 n$$

Proof: Let T_1 = time taken by a single processor system,

f_k = probability of k PEs working simultaneously

(f_k can be thought of as program fraction with a degree of parallelism k)

T_n = Time taken by the parallel machine with n processors.

$T_n = [f_1/1 + f_2/2 + f_3/3 + .. + f_n/n\ ] .T_1$

The speedup $S < T_1/T_n = 1 /(f_1/1 + f_2/2 + f_3/3 + ... + f_n/n\ )$.

Assume (for simplicity) $f_k = 1/n$ for all $k = 1, 2,... n$ (*i.e.*, degree of parallelism is equal for all k),

$$T_1/T_n = 1/(1/n\ (1/1 + 1/2 + 1/3 + ... + 1/n))$$

Hence $$S \leq \frac{n}{\log_2 n}$$

A plot of speedup as a function of n for the above discussed bounds and the typical expected speedup (linear) is presented in Fig. 1.22. Lee [1980] presented an empirical table for speedup for different values of n. His observations are listed in Table 1.2. It is to be noted from Fig. 1.22 that the speedup curve, using Flynn's bound, for large values of n have a linear shape. This is so because, the function $n/\log n$ approximates to a straight line when $n > 1000$. The observations made by Lee are thus derivable from Folk Theorem 1.3. Figure 1.22 shows the speedup as a function of the number of processors. The studies on the execution time, the effective parallelism as function of the number of processors was studied by Jordon [1984] for HEP MIMD architecture. Figures 1.23 and 1.24 show the graphs depicting the asymptotic time reduction and the saturation of the available degree of parallelism for two important problems, viz., numerical integration and LU decomposition. Table 1.3 gives Linpack performance of some of the important machines of 80-90's [Dongara 1992].

Table 1.2 Speedup and the Number of Processors

Value of n (Number of Processors)	*Value of S (Speedup)*
1	O(n)
Few 10's up to 100	O(n log n)
Around 1000	O(log n) < S < O(n/log n)
10000 and above	O(n)

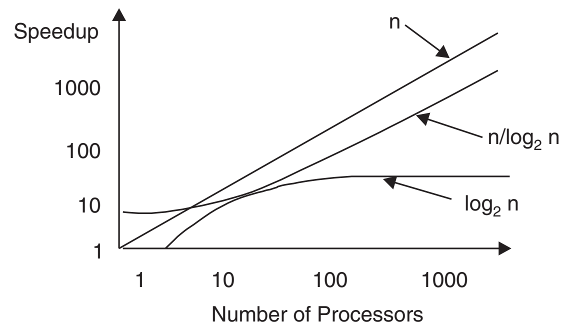


Fig. 1.22 Speedup as a function of the number of processors

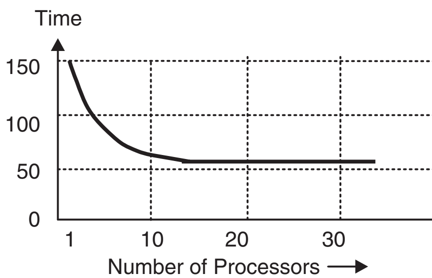


Fig. 1.23 Asymptotic Time Reduction

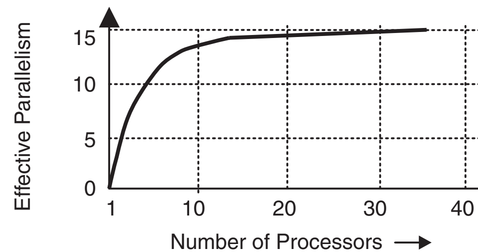


Fig. 1.24 Effective Parallelism

Table 1.3 Linkpack Performance of Some Computer [Dongara 1992]

Computer	*Performance Megaflops(Mflops)*		
	Problem Sizes		
	Peak	*100 × 100*	*1000 × 1000*
1. Cray YMP C90 4.2 ns clock CF77.5			
16 Processors	16000	479	9715
4 Processors	4000	388	3272
1 Processor	1000	387	874
2. Fujitsu VP2200 10.4 ns clock Fortran 77EX/VPV11L10	100	127	8423
3. IBM ES/9000 9 ns clock under VAST-2/VS Fortran	2664		1457
4. CDC Cyber 2000V Fortran V2		32	
5. IBM RISC System/6000-550 (41 MHz) under v2.2 xlf-P-Wp, ea 478	150	26	132
6. HP 9000/730 (66MHz) Under HP UX 8.05 f77 +OP4+O3	66	24	
7. nCUBE 2 1024 Processors	2400		258
256 Processors	602		165
16 Processors	38		26.1
1 Processor	2.35		2.02
8. Sun Sparc station 1 f77 1.3.1		1.4	
9. CDC 7600 CHAT			1.2
10. Inmos T 800(20 MHz) Fortran 3L		0.37	

Today's top machine IBM Blue gene was conceived as scalable 1 Million processor machine with 32 (2^5) Processors per chip, 64 (2^6) chips per board with a tower cabinet with 8 (2^3) boards and 64(2^6)towers, giving total of $2^{(5+6+3+6)} = 2^{20}$ = 1 million, and was planned 5 years back based on PowerPC architecture. An installed version today (in 2006) is faster than the sum of all the 500 top machines in 2001 and gives sustained 280.6 TeraFlops. Table 1.4 shows top three supercomputers in 2005 with their configurations and speeds in teraflops (1 Teraflop = 1000 MFLOPS), while Fig. 1.25 shows the performance trends of the supercomputers.

