UNIT 1 OPERATING SYSTEM FOR PARALLEL COMPUTER

Structure **Page Nos.**

1.0 INTRODUCTION

In Blocks 1 and 2, we have discussed parallel computing architectures and parallel algorithms. This unit discusses the additional requirements at operating system and software levels which will make the parallel programs run on parallel hardware. Collectively, these requirements define the parallel program development environment. A parallel programming environment consists of available hardware, supporting languages, operating system along with software tools and application programs. The hardware platforms have already been discussed in the earlier units. These topics include discussion of shared memory systems, message passing systems, vector processing; scalar, superscalar, array and pipeline processors and dataflow computers. This unit also presents a case study regarding operating systems for parallel computers.

1.1 OBJECTIVES

After studying this unit you will be able to describe the features of software and operating systems for parallel computers.

In particular you should be able to explain the following:

- various additional requirements imposed at OS level for parallel computer systems;
- parallel programming environment characteristics;
- multitasking environment, and
- features of Parallel UNIX.

1.2 PARALLEL PROGRAMMING ENVIRONMENT CHARACTERISTICS

The parallel programming environment consists of an editor, a debugger, performance evaluator and programme visualizer for enhancing the output of parallel computation. All programming environments have these tools in one form or the other. Based on the features of the available tool sets the programming environments are classified as basic, limited, and well developed. The Basic environment provides simple facilities for program tracing and debugging. The limited integration facilities provide some additional tools for parallel debugging and performance evaluation. Well-developed environments provide most advanced tools of debugging programs, for textual graphics interaction and for parallel graphics handling.

There are certain parallel overheads associated with parallel computing. The parallel overhead is the amount of time required to coordinate parallel tasks, as opposed to doing useful work. These include the following factors:

i) Task start up time
ii) Synchronisations
iii) Data communications.

Besides these hardware overheads, there are certain software overheads imposed by parallel compilers, libraries, tools and operating systems.

The parallel programming languages are developed for parallel computer environments. These are developed either by introducing new languages (e.g. occam) or by modifying existing languages like, (FORTRAN and C). Normally the language extension approach is preferred by most computer designs. This reduces compatibility problem. High-level parallel constructs were added to FORTRAN and C to make these languages suitable for parallel computers. Besides these, optimizing compilers are designed to automatically detect the parallelism in program code and convert the code to parallel code.

In addition to development of languages and compilers for parallel programming, a parallel programming environment should also have supporting tools for development and text editing of parallel programmes.

Let us now discuss the examples of parallel programming environments of Cray Y-MP software and Intel paragaon XP/S.

The Cray Y-MP system works with UNICOS operating system. It has two FORTRAN compilers CFT 77 and CFT for automatic vector code generation. The system software has large library of routines, program management utilities, debugging aids and assembler UNICOS written in C. It supports optimizing, vectorizing, concurrentising facilities for FORTRAN compilers and also has optimizing and vetorizing C compiler. The Cray Y-MP has three multiprocessing/multitasking methods namely, (i) macrotasking, (ii) microtasking, (iii) autotasking. Also, it has a subroutine library, containing various utilities, high performance subroutines along with math and scientific routines.

The Intel Paragaon XP/S system is an extension of Intel iPSC/860 and Delta systems, and is a scalable and mesh connected multicompiler which is implemented in a distributed memory system.

The processors that for nodes of the system are 50 MHz i860 XP Processors. Further, it uses distributed UNIX based OS technology. The languages supported by Paragaon include C, C++, Data Parallel Fortran and ADA. The tools for integration include FORGE and Cast parallelisation tools. The programming environment includes an Interactive Parallel Debugger (IPD).

1.3 SYNCHRONIZATION PRINCIPLES

In multiprocessing, various processors need to communicate with each other. Thus, synchronisation is required between them. The performance and correctness of parallel execution depends upon efficient synchronisation among concurrent computations in multiple processes. The synchronisation problem may arise because of sharing of writable data objects among processes. Synchronisation includes implementing the order of operations in an algorithm by finding the dependencies in writable data. Shared object access in an MIMD architecture requires dynamic management at run time, which is much more complex as compared to that of SIMD architecture. Low-level synchronization primitives are implemented directly in hardware. Other resources like CPU, Bus and memory unit also need synchronisation in Parallel computers.

To study the synchronization, the following dependencies are identified:

i) **Data Dependency:** These are WAR, RAW, and WAW dependency.

ii) **Control dependency:** These depend upon control statements like GO TO, IF THEN, etc.

iii) **Side Effect Dependencies:** These arise due to exceptions, Traps, I/O accesses.

For the proper execution order as enforced by correct synchronization, program dependencies must be analysed properly. Protocols like wait protocol, and sole access protocol are used for doing synchronisation.

1.3.1 Wait protocol

The wait protocol is used for resolving the conflicts, which arise because of a number of multiprocessors demanding the same resource. There are two types of wait protocols: busy-wait and sleep-wait. In busy-wait protocol, process stays in the process context register, which continuously tries for processor availability. In sleep-wait protocol, wait protocol process is removed from the processor and is kept in the wait queue. The hardware complexity of this protocol is more than busy-wait in multiprocessor system; if locks are used for synchronization then busy-wait is used more than sleep-wait.

Execution modes of a multiprocessor: various modes of multiprocessing include parallel execution of programs at (i) Fine Grain Level (Process Level), (ii) Medium Grain Level (Task Level), (iii) Coarse Grain Level (Program Level).

For executing the programs in these modes, the following actions/conditions are required at OS level.

i) Context switching between multiple processes should be fast. In order to make context switching easy multiple sets should be present.

ii) The memory allocation to various processes should be fast and context free.

iii) The Synchronization mechanism among multiple processes should be effective.

iv) OS should provide software tools for performance monitoring.

1.3.2 Sole Access Protocol

The atomic operations, which have conflicts, are handled using sole access protocol. The method used for synchronization in this protocol is described below:

1) **Lock Synchronization:** In this method contents of an atom are updated by requester process and sole access is granted before the atomic operation. This method can be applied for shared read-only access.

2) **Optimistic Synchronization:** This method also updates the atom by requester process, but sole access is granted after atomic operation via abortion. This technique is also called post synchronisation. In this method, any process may secure sole access after first completing an atomic operation on a local version of the atom, and then executing the global version of the atom. The second operation ensures the concurrent update of the first atom with the updation of second atom.

3) **Server synchronization:** It updates the atom by the server process of requesting process. In this method, an atom behaves as a unique update server. A process requesting an atomic operation on atom sends the request to the atom's update server.

Check Your Progress 1

1) In sleep-wait synchronization mechanism, we know a process is removed/suspended from the processor and put in a wait queue. Suggest some fairness policies for reviving removed/suspended process
……………………………………………………………………………………………
……………………………………………………………………………………………
……………………………………………………………………………………………
……………………………………………………………………………………………

1.4 MULTI TASKING ENVIRONMENT

Multi tasking exploits parallelism by:

1) Pipelining functional units are pipe line together
2) Concurrently using the multiple functional units
3) Overlapping CPU and I/O activities.

In multitasking environment, there should be a proper mix between task and data structures of a job, in order to ensure their proper parallel execution.

In multitasking, the useful code of a programme can be reused. The property of allowing one copy of a programme module to be used by more than task in parallel is called reentrancy. Non-reentrant code can be used only once during lifetime of the programme. The reentrant codes, which may be called many times by different tasks, are assigned with local variables.

Shared variable programme structures

The concept of shared variable has already been discussed above. In this section, we discuss some more concepts related to the shared programme.

1.4.1 Concept of Lock

Locks are used for protected access of data in a shared variable system. There are various types of locks:

1) **Binary Locks:** These locks are used globally among multiple processes.

2) **Deckkers Locks:** These locks are based on distributed requests, to ensure mutual exclusion without unnecessary waiting.

1.4.2 System Deadlock

A deadlock refers to the situation when concurrent processes are holding resources and preventing each other from completing their execution.

The following conditions can avoid the deadlock from occurring:

1) **Mutual exclusion:** Each process is given exclusive control of the resources allotted to it.

2) **Non-preemption:** A process is not allowed to release its resources till task is completed.

3) **Wait for:** A process can hold resources while waiting for additional resources.

4) **Circular wait:** Multiple processes wait for resources from the other processes in a circularly dependent situation.

