CHAPTER-I
INTRODUCTION

We being by discussing a number of examples of congestion. The things common to all the systems that we shall consider is a flow of customers requiring service/there being some restriction on the service that can be provided. For example, the customer may be aircraft requiring to take-off, the restriction on ‘service’ being that only one aircraft can be on the runway at a time. Or, the customers may be patients arriving at an out-patients clinic to see a doctor. The restriction on services is again that only one customer can be served at a time, and, infact, these two examples are both cases of what we shall call the single server queue. An example of a multi server queue is a queue for buying stamps at a British main post office, or for having goods checked at a super market. Here the restriction is that not more than, say in customary can be served at a time.

Some other examples of systems where the number of customers that can be served at a time is restricted, are the following:

(a) articles pass along a conveyor belt and are to be packed into boxes,
(b) machines stop from time to time and require attention by an operative before restarting, the operative being able to attend to only one machine at a time.
(c) numerous problems connected with telephone exchanges;
(d) items of work come into, say, a testing department or a typing pool the number of times that can be dealt with being limited by the number of workers in the department.

In the examples given so far, the restriction on service is that not more than a limited number of customers can be served at a time, and congestion arises because the assured customers must queue up and await their turn for services. Sometimes, however the restriction is that service is only available during limited periods, during these periods there may or may not be a limit on the number of customers that can be served at a time. For example, if the customers are pedestrians waiting to cross a by road at an uncontrolled. Crossing, service is available only when a sufficient gab develops in the traffic; when this occurs a large number of customers may be able to cross simultaneously. It the customers are car waiting to move at traffic lights, or waiting to move from a minor road into a major road, service is restriction both by the need to wait
until a 'free period' occurs, and by the need to wait until previous customers in
the queue have been served i.e. we have a mixture of the two types of restriction
on services. Another example is a queue of people queueing a taxi; here, whether
or not a particular customer can be served immediately on arrival depends on
whether or nor taxi are free in the rank.

Another type of application, closely related to the single-server queue,
concerns the size of stores suppose that the customers, in the previous
discussion, are items placed from time to time in a store, but that not more than
a limited number of item can be stored at once. The service of a customer is the
withdrawal of an item from the store for use. We are interested in the relation
between the maximum context of the store and the chance that the store will be
found empty, when a call for a fresh item is made.

1. QUEUEING MODEL CONCEPTS:

A flow of customers from infinite/finite peoples towards the service
facility forms a queue. We can say Queue as a waiting line also because on
account of lack of capability to serve the customers all at a time.

By the term 'customer' we mean the arriving unit that requires some
services to be performed. The customer may be of persons, machines, computers,
vehicles, parts etc. Queues stands for a number of customers waiting to be
serviced. The Queue does not include the customer being serviced.

Characteristics of Queueing System :

A queueing system is specified completely by the following five basic
characteristics :

The Input Process :- It expresses the mode of arrival of customer at the service
facility governed by some probability law. The number of customers emanates
from finite or infinite sources. The customers may arrive at the service facility
in batches of fixed six or a variable size or one by one. When more than one
arrival is allowed to enter the system simultaneously batches. It is also necessary
to now the reaction of a customer of a customer upon entering the system. A
customer may decide to wait no matter how long the queue becomes, or if the
queue is too long to suit him, may decide not to enter it. If a customer decides
not to enter the queue because of it's huge length, he is said to have balked.

On the other hand, a customer may enter the queue, but after some time
loses patience and decides to leave. In this case be is said to have reneged. In the
case when there are two or more parallel queues, the customer may more from 
one queue to another for his personal economic gains, that is jockey for position.

The final factor to be considered regarding the input process is the manner 
in which the arrival patterns change with time. The input process which does not 
change with time is called a stationary input process. If it is time dependent 
then the process is termed as transient.

