UNIT 2 PERFORMANCE EVALUATIONS

Structure **Page Nos.**

2.0 INTRODUCTION

In the earlier blocks and the previous section of this block, we have discussed a number of types of available parallel computer systems. They have differences in architecture, organisation, interconnection pattern, memory organisation and I/O organisation. Accordingly, they exhibit different performances and are suitable for different applications. Hence, certain parameters are required which can measure the performance of these systems.

In this unit, the topic of performance evaluation explains those parameters that are devised to measure the performances of various parallel systems. Achieving the highest possible performance has always been one of the main goals of parallel computing. Unfortunately, most often the real performance is less by a factor of 10 and even worse as compared to the designed peak performance. This makes parallel performance evaluation an area of priority in high-performance parallel computing. As we already know, sequential algorithms are mainly analyzed on the basis of computing time i.e., time complexity and this is directly related to the data input size of the problem. For example, for the problem of sorting n numbers using bubble sort, the time complexity is of $O(n^2)$. However, the performance analysis of any parallel algorithm is dependent upon three major factors viz. time complexity, total number of processors required and total cost. The complexity is normally related with input data size (n).

Thus, unlike performance of a sequential algorithm, the evaluation of a parallel algorithm cannot be carried out without considering the other important parameters like the total number of processors being employed in a specific parallel computational model. Therefore, the evaluation of performance in parallel computing is based on the parallel computer system and is also dependent upon machine configuration like PRAM,

combinational circuit, interconnection network configuration etc. in addition to the parallel algorithms used for various numerical as well non-numerical problems.

This unit provides a platform for understanding the performance evaluation methodology as well as giving an overview of some of the well-known performance analysis techniques.

2.1 OBJECTIVES

After studying this unit, you should be able to:

- describe the Metrics for Performance Evaluation;
- tell about various Parallel System Overheads;
- explain the speedup Law; and
- enumerate and discuss Performance Measurement Tools.

2.2 METRICS FOR PERFORMANCE EVALUATION

In this section, we would highlight various kinds of metrics involved for analysing the performance of parallel algorithms for parallel computers.

2.2.1 Running Time

The running time is the amount of time consumed in execution of an algorithm for a given input on the N-processor based parallel computer. The running time is denoted by T(n) where n signifies the number of processors employed. If the value of n is equal to 1, then the case is similar to a sequential computer. The relation between Execution time vs. Number of processors is shown in *Figure 1*.

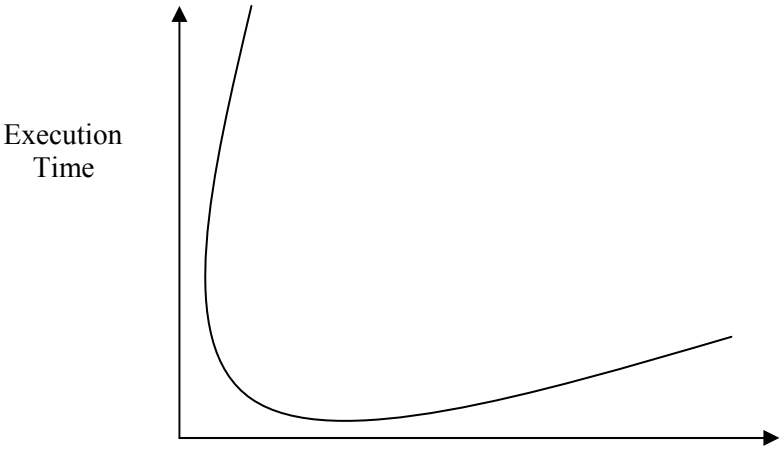

Figure 1: Execution Time vs. number of processors

It can be easily seen from the graph that as the number of processors increases, initially the execution time reduces but after a certain optimum level the execution time increases as number of processors increases. This discrepancy is because of the overheads involved in increasing the number of processes.

2.2.2 Speed Up

Speed up is the ratio of the time required to execute a given program using a specific algorithm on a machine with single processor (i.e. T (1) (where n=1)) to the time required

to execute the same program using a specific algorithm on a machine with multiple processors (i.e. T(n)). Basically the speed up factor helps us in knowing the relative gain achieved in shifting from a sequential machine to a parallel computer. It may be noted that the term T(1) signifies the amount of time taken to execute a program using the best sequential algorithm i.e., the algorithm with least time complexity.

$$S(n) = \frac{T(1)}{T(n)}$$

Let us take an example and illustrate the practical use of speedup. Suppose, we have a problem of multiplying n numbers. The time complexity of the sequential algorithm for a machine with single processor is O(n) as we need one loop for reading as well as computing the output. However, in the parallel computer, let each number be allocated to individual processor and computation model being used being a hypercube. In such a situation, the total number of steps required to compute the result is log n i.e. the time complexity is O(log n). *Figure 2,* illustrates the steps to be followed for achieving the desired output in a parallel hypercube computer model.