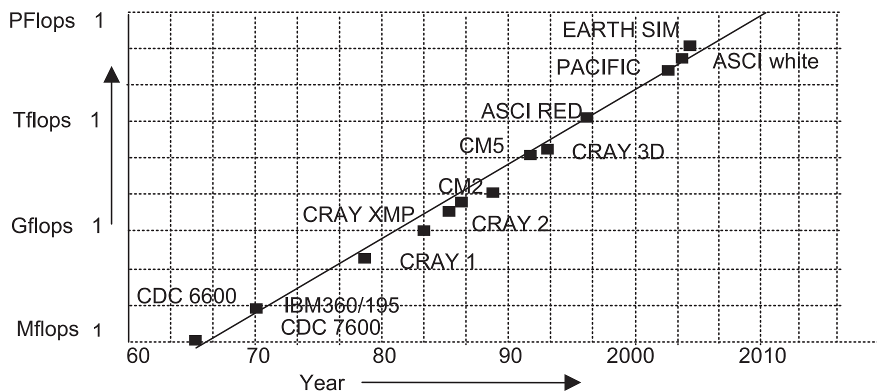


Fig. 1.25 Supercomputers Performance Trends

Table 1.4 The Quest for Higher Performance [IEEE 2005]

Name	*Purpose*	*No. of Processors, RAM and Disk capacity*	*Speed and cost*	*Technology*	*Full System*
IBM Blue Gene/L At LLNL California	Material and Nuclear simulation	32,768 proc's, 8 TB RAM , 28 TB disk storage Linux and Custom OS	**71 TFLOPS**, $100 M	IBM Dual-proc. Power chips (10-15 W power)	130k-proc. 360 TFLOPS
SGI Columbia NASA Ames Lab	Aerospace simulation, climate research	10,240 proc's, 20 TB, 440 TB disk storage Linux	**52 TFLOPS**, $50 M	20x Altix (512 Itanium2) linked by by Infiniband	
NEC Earth Sim Center Yokohama	Atmospheric, oceanic, and earth	5,120 proc's, 10 TB, 700 TB disk storage Linux	**36 TFLOPS** $400 M	Built of custom vector microprocessors	

1.6 PERFORMANCE METRICS FOR PROCESSORS

Performance metrics are the measures (numbers) to quantify the processor's performance and hence are useful in making comparisons. Execution time of the running jobs, from a mix of jobs, on a processor as measure can be useful. In fact, today, where all kinds of peak performance based on the clock speed is being pushed by the manufacturers; the execution time is the real measure of the performance of a processor.

Arithmetic Mean

It provides a simple average for the time T_A taken by n different programs run on the system. These programs typically characterize the job mix of the user organization.

$$T_A = \left(\sum_{k=1}^{n} P_k\ T_k\right)/n$$

where T_k is the runtime of a Job k whose probability of execution is P_k.

Weighted Arithmetic Mean

Since the frequency of running a job may be different for different jobs, a better measure is needed that takes this factor into account.

Let M_k be speed for k^{th} job and P_k = probability of running a job k whose run time is T_k. Weighted arithmetic mean is a better measure and may be defined as time T_W given by:

$$T_W = \left(\sum_{k=1}^{n} P_k\ T_k\right)/n.$$

If we do not have the execution times and instead the speeds are available (speed is treated as the inverse of the execution time), we could use speeds and obtain the harmonic mean S_H of the speeds. It is defined as:

$$S_H = n / \prod_{k=1}^{n} (P_k / M_k),$$

Geometric Mean Time Metric: R_G

When speeds are known as relative to the speed of some processor and not its absolute value, we define a normalized metric in such situations. In that case, a normalized (to the reference machine) metric could be defined. Geometric Mean R_G is defined as nth root of the product of all ratios, *i.e.*,

$$R_G = \sqrt[n]{\Pi R_k}, \; k = 1, 2, \ldots n, \text{ where } R_k = \text{Time on CPU/Time on reference CPU}$$

It exhibits a nice property, *i.e.*, Ratio of the means = Mean of the ratios (independent of running times of individual programs). Hence, this measure is better than arithmetic mean but still does not form accurate prediction model for the performance.

Amdahl's Law Revisited:

We formulate the overall speedup for a processor, which has the specific feature that makes it run certain instruction/s faster, due to hardware enhancements. These enhancements are useful only if these instruction/s form a part of program code.