1.4.3 Deadlock Avoidance

To avoid deadlocks two types of strategies are used:

1) **Static prevention:** It uses P and V operators and Semaphores to allocate and deallocate shared resources in a multiprocessor. Semaphores are developed based on sleep wait protocol. The section of programme, where a deadlock may occur is called critical section. Semaphores are control signals used for avoiding collision between processes.

   P and V technique of Deadlock prevention associates a Boolean value 0 or 1 to each semaphore. Two atomic operators, P and V are used to access the critical section represented by the semaphore. The P(s) operation causes value of semaphore s to be increased by one if s is already at non-zero value.

   The V(s) operation increases the value of s by one if it is not already one. The equation s=1 indicates the availability of the resource and s=0 indicates non-availability of the resource.

   During execution, various processes can submit their requests for resources asynchronously. The resources are allocated to various processors in such a way that they do not create circular wait. The shortcoming of the static prevention is poor resource utilisation.

2) Another method of deadlock prevention is **dynamic deadlock avoidance**. It checks deadlocks on runtime condition, which may introduce heavy overhead in detecting potential existence of deadlocks.

Check Your Progress 2

1) Explain the spin lock mechanism for synchronisation among concurrent processes. Then define a binary spin lock.

……………………………………………………………………………………
……………………………………………………………………………………
……………………………………………………………………………………
……………………………………………………………………………………

1.5 MESSAGE PASSING PROGRAMME DEVELOPMENT ENVIRONMENT

In a multicomputer system, the computational load between various processors must be balanced. To pass information between various nodes, message-passing technique is used. The programming environment of a multicomputer includes a host runtime system and resident operating system called Kernel in all the node computers. The host system provides uniform communication between processes without intervention of the nodes, host workstation and network connection. Host processes are located outsides the nodes. The host environment for hypercube computer at Caltech is a UNIX processor and uses UNIX and language processor utilities to communicate with node processes. The Kernel is separately located in each node computer that supports multiprogramming with an address space confined by local memory. Many node processes can be created at each node. All node processes execute concurrently in different physical node or interleaved through multiprogramming within the same node. Node processes communicate with each other by sending or receiving messages.

The messages can be of various types. A particular field of all messages can be reserved to represent message type. The message passing primitives are as follows:

- Send (type, buffer, length, node process)
- Receive (type, buffer, length)

Where type identifies the message type, buffer indicates location of the message, length specifies the length of the message, node designates the destination node and process specifies process ID at destination node. Send and receive primitives are used for denoting the sending and receiving processes respectively. The buffer field of send specifies the memory location from which messages are sent and the buffer field of receive indicates the space where arriving messages will be stored.
The following issues are decided by the system in the process of message passing:

1) Whether the receiver is ready to receive the message
2) Whether the communication link is established or not
3) One or more messages can be sent to the same destination node

The message passing can be of two types: Synchronous and Asynchronous

In **Synchronous**, the message passing is implemented on synchronous communication network. In this case, the sender and receiver processes must be synchronized in time and space. Time synchronization means both processes must be ready before message transmission takes place. The space synchronization demands the availability of interconnection link.

In **Asynchronous** message passing, message-sending ands receiving are not synchronized in time and space. Here, the store and forward technology may be used. The blocked messages are buffered for later transmission. Because of finite size buffer the system may be blocked even in buffered Asynchronous non-blocking system.

1.6 UNIX FOR MULTIPROCESSOR SYSTEM

The UNIX operating system for a multiprocessor system has some additional features as compared to the normal UNIX operating system. Let us first discuss the design goals of the multiprocessor UNIX. The original UNIX developed by Brian Kernighan and Dennis Ritchie was developed as portable, general purpose, time-sharing uniprocessor operating system.

The OS functions including processor scheduling, virtual memory management, I/O devices etc, are implemented with a large amount of system software. Normally the size of the OS is greater than the size of the main memory. The portion of OS that resides in the main memory is called kernel. For a multiprocessor, OS is developed on three models viz: Master slave model, floating executive model, multithreaded kernel. These UNIX kernels are implemented with locks semaphores and monitors.
Let us discuss these models in brief.

1) **Master slave kernel:** In this model, only one of the processors is designated as Master.

   The master is responsible for the following activities:

   i) running the kernel code

   ii) handling the system calls

   iii) handling the interrupts.

   The rest of the processors in the system run only the user code and are called slaves.

2) **Floating-Executive model:** The master-slave kernel model is too restrictive in the sense that only one of the processors viz the designated master can run the kernel. This restriction may be relaxed by having more than one processors capable of running the kernel and allowing additional capability by which the master may float among the various processors capable of running the kernel.

3) **Multi-threaded UNIX kernel:** We know that threads are light-weight processors requiring minimal state information comprising the processor state and contents of relevant registers. A thread being a (light weight) process is capable of executing alone. In a multiprocessor system, more than one processors may execute simultaneously with each processor possibly executing more than one threads, with

the restriction that those threads which share resources must be allotted to one processor. However, the threads which do not share resources may be allotted to different processors. In this model, in order to separate multiple threads requiring different sets of kernel resources spin locks or semaphores are used.

1.7 SUMMARY

In this unit, various issues relating to operating systems and other software requirements for parallel computers are discussed.

In parallel systems, the issue of synchronisation becomes very important, specially in view of the fact more than one processes may require the same resource at same point of time. For the purpose, synchronisation principles are discussed in section 1.3. For the purpose of concurrent sharing of the memory in multitasking environment, the concepts of lock, deadlock etc are discussed in section 1.4. Finally, extensions of UNIX for multiprocessor systems are discussed in section 1.6.

1.8 SOLUTIONS/ANSWERS

Check Your Progress 1

1) Out of large number of fairness policies, the following three policies are quite well known:

   1) First-In-First-Out (FIFO) policy
   2) Upper-bounded misses policy
   3) Livelock-free policy

   First-In-First-Out (FIFO) as the name suggests that no process will be served out of turn. The disadvantage is that in some cases cost of computation and memory requirement may be too high.

   The second fairness policy may be implemented in a number of ways. Depending upon the implementation, the implementation costs and memory requirements may be reduced.

Check Your Progress 2

1) Under the centralized shared (CS) memory, the gate for entry and exist to CS is controlled by a single binary variable say y, which is then shared by all processes attempting to access CS.

   <u>For defining the binary spin lock</u>
   Let y be the single variable for entry to the gate. Initially the value of y is set to 0 (zero) indicating that entry by any one process is allowed. Then spin lock controlling mechanism continuously checks one by one all processes in turn, whether any one of these processes needs to access the memory. Once, any of the processes is found need to access CS, that process say Pp is allowed to access the CS and the entry-allowing variable y is assigned 0 (zero) indicating next process in queue requiring access to CS is allowed to access CS.

1.9 FURTHER READINGS

1) Kai Hwang: *Advanced Computer Architecture: Parallelism, Scalability, Programmability*, Tata-McGraw-Hill (2001)

2) Thomas L. Casavant, Pavel Tvrdik, Frantisck Plasil, *Parallel Computers: Theory and Practice*, IEEE

UNIT 2 PERFORMANCE EVALUATIONS

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*.

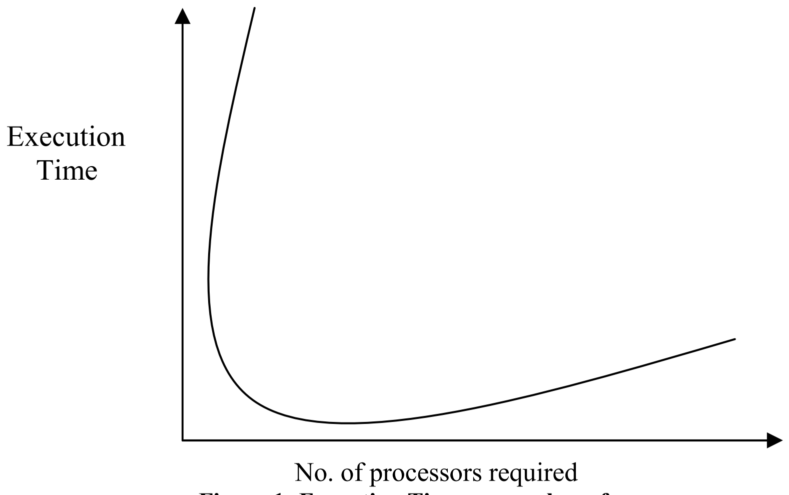

Figure 1: Execution Time vs. number of processors

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.