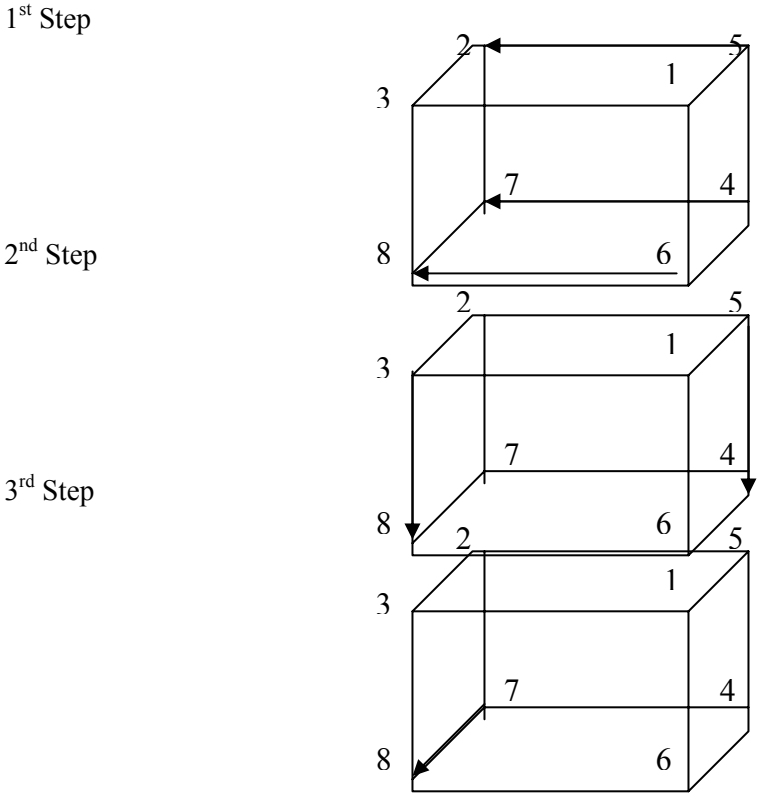

Figure 2: Steps followed for multiplying n numbers stored on n processors

As the number of steps followed is equal to 3 i.e., log 8 where n is number of processors. Hence, the complexity is O(log n).

In view of the fact that sequential algorithm takes 8 steps and the above-mentioned parallel algorithm takes 3 steps, the speed up is as under:

$$S(n) = \frac{8}{3}$$

As the value of S(n) is inversely proportional to the time required to compute the output on a parallel number which in turn is dependent on number of processors employed for performing the computation. The relation between S(n) vs. Number of processors is shown in *Figure 3*. The speedup is directly proportional to number of processors,

therefore a linear arc is depicted. However, in situations where there is parallel overhead, the arc is sub-linear.

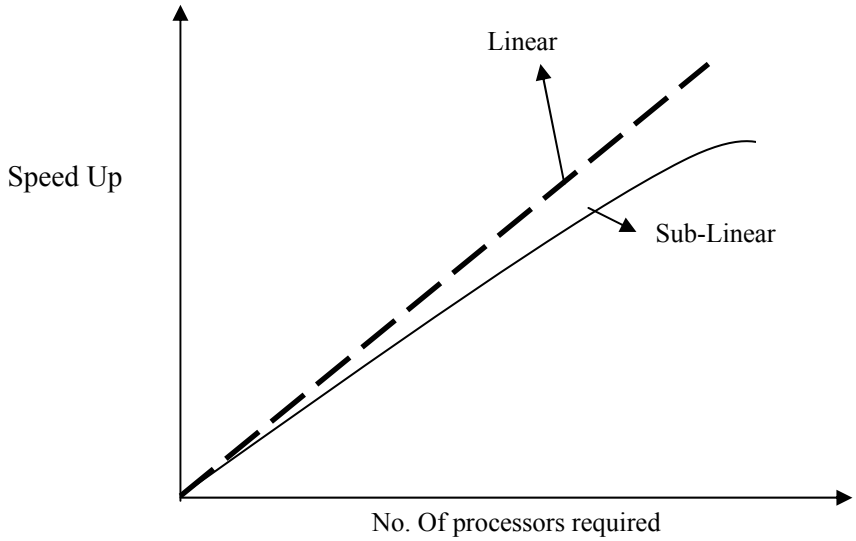

Figure 3: Speed-up vs. Number of Processors

2.2.3 Efficiency

The other important metric used for performance measurement is efficiency of parallel computer system i.e. how the resources of the parallel systems are being utilized. This is called as degree of effectiveness. The efficiency of a program on a parallel computer with *k* processors can be defined as the ratio of the relative speed up achieved while shifting the load from single processor machine to *k* processor machine where the multiple processors are being used for achieving the result in a parallel computer. This is denoted by E(k).

E_k is defined as follows:

$$E(k) = \frac{S(k)}{k}$$

The value of E(k) is directly proportional to S(k) and inversely proportional to number of processors employed for performing the computation. The relation between E(k) vs. Number of processors is shown in *Figure 4*.

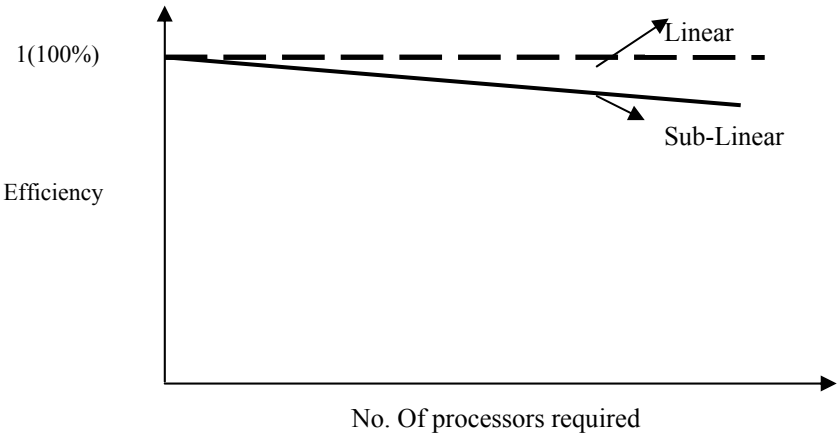

Figure 4: Efficiency vs. Number of Processors

Assuming we have the multiplication problem as discussed above with *k* processors, then the efficiency is as under:

$$E(k) = \frac{T(1)}{T(n) * K} = \frac{N}{\log n * N}$$

$$E(n) = \frac{1}{\log(n)}$$

Now, supposing we have X processors i.e. X < K and we have to multiply n numbers, in such a situation the processors might be overloaded or might have a few overheads. Then the efficiency is as under:

$$E(X) = \frac{T(1)}{T(X) * X}$$

Now, the value of T(X) has to be computed. As we have n numbers, and we have X processors, therefore firstly each processor will multiply *n/X* numbers and consequently process the X partial results on the X processors according to the method discussed in *Figure 1*. The time complexity is equal to the sum of the time to compute multiplication of k/X numbers on each processor i.e., O(k/X) and time to compute the solution of partial results i.e. log (X)

$$E(X) = \frac{K}{(K / X + \log(X)) * X}$$

$$E(X) = \frac{(K / X)}{(K / X + \log(X))}$$

Dividing by X/K we get

$$E(X) = \frac{1}{(1 + (X / K) * \log(X))}$$

It can be concluded from the above statement that if N is fixed then the efficiency i.e. E(X) will decrease as the value of X increases and becomes equivalent to E(N) in case X=N. Similarly, if X is fixed then the efficiency i.e., E(X) will increase as the value of X i.e. the number of computations increases.