Folk Theorem 1.4:

$$\text{Overall Speedup} = \frac{1}{(1 - \text{Fraction enhanced}) + \dfrac{\text{Fraction enhanced}}{\text{Speedup enhanced}}}$$

Proof:

Let F = fraction of instructions that use the enhanced feature (Fraction enhanced)

S_F = speedup enhanced for the fraction of instructions

T = execution time without enhancement

T_F = execution time with enhancement

T_F = time for the part without enhancement + time for the part with enhancement

$= (1 - F).T + F \,.\, T/S_F$

Overall Speedup = T / T_F

$$\text{Overall Speedup} = \frac{T}{(1 - F)\,.\,T + F\,.\,\dfrac{T}{S_F}}$$

$$\text{Overall Speedup} = \frac{1}{(1 - F) + \dfrac{F}{S_F}}$$

Example

A program has 50% of the code that refers to the main memory (RAM), out of which 95% refers to the cache. If we have RAM of 100 ns and cache of 10 ns., then $S_F = 10$. Fraction enhanced

will be 50% * 0.95 = 0.475 (We assume here that the cache time is predominant in the execution cycle of these 50% instructions. (Ignoring other cycles for simplicity), overall speedup of a processor with the cache is

$$= 1/(0.525 + 0.0475\ ) = 1/\ 0.5725 = 1.746$$

The CPU shall run 1.746 times faster with a cache than CPU with no cache.

Calculating CPU Performance

All commercial machines are synchronous with some clock tick driving its work, hence, a useful basic metric could be clock frequency in GHz (1 GHz = 1000 MHz). Thus, clock cycle time = 1/clock frequency (1 GHz gives clock cycle time of 1 nano-second).

$$\text{CPU Time} = \frac{\text{CPU Cycles for a program}}{\text{Clock Frequency}}$$

We tend to count the instructions executed, *i.e.* Instruction Count (IC). A program may have N instructions in its object code, but that is not IC., What is important, is the dynamic count of the instructions actually executed. (A program may skip certain instructions due to specified conditions, may repeat others due to loops etc). Clocks required per Instruction (CPI) is a figure of merit and is given by:

$$\text{CPI} = \frac{\text{CPU Clock cycles for a program}}{\text{IC}}$$

(The great RISC vs. CISC debate where RISC proponents say one clock per instruction for RISC machines in contrast to CISC).

$$\text{CPU time} = \text{IC} \times \text{CPI} \times \text{Cycle Time} = \frac{\text{IC} \times \text{CPI}}{\text{Clock Frequency}}$$

Cycle Time, CPI and IC

These are interdependent and making one better often makes another worse. Cycle time depends on hardware technology and organization. CPI depends on organization and Instruction Set Architecture (ISA), while IC depends on ISA and compiler technology. Often CPI's are easier to deal with on a per instruction basis.

Where should Architect Spend Resources?

The frequency of the various instructions supposed to run on the machine is counted. Based on this information, it is easy to work out where the designer requires to spend resources to get optimum benefits.

$$\text{CPU time} = \text{Cycle Time} \times \sum_{k=1}^{n} \text{CPI}_k \times \text{IC}_k$$

Consider a case shown in Table 1.5 that gives the instruction frequency (mix) for the specific applications on the processor and its current design giving the clock cycles utilized per instruction in these categories.

Clock cycles IC

Table 1.5 Processor Time Spent in Typical Mix of Instructions

Observed MIX			*Time spent (calculated)*	*Percentage*
Operation	*Frequency*	*CPI*	$CPI_k * IC_k$	*%Time spent*
ALU	50	1	0.5*1 = 0.5	0.5/1.5 = 33%
LOAD	20	2	0.2*2 = 0.4	0.4/1.5 = 27%
STORE	10	2	0.10*2 = 0.2	0.2/1.5 = 13%
BRANCH	20	2	0.20*2 = 0.4	0.4/1.5 = 27%
Total	100	Total	1.5	

Since technology may not allow all the improvements at once, they must come in some order of priority. From the table 1.5, it is clear what the improvement priority should be, i.e., first improve ALU, then improve Load from memory and then branch executions.

Overall Effects of Enhancements : An Example

Consider a processor and program run characteristics with current and improved version is shown below. Work out the speedup obtained by improvements.

Table 1.6 Example Program Mix

Instruction Type	*Instruction Count*	*Execution Time per Inst. (Current Version 1GHz)*	*Execution Time per Inst. (Improved Version 1.5GHz)*
Memory reference	20%	4 **clock cycles**	**1 clock cycles**
Arithmetic	50%	8 **clock cycles**	**2 clock cycles**
Branch	20%	3 **clock cycles**	**1 clock cycles**
Others	10%	1 **clock cycles**	**1 clock cycles**

Instruction Type	*IC*	*Total Execution Time (Current Version 1GHz)*	*Total Execution Time (Improved version 1.5GHz)*
Memory reference instructions	20%	0.2*4 = 0.8 Units	0.20 * 1 = 0.2 Units
Arithmetic instructions	50%	0.50*8 = 4.0 Units	0.50* 2 = 1.0 Units
Branch Instructions	20%	0.2*3 = 0.6 Units	0.20 * 1 = 0.2 Units
Other instructions	10%	0.1.*1 = 0.1 Units	0.1 * 1 = 0.1 Units
Total	100%	5.5 Units	1.5 Units