**The Queue Discipline:** – It is a rule according to which customer are selected 
for service when a queue has been formed. The most common discipline is the 
“first in, first out” (FIFO) rule under which the customers are serviced in the 
strict order of their arrivals. Other queue disciplines include: “last in, first out” 
(LIFO) rule according to which the last arrival in the system is serviced first, 
“selection for service in random order” (SIRO) rule according to which the 
arrivals as serviced the randomly irrespective of their arrivals in the systems; 
and a variety of priority schemes – according to which a customers service is 
done n preference over some other customer’s services.

Under priority discipline, the service is of two types. In the first, which 
is called pre-emptive, the customers of high priority are given service over the 
low priority customer. In the second type, called the non-pre-emptive, a customer 
of low priority is serviced before a customer of high priority is entertained for 
service.

In the case of parallel channels “fastest server rule” (FSR) is adopted. 
For its discussion we suppose that the customer arrive before parallel service 
channels. If only one service channel is free, then incoming customer is assigned 
to free service channel. But it will be more efficient to assume that an incoming 
customer is to be assigned a service of largest service rate among the free ones.

**The Service Mechanism:** : This means the arrangement of server is facility to 
server the customers. If there are infinite number of server then all the customer 
are served instantaneously on arrival and there will be no queue. If the number 
of servers is finite, then the customers are served according to a specific order. 
Further, the customers may be served in batches of fixed size of variable size 
rather than individually by the same server, such as a computer with parallel 
processing or people boarding a bus. The service system in this case is termed 
as bulk service system. Some times, the service rate may also depend on the 
number of customers, waiting for service. The situation in which service depends
upon the number of waiting customers is referred to as a state dependent system.

The capacity of the system: some of the queueing processes admit the physical limitation to the amount of waiting room; so that when the waiting line reaches certain length, no further customers are allowed to enter until space becomes available by a service completion. Such types of situation are referred to as finite source queues. That is, there is a finite limit to the maximum queue size.

Service Channels: — When there are several service channels available to provide service, much depends upon their arrangements. They may be arranged in parallel or in series or a more complex combination of both depending on the design of the system’s service mechanism.

By parallel channels we means a number of channels providing identical service facilities so that several customers may be serviced simultaneously. Customer may wait in a single queue until one of the service channels is ready to serve.

For series channels, a customer must pass successively through all the ordered channels before service is completed. The situation may be seen in public officer where part of the service are done at different service counters.

A queueing system is called a one-server model when the system has one server only and a multiple server model when the system has a number of parallel channels each with one server.

1.2 Parallel Processing Concepts: The following introduces you to significant concepts concerning parallel processing and it’s efficient realization within a number of different types of hardware and software environments.

Overview and Advantages of Parallel Processing: Parallel processing is the use of multiple processors to execute different parts of the same program simultaneously. The main goal of parallel processing is to reduce wall-clock time.

Imagine yourself having to order a deck of playing cards: a typical solution would be to first order them by suit, and then by rank-order within each suit”. If there were two of you doing this, you could split the deck between you and both could follow the above strategies, combining your partial solution at the end; or one of your could sort by suit, and the other by order within the suits,
both of you working simultaneously. Both of these scenarios are examples of the application of parallel processing to a particular task, and the reason for doing so is very simple: reduce the amount of time before achieving a solution. You can use this analogy to see indications of both the power and the weakness of the parallel approach, by taking it gradually to its extreme: as you increase the number of helpers involved in a particular task you will generally experience in characteristic speed up curve demonstrating how up-to-certain-number of helpers is beneficial but any over that simply get in each other’s way and reduce the overall time to completion. Consider, for e.g., how little it would help to have 52 people crowding around a table each responsible for putting one particular card into its proper place in the deck → this is exactly what is meant by the adage too many coops spoil the both.

Advantages of Parallel Processing:

1. **Reduce Wall Clock Time**: You will notice that it is not simply time that is being reduced, but wall-clock-time, other kinds of “time” could have been emphasized, for example CPU time, which is a count of the exact number of CPU cycles spent working on you job, but wall-clock is considered to be the most significant because it is what the researcher want to spend as little of as possible waiting for the solution: your own time. What is considered to be “acceptable” differs with the situation, and invalues characteristics of the user, the particular code being run and the system it is being run on. But in all case, it is safe to assume that the user is generally going to be more pleased the faster a solution is produced.