The other performance metrics involve the standard metrics like MIPS and Mflops. The term MIPS (Million of Instructions Per Second) indicates the instruction execution rate. Mflops (Million of Floating Point Operations per Second) indicates the floating-point execution rate.

2.3 FACTOR CAUSING PARALLEL OVERHEADS

Figures 2,3 and *4* clearly illustrate that the performance metrics are not able to achieve a linear curve in comparison to the increase in number of processors in the parallel computer. The reason for the above is the presence of overheads in the parallel computer which may degrade the performance. The well-known sources of overheads in a parallel computer are:

1) Uneven load distribution
2) Cost involved in inter-processor communication
3) Synchronization
4) Parallel Balance Point

The following section describes them in detail.

2.3.1 Uneven Load Distribution

In the parallel computer, the problem is split into sub-problems and is assigned for computation to various processors. But sometimes the sub-problems are not distributed in a fair manner to various sets of processors which causes imbalance of load between various processors. This event causes degradation of overall performance of parallel computers.

2.3.2 Cost Involved in Inter-Processor Communication

As the data is assigned to multiple processors in a parallel computer while executing a parallel algorithm, the processors might be required to interact with other processes thus requiring inter-processor communication. Therefore, there is a cost involved in transferring data between processors which incurs an overhead.

2.3.3 Parallel Balance Point

In order to execute a parallel algorithm on a parallel computer, K number of processors are required. It may be noted that the given input is assigned to the various processors of the parallel computer. As we already know, execution time decreases with increase in number of processors. However, when input size is fixed and we keep on increasing the number of processors, in such a situation after some point the execution time starts increasing. This is because of overheads encounteed in the parallel system.

2.3.4 Synchronization

Multiple processors require synchronization with each other while executing a parallel algorithm. That is, the task running on processor X might have to wait for the result of a task executing on processor Y. Therefore, a delay is involved in completing the whole task distributed among K number of processors.

Check Your Progress 1

1) Which of the following are the reasons for parallel overheads?
   A) Uneven load distribution
   B) Cost involved in inter-processor communication
   C) Parallel Balance Point
   D) All of the above

2) Which of the following are the performance metrics?
   A) MIPS
   B) Mflops
   C) Parallel Balance Point
   D) A and B only

3) Define the following terms:
   A) Running Time
   B) Speed Up
   C) Efficiency

2.4 LAWS FOR MEASURING SPEED UP PERFORMANCE

To measure speed up performance various laws have been developed. These laws are discussed here.

2.4.1 Amdahl's Law

Remember, the speed up factor helps us in knowing the relative gain achieved in shifting the execution of a task from sequential computer to parallel computer and the performance does not increase linearly with the increase in number of processors. Due to the above reason of saturation in 1967 Amdahl's law was derived. Amdahl's law states that a program contains two types of operations i.e. complete sequential operations which must be done sequentially and complete parallel operations which can be executed on multiple processors. The statement of Amdahl's law can be explained with the help of an example.

Let us consider a problem say P which has to be solved using a parallel computer. According to Amdahl's law, there are mainly two types of operations. Therefore, the problem will have some sequential operations and some parallel operations. We already know that it requires T (1) amount of time to execute a problem using a sequential machine and sequential algorithm. The time to compute the sequential operation is a fraction α (alpha) ($\alpha \leq 1$) of the total execution time i.e. T (1) and the time to compute the parallel operations is (1- α). Therefore, S (N) can be calculated as under:

$$S(N) = \frac{T(1)}{T(N)}$$

$$S(N) = \frac{T(1)}{X * T(1) + (1-\alpha) * \frac{T(1)}{N}}$$

Dividing by T(1)

$$S(N) = \frac{1}{\alpha + \frac{1-\alpha}{N}}$$

Remember, the value of α is between 0 and 1. Now, let us put some values of α and compute the speed up factor for increasing values of number of processors. We find that the S(N) keeps on decreasing with increase in the value of α (i.e. number of sequential operations as shown in *Figure 5*).

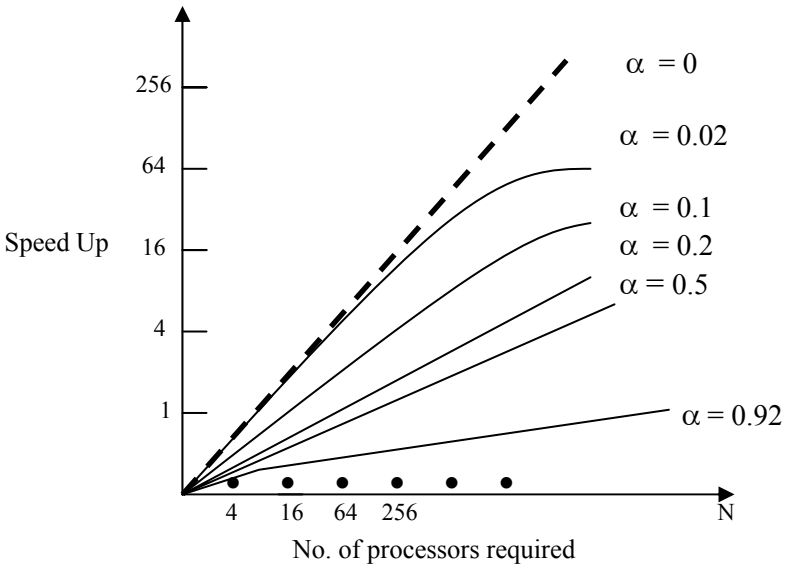

Figure 5: Speed-up vs. Number of Processors

The graph in *Figure 5* clearly illustrates that there is a bottleneck caused due to sequential operations in a parallel computer. Even when the number of sequential operations are more, after increasing the number of processors, the speed up factor S (N) degrades.