Units are different since clock rates are different

Speedup = Time taken by old version/time by new version

= (5.5% clock period of current version)/(1.5* clock period of improved version)

$$= \frac{5.5}{1.5} \times \frac{1.5}{1} = 5.5$$

Measuring Performance

The measurement of the performance is done by instruction level simulator and the timing analyzer. It is still difficult due to the memory effects and the dependencies and stalls in the pipe. MIPS (Million Instructions Per Second) are a good measure of performance for RISC machines (Today it is GIPS or TIPS, *i.e.*, Giga Instructions Per Second or Trillion Instructions Per second). In general we have to use Synthetic benchmarks (Mix of programs which are run) for predicting the real program performance. Parallel processor companies track the peak performance, making the users to believe that this would be the performance (like motor vehicle

Memory reference Arithmetic Branch

manufacturers specifying kilometers per liter of petrol. In the case of vehicles, the actual performance may go wrong by a factor of three or four, but in computers the performance could go wrong by a factor of 10 and may go wrong by as much as 100 times). The architects go by benchmarks. There are a number of benchmarks dealing with the processor's performance. The basic idea in all these is to model sample programs whose run on the machines will give idea of their real performance.

One can use actual application programs, OS Kernels, Toy benchmarks and benchmarks designed using known mix of codes. In 1992, benchmark SPECInt92 had six integer programs and SPECfp92 had 14 floating point programs. Programs SPECint95 had new set of programs. SPECint95 had 8 integer programs and SPECfp95 (with 10 floating point programs). Latest is SPEC 2000 (SPEC stands for System Performance Evaluation Cooperative).

1.5 PARALLEL PROGRAMMING MODELS

There are several parallel programming models in common use:

(*a*) Shared Memory

(*b*) Message Passing

(*c*) Threads

(*d*) Data Parallel

Parallel programming models exist as an abstraction above the hardware and the memory architectures. These models are not specific to a particular type of machine or memory architecture, but are abstractions that could be built over any hardware. If the abstraction and the hardware have a wide gap, the program will suffer from exceedingly large access time. For example, a shared memory abstraction could be built over the message passing hardware, but it will have large access time for the shared global memory access. However, it could appear to the user as a single shared memory (global address space). Generically, this approach is referred to as "virtual shared memory". Conversely, a message passing abstraction on a shared memory machine such as MPI (Message Passing Interface) on SGI Origin. It employs the Closely Coupled Non-Uniform Memory Access(CNUMA) type of shared memory architecture, where every task has direct access to global memory.

The decision regarding which model to use is often a combination of personal choice and the availability. There is no "best" model, although there certainly are better implementations of some models over others. One must look at the cost (time spent in data access) of using the model on the underlying architecture.

Shared Memory Model

In the shared-memory programming model, tasks share a common address space, which they read and write asynchronously. Various mechanisms like locks/ semaphores are used to control the access to the shared memory. An advantage of this model from the programming point of view is that the notion of data "ownership" is not present, so there is no need to explicitly specify the communication of data between the tasks. This leads to simplified program development. Major disadvantage is due to the poor locality understanding and managing data locally. On shared memory platforms, the native compilers translate user program variables into actual memory addresses, which are global.

Threads Model

The essence of the thread model is shown in Fig. 1.26. Main program P is scheduled to run by the OS. P loads and gets all the system and the user resources so as to run. P runs its code and may create number of threads as entities that run in parallel. Threads can be scheduled and run by OS concurrently. Each thread may have a local data. A thread also shares the entire resources of P including global memory of P.

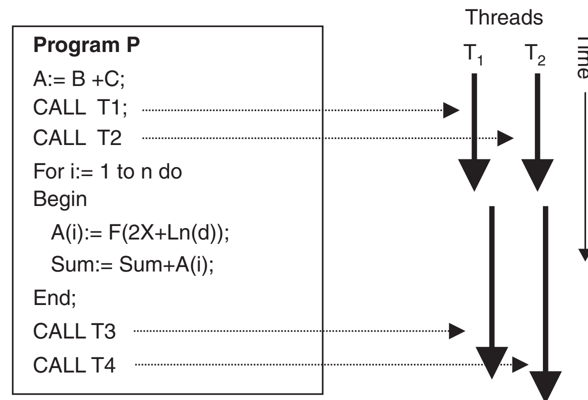


Fig. 1.26 Threads Programming Model

A thread's work may best be described as a subprogram within the main program. Any thread can execute any subroutine at the same time as the other threads. Threads communicate with each other through global memory (updating addressed locations). This requires synchronization constructs to ensure that more than one thread is not updating the same global address at any time. Threads can come and go, but P remains present to provide the necessary shared resources until the application run has completed. Threads are commonly associated with shared memory architectures and also the operating systems and today Sun Microsystem's Java programming language has excellent thread support.

Implementations:

Historically, hardware vendors have implemented their own proprietary versions of threads. These implementations differed substantially from each other making it difficult for programmers to develop portable threaded applications. Independent standardization efforts have resulted in two very different implementations of threads, viz: Posix Threads and OpenMP.