2. **Cheapest possible Solution Strategy**: Running programme cost money; different ways of achieving the same solution could have significantly different costs. If you are in a fiscally tight situation, you may have no reasonable recourse to a parallel strategy if it costs more than your budget allows. At the same time, you may find that running your program in parallel across a large number of workstation – type computers could cost considerably less than submitting it to a large, mainframe – style mega-beast.

3. **Local Versus Non-Local Resources**: “Locality” here usually refers to either “geographical locality” or “political locality”. The former is just another way of saying that you want all of your processes to “be” “close” to one another in terms of communications; the latter indicates that you
only want to use resources that are administratively open to you. Both can have a bearing on “cost”, e.g., the more communication latency incurred by your application, the longer it will run, and the more you might be charged, while use of resources “owned” by other organizations may also carry charges.

4. **Memory Constraints**: One of the first resources to get exhausted is local memory – especially for Grand Challenge level projects, the amount of memory available to a single system is rarely going to be sufficient to the computation and data-strong needs encountered during application runs.

The situation is greatly alleviated by having access to the aggregate memory made available by distributed computing environment: main memory requirements can be spread across the various processors engaged in the cooperative computation, and long term storage (tape and disk) can be accommodated at different cities (indeed, data security can be enhanced by arranging for multiple copies to be maintained at distinct locations in the distributed environment).

1.3 **Parallel Computation Paradigms**: Defining paradigms hard relevance in guiding our thoughts in a determined logical conclusion. At abstract level following these broad paradigms we determine how a parallel program can eventually be implemented and on which machine model. A parallel computation can progress either in synchronous or in asynchronous manner.

**Synchronous Computation** : One of the main concern in any form of computation is to ensure that required items of data reach at right place in right time to interact with each other. In addition to this a synchronous parallel computation progress with a global synchronization mechanism”. This implies that the operation on different items of data take exactly the same time for the same operation.

**1. Pipeline Computation** : Pipelining one of the most primitive forms of synchronous computation. This is illustrated by using an example of pipeline integer adder.

Suppose, we want to add a pair of integer vectors \((A_1, A_2, \ldots)\) and \((B_1, B_2, \ldots)\), where \(A_i = (a_{i1}, a_{i2}, \ldots a_{in})\) and \(B_i = (b_{i1}, b_{i2}, \ldots b_{in})\) in their finary representations. The pipeline integer adder for \(k = 3\) is illustrated in the figure 1.1
The value \( a_{ij} \) and \( b_{ij} \) march towards the pipeline processors as shown. The value of each of the initial carry \( (c_{0i}) \) entering rightmost processor is 0. The bits \( a_{ij}, j = 0, 1, 2 \) enter from above and march towards the processor with a time lag of one cycle between consecutive bits. Similarly, the bits \( b_{ij}, j = 0, 1, 2 \), enter the process through the input line at bottom by a time lag of one cycle as above. A cycle is equal to the time taken by a processor of the pipeline to sum three bits arriving at its three input lines and output the sum and the carry at its two output lines as depicted in the fig 1.1. In a pipeline computation a steady state is reached when all stage of the pipeline are full.

Once the steady state is reached one output is produced per cycle. Therefore, pipeline parallelism is effective for handling batches of data when operations on them can be proper down into distinct sub-operations.

2. SIMD Computation: – In a SIMD computer all enabled processors execute the same instruction simultaneously on different data values. These identical hence take same time for an operation. The major issue in SIMD computation is that of partitioning data and distributing partitions evenly across the processors. Following this the processing is done in parallel by all processing. Therefore SIMD computation is said to exhibit data parallelism. One of the simple example where we ca use SIMD computation is finding the sum of a set of numbers which are initially distributed over the processors. The sum is computed by each processor for the subset of numbers assigned to it. These intermediate sums are added by the cooperation of pairs of processors in a finery tree structured fashion. As number of nodes in a full binary tree at a level is equal to half that of total number of nodes at one level down, the number of intermediate sums generated at each iteration reduces by a factor of two over the previous
interaction. If there are $n$ number to be summed with $p$ processors using data parallelism, it requires $O\left(\frac{n}{p} + \log p\right)$ units of time discounting the communication overheads.