The sequential fraction i.e. α can be compared with speed up factor S(N) for a fixed value of N say 500. *Figure 6* illustrates a pictorial view of effect of Amdahl's law on the speed up factor.

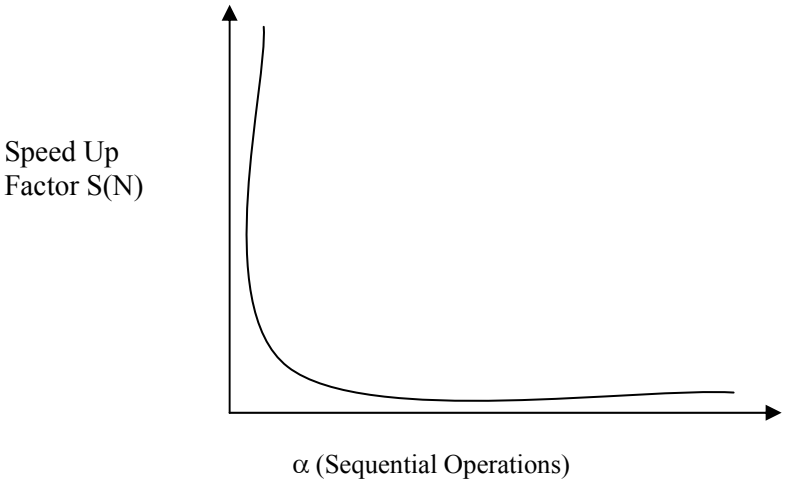

Figure 6: S (n) vs. α (Graph is not to scale)

The outcomes of analysis of Amdahl's law are:

1) To optimize the performance of parallel computers, modified compilers need to be developed which should aim to reduce the number of sequential operations pertaining to the fraction α.
2) The manufacturers of parallel computers were discouraged from manufacturing large-scale machines having millions of processors.

There is one major shortcoming identified in Amdahl's law. According to Amdahl's law, the workload or the problem size is always fixed and the number of sequential operations mainly remains same. That is, it assumes that the distribution of number of sequential operations and parallel operations always remains same. This situation is shown in

Figure 7 and the ratio of sequential and parallel operations are independent of problem size.

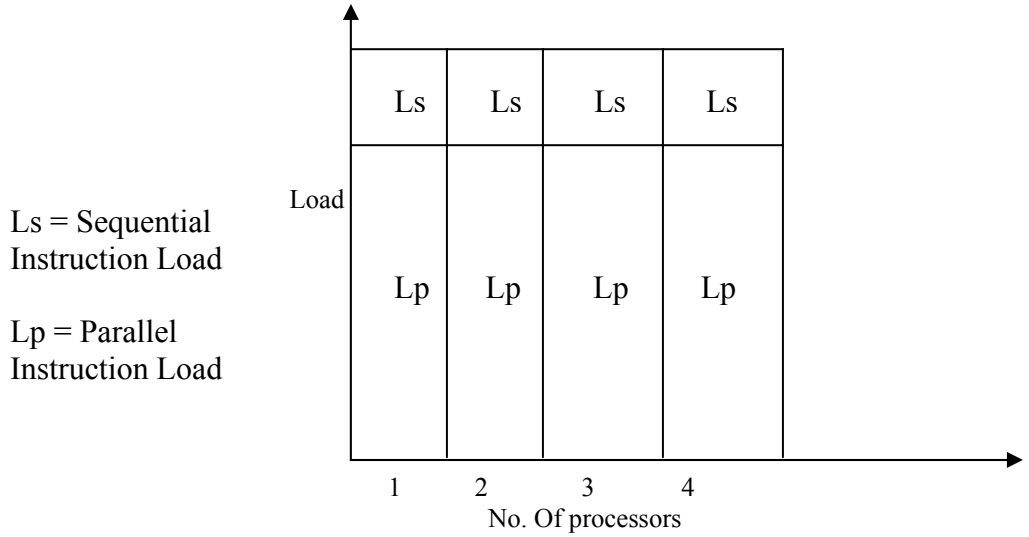

Figure 7: Fixed load for Amdahl's Law

However, practically the number of parallel operations increases according to the size of problem. As the load is assumed to be fixed according to Amdahl's law, the execution time will keep on decreasing when number of processors is increased. This situation is shown in *Figure 8*. This is in Introduction to the processes operation.

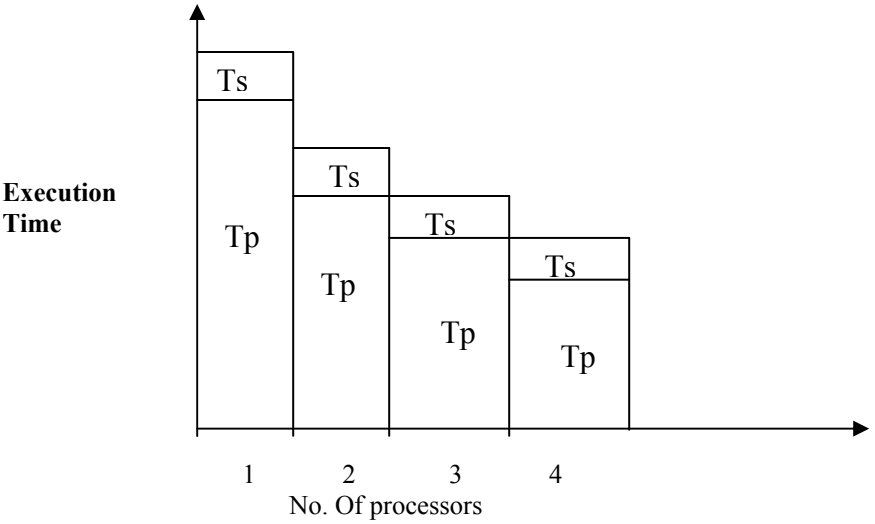

Figure 8: Execution Time decreases for Amdahl's Law

2.4.2 Gustafson's Law

Amdahl's law is suitable for applications where response time is critical. On the other hand, there are many applications which require that accuracy of the resultant output should be high. In the present situation, the computing power has increased substantially due to increase in number of processors attached to a parallel computer. Thus it is possible to increase the size of the problem i.e., the workload can be increased. How does this operate? Gustafson's Law relaxed the restriction of fixed size of problem and aimed at using the notion of constant execution time in order to overcome the sequential bottleneck encountered in Amdahl's Law. This situation which does not assume a fixed workload is analysed by Gustafson. Thus, Gustafson's Law assumes that the workload may increase substantially, with the number of processors but the total execution time should remain the same as highlighted in *Figures 9* and *10.*