POSIX Threads

It is a set of library functions (an IEEE standard called IEEE POSIX 1003.1c standard 1995) to be used in C programs. It has support to create threads that run concurrently. The threads created by a process share all the resources of the process, other support includes joining and destroying the threads, synchronize the threads if required along with the other support required for parallel programming.

OpenMP

OpenMP is an Application Program Interface (API), jointly defined by a group of major computer hardware and software vendors. OpenMP provides a portable, scalable model for developers of shared memory parallel applications. The API supports C/C++ and FORTRAN on most of the architectures, including UNIX & Windows NT. The major features of OpenMP are its various constructs and directives for specifying parallel regions, work sharing, synchronization and data environment.

Message Passing Model

Multiple tasks can reside on the same physical machine as well across an arbitrary number of machines. Tasks exchange data through communications by sending and receiving messages(Fig. 1. 27). Data transfer usually requires cooperative operations to be performed by each process. For example, a send operation must have a matching receive operation. From a programming point of view, message passing implementations commonly comprise a library of subroutines that are embedded in the source code. The programmer determines all the parallelism required.

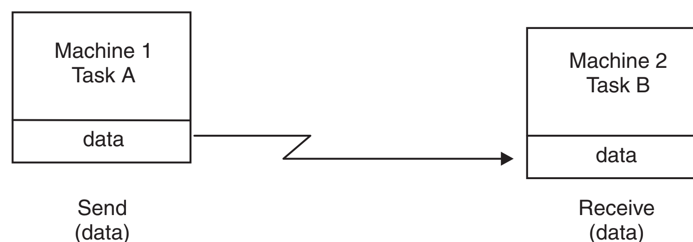


Fig. 1.27 Message Passing Model

A variety of message passing libraries have been available since last 25 years but implementations differed from each other making it difficult for programmers to develop portable applications. The MPI (Message Passing Interface) Forum was formed in 1992 to recommend a standard interface for message passing implementations. Message Passing Interface, MPI part-1 was released in 1994 while MPI-2 was released in 1996. These are available on the website www.mcs.anl.gov/projects/mpi/standard.html. MPI is now the industry standard for message passing, replacing most of the other message passing implementations. MPI implementations usually don't use a network for task communications. Instead, they use shared memory (memory copies) for performance reasons since MPI was designed for high performance on both, the massively parallel machines and also on the workstation clusters.

Data Parallel Model

Parallel computing work in this model focuses on performing operations on a data set. The data set is organized into a common structure, such as an array. A set of tasks work collectively on the same data structure, each task working on part of it. Tasks perform the same operation on their part of work.

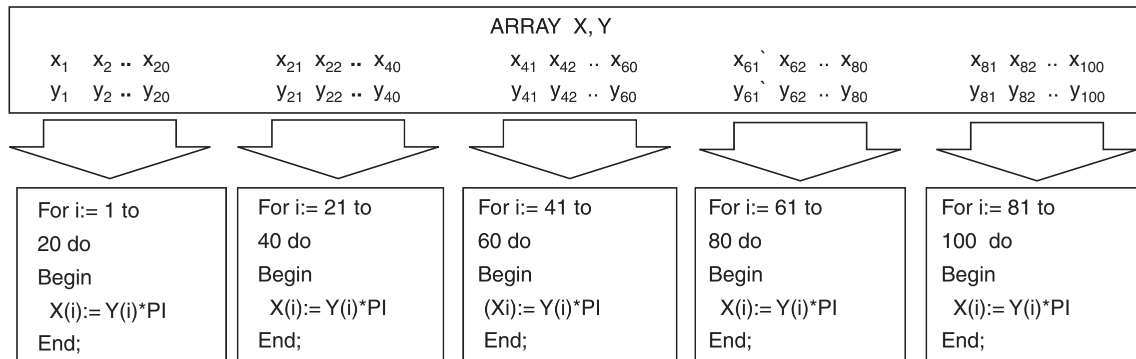


Fig. 1.28 Data Parallel Model

For example, create an array X by multiplying Y elements by value of PI as shown in Fig.1.28, where each processing element works on 20 distinct elements of the arrays. In shared memory architectures, all the tasks have access to the data structure through global memory. On the other hand, in the distributed memory architectures, the data structure is split up in pieces, each piece resides in the local memory of a task. Programming the data parallel model is accomplished by writing a program with data parallel constructs. The constructs can be calls to a data parallel subroutine library or, compiler directives recognized by a data parallel compiler. High Performance FORTRAN (HPF) provides extensions to Fortran 90 to support data parallel programming.

1.7 PARALLEL ALGORITHMS

Studies in the design of parallel algorithms are interesting. Any algorithm development and analysis studies involve how best one can do the given job. Sequential algorithms have been extensively studied for several important problems. One important measure of the performance of any algorithm is the time complexity. It is the time required to execute the algorithm specified as some function of the problem size. Another important performance measure is the space complexity defined as memory required by the algorithm. It is also specified as some function of the problem size. Many times, the time and space complexities depend on the data structure used. Another important measure therefore, is the preprocessing time complexity to generate the desired data structure. Algorithms giving the best figures for the above measures naturally shall be preferred. To determine the theoretical lower bound on the time and space complexities is an important research activity.