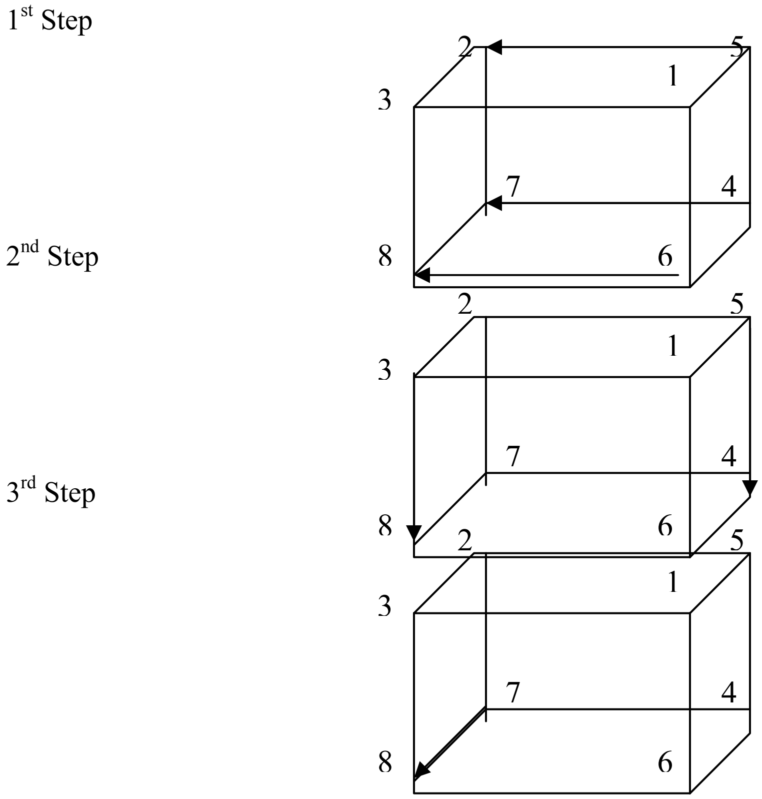

Figure 2: Steps followed for multiplying n numbers stored on n processors

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.

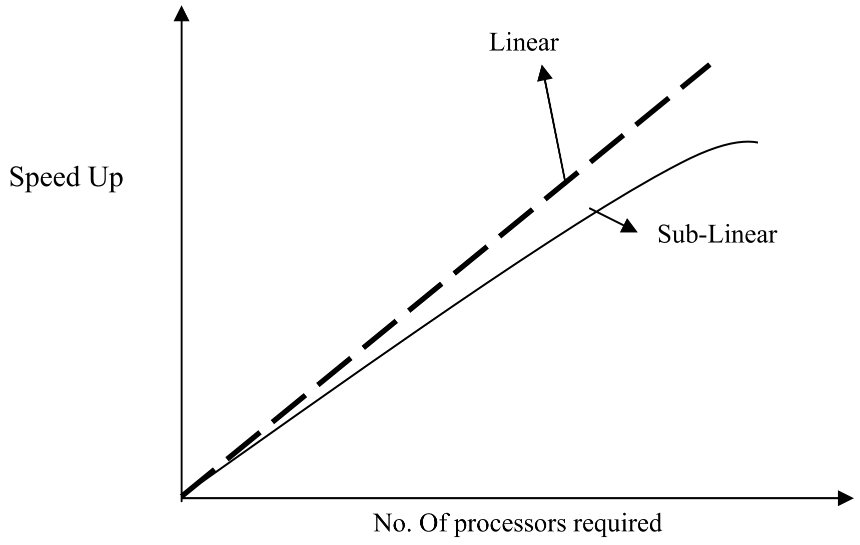

Figure 3: Speed-up vs. Number of Processors

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*.

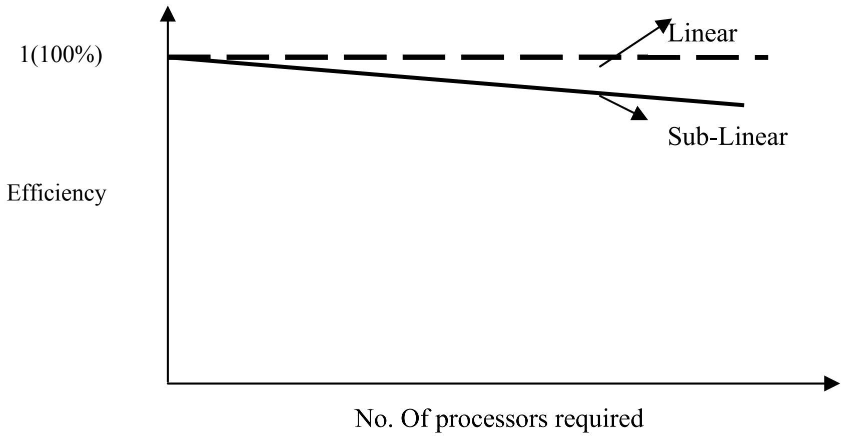

Figure 4: Efficiency vs. Number of Processors

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

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

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

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*).

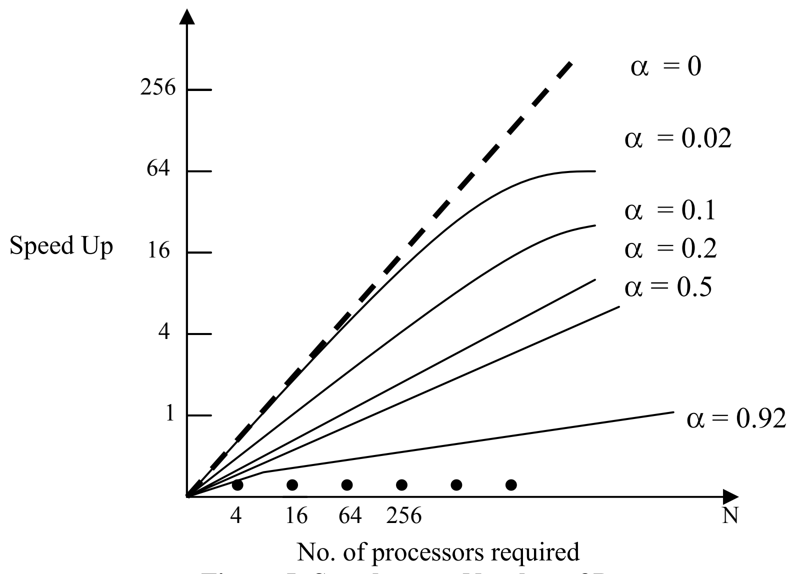

Figure 5: Speed-up vs. Number of Processors

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.

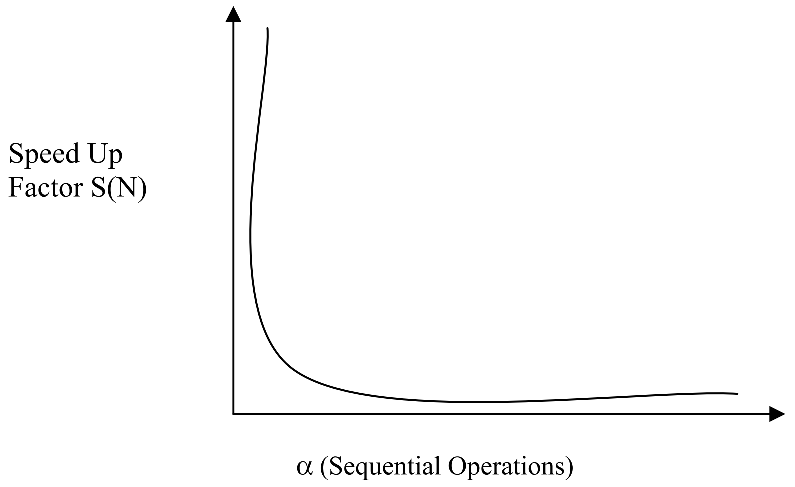

Figure 6: S (n) vs. α (Graph is not to scale)

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.