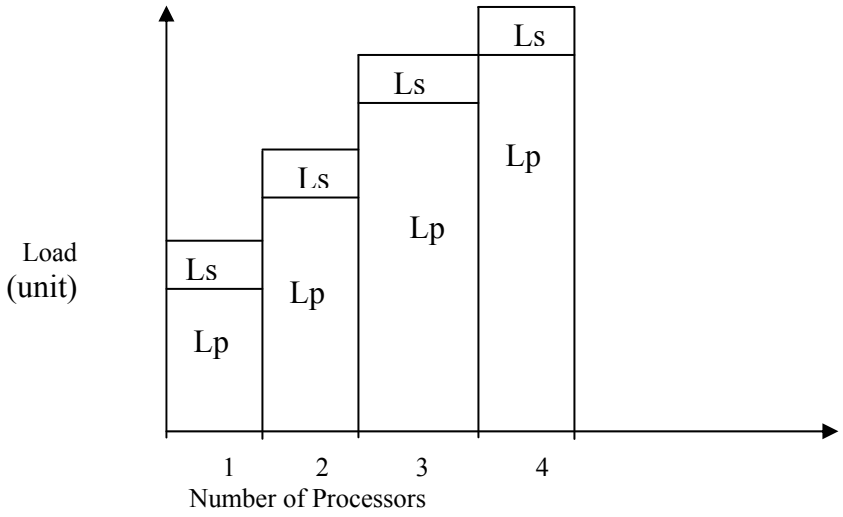

Figure 9: Parallel Load Increases for Gustafson's Law

According to Gustafson's Law, if the number of parallel operations for a problem increases sufficiently, then the sequential operations will no longer be a bottleneck.

The statement of Gustafson's law can be explained with the help of an example. Let us consider a problem, say P, which has to be solved using a parallel computer. Let T_s be the time taken which is considered as constant for executing sequential operations. Let T_p (N, L) be the time taken for running the parallel operations with L as total load on a parallel computer with N processors. The total time taken for finding the solution of problem is T=Ts + Tp. Therefore, S (N) can be calculated as under:

$$S(N) = \frac{T(1)}{T(N)}$$

$$S(N) = \frac{T_s + T_p(1, L)}{T_s + T_p(N, L)}$$

Also, Tp (1, L)= N Tp (N, L) i.e. time taken to execute parallel operations having a load of L on one processor is equal to the N multiplied by the time taken by one computer having N processor. If α be the fraction of sequential load for a given problem i.e.,*

$$\alpha = \frac{T_s}{T_s + T_p(N, L)}$$

Substituting Tp (1, L)= N Tp (N, L) we get,*

$$S(n) = \frac{T_s + N * T_p(N, L)}{T_s + T_p(N, L)}$$

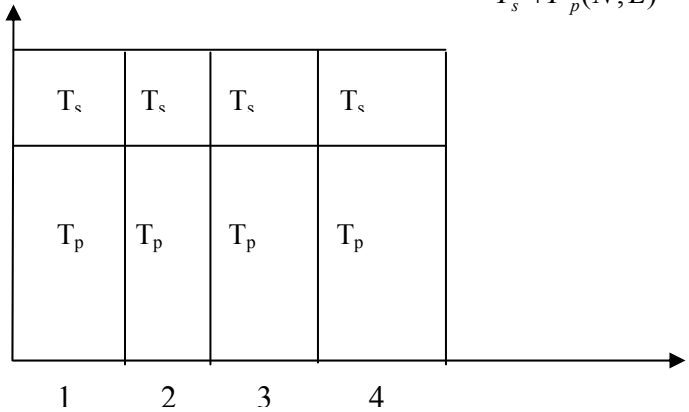

Figure 10: Fixed Execution Time for Gustafson's Law

$$S(N) = \frac{T_s}{T_s + T_p(N,L)} + \frac{N * T_p(N,L)}{T_s + T_p(N,L)}$$

Now, let us change the complete equation of S(N) in the form of X. We get

*S(N) = α +N * (1-α)*

S(N) = N –α * (N-1)

Now, let us put some values of α and compute the speed up factor for increasing values of α i.e. sequential factor of work load for a fixed number of processors say N. *Figure 11* illustrates a pictorial view of effect of Gustafson's Law on the speed up factor. The graph shows as the value of α increases, the speed up factor increases. This decrease is because of overhead caused by inter-processor communication.