Another form of synchronous parallel computation proposed initially by King [Kun 80], is known as systolic computation and is a clever overlapping combination of SIMD and pipeline paradigms.

**Asynchronous Computation:** Asynchronous parallel computation does not require execution of instruction by different processors in lock step manner. It is therefore, the most general form of parallel computation. The processor interaction depends on explicit locks and flow of control. There are two different organizations, which support asynchronous parallel computation, viz., MIMD and dataflow. The machines based on data flow model have not been commercially successful so far because of enormous overheads for processing graphs. Our discussion regarding asynchronous parallel computation will, therefore be restricted to MIMD.

The two distinctive features in MIMD paradigm that makes it the most general form of parallel computation are:

1. MIMD computation is decomposed into a collection of heterogeneous task which can operate concurrently.
2. MIMD machines do not employ an centralized or global synchronization mechanism.

Since, heterogeneous collection of tasks operate simultaneously, parallel computation requiring infrequent task interactions is ideal for MIMD machine organizations. In a MIMD computer, the processors may communicate through a shared memory. The access to shared memory can be determined by a common switch mechanism. Such a machine is referred to as a multiprocessor C.mmp developed by Carnegie-Mellon University (CMU) is an example of a MIMD machine with a shared memory. There are other MIMD models, in which the processors communicate between themselves through direct exchange of message over the interconnection network. Several hopes may be necessary which do not have direct links. Such a machine is known as a multicomputer. In a MIMD machines synchronization is achieved either by explicit exchange of messages among processors or by setting software locks to prevent other processors using variables in the shared memory when it is used.
An example of MIMD computation can be provided by sum of a set of numbers. The addition may be considered to proceed in two phases, viz., (i) distribution phase, (and) (ii) collection partitions the set into two subsets. These subset may be of different sizes. One subset is retained by the processor and the other subset is passed on the another processor. Both of these processors now operate on their respective subsets. Of more processors are available, the first two processors can again partition the respective subsets assigned to them and share the data with other processors. Sharing of data, as mentioned above, can be continued recursively till all the processors receive their chunk of data. After the distribution is over, processors complete the sum of numbers assigned to them. In collection phase, the processors, which received the chunk of data, pass on their results to the processors from where they had received the data. After receiving partial results from its immediate successor processor each processor would update its local sum. The process is continued until the sum of the entire set at the processor, which initiated the distribution phase. This procedure is similar to that of finding sum using a SIMD or a tree machine. However in this case we relax the load balancing and processor requirement to generate tasks of unequal grain size. This introduce asynchrony in computation and hence heterogeneous. MIMD computation is ideally suitable to support truly concurrent data base system. This is because a generalized concurrent data base can viewed as an asynchronous algorithm which run several concurrent processes to execute parallel transactions. However, it allow little or no control to designer over the set of transactions that may run in the system. It leads to costly synchronization, which may need by performed to preserve the integrity of the database.

1.4 A Taxonomy of parallel Algorithms:

Although certain basic paradigms used for parallel computation on the basis of available architectural support have been identified, we have not provided a taxonomy of parallel algorithms vis-a-vis parallel architecture. Specially, such a taxonomy helps us study structure of parallel algorithm under a unified framework. It also helps us in matching parallel algorithms to appropriate parallel architectures.