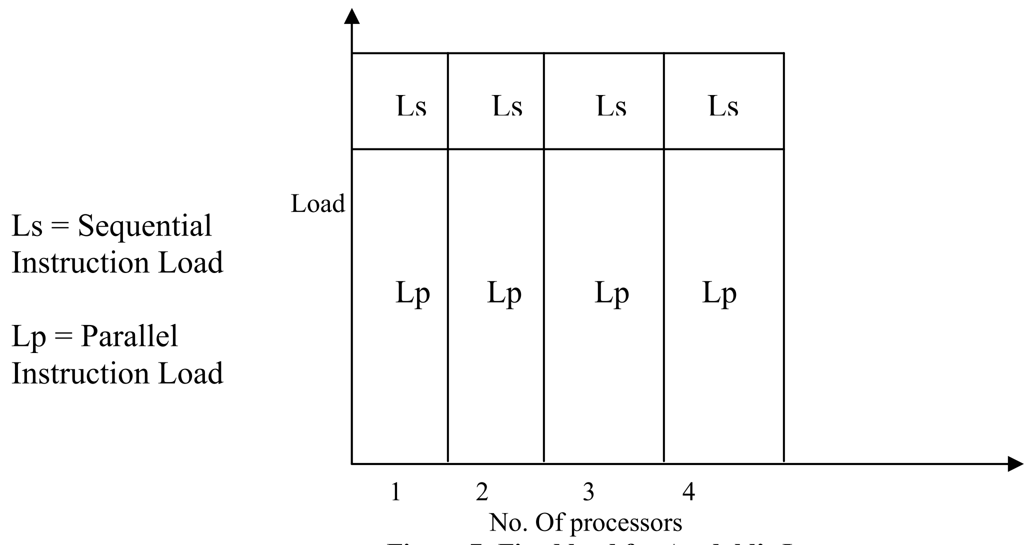
Ls = Sequential Instruction Load
Lp = Parallel Instruction Load
Load
Ls
Ls
Ls
Ls
Lp
Lp
Lp
Lp
1
2
3
4
No. Of processors

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.

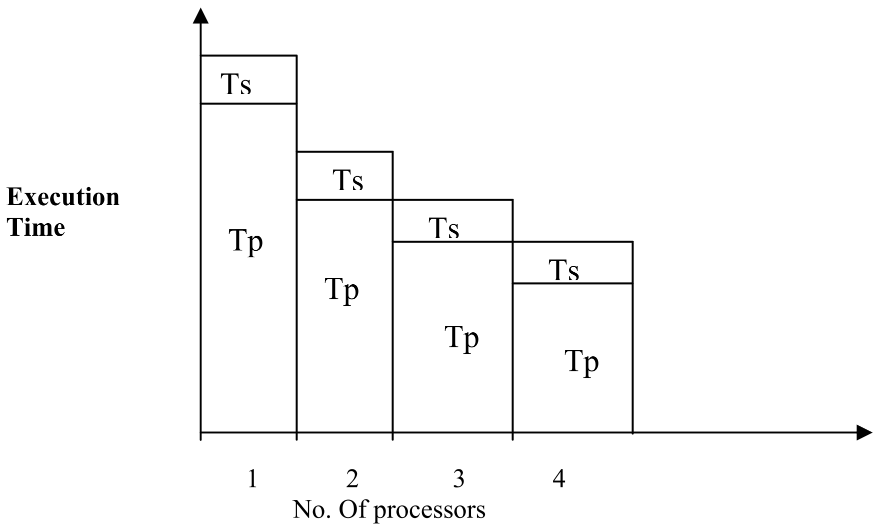

Figure 8: Execution Time decreases for Amdahl's Law

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.*

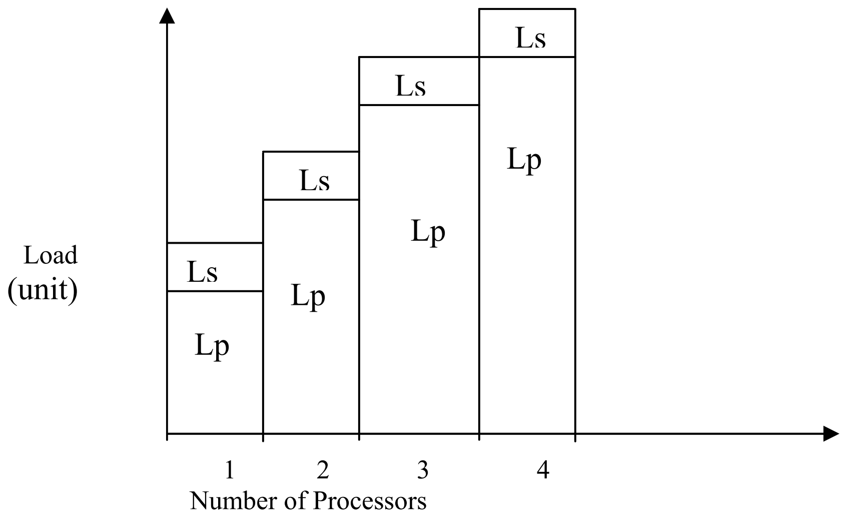

Figure 9: Parallel Load Increases for Gustafson's Law

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,*

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

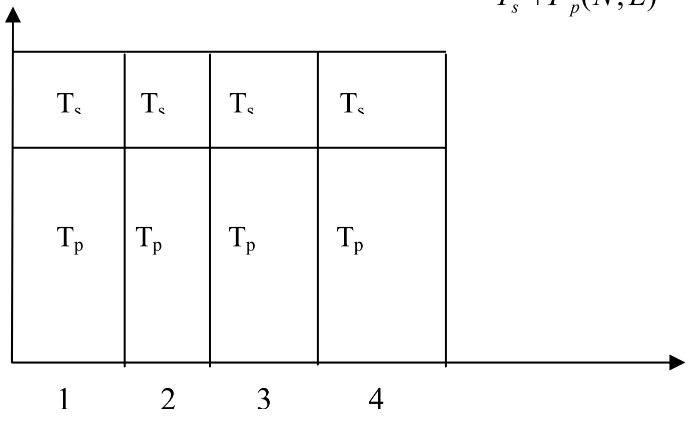

Figure 10: Fixed Execution Time for Gustafson's Law

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

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.

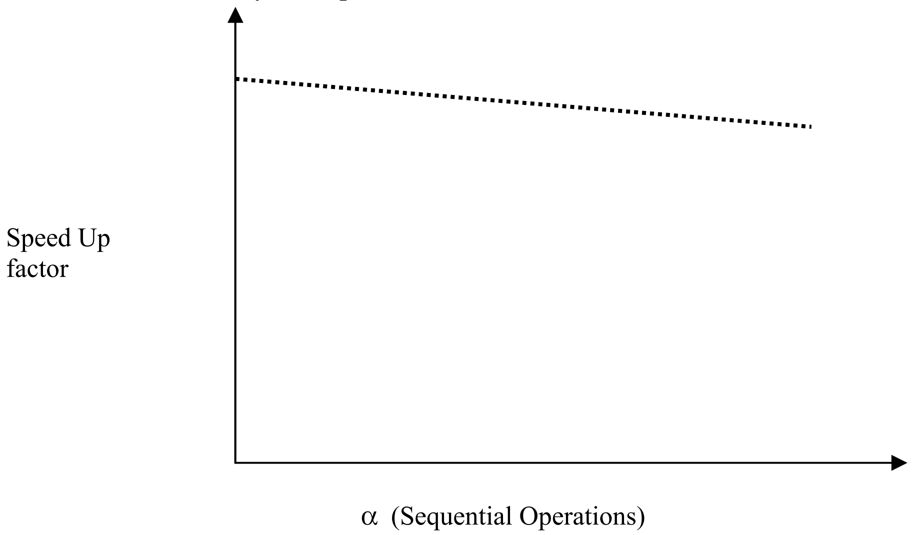

Figure 11: S (N) vs. α (Graph is not to scale)

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*.