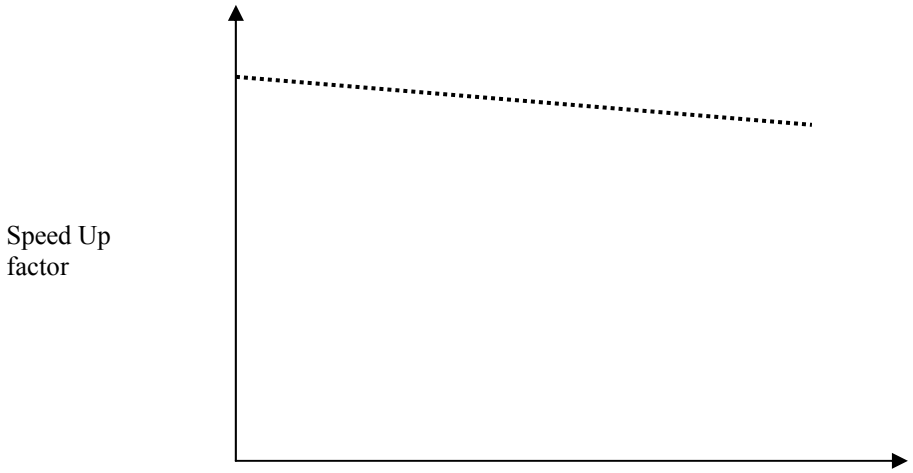

Figure 11: S (N) vs. α (Graph is not to scale)

2.4.3 Sun and Ni's Law

The Sun and Ni's Law is a generalisation of Amdahl's Law as well as Gustafson's Law. The fundamental concept underlying the Sun and Ni's Law is to find the solution to a problem with a maximum size along with limited requirement of memory. Now-a-days, there are many applications which are bounded by the memory in contrast to the processing speed.

In a multiprocessor based parallel computer, each processor has an independent small memory. In order to solve a problem, normally the problem is divided into sub-problems and distributed to various processors. It may be noted that the size of the sub-problem should be in proportion with the size of the independent local memory available with the processor. Now, for a given problem, when large set of processors is culminated together, in that case the overall memory capacity of the system increases proportionately. Therefore, instead of following Gustafson's Law i.e., fixing the execution time, the size of the problem can be increased further such that the memory could be utilized. The above technique assists in generating more accurate solution as the problem size has been increased.

State the law and then inter part discuss it.

Check Your Progress 2

1) Which of the following laws deals with finding the solution of a problem with maximum size along with limited requirement of memory?

   a) Amdahl's law
   b) Gustafson's law
   c) Sun and Ni's law
   d) None of the above

2) Which of the following laws state the program, which contains two types of operations i.e. complete sequential operations which must be done sequentially and complete parallel operations which can be executed on multiple processors?

   a) Amdahl's law
   b) Gustafson's law
   c) Sun and Ni's law
   d) None of the above

3) Which of the following law used the notion of constant execution time?

   a) Amdahl's law
   b) Gustafson's law
   c) Sun and Ni's law
   d) None of the above

2.5 TOOLS FOR PERFORMANCE MEASUREMENT

In the previous units, we have discussed various parallel algorithms. The motivation behind these algorithms has been to improve the performance and gain a speed up. After the parallel algorithm has been written and executed, the performance of both the algorithms is one of the other major concerns. In order to analyze the performance of algorithms, there are various kinds of performance measurement tools. The measurement tools rely not only on the parallel algorithm but require to collect data from the operating system and the hardware being used, so as to provide effective utilisation of the tools.

For a given parallel computer, as the number of processors and its computational power increases, the complexity and volume of data for performance analysis substantially increases. The gathered data for measurement is always very difficult for the tools to store and process it.

The task of measuring the performance of the parallel programs has been divided into two components.

1) **Performance Analysis**: It provides the vital information to the programmers from the large chunk of statistics available of the program while in execution mode or from the output data.

2) **Performance Instrumentation:** Its emphasis on how to efficiently gather information about the computation of the parallel computer.

2.6 PERFORMANCE ANALYSIS

In order to measure the performance of the program, the normal form of analysis of the program is to simply calculate the total amount of CPU time required to execute the various part of the program i.e., procedures. However, in case of a parallel algorithm running on a parallel computer, the performance metric is dependent on several factors such as inter-process communication, memory hierarchy etc. There are many tools such as ANALYZER, INCAS, and JEWEL to provide profiles of utilization of various resources such as Disk Operations, CPU utilisation, Cache Performance etc. These tools even provide information about various kinds of overheads. Now, let us discuss various kinds of performance analysis tools.

2.6.1 Search-based Tools

The search-based tools firstly identify the problem and thereafter appropriately give advice on how to correct it.

AT Expert from Cray Research is one of the tools being used for enhancing the performance of Fortran programs with the help of a set of rules which have been written with Cray auto-tasking library. The Cray auto-tasking library assists in achieving the parallelism. Basically, the ATExpert analyses the Fortran program and tries to suggest complier directives that could help in improving the performance of the program.

Another tool called *"Performance Consultant"* is independent of any programming language, model and machines. It basically asks three questions i.e. WHY, WHERE and WHEN about the performance overheads and bottlenecks. These three questions form the 3 different axes of the hierarchal model. One of the important features of Performance Consultant tool is that it searches for bottlenecks during execution of program. The above-mentioned features assist in maintaining the reduced volume of data. The WHY axis presents the various bottlenecks such as communication, I/O etc. The WHERE axis defines the various sources which can cause bottlenecks such as Interconnection networks, CPU etc. The WHEN axis tries to separate the set of bottlenecks into a specific phase of execution of the program.

2.6.2 Visualisation

Visualization is a generic method in contract to search based tools. In this method visual aids are provided like pictures to assist the programmer in evaluating the performance of parallel programs. One of visualization tools is *Paragraph*. The Paragraph supports various kinds of displays like communication displays, task information displays, utilisation displays etc.

Communication Displays

Communication displays provide support in determining the frequency of communication, whether congestion in message queues or not, volume and the kind of patterns being communicated etc.

Communication Matrix

A communication matrix is constructed for displaying the communication pattern and size of messages being sent from one processor to another, the duration of communication etc. In the communication matrix, a two dimensional array is constructed such that both the horizontal and vertical sides are represented with the various processors. It mainly provides the communication pattern between the processors. The rows indicate the processor which is sending the message and columns indicate the processor which is going to receive the messages as shown in *Figure 12*.