There are three orthogonal dimensions for space of parallel algorithms, which may be designed for different application, viz.,

1. Concurrency control
2. Module Granularity

3. Communication Geometry

A parallel algorithm can be viewed as a collection of independent task-modules, which execute in parallel and communicate with each during execution of algorithm. As several task-modular execute in parallel, concurrency control is to be exercised to synchronize these tasks and produce correct computation results. Concurrency control can be applied in one of the following ways:

- By central monitoring
- Through distributed control
- Using shared data

The centralized concurrency control is synchronous in nature whereas control through shared data is completely asynchronous. Distributed control can be either synchronous or asynchronous.

The module granularity of a parallel algorithm determines how frequently the different task-modules comprising the algorithm communicate with each other i.e., the module granularity provides a measure for the maximal amount of computation a typical task module can do before it needs to communicate with other modular. Parallel algorithms with smaller module granularity will require frequent intermodule communication compared to algorithms with larger module granularity. It is, therefore desirable to provide proper data paths in hardware to match the communication requirements with appropriate module granularity. It suffices to classify module granularities of parallel algorithms into three broad classes,

- Small constant
- Small
- Medium or large

Parallel algorithms with constant module granularity can typically perform only a constant number of instructions without intermodule communication. The systolic architectures are most suitable for this class of parallel algorithms. MIMS machine relates to other end of the spectrum. Parallel algorithms with medium or large module granularity would require less communication overheads and therefore, are suitable for implementation on MIMS architectures. SIMD architectures support single instruction stream, and require some sort of broadcast capabilities in their control units. These machines
are more appropriate for parallel algorithms with both small or large module granularities.

Geometric layout of inter dependencies among different task modules of a parallel algorithm is known as communication geometry. Therefore communication geometry in some sense relate to pattern of interconnections among processing elements desirable for parallel algorithms.

1.5 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 ideas implemented by various machines, a particular characterization may not be adequate in reflecting the mechanize architecture and capability. Several different ideas are presented in this section to describe the architectural characterization for a given machine.

1.5.1 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:

1. Stream of instructions, i.e., sequence of instruction executed by the machine.

2. Stream of data, i.e., sequence of data including input, temporary or partial results referenced by instructions.

Computer architectures are characterized by the multiplicity of hardware of server instruction and data streams. Following combinations are natural.

Figure 1.2 shows the architectures based on the above 4 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 on its own data steam. The examples are ILLIA C IV [Bernes 1968].

ICL DAP and many others. The conventional multiprocessors are based on MIMD architectures since individual processors operate on programs having their own instructions and data. MISD architecture is possible for a specialized application, where same data stream is required to be transformed simultaneously in a number of ways. There are no visible examples of MISD architecture.
1.5.2 Shore’s Classification

Unlike Flynn, Shore [1972]-classified the computers on the basis of organization of the constituent elements in the computer.

Machine 1

These are conventional non Neumann architectures with following units
in single quantities.

(a) Control Unit (CU)
(b) Processing Unit (PU)
(c) Instruction Memory (IM) and Data Memory (DM)

A single DM read produces all 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 gain includes both the scalar computers (e.g., IBM 360/91, CDC 7600, etc.) and the pipelined vector computers (e.g., Cracy), Cyber 205). Fig. 1.3(a) shows the organization of such machine. Note that processing is characterized as horizontal (number of bits in parallel as a word).

![Diagram of Machine 1]

**Fig. 3(a) Machine 1**

**Machine 2**: This organization is similar to machine 1 except that DM fetches a bit slice from all the words in the memory and PU is organized to perform the operations in a bit serial manner on all the words. 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 fig. 1.3(b). Examples are ICL DAP [1979] and MPP [Bacher 1980].

![Diagram of Machine 2]

**Fig. 3(b) Machine 2**
**Machine 3**: This machine is a combination of machine 1 and machine 2. It could be characterized having a memory as an array of bits with both horizontal and vertical reading and processing possible. The machine thus will have both vertical and horizontal processing units. Well known example is the machine Omenn 60 [Higbie 1973]. Fig. 1.3 (c) shows the architecture machine 3.

![Fig. 3(c) Machine 3](image)

**Machine 4**: This architecture 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]. Absence of connection between PEs limits the applicability of the machine.