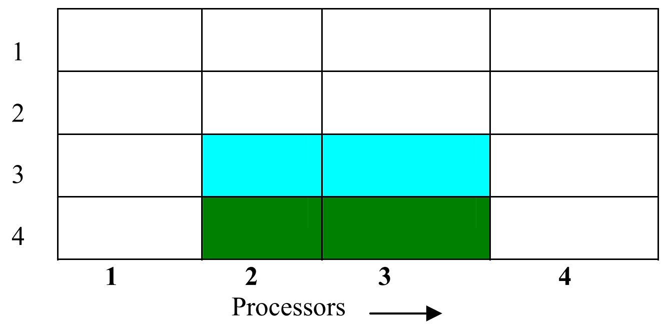

Figure: 12 Communication Matrix

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.

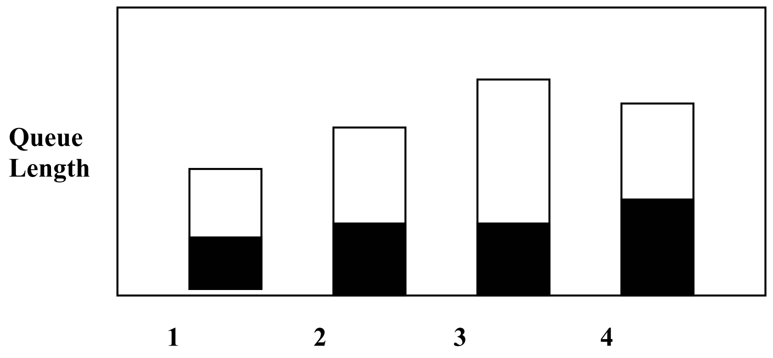

Figure 13: Message Queues

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*.

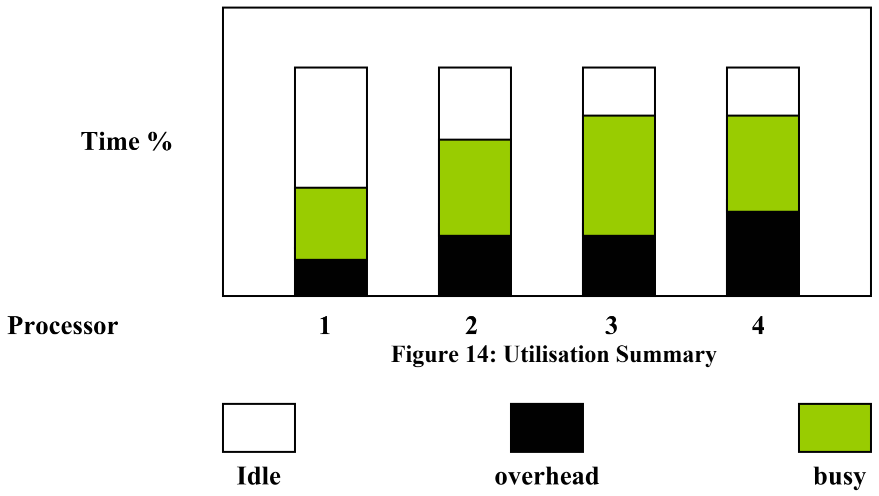

Figure 14: Utilisation Summary

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*.

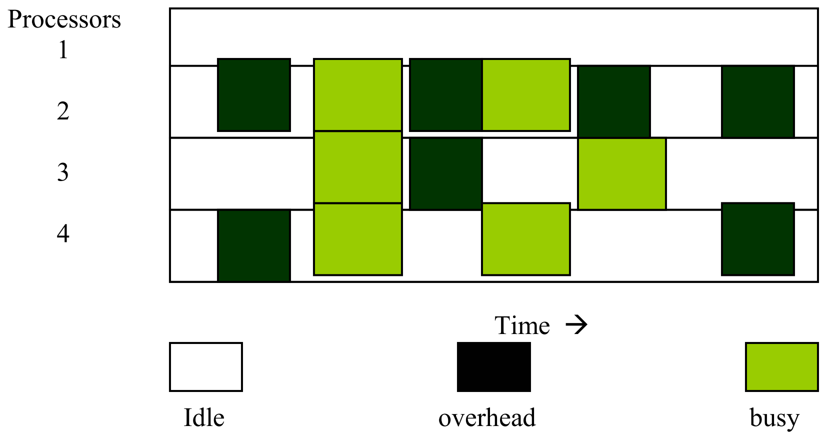

Figure 15: Utilisation Count

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.*

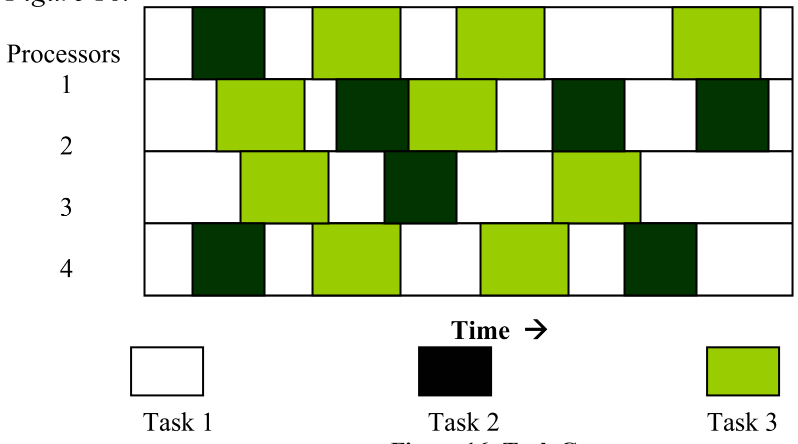

Figure 16: Task Gantt

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.*

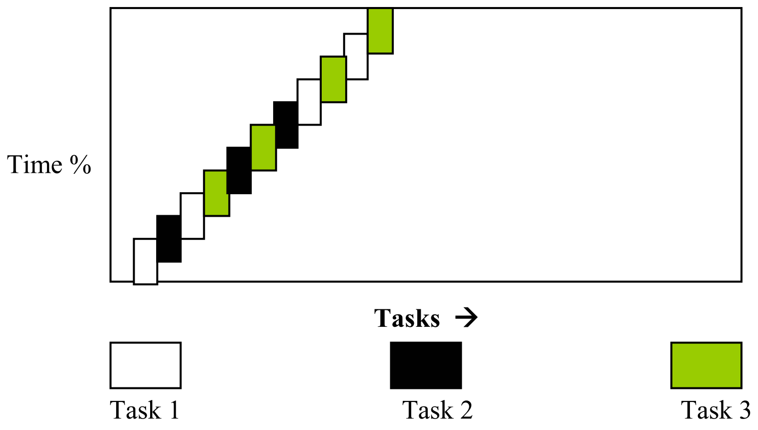

Figure 17: Summary of Tasks

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)

UNIT 3 RECENT TRENDS IN PARALLEL COMPUTING

Structure **Page Nos.**

3.0 INTRODUCTION

This unit discusses the current trends of hardware and software in parallel computing. Though the topics about various architectures, parallel program development and parallel operating systems have already been discussed in the earlier units, here some additional topics are discussed in this unit.

3.1 OBJECTIVES

After studying this unit you should be able to describe the

- various parallel programming models;
- concept of grid computing;
- concept of cluster computing;
- IA-64 architecture, and
- concept of Hyperthreading.

3.2 RECENT PARALLEL PROGRAMMING MODELS

A model for parallel programming is an abstraction and is machine architecture independent. A model can be implemented on various hardware and memory architectures. There are several parallel programming models like Shared Memory model, Threads model, Message Passing model, Data Parallel model and Hybrid model etc.

As these models are hardware independent, the models can (theoretically) be implemented on a number of different underlying types of hardware.

The decision about which model to use in an application is often a combination of a number of factors including the available resources and the nature of the application. There is no universally best implementation of a model, though there are certainly some implementations of a model better than others. Next, we discuss briefly some popular models.

1) Shared Memory Model

In the shared-memory programming model, tasks share a common address space, which they read and write asynchronously. Various mechanisms such as locks / semaphores may be used to control access to the shared memory. An advantage of this model from the programmer's point of view is that program development can often be simplified. An important disadvantage of this model (in terms of performance) is that for this model, data management is difficult.

2) Threads Model

In this model a single process can have multiple, concurrent execution paths. The main program is scheduled to run by the native operating system. It loads and acquires all the necessary softwares and user resources to activate the process. A thread's work may best be described as a subroutine within the main program. Any thread can execute any one subroutine and at the same time it can execute other subroutine. Threads communicate with each other through global memory. This requires Synchronization constructs to insure that more than one thread is not updating the same global address at any time. Threads can be created and destroyed, but the main program remains live to provide the necessary shared resources until the application has completed. Threads are commonly associated with shared memory architectures and operating systems.