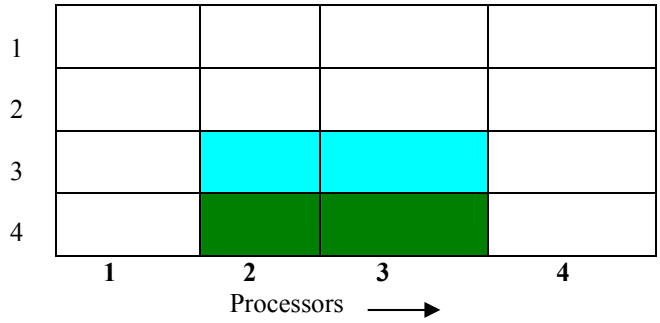

Figure: 12 Communication Matrix

Communication Traffic

The Communication Traffic provides a pictorial view of the communication traffic in the interconnection network with respect to the time in progress. The Communication Traffic displays the total number of messages, which have been sent but not received.

Message Queues

The message queue provides the information about the sizes of queues under utilization of various processors. It indicates the size of each processor incoming message queue, which would be varying depending upon the messages being sent, received or buffered as shown in *Figure 13*. It may be noted that every processor would be having a limit on maximum length of its queue. This display mainly helps in analyzing whether the communication is congestion free or not.

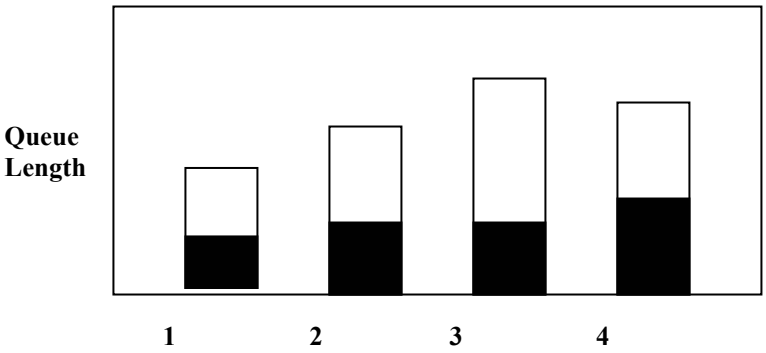

Figure 13: Message Queues

Processors Hypercube

This is specific to In the hypercube: Here, each processor is depicted by the set of nodes of the graph and the various arcs are represented with communication links. The nodes status can be idle, busy, sending, receiving etc. and are indicated with the help of a specific color. An arc is drawn between two processors when message is sent from one node to another and is deleted when the message is received.

Utilisation Displays

It displays the information about distribution of work among the processors and the effectiveness of the processors.

Utilisation Summary

The Utilisation Summary indicates the status of each processor i.e. how much time (in the form of percentage) has been spent by each processor in busy mode, overhead mode and idle mode as shown in *Figure 14*.

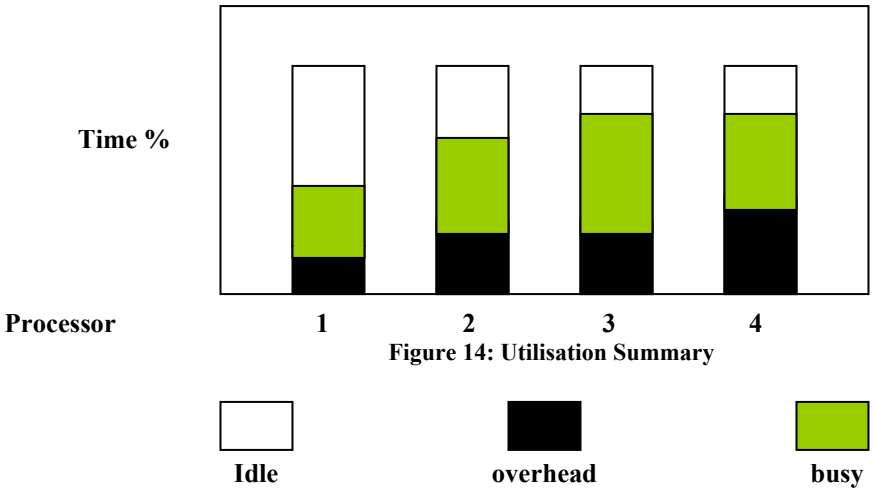

Figure 14: Utilisation Summary

Utilisation Count

The Utilisation Count displays the status of each processor in a specific mode i.e. busy mode, overhead mode and idle mode with respect to the progress in time as shown in *Figure 15*.

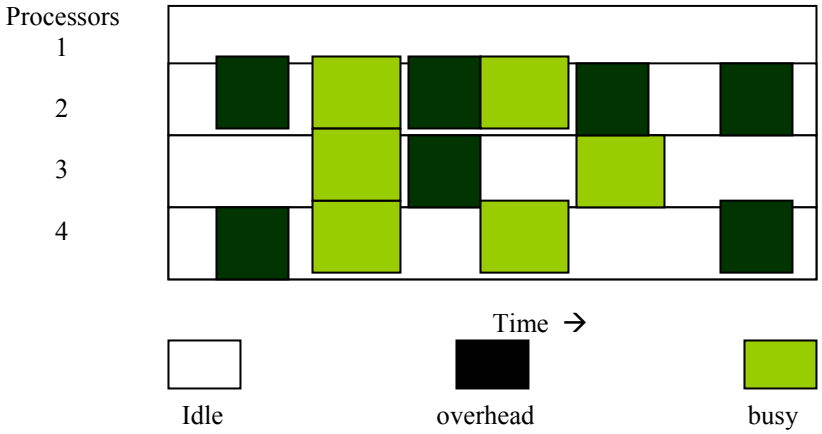

Figure 15: Utilisation Count

Gantt chart: The Gantt chart illustrates the various activities of each processor with respect to progress in time in idle-overhead -busy modes with respect to each processor.

Kiviat diagram: The Kiviat diagram displays the geometric description of each processor's utilization and the load being balanced for various processors.

Concurrency Profile: The Concurrency Profile displays the percentage of time spent by the various processors in a specific mode i.e. idle/overhead/busy.

Task Information Displays

The Task Information Displays mainly provide visualization of various locations in the parallel program where bottlenecks have arisen instead of simply illustrating the issues that can assist in detecting the bottlenecks. With the inputs provided by the users and through portable instrumented communication library i.e., PICL, the task information displays provide the exact portions of the parallel program which are under execution.