![Fig. 3(d) Machine 4](image)

![Fig. 3(e) Machine 5](image)

![Fig. 3(f) Machine 6](image)

**Machine 5**: This class of machines is similar to the machine 4 with the addition of communication between the PEs. Figure 1.3. (d) shows the idea of the architecture while figure 1.3.(e) show the same for machine 5. (LLIAC iv and many SIMD processors fall in this category.

**Machine 6**: Machine 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. 1.3 (f) includes the logic in memory itself and is called associate processor. Machine based on such architectures span a range from simple associative memories to complex associative processors.
1.5.3 Feng’s Classification :

Feng [1972] also proposed a scheme on the basis of degree of parallelism to classify 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 bit and word levels to produce the following classification:

<table>
<thead>
<tr>
<th>Classification</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSBS (Word Serial / Bit serial)</td>
<td>No conceivable implementation</td>
</tr>
<tr>
<td>WPBS (Word Parallel Bit serial)</td>
<td>(Staran)</td>
</tr>
<tr>
<td>WSBP (Word Serial Bit Parallel)</td>
<td>Conventional Computers</td>
</tr>
<tr>
<td>WPBP (Word Parallel Bit Parallel)</td>
<td>ILLIAL IV</td>
</tr>
</tbody>
</table>

The maximum degree of parallelism is given by the product of the number of bits in a word and the number of words processed in parallel. Some of the commercial available machines are classified in fig. 1.4. in the above diagram x axis represents the number of words processed in parallel. A machine is place at a paint in the space defined by these (x,y) co-ordinates. The maximum degree of parallelism in a machine is given by the area of the rectangle formed by the origin and the paint representing the machine. For example, Cray 1 has an area of 64, while the IBM 370/168 has an area of 32. It is to be noted again that the
scheme fails to project the concurrency in pipeline processor.

1.5.4 **Handler's Classification**: Feng's scheme, while indicating the degree of parallelism does not account for the concurrency handled by the pipelined designs. Handler's scheme allows the pipelining to be specified. In fact the scheme proposed allow one to list the machine's features as a record of information about the machine.

Essence of the Handler's Scheme [1977] allows the identification of parallelism and degree of pipelining built in the hardware structure of the system. Handler defined some of the terms used in the record as:

- **PCUs** = Processor Control Units
- **ALU** = Arithmetic Logic Units
- **BLC** = Bit Level Circuits (e.g. combinational circuit need for 1 bit processing in the ALU)
- **PEs** = Processing Elements
- **CU** = Control Unit

A computing system C can be characterized by a triple as below:

\[ T(c) = (K \times K', D \times D', W \times W') \]

where

- **K** = number of PCUs in the system
- **K'** = number of processors that are pipelined.
- **D** = number of ALUs in the system
- **D'** = word length of ALU or PE
- **W'** = number of pipeline stages in ALU or PE.

For simplicity, if the value of the second element of an pair is one, then it is omitted. To illustrate the idea, I am giving one example.

The Cray 1 system is a single CPU system 12 pipelined functional units. The length of pipeline in these units vary from 1 to 14. Cray 1 word length is 64 bits. The system thus may be characterized as:

\[ T(CRAY\ 1) = (1 \times 1, 12 \times 8, 64 \times (1-14)) \]

1.6 **What previous Researchers have done on it**: The quantification of performance characterization of a computer applicable across several
applications is not simple. Even for sequential computers, we do not have appropriate grading numbers, which truly reflect the power of a particular machine.

Let

\[
\begin{align*}
    n & = \text{number of processors} \\
    T_s & = \text{Single processor execution time} \\
    T_n & = \text{n processor execution time} \\
    S & = \text{Speedup}.
\end{align*}
\]

The speedups is measure of performance by parallel computer and is the ratio time taken by single processor system and that taken by a parallel processing system. So speed up is given by:

\[
S = \frac{T_s}{T_n}
\]

1.6.1 Folk Theorem 1.1: \(1 \leq S \leq n\)

Parallel processor can act as a sequential processor using one PE. Therefore parallel processor is or fast as a sequential processor. Conversely, a sequential processor can simulate a parallel processor. Each time step of parallel machine, single processor takes \(n\) steps.