Parallel algorithms are the algorithms to be run on parallel machines. The most important measure of performance of a parallel algorithm is how fast one can solve a given problem using as many processors as required. Since the parallel algorithm uses several processors, the complexity of communication among the processors is also an important measure. The communication complexity in turn depends on the communication hardware supported by the machine. Therefore, an algorithm may fare badly on one machine and do much better on another. Due to this reason, the mapping of the algorithms on architectures is an important activity in the study of parallel algorithms. Speedup and efficiency are also important

performance measures for a parallel algorithm when mapped on to a given architecture. A parallel algorithm for a given problem may be developed using one or more of the following:

1. Detect and exploit the inherent parallelism available in the existing sequential algorithm.
2. Independently invent a new parallel algorithm.
3. Adapt an existing parallel algorithm that solves a similar problem.

Studies on the structure of problems to be solved mechanically are an exciting field since ancient times. These studies provide an insight into the problem structure and its properties, which are useful in many ways including those for finding the parallelism available. Many times the internal structure of the problem solving process is so complex that people have to work for centuries to gain an insight into the problem. Four color problems, linear programming problems are the few problems that are being extensively studied. Some of the important problems that have been extensively studied for parallel algorithms include sorting, Fast Fourier Transform (FFT), search and graph problems and matrix operations.

1.8 DISTRIBUTED PROCESSING

Distributed processing is also a form of parallel processing but for a different purpose. It has a different working model of the computers and the computing process. Parallel computers are built to reduce the computation time of a given problem. Distributed computing services are used primarily for the reasons of physical distribution of sub-jobs. For example, a banking application requires the processing of transactions generated through the distant teller machines. Central important idea of distributed computing environment is to provide convenience of computing and interaction with the system. Of course, due to the convenience given to the users, the time required to do a job in the overall system reduces drastically. Since computing facility could be composed of a number of computing nodes linked through a local/ wide area network, the communication cost and the way the communication is done are important. Moreover, the system integrity under node and or link failures must be maintained. These systems thus need to have a different look at the computing process unlike the parallel computing system which assumes close coupling between the processors, and the communication failures etc., are not of much importance in the design of the system hardware and software. However, the cost of communication in these should also be modeled.

There are several fundamental problems in distributed processing. These problems arise because of the various fundamental processes like time uncertainty (due to differing local (to nodes) times in the system clocks), incomplete information about others (nodes and links) in the system, duplicate information which may not be consistent at all the times, several managers supposedly working independently and still cooperating. It is to be noted that many abstract models of concurrency in the distributed system and many of the solutions to fundamental problems, some of them mentioned above, will be useful in understanding the abstract distributed computing process better.

Abstract concurrent programming systems have been studied extensively since late 60's. Some of the fundamental problems in these systems were debated and required a decade [Dijkstra 1965, Knuth 1966 and Hoare 1978] to find the satisfactory solutions. Looking at the

parallel computing process in an abstract way is highly challenging and we have just made a beginning in late 90's in solving problems in distributed systems for the parallel and distributed programs.

Developments in parallel and distributed architectures and parallel algorithms continue at increasing pace. There are several publications on this subject. Readers should find articles on parallel computing in ACM computing surveys, ACM Transaction on Programming languages and Systems, IEEE Computer, Communications of ACM, IEEE Transactions on Computers, Journal of Parallel Programming, Journal of Parallel and Distributed Programming, Parallel Computing, ACM Transactions on Computer Systems, Journal of Parallel and Distributed Computing (Academic Press), Journal of Parallel and Distributed Computing (Elsevier) and International Journal of High Performance Systems Architecture (IJHPSA) to begin publication by January 2007, by Interscience publishers. There are number of special issues on Communications of ACM and IEEE computer devoted to parallel computing or related topics. Other interested publication is IEEE Micro.

EXERCISES

1.1 Prove that n stage pipeline processor can be at most n times faster than a corresponding non-pipelined serial processor.

1.2 Give examples of granularities of parallelism known to you.

1.3 Prove Amdahl's law.

1.4 Why the speedup is approximately linear for n processor parallel system with large values of n.

1.5 Give two reasons why Amdahl's law should hold? Is it possible that there are cases where the law may fail? If yes, explain.

1.6 Why could Folk theorem 1.1 fail?

1.7 Is Minsky's conjecture true and under what conditions?

1.8 Do Flynn's classification scheme fail for pipelined computers? Explain.

1.9 Do you find any machine that does not fit into Shore's classification scheme? Explain.

1.10 Why do you need parallel computers? Give one analogy to explain your answer.

1.11 Do you believe that technology development will give boost to computing speed rather than parallel processing in near future?

1.12 Why do you think that the parallel processing with large number of nodes has been still in infancy as far as general purpose computing is concerned?