Task Gantt

The Task Gantt displays the various tasks being performed i.e., some kind of activities by the set of processors attached to the parallel computer as shown in *Figure 16.*

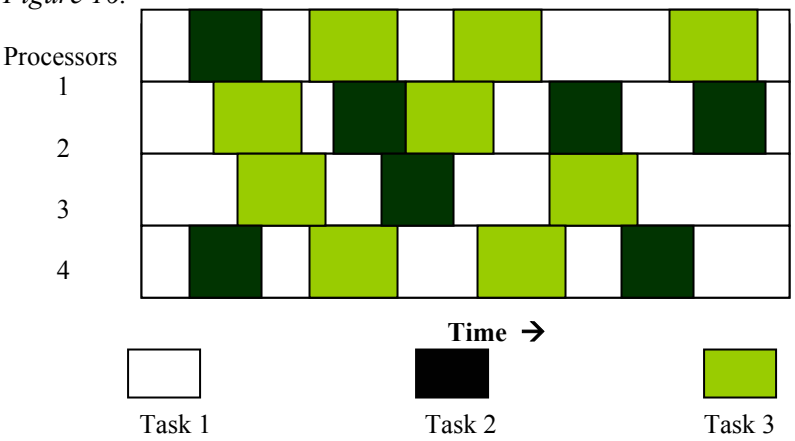

Figure 16: Task Gantt

Summary of Tasks

The Task Summary tries to display the amount of duration each task has spent starting from initialisation of the task till its completion on any processor as shown in *Figure 17.*

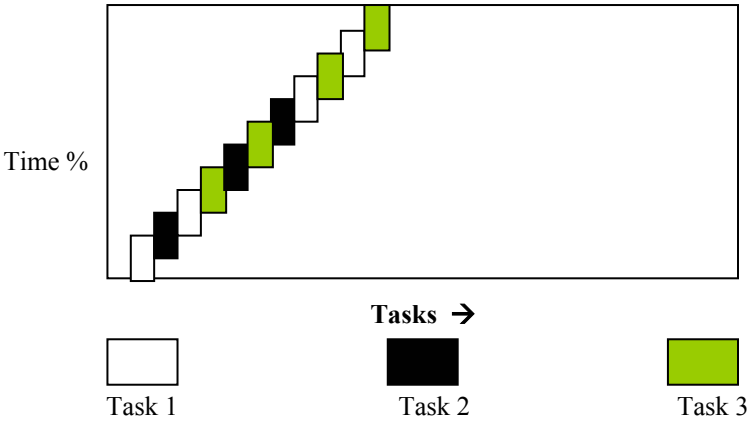

Figure 17: Summary of Tasks

2.7 PERFORMANCE INSTRUMENTATION

The performance instrumentation emphasizes on how to efficiently gather information about the computation of the parallel computer. The method of instrumentation mainly attempts to capture information about the applications by adding some kind of instrument like a code or a function or a procedure etc. in to the source code of the program or may be at the time of execution. The major effect of such instrumentation is generation of some useful data for performance tools. The performance instrumentation sometimes causes certain problems known as intrusion problems.

Check Your Progress 3

1) Which of the following are the parts of performance measurement?
   (a) Performance Analysis
   (b) Performance Instrumentation
   (c) Overheads
   (d) *A and B*

2) List the various search-based tools used in performance analysis.

……
……
……
……

3) List the various visualisation tools employed in performance analysis.

……
……
……
……

2.8 SUMMARY

The performance analysis of any parallel algorithm is dependent upon three major factors i.e., Time Complexity of the Algorithm, Total Number of Processors required and Total Cost Involved. However, it may be noted that the evaluation of performance in parallel computing is based on parallel computer in addition to the parallel algorithm for the various numerical as well non-numerical problems. The various kinds of performance metrics involved for analyzing the performance of parallel algorithms for parallel computers have been discussed e.g., Time Complexity, Speed-Up, Efficiency, MIPS, and Mflops etc. The speedup is directly proportional to number of processors; therefore a linear arc is depicted. However, in situations where there is parallel overhead in such states the arc is sub-linear. The efficiency of a program on a parallel computer with say N processors can be defined as the ratio of the relative speed up achieved while shifting from single processor machine to n processor machine to the number of processors being used for achieving the result in a parallel computer. The various sources of overhead in a parallel computer are; *Uneven load distribution, Cost involved in inter-processor communication, Synchronization, Parallel Balance Point etc.*

In this unit we have also discussed three speed-up laws i.e., Gustafson's law, Amdahl's law, Sun and Ni's law. The performance of the algorithms is one of the other major concerns. In order to analyze the performance of algorithms, there are various kinds of performance measurement tools. The measurement tools rely not only on the parallel algorithm but are also require to collect data from the operating system, the hardware being used for providing effective utilization of the tools. The task of measuring the performance of the parallel programs has been divided into two components i.e. Performance Analysis and Performance Instrumentation. The various tools employed by the above two components have also been discussed in this unit.

2.9 SOLUTIONS/ANSWERS

Check Your Progress 1

1) D
2) D
3) The running time defines the amount of time consumed in execution of an algorithm for a given input on the N processor based parallel computer. The running time is denoted by T(n) where n signifies the number of processors employed.
Speed up defines the ratio of the time required to execute a given program using a specific algorithm on a machine with single processor i.e. T (1) (where n=1) to the time required to execute a given program using a specific algorithm on a machine with multiple processor i.e. T(n).

Efficiency of the machine i.e. how are the resources of the machine e.g. processors being utilized (effective or ineffective)

Check Your Progress 2

1) C
2) A
3) B

Check Your Progress 3

1) D
2) ATExpert from Cray Research, Performance Consultant etc.
3) Utilisation Displays, Communication Displays and Task Information displays.

2.10 FURTHER READINGS

Parallel Computers - *Architecture and Programming*, Second edition, by V.Rajaraman and C.Siva Ram Murthy (Prentice Hall of India)