We called the theorem as folk theorem. This is so because the arguments in the proof does not cater for real machines. People believe using above arguments that the theorem should be true. It is not difficult to demolish the above theorem. David P Hambold and et al [1990] modeled speedup's greater than \(n\).

The efficiency of a parallel computer is defined as speed up divided by the number of processor, i.e., the normalized speed up gives the efficiency,

\[
e = \frac{s}{n}, \text{ where } e \text{ is the efficiency.}
\]

Another view of the efficiency is derived from the analog of effacing in other machines with which we are familiar. We bring out the ideas as follows.

Let the machine run for some period \(T\) and solve a problem. A parallel computer contains number of PEs, each PE, when employed for a work for a
unit time shall do some computing work. Work done in a unit of time is called power.

When the computing power is employed over a period of time work is done by consuming the computing energy. A computing energy of a PE is its use over time. A graph showing computing power and time shall thus depict the energy. A single PE has some power, n PEs have n time the computing power of a single PE. An energy diagram for a computing machine is shown in fig. 1.5.

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

\[ e = \frac{\text{Energy used in doing work}}{\text{Total energy supplied}} \]

The two energies referred to above are shown in fig. 1.5. The loss of energy is mainly energy conversion machine occurs because of heat losses due to various reasons. Similarly in computing machine also we have losses occurring in the computational energy. This happens mainly because some computing power could not be used while doing the work or some power is wasted in the losses occurring in the process because of the mismatch in the architecture and the algorithm. According to the Folk theorem 1.1, for n processor parallel machine, the upper limits on the speed up and efficiency are n and 1 respectively but in practice both of these will usually be much smaller for large n. Following are
the basic losses in solving problems using parallel computers:

(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.

1.6.2 Amdahl's law:

Amdahl's law [Amdahl 1967] is based on a very simple observation. A program requiring total time $T$ for sequential execution shall have some part which is inherently sequential (which cannot parallelised). In terms of a total time taken for solving a problem, this fraction called sequential fraction of computing time (SFS) is an important parameter about a program.

Let

$$f = \text{sequential fraction for a given program.}$$

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

$$S \leq \frac{1}{f + \left(1 - \frac{f}{n}\right)}$$

Assume that total time is $T$. Then the sequential component of this time is $fT$. 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 $(1-f)\frac{T}{n}$. This time cannot be reduced further. Total time taken by the parallel computer thus is atleast $fT + (1-f) - \frac{T}{n}$, while the sequential processor takes time $T$. The speedily $S$ is limited thus by:

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

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

This expression throws some light on the way parallel computers should
be built. A computer architect is faced with following two basic approaches while designing a parallel computer;

1. Connect small number of extremely powerful processors.
2. Connect very large number of inexpensive processors.

**1.6.3 Folk Theorem 1.2 :** Consider two parallel Me and Ma. The computer Me is built using the approach (connect small number of extremely powerful processors) such that each processor is capable of executing computations at a speed of M megaflops. The computer Ma is built using approach (connect very large number of inexpensive processors) and each processor of Ma executes r-m Megaflops, where 0 < r < |.

If the machine Me attempts a computation whose inherently sequential fraction f is greater than r then Ma will execute the computations more slovenly than a single processor of the computer Me.