1.13 How fast can one add n numbers assuming that the number of adders available is unlimited?

1.14 A processor had no floating point unit in earlier version but was added to it later. The speedup for floating point operations is 500 compared to software implementation of floating point routines. Find the speedup for programs that has floating point operations in the original machine consuming 10%, 20% and 90% of time.

1.15 A machine is run on many applications and the instruction mix is collected. Loads/Store are 10%, Integer add/sub 15%, FP add/sub 50%, FP multiply divide 5% and others 5% and Branches 15%. The clock cycles consumed by these instructions are: Loads 2, Integer add/sub 1, FP add/sub 5, FP mult/divide 20, others 1. Find which component of the architecture requires enhancement first. After incorporating the enhancement which makes clock cycle requirement as 2, find the overall speedup.

1.16 Answer TRUE /FALSE with one line justification.

(*a*) A 5 processor system gives higher efficiency than a 2 processor system for a given value of sequential computing fraction.

(*b*) A program with sequential computing fraction of 0.2, run on a parallel system having 100 processors gives the speedup greater than 8.

1.17 Fill in the blanks:

(*a*) To obtain a speedup of 4, a 5-stage pipe requires to run —— tasks.

(*b*) Amdahl's law gives possible maximum speedup as ————

(*c*) Pipeline P1 has 8 stages, while pipeline P2 has 7 stages, the speedup of chained pipe P1,P2 for 136 tasks is ————-

1.18 A program has following fractions of computation with degrees of parallelism as below

f_1 sequential fraction = 0.10

f_3 parallel fraction of degree of parallelism 3 = 0.60

f_6 parallel fraction of degree of parallelism 6 = 0.30

All other fractions are = 0.00.

A machine with 8 processors is used to run the program. Draw an energy diagram for machine. Using energy diagram, find the efficiency and speedup for the program. (no other formula or derivation to be used).

1.19 A processor and program run characteristics with current and improved version is shown below. Work out the speed up obtained by improvements.

Instruction Type	*Instruction Count*	*Execution Time (Current version 1GHz)*	*Execution Time (Improved version 1.5GHz)*
Memory reference instructions	20%	4 clock cycles	1 clock cycles
Arithmetic instructions	50%	8 clock cycles	2 clock cycles
Branch Instructions	20%	3 clock cycles	1 clock cycles
Other instructions	10%	1 clock cycles	1 clock cycles

1.20 A program has following fractions of computation with degrees of parallelism as:

f_1 sequential fraction = 0.10

f_2 parallel fraction of degree 2 = 0.20

f_3 parallel fraction of degree 3 = 0.20

f_{10} parallel fraction of degree 10 = 0.50

All other fractions are = 0.00.

Let

Machine M_1 be a single PE with 100 MOPS speed and

Machine M_2 be a 10 PE machine with each PE having 10 MOPS speed.

(*a*) Which of the machines M_1 and M_2 take less time to complete the program ?

(*b*) Draw the energy diagrams for running the problem on machines M_1 and M_2 and from it find the efficiencies for machines M_1 and M_2 in solving the problem.

REFERENCES

Amdahl, G. [1967], "Validity of Single Processor Approach to Achieving Large Scale Computing Capabilities", Proceedings AFIPS Computer Conference, Vol. 30, 1967, pp. 483-485.

Barnes, G.H., et al. [1968], "The ILLIAC IV Computer,", IEEE Trans on Computers, August 1968, pp. 746-757.

Batcher, K.E.[1980], "Design of Massively Parallel Processor", IEEE Trans on Computers, C-29, Sept. 1980, pp. 836-840.

Bhattacharya, M.K. and Bhujade, M. R. (Guide) [1989], "Neural Network Simulation Studies and Usage", M. Tech. Dissertation, Computer Science and Engineering, Indian Institute of Technology, Bombay, 1989.

Bhujade, M. R. [1987], "Lecture Notes on Parallel Computing", Computer Science and Engineering, Indian Institute of Technology, Bombay, 1987.

Davis, R. and Thomas Dave, "Systolic Array Chip Matches the Pace of High Speed Processing", Electronic Design, October 1984.

Dijkstra, E.W. [1965], "The Solution of a Problem in Concurrent Programming Control", Communication of ACM, Vol.8, Sept. 1965, p. 569.

Dennis, J.B. and Misunas, D.P.[1975], "A Preliminary Architecture for a Basic Data Flow Processor", Proc. IEEE 2nd International Symp. on Computer Architecture, Jan. 1975, pp. 126-132.

Dennis, J.B, [1980], "Data Flow Supercomputers", IEEE Computer, Nov.1980, pp.48-56.

Dongarra, J. [1992], "Performance of Various Computers Using Standard Linear Equation Software", ACM CAN March 1992.

Duncan, R. [1990], "A Survey of Parallel Computer Architectures", IEEE Computer, Vol.23, Feb. 1990, pp. 5-16.

Erico Guizzo [2005], IBM Reclaims Supercomputer Lead , IEEE Spectrum, Feb. 2005, pp. 15-16

Feng, T.Y. [1972], "Some Characteristics of Associative/Parallel Processing", Proc.1972 Sigmore Computer Conference, Syracuse University, NY 1972 pp. 5-16.

Flynn, M. J. [1966], "Very High Speed Computer System", Proceedings of IEEE, Vol.54, pp. 1901-1909.

Grossberg, S. [1988], "Neural Network Research: From Personal Perspective", Electronic Engineering Times, March 7,1988, A12.

Grossberg, S. [1986], "Competitive Learning From Interactive Activation to Adaptive Resonance", Cognitive Science vol. 11 pp. 23-63.

Helmbold, D. P and McDowell, C.E. [1990], "Modeling Speed up (n) Greater Than n", IEEE Trans. On Parallel and Distributed Systems, Vol.1, No.2, April 1990, pp. 250-256.

Higbie, L. C.[1973], "Supercomputer Architecture", IEEE Transactions on Computer, December 1973, pp. 48-58.

Hillis, W.D., "The Connection Machine, MIT Press, Cambridge, Mass. 1985.

Hoare CAR [1978], "Communicating Sequential Processes," Communications of ACM, Vol.21, August 1978, pp. 666-677.

Holmgren, Don [2002], "Comparison of AMD MP and Xeon Performance", http://lqcd.fnal.gov/benchmarks/amd_xeon.html last modified 15th Feb. 2002.

Hopfield, J. J. and Tank, D. W. [1986], "Computing with Neural Circuits: A Model", Science Vol. 233, August 1986, pp. 625-633.

Hutcheson, G. Dan [2004], "The First Nano Chips", Scientific American, March 2004.

Johnson, Eric E., "Completing an MIMD Multiprocessor Taxonomy", ACM, Computer Architecture News, Volume 16 , Issue 3 (June 1988) pp. 44-47.

Jordon, H. F. 1984], "Experience with Pipelined Multiplier Instruction Streams", Proceedings of IEEE, Vol. 72, No.1, January 1984, pp. 113-123.

Knuth, D. E. [1966], "Additional Comments on a Problem in Concurrent Programming Control, Communications of ACM, May 1966, pp. 321-322.

Kohonen, T. [1978], “Associative Memory: A System Theoretical Approach” , Springer Verlag, New York, 1978.

Kuck, D. J. and Strokes, R. A. [1982], “The Burroughs Scientific Processor (BSP)”, IEEE Trans. on Computers, Vol. C-31, May 1982, pp. 363-376.

Lee, R. [1980], “Empirical Results on Speed Efficiency, Redundancy and Quality of Parallel Computations, International on Parallel Processing, August 1980, pp. 91-96.

Minsky, M. [1970], “Form and Content in Computer Science”, ACM Turing Lecture, JACM, Vol. 17, 1970, pp. 197-215.

Parkinson D. and John Lin(Ed) [1990], Parallel Commuting with DAP, Pitmann London, MIT Press, 1990.

Parkinson D. [1986], “Parallel Efficiency Can be Greater Than Unity, Parallel Processing, Vol. 3, 1986, pp. 261-262.

Reddaway, S. F. [1979], “DAP A Distributed Array Processor”, 1st IEEE ACM Annual Symposium on Computer Architecture, Florida, 1979.

Russell, R. M [1978], “The CRAY 1 Computer System”, Communications of ACM, January 1978, pp. 63-72.

Stone, J. E. [1973], “Second Thoughts on Parallel Processing”, International Journal of Computer and Electrical Engineering, Vol. 1, 1973, pp. 95-109.

SIPs, H. J. [1984], “Bit Sequential Arithmetic for Parallel Computers, IEEE Trans on Computers, January 1984, pp. 7-20.

Skillicom, D. B. [1988], “A Taxonomy for Computer Architectures”, IEEE Computer, Vol. 21, November 1988, pp. 46-57.

Thurber, K. J. [1976], “Large Scale Computer Architecture Parallel and Associative Processors”, Hayden Book Company, NJ, 1976.

Thurber, K. J. [1979a], “Parallel Processor Architecture Part I: General Purpose Systems”, Computer Design, January 1979, pp. 89-97.

Thurber, K. J. [1979b], “Parallel Processor Architecture Part II: Special Purpose Systems”, Computer Design, February 1979, pp. 103-114.

Thurber, K. J. and Walad, L. D. [1975], “Associative and Parallel Processors, ACM Computing Survey Vol.7, Dec. 1975, pp. 215-255.

Tomasulo, R. M.[1967], “An Efficient Algorithm for Exploiting Multiple Arithmetic Units”, IBM Journal of Research and Development, January 1967, pp. 25-33.

Victor V. Zhirnov, et al.[2003], “Limits to Binary Logic Switch Scaling-A Gedanken Model”, Proceedings of IEEE Vol. 91, No.11, Nov. 2003, pp. 1934-1938.

Watson, W. J. [1972], “The TI-ASC, A Highly Modular and Flexible Supercomputer Architecture, Proc. AFIPS Fall Joint Computer Conference, AFIPS Press, NJ, 1972 pp. 221-228.