3) Message Passing Model

In the message-passing model, there exists a set of tasks that use their own local memories during computation. Multiple tasks can reside on the same physical machine as well across an arbitrary number of machines. Tasks exchange data by sending and receiving messages. In this model, data transfer usually requires cooperation among the operations that are performed by each process. For example, a send operation must have a matching receive operation.

4) Data Parallel Model

In the data parallel model, most of the parallel work focuses on performing operations on a data set. The data set is typically organised into a common structure, such as an array or a cube. A set of tasks work collectively on the same data structure with each task working on a different portion of the same data structure. Tasks perform the same operation on their partition of work, for example, “add 3 to every array element” can be one task. In shared memory architectures, all tasks may have access to the data structure through the global memory. In the distributed memory architectures, the data structure is split up and data resides as “chunks” in the local memory of each task.

5) Hybrid model

The hybrid models are generally tailormade models suiting to specific applications. Actually these fall in the category of mixed models. Such type of application-oriented models keep cropping up. Other parallel programming models also exist, and will continue to evolve corresponding to new applications.

In this types of models, any two or more parallel programming models are combined. Currently, a common example of a hybrid model is the combination of the message passing model (MPI) with either the threads model (POSIX threads) or the shared memory model (OpenMP). This hybrid model lends itself well to the increasingly common hardware environment of networked SMP machines.

Another common example of a hybrid model is combining data parallel model with message passing model. As mentioned earlier in the data parallel model, data parallel implementations (F90, HPF) on distributed memory architectures actually use message passing to transmit data transparently between tasks and the programmer.

6) Single Program Multiple Data (SPMD)

SPMD is actually a "high level" programming model that can be built upon any combination of the previously mentioned parallel programming models. A single program is executed by all tasks simultaneously. SPMD programs usually have the necessary logic programmed into them to allow different tasks to branch or conditionally execute only those parts of the program they are designed to execute. That is, tasks do not necessarily have to execute the entire program, they may execute only a portion of it. In this model, different tasks may use different data.

7) Multiple Program Multiple Data (MPMD)

Like SPMD, MPMD is actually a "high level" programming model that can be built upon any combination of the previously mentioned parallel programming models.
MPMD applications typically have multiple executable object files (programs). While the application is being run in parallel, each task can be executed on the same or different program. In this model all tasks may use different data.

3.3 PARALLEL VIRTUAL MACHINE (PVM)

PVM is basically a simulation of a computer machine running parallel programs. It is a software package that permits a heterogeneous collection of Unix and/or Windows computers hooked together by a network to be used as a single large parallel computer. It includes a combination of good features of various operating systems and architectures. Thus large computational problems can be solved more cost effectively by using the combined power and memory of many computers. The software is highly portable. The source, which is available free through netlib, has been compiled on a range of computing machines from laptops to CRAY machines.

PVM enables users to exploit their existing computer hardware to solve much larger problems at minimal additional cost. Hundreds of sites around the world are using PVM to solve important scientific, industrial, and medical problems. Further, PVM's are being used as an educational tools to teach parallel programming. With tens of thousands of users, PVM has become the de facto standard for distributed computing world-wide.

Check Your Progress 1

1) Explain the operations in the following message passing operating system model
   1) Object Oriented Model
   2) Node addressed model
   3) Channel addressed model

2) Explain various Multi Processor Execution Node.

   …………………………………………………………………………………………
   …………………………………………………………………………………………
   …………………………………………………………………………………………
   …………………………………………………………………………………………

3) What are the levels at which multitasking is exploited?

   …………………………………………………………………………………………
   …………………………………………………………………………………………
   …………………………………………………………………………………………
   …………………………………………………………………………………………

3.4 GRID COMPUTING

Grid Computing means applying the resources of many computers in a network simultaneously to a single problem for solving a scientific or a technical problem that requires a large number of computer processing cycles or accesses to large amounts of data. Grid computing uses software to divide and distribute pieces of a program to as many as several thousand computers. A number of corporations, professional groups and university consortia have developed frameworks and software for managing grid computing projects.

Thus, the Grid computing model allows companies to use a large number of computing resources on demand, irrespective of where they are located. Various computational tasks can be performed using a computational grid. Grid computing provides clustering of remotely distributed computing environment. The principal focus of grid computing to date has been on maximizing the use of available processor resources for compute-intensive applications. Grid computing along with storage virtualization and server virtualization enables a utility computing. Normally it has a Graphical User Interface (GUI), which is a program interface based on the graphics capabilities of the computer to create screens or windows.

Grid computing uses the resources of many separate computers connected by a network (usually the internet) to solve large-scale computation problems. The SETI@home project, launched in the mid-1990s, was the first widely-known grid computing project, and it has been followed by many others project covering tasks such as protein folding, research into drugs for cancer, mathematical problems, and climate models.

Grid computing offers a model for solving massive computational problems by making use of the unused resources (CPU cycles and/or disk storage) of large numbers of disparate, often desktop, computers treated as a virtual cluster embedded in a distributed telecommunications infrastructure. Grid computing focuses on the ability to support computation across administrative domains which sets it apart from traditional computer clusters or traditional distributed computing.

Various systems which can participate in the grid computing as platform are:
Windows 3.1, 95/98, NT, 2000 XP , DOS, OS/2, , supported by Intel (x86);
Mac OS A/UX (Unix) supported by Motorola 680 x0;
Mac OS , AIX (Unix), OS X (Unix) supported by Power PC;
HP / UX (Unix) supported by HP 9000 (PA – RISC);
Digital Unix open VMS, Windows NT spported by Compaq Alpha;
VMS Ultrix (Unix) supported by DEC VAX;
Solaris (Unix) supported by SPARC station;
AIX (Unix) supported by IBM RS / 6000;
IRIS (Unix) supported by Silicon Graphics workstation.

3.5 CLUSTER COMPUTING

The concept of clustering is defined as the use of multiple computers, typically PCs or UNIX workstations, multiple storage devices, and their interconnections, to form what appears to users as a single highly available system. Workstation clusters is a collection of loosely-connected processors, where each workstation acts as an autonomous and independent agent. The cluster operates faster than normal systems.

In general, a 4-CPU cluster is about 250~300% faster than a single CPU PC. Besides, it not only reduces computational time, but also allows the simulations of much bigger computational systems models than before. Because of cluster computing overnight

analysis of a complete 3D injection molding simulation for an extremely complicated model is possible.
Cluster workstation defined in ***Silicon Graphics*** project is as follows:

"A distributed workstation cluster should be viewed as single computing resource and not as a group of individual workstations".

The details of the cluster were: 150MHz R4400 workstations, 64MB memory, 1 GB disc per workstation, 6 x 17" monitors, Cluster operating on local 10baseT Ethernet, Computing Environment Status, PVM, MPI (LAM and Chimp), Oxford BSP Library.

The operating system structure makes it difficult to exploit the characteristics of current clusters, such as low-latency communication, huge primary memories, and high operating speed. Advances in network and processor technology have greatly changed the communication and computational power of local-area workstation clusters. Cluster computing can be used for load balancing in multi-process systems

A common use of cluster computing is to balance traffic load on high-traffic Web sites. In web operation a web page request is sent to a manager server, which then determines which of the several identical or very similar web servers to forward the request to for handling. The use of cluster computing makes this web traffic uniform.

Clustering has been available since the 1980s when it was used in DEC's VMS systems. IBM's SYSLEX is a cluster approach for a mainframe system. Microsoft, Sun Microsystems, and other leading hardware and software companies offer clustering packages for scalability as well as availability. As traffic or availability assurance increases, all or some parts of the cluster can be increased in size or number. Cluster computing can also be used as a relatively low-cost form of parallel processing for scientific and other applications that lend themselves to parallel operations

An early and well-known example was the project in which a number of off-the-shelf PCs were used to form a cluster for scientific applications.

Windows cluster is supported by the Symmetric Multiple Processors (SMP) and by Moldex3D R7.1. While the users chose to start the parallel computing, the program will automatically partition one huge model into several individual domains and distribute them over different CPUs for computing. Every computer node will exchange each other's computing data via message passing interface to get the final full model solution. The computing task is parallel and processed simultaneously. Therefore, the overall computing time will be reduced greatly.

Some famous projects on cluster computing are as follows:

i) **High Net Worth Project**: (developed by: Bill McMillan, JISC NTI/65 – The HNW Project, University of Glasgow, (billm@aero.gla.ac.uk)

The primary aims / details of the project are:

- Creation of a High Performance Computing Environment using clustered workstations
- Use of pilot facility,
- Use of spare capacity,
- Use of cluster computing environment as a Resource Management system,
- Use of ATM communications,
- Parallel usage between sites across SuperJANET,
- Promoting Awareness of Project in Community.

ii) **The primary aims/details of Load Sharing Facility Resource Management Software(LSFRMS)** are better resource utilisation by routing the task to the most appropriate system and better utilisation of busy workstations through load sharing, and also by involving idle workstations.

Types of Users of the LSFRMS:

- users whose computational requirements exceed the capabilities of their own hardware;
- users whose computational requirements are too great to be satisfied by a single workstation;
- users intending to develop code for use on a national resource;
- users who are using national resources
- users who wish to investigate less probable research scenarios.

Advantages of using clusters:

- Better resource utilisation;
- Reduced turnaround time;
- Balanced loads;
- Exploitation of more powerful hosts;
- Access to non-local resources;
- Parallel and distributed applications.

Check Your Progress 2

1) Name any three platforms which can participate in grid computing.

…………………………………………………………………………………
…………………………………………………………………………………
…………………………………………………………………………………
…………………………………………………………………………………

2) Explain the memory organization with a cluster computing.

…………………………………………………………………………………
…………………………………………………………………………………
…………………………………………………………………………………
…………………………………………………………………………………

3) Find out the names of famous cluster computer projects.

…………………………………………………………………………………
…………………………………………………………………………………
…………………………………………………………………………………
…………………………………………………………………………………

4) Explain various generations of message passing multi computers.

…………………………………………………………………………………
…………………………………………………………………………………
…………………………………………………………………………………
…………………………………………………………………………………

3.6 INTEL ARCHITECTURE – 64 (IA-64)

IA-64 (Intel Architecture-64) is a 64-bit processor architecture developed in cooperation by Intel and Hewlett-Packard, implemented by processors such as Itanium. The goal of Itanium was to produce a “post-RISC era” architecture using EPIC(Explicitly Parallel Instruction Computing).

1 EPIC Architecture

In this system a complex decoder system examines each instruction as it flows through the pipeline and sees which can be fed off to operate in parallel across the available execution units — *e.g.*, a sequence of instructions for performing the computations
A = B + C and
D = F + G

These will be independent of each other and will not affect each other, and so they can be fed into two different execution units and run in parallel. The ability to extract instruction level parallelism (ILP) from the instruction stream is essential for good performance in a modern CPU.

Predicting which code can and cannot be split up this way is a very complex task. In many cases the inputs to one line are dependent on the output from another, but only if some other condition is true. For instance, consider the slight modification of the example noted before, A = B + C; IF A==5 THEN D = F + G. In this case the calculations remain independent of the other, but the second command requires the results from the first calculation in order to know if it should be run at all.

In these cases the circuitry on the CPU typically "guesses" what the condition will be. In something like 90% of all cases, an IF will be taken, suggesting that in our example the second half of the command can be safely fed into another core. However, getting the guess wrong can cause a significant performance hit when the result has to be thrown out and the CPU waits for the results of the "right" command to be calculated. Much of the improving performance of modern CPUs is due to better prediction logic, but lately the improvements have begun to slow. Branch prediction accuracy has reached figures in excess of 98% in recent Intel architectures, and increasing this figure can only be achieved by devoting more CPU die space to the branch predictor, a self-defeating tactic because it would make the CPU more expensive to manufacture.

IA-64 instead relies on the compiler for this task. Even before the program is fed into the CPU, the compiler examines the code and makes the same sorts of decisions that would otherwise happen at "run time" on the chip itself. Once it has decided what paths to take, it gathers up the instructions it knows can be run in parallel, bundles them into one larger instruction, and then stores it in that form in the program.

Moving this task from the CPU to the compiler has several advantages. First, the compiler can spend considerably more time examining the code, a benefit the chip itself doesn't have because it has to complete the task as quickly as possible. Thus the compiler version can be considerably more accurate than the same code run on the chip's circuitry. Second, the prediction circuitry is quite complex, and offloading a prediction to the compiler reduces that complexity enormously. It no longer has to examine anything; it simply breaks the instruction apart again and feeds the pieces off to the cores. Third, doing the prediction in the compiler is a one-off cost, rather than one incurred every time the program is run.

The downside is that a program's runtime-behaviour is not always obvious in the code used to generate it, and may vary considerably depending on the actual data being processed. The out-of-order processing logic of a mainstream CPU can make decisions on the basis of actual run-time data which the compiler can only guess at. It means that it is possible for the compiler to get its prediction wrong more often than comparable (or simpler) logic placed on the CPU. Thus this design this relies heavily on the performance of the compilers. It leads to decrease in microprocessor hardware complexity by increasing compiler software complexity.

Registers: The IA-64 architecture includes a very generous set of registers. It has an 82-bit floating point and 64-bit integer registers. In addition to these registers, IA-64 adds in a register rotation mechanism that is controlled by the Register Stack Engine. Rather than the typical spill/fill or window mechanisms used in other processors, the Itanium can rotate in a set of new registers to accommodate new function parameters or temporaries. The register rotation mechanism combined with predication is also very effective in executing automatically unrolled loops.

Instruction set: The architecture provides instructions for multimedia operations and floating point operations.

The Itanium supports several bundle mappings to allow for more instruction mixing possibilities, which include a balance between serial and parallel execution modes. There was room left in the initial bundle encodings to add more mappings in future versions of IA-64. In addition, the Itanium has individually settable predicate registers to issue a kind of runtime-determined "cancel this command" directive to the respective instruction. This is sometimes more efficient than branching.

Pre-OS and runtime sub-OS functionality

In a raw Itanium, a "Processor Abstraction Layer" (PAL) is integrated into the system. When it is booted the PAL is loaded into the CPU and provides a low-level interface that abstracts some instructions and provides a mechanism for processor updates distributed via a BIOS update.
During BIOS initialization an additional layer of code, the "System Abstraction Layer" (SAL) is loaded that provides a uniform API for implementation-specific platform functions.

On top of the PAL/SAL interface sits the "Extensible Firmware Interface" (EFI). EFI is not part of the IA-64 architecture but by convention it is required on all IA-64 systems. It is a simple API for access to logical aspects of the system (storage, display, keyboard, etc) combined with a lightweight runtime environment (similar to DOS) that allows basic system administration tasks such as flashing BIOS, configuring storage adapters, and running an OS boot-loader.

Once the OS has been booted, some aspects of the PAL/SAL/EFI stack remain resident in memory and can be accessed by the OS to perform low-level tasks that are implementation-dependent on the underlying hardware.

IA-32 support

In order to support IA-32, the Itanium can switch into 32-bit mode with special jump escape instructions. The IA-32 instructions have been mapped to the Itanium's functional units. However, since, the Itanium is built primarily for speed of its EPIC-style instructions, and because it has no out-of-order execution capabilities, IA-32 code executes at a severe performance penalty compared to either the IA-64 mode or the Pentium line of processors. For example, the Itanium functional units do not automatically generate integer flags as a side effect of ordinary ALU computation, and do not intrinsically support multiple outstanding unaligned memory loads. There are also IA-32 software emulators which are freely available for Windows and Linux, and these emulators typically outperform the hardware-based emulation by around 50%. The Windows emulator is available from Microsoft, the Linux emulator is available from some Linux vendors such as Novell and from Intel itself. Given the superior performance of the software emulator, and despite the fact that IA-32 hardware accounts for less than 1% of the transistors of an Itanium 2, Intel plan to remove the circuitry from the next-generation Itanium 2 chip codenamed "Montecito".

3.7 HYPER-THREADING

Hyper-threading, officially called **Hyper-threading Technology** (**HTT**), is Intel's trademark for their implementation of the simultaneous multithreading technology on the Pentium 4 microarchitecture. It is basically a more advanced form of Super-threading that was first introduced on the Intel Xeon processors and was later added to Pentium 4 processors. The technology improves processor performance under certain workloads by providing useful work for execution units that would otherwise be idle for example during a cache miss.

Features of Hyper-threading

The salient features of hyperthrading are:

i) Improved support for multi-threaded code, allowing multiple threads to run simultaneously.

ii) Improved reaction and response time, and increased number of users a server can support.

According to Intel, the first implementation only used an additional 5% of the die area over the "normal" processor, yet yielded performance improvements of 15-30%.
Intel claims up to a 30% speed improvement with respect to otherwise identical, non-SMT Pentium 4. However, the performance improvement is very application dependent, and some programs actually slow down slightly when HTT is turned on.

This is because of the replay system of the Pentium 4 tying up valuable execution resources, thereby starving for the other thread. However, any performance degradation is unique to the Pentium 4 (due to various architectural nuances), and is not characteristic of simultaneous multithreading in general.

Functionality of hypethread Processor

Hyper-threading works by duplicating those sections of processor that store the architectural state—but not duplicating the main execution resources. This allows a Hyper-threading equipped processor to pretend to be two "logical" processors to the host operating system, allowing the operating system to schedule two threads or processes simultaneously. Where execution resources in a non-Hyper-threading capable processor are not used by the current task, and especially when the processor is stalled, a Hyper-threading equipped processor may use those execution resources to execute the other scheduled task.

Except for its performance implications, this innovation is transparent to operating systems and programs. All that is required to take advantage of Hyper-Threading is symmetric multiprocessing (SMP) support in the operating system, as the logical processors appear as standard separate processors.

However, it is possible to optimize operating system behaviour on Hyper-threading capable systems, such as the Linux techniques discussed in Kernel Traffic. For example, consider an SMP system with two physical processors that are both Hyper-Threaded (for a total of four logical processors). If the operating system's process scheduler is unaware of Hyper-threading, it would treat all four processors similarly.

As a result, if only two processes are eligible to run, it might choose to schedule those processes on the two logical processors that happen to belong to one of the physical processors. Thus, one CPU would be extremely busy while the other CPU would be completely idle, leading to poor overall performance. This problem can be avoided by improving the scheduler to treat logical processors differently from physical processors; in a sense, this is a limited form of the scheduler changes that are required for NUMA systems.

The Future of Hyperthreading

Current Pentium 4 based MPUs use Hyper-threading, but the next-generation cores, Merom, Conroe and Woodcrest will not. While some have alleged that this is because Hyper-threading is somehow energy inefficient, this is not the case. Hyper-threading is a particular form of multithreading, and multithreading is definitely on Intel roadmaps for the generation after Merom/Conroe/Woodcrest. Quite a few other low power chips use multithreading, including the PPE from the Cell processor, the CPUs in the Playstation 3 and Sun's Niagara. Regarding the future of multithreading
the real question is not whether Hyper-threading will return, as it will, but rather how it will work. Currently, Hyper-threading is identical to Simultaneous Multi-Threading, but future variants may be different. In future trends parallel codes have been easily ported to the Pilot Cluster, often with improved results and Public Domain and Commercial Resource Management Systems have been evaluated in the Pilot Cluster Environment.

The proposed enhancements in these architecture are as follows:

- High Performance language implementation
- Further developments in Heterogeneous Systems
- Management of wider domains with collaborating departments
- Faster communications
- Inter-campus computing

Check Your Progress 3

1) How is performance enhanced in IA-64 Architecture?
……………………………………………………………………………………
……………………………………………………………………………………
……………………………………………………………………………………
……………………………………………………………………………………
2) How is performance judged on the basis of run time behaviour?
……………………………………………………………………………………
……………………………………………………………………………………
……………………………………………………………………………………
……………………………………………………………………………………
……………………………………………………………………………………
3) List some data parallelism feature of a modern computer architecture.
……………………………………………………………………………………
……………………………………………………………………………………
……………………………………………………………………………………
……………………………………………………………………………………

3.8 SUMMARY

This unit discusses basic issues in respect of latest trends of hardware and software technology for parallel computing systems. The technology of parallel computing is the outcome of four decades of research and industrial advances in various fields like microelectronics, integration technologies, advanced processors, memory and peripheral devices, programming language development, operating system developments etc.

The high performance computers which are developed using these concepts provide fast and accurate solutions to scientific, engineering, business, social and aerospace applications.

The unit discusses some of the latest architectures. However, the learner is advised to get the knowledge about further ongoing developments from related websites.

3.9 SOLUTIONS/ANSWERS

Check Your Progress 1

1) i) **Object Oriented Model:** This model treats multiple tasks or processor concurrently, each task using its own address space. In this model multitasking is practiced at each node.
The communication between different tasks is carried out by either at synchronous or asynchronous message passing protocol. Message passing supports IPC as well as process migration or load balancing.

ii) **Node Address Model:** In this only one task runs on each node at a time. It is implemented with asynchronous scheme. The actual topology is hidden from the user. Some models use store and forward message routing and some models use wormhole routing.

iii) **Channel Addressed Model:** This model runs one task on each node. Data are communicated by synchronous message passing for communication channel.

2) a) Multi Processor super computers are built in vector processing as well as for parallel processing across multiple processors. Various executions models are parallel execution from fine grain process level.
b) Parallel execution from coarse grain process level
c) Parallel execution to coarse grain exam level

3) Multi tasking exploits parallelism at the following level:

i) The different functional unit is pipeline together.
ii) Various functional units are used concurrently.
iii) I/O and CPU activities are overlapped.
iv) Multiple CPUs can operate on a single program to achieve minimum execution time.

In a multi tasking environment, the task and the corresponding data structure of a program are properly pertaining to parallel execution in a such a manner that conflicts will not occur.

Check Your Progress 2

1) The following platform can participate in grid computing.
i) Digital, Unit, Open, VMS
ii) AIX supported by IBM RS/6000.
iii) Solaris (Unix supported by SPARC station)

2) In cluster computer the processor are divided into several clusters. Each cluster is a UMA or NUMA multiprocessor. The clusters are connected to global shared memory model. The complete system works on NUMA model.

All processing elements belonging to it clusters are allowed to access the cluster shared memory modules.

All clusters have access to global memory. Different computers systems have specified different access right among Internet cluster memory. For example, CEDAR multi processor built at University of Illinois adopts architecture in each cluster is Alliant FX/80 multiprocessor.

3) Collect them from the website.

4) Multi computers have gone through the following generations of development. The first generally ranged from 1983-87. It was passed on Processor Board Technology using Hypercube architecture and Software controlled message switching. Examples of this generation computers are Caltech, Cosmic and Intel, PEPSC.

The second generation ranged from 1988-92. It was implemented with mesh connected architecture, hardware access routing environment in medium grain distributed computing. Examples of such systems are Intel Paragon and Parsys super node 1000.

The third generation ranged from 1983-97 and it is an era of fine grain computers like MIT, J Machine and Caltech mosaic.

Check Your Progress 3

1) The performance in IA 64 architecture is enhanced in a modern CPU due to vector prediction logic and in branch prediction technique the path which is expected to be taken as is anticipated in advance, in order to preset the required instruction in advance. Branch prediction accuracies has reached to more than 98% in recent Intel architecture, and such a high figure is achieved by devoting more CPU die space to branch prediction.

2) The run time behaviour of a program can be adjudged only at execution time. It depends on the actual data being processed. The out of order processing logic of a main stream CPU can make itself on the basis of actual run time data which only compiler can connect.

3) Various data parallelism depends as specified as to how data is access to distribute in SIMD or MIMD computer. These are as follows:

   i) **Run time automatic decomposition:** In this data are automatically distributed with no user intervention.

   ii) **Mapping specifications:** It provides the facility for user to specify the communication pattern.

   iii) **Virtual processor support:** The compiler maps virtual process dynamically and statically.

   iv) **Direct Access to sharing data:** Shared data can be directly accessed without monitor control.

3.10 FURTHER READINGS

1) Thomas L. Casavant, Pavel Tvrdik, Frantisck Plasil, *Parallel Computers: Theory and Applications*, IEEE Computer Society Press (1996)

2) Kai Hwang: *Advanced Computer Architecture: Parallelism*, Scalability and Programmability, Tata-McGraw-Hill (2001)