Let

\[
\begin{align*}
W &= \text{Total work} \\
M &= \text{speed of PE of Me} \\
r.m &= \text{speed of a PE of Ma} \\
F. W &= \text{Sequential work of the job.} \\
T_{(Ma)} &= \text{time taken by Ma for the work W.} \\
T_{(Me)} &= \text{time taken by Me for the work W.} \\
\end{align*}
\]

Time taken by any computer is:

\[
T = \frac{\text{Amount of work}}{\text{Speed}}.
\]

\[
T(Ma) = \text{Time for sequential part} + \text{Time for parallel part}
\]

\[
= \frac{f \cdot W \cdot (1-f) \cdot W/n}{r \cdot M \cdot r \cdot m} + \frac{1-f \cdot W}{r \cdot M \cdot r \cdot n \cdot m}
\]

20
\[ T(M) = \frac{W}{M} \] (assume only one PE) from (1) and (2) it is clear that if \( F > r \) then \( T(Ma) > T(Me) \). The above theorem empties that a sequential component fraction acceptable for the Me may not be acceptable for the machine Ma. It does no good to have a very large number of processor power that goes waste. Processor must maintain some level of efficiency. Let us see how the efficiency 'e' and sequential fraction f are related.

\[ S \leq \frac{1}{f + (1 - f)/n} \]

the efficiency \( e = s/n \) and hence,

\[ e \cdot n \leq \frac{1}{f + (1 - f)/n} \]

\[ e \leq \frac{1}{f \cdot n + 1 - f} \]

It says that for constant efficiency, the fraction of sequential component of an algorithm must be inversely proportional to the number of processors. Figure 1.6 Show the graph of sequential component and efficiency with n as a parameter.

![Fig. 6. Efficiency and the sequential fraction](image)
The idea using very large number of processors may thus be good only for specific applications for which it is known that the algorithms have a very small sequential fraction $f$.

1.6.4. Minsky's conjecture [Minsky 1970]: For a parallel computer with $n$ processors, then the speed up $S$ shall be proportional to $\log_2 n$.

The Minsky's conjecture was very bad for proponents for large-scale parallel architectures. Flynn and Henessy [1980] gave yet another formulation, which is presented by the following Folk theorem.

1.6.5 Folk Theorem 1.3:

Speed up of a $n$ processor parallel system is limited by:

Let $T_1 =$ Time taken by single processor system,

$Q_i =$ probably of $i$ PEs working simultaneously.

$T_n =$ Time required by parallel machine with $n$ processors.

The time $T_n \geq \left[ \frac{q_1}{1} + \frac{q_2}{2} + \frac{q_3}{3} + \ldots + \frac{q_n}{n} \right] T_1$

The speed up $S = \frac{T_1}{T_n} \leq \frac{1}{(q_1/1 + q_2/2 + q_3/3 + \ldots + N)}$

Assuming $q_i = 1/n$ gives

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

A plot of speedup as a function of $n$ for the above discussed bounds and the typical expected speed up (linear) is presented in fig 1.7. Lee [1980] presented an empirical table for speed up for different ranges of the values of $n$. His observations are listed in Table 1.1.

<table>
<thead>
<tr>
<th>Value of $n$ (number of processors)</th>
<th>Speed up</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$(n)$</td>
</tr>
<tr>
<td>up to 100</td>
<td>$0 \ (n/\log \ n)$</td>
</tr>
<tr>
<td>around 1000</td>
<td>$0 \ (\log \ n) &lt; S &lt; 0 \ (0 \ n/\text{gn})$</td>
</tr>
<tr>
<td>10000 and above</td>
<td>0(n)</td>
</tr>
</tbody>
</table>

Table 1.1. Speed up and the number of processors.
Fig. 7 Speed up as a Function of number of processors

Fig. 8 Asymptotic Time Reduction

Fig. 9 Effective Parallelism
It is to be noted from fig. 1.7 that the speed up curve using Flynn’s bound for large values of n have a linear shape. This is so because the function n/log n approximately to a straight line when n > 1000. The observations due to Lee are thus derivable from Folk theorem 1.3. Fig 1.7 shows the speed up as a function of the number of processors. The studies on execution time, effective parallelism as f^n of number of processors were made by Jordan [1984] for HEP MIMD architecture. Figure 1.9 shows the graphs depicting the saturation of the available degree of parallelism and Figure 1.8 shows the graph for time reduction for two important problems, viz., numerical integration and <v decomposition.
REFERENCES:


