3. PARALLEL PROCESSING ISSUES

3.1 GENERAL

The art of parallel programming lies in being able to decompose a given problem, functionally and according to the data distribution, across processors. Over the years, work in parallel processing architectures have focused on a number of different issues and resulted in different approaches for solving them. The resulting architectures have been fundamentally so disparate that the programming issues vary significantly across them. An algorithm that gives a very high speedup on one processors architecture can give a very poor performance on another. Even for the MIMD class of parallel machines, difference between the shared memory and distributed memory machines force developers to adopt different data function decomposition strategies for the same problem on these classes' machines.

The fundamental concept behind high performance computing is to use more number of resources to solve a given problem. But using more resources cannot speedup every task. There are restrictions and dependencies to be considered. Based on such issues, computations may be of one of the following characteristics [2].

Embarrassingly Parallel Computations: An ideal parallel computation that can be divided into a number of completely independent parts each of which can be executed by a separate processor without any communication between them, is known as embarrassingly parallel computation or pleasantly parallel computation. Parallelizing these problems should be obvious and requires no special techniques to obtain working solution. Each process requires different data and produces results from its input without any need for results from other processors. This situation will give the maximum possible speedup if all the available processors can be assigned processes for the total duration of the computation. Example of such process can be given as follows:

a = b + c d = e + f

As the variables used in both the processes are altogether independent of each other, both the processes can run in parallel without having any effect on the final result. So such processes are called embarrassingly parallel computations.

Nearly Embarrassingly Parallel Computations: The computations that require results to be distributed and collected and combined in some way are known as nearly embarrassingly parallel computations. So in such computations, initially and finally a single process must be operating alone. If dynamic process creation is used, a common approach is the master-slave organization. First, a master approach will be started that will initiate appropriate slave processes.

Inherently Sequential Computations: The computations, in which the parallelism of processes is not at all possible, are known as inherent sequential computations. Consider the following example consisting two processes of single computation task:

```
a = b + c          d = a * a
```

As in the above processes the value of variable "d" is dependent on the value of variable "a", it is necessary to modify the value of "a" before using it to calculate the value of "d". So these two processes cannot run in parallel and must be run sequentially. This computation is an example of inherent sequential computing.

For better computational efficiency embarrassingly parallel problems and nearly embarrassingly parallel problems can be implemented on multi processor or multi computer systems. Multi processing system consisting of number of processors connected together and utilizing common or different memory module is known as tightly coupled multiprocessing systems. This is also considered as dedicated hardware for parallel processing. Alternative system consisting of network of personal computers is known as loosely coupled multi computer system or distributed computing platform. Many times parallel computing and distributed computing are used interchangeably. It can be safely said that the computation can be distributed within a parallel computing environment, or that the distributed computing environment can be used to exploit the parallelism in some computations.

Various issues related to tightly coupled and loosely coupled multiprocessing systems like hardware requirements, operating system, connection topology, and software tools available etc. are discussed in subsequent sections [93, 94, 95].

3.2 TIGHTLY COUPLED MULTIPROCESSING SYSTEMS

Generally multiprocessor machines are constructed using general-purpose microprocessors and much of the functionality required of a processing unit has to be compensated by the operating system software.

3.2.1 Desirable Characteristics

Process recoverability: If the processor fails, another processor must be assigned to the process / task and must be continued from the state where the processor failed. This can be achieved by maintaining a shared register file that keeps a record of the state for which active process in the system.

Efficient Context switching: When parallel machine have more processes than the number of processors, there is a need for swapping the processes in and out of context. The processor must have efficient mechanism to support operating systems in switching the context efficiently.

Large virtual and physical address space: With the increase in the size of problem to be solved by parallel machine, there is growing need for larger memory space and addressing capabilities.

Effective synchronization primitives: Multiple processes of parallel program need to cooperate to compute the results. This can be done by sharing data and accessing common data while maintaining integrity. Such synchronization must be supported at instruction level by the processors to facilitate switchover of access.

Interprocess communication mechanism: The communication between cooperating processes takes place in form of signals, messages, and interrupts.

3.2.2 Connection Topology

The sharing capability is provided through a set of two interconnection networks - one between processors and memory modules and the other between the

processors and the I/O subsystems. The basic types of interconnections are as follows:

Time shard or common bus: It is a common communication path connecting all the functional units. As the bus is shared, a mechanism must be provided to resolve contention between various processors and devices for access to bus for interconnection. Refer Fig 3.1. A centralized bus controller simplifies the conflict resolution but may have performance bottleneck. The overall system capacity is limited by the bus transfer rate. Time share bus organization is appropriate for small system upto 10 units.

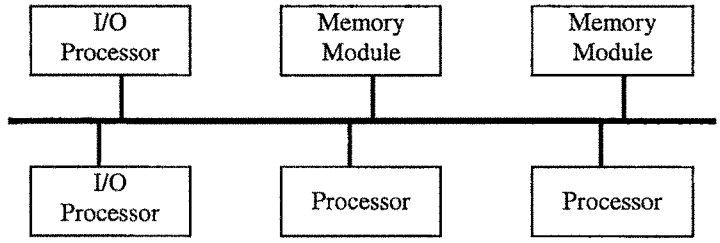

FIG. 3.1 TIME – SHARED BUS

Crossbar switch: It provides complete connectivity with respect to the memory modules because as shown in Fig. 3.2 there is a separate bus associated with respect to the memory modules. The maximum number of transfer taken place simultaneously depends on number of memory modules and the bandwidth of the buses. The crossbar switch mechanism provides for multiple streams of communication between the modules to be active in parallel. The access to devices and memory modules by the processors is arbitrated by switch. This type of interconnection system is complex to design and expensive to fabricate.

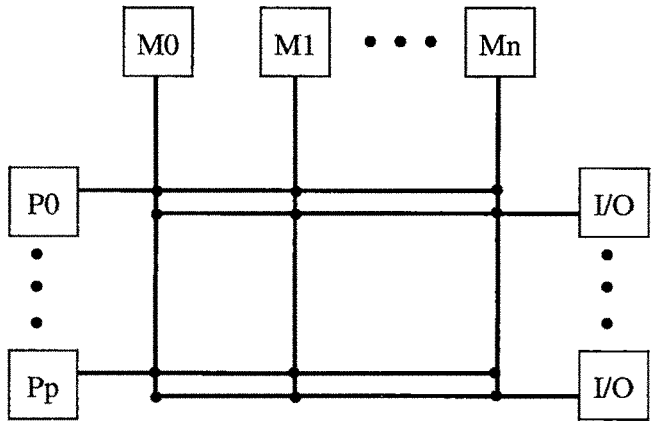

FIG. 3.2 CROSS - BAR SWITCH

Hierarchical organization of processors: Although all memory locations of any of the memory modules are accessible from any processors, accessing some subset of the address space is cheaper if the memory devices on which the addresses are mapped sit on the same bus as the processor. The compiler must generate loadable code such that the data required by certain task is accessible using only the local bus as far as possible, by the processor that executes the code for that task. Hierarchical bus architecture as shown in Fig. 3.3 is utilized properly through software, gives the best price to performance ratio.

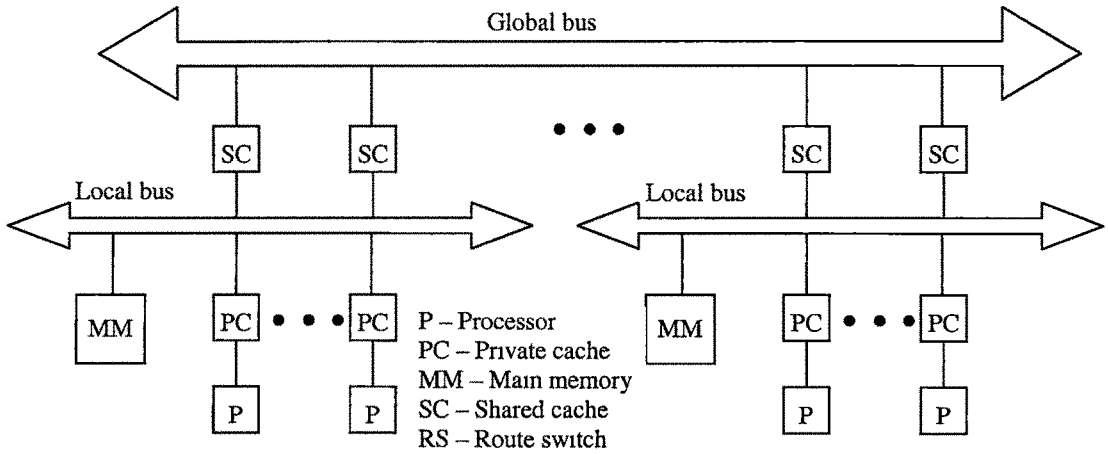

FIG. 3.3 HIERARCHICAL BUS

To reduce bus connection and average memory access time, all the processors in multiprocessor machine normally have a local cache memory. The presence of private caches necessarily introduces problem of cache coherence, which may result in data inconsistency. That is, several copies of the same data may exist in different caches at any given time. This is a potential problem especially in asynchronous parallel algorithms, which do not possess explicit synchronization stages of the computations. The possibilities of having several processors using different copies of the same data must be avoided if the system has to perform correctly. Hence data consistency must be enforced in the caches.

3.2.3 Parallel Programming Models

A programming model is a collection of program abstractions providing the programmer a simplified and transparent view of computer hardware and software. Parallel programming models are specifically designed for multiprocessors or vector / SIMD computers. Five such models are described

here. Parallel program is collection of processes or tasks. The models described here differ in the way these processes share data, achieve synchronization and communicate.

(i) Shared Memory Model

Multiprocessor programming is based on the use of shared variables in commonly accessible memory for communication and sharing data. Besides sharing variables in a common address space, communication also takes place through software signals and interrupts. Since multiple processors may attempt to access the shared data as shown in Fig.3.4, data memory management or locking is required to ensure the integrity of data and handle conflicts.

This model is also available in the form of multithreading libraries. Various techniques are available for scheduling of processes across the subtasks to be carried out to solve the given instance of a problem. In such models it is the programmer's responsibility to decompose the problem effectively and employ either static or dynamic scheduling of processes for achieving maximum parallelism.

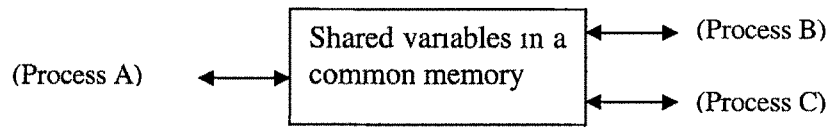

FIG. 3.4 SHARED MEMORY MODEL

(ii) Message Passing Model

Multicomputers employ message passing as the mechanism for inter process communication. Two processes residing at different nodes communicate with each other by passing messages over communication channel. The message may be instruction, data, and synchronization or interrupt signals. The communication delay in message delivery is much longer than in case of a shared memory model.

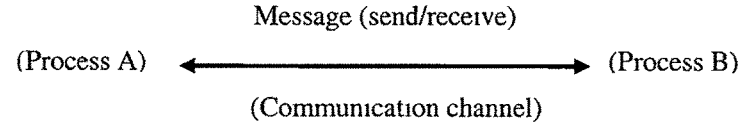

FIG. 3.5 MESSAGE PASSING MODEL

Two types of message passing models can be implemented: Synchronous and Asynchronous. Synchronous message passing requires both sender and the receiver to be synchronized in time for transfer of message. No buffering of message is done by the communication channel. The receiver is always blocked waiting for the message to arrive, the sender also remains blocked till the receiver receives and acknowledges the message. On the other hand asynchronous message passing does not impose blocking on the sender. The message gets buffered by the communication channel and is delivered to the target processes when they choose to look for the message. Message passing model does not require mechanism for mutual exclusion for access to shared data because there is no way for processors to share each other's address space for data exchange. All data sharing is done through message passing.

(iii) Data Parallel Model

It is appropriate for SIMD machines because the data is distributed to the processors and each executes the same set of instructions. The programming model for data parallel SIMD processors is an extension of the sequential programming. For example Fortran90 is specifically tailored for data parallelism. Data parallel programs require the use of pre-distributed data sets. Thus choice of parallel data structure plays a significant role in data parallel programming.

(iv) Object Oriented Model

Mapping of execution units to object is naturally achieved in this model. Objects are dynamically created and manipulated. Processing is performed by sending and receiving messages among the objects. Concurrent programming models are built up from low level objects such as processors, threads, queues into high level objects like monitors and program modules.

(v) Functional and Logic Model

A functional programming language emphasizes the functionality of a program and should not produce any side effect during execution. There is no concept of storage, assignment and branching in functional programming. The lack of side effects opens up much more opportunity for parallelism. The evaluation of a function produces the same value regardless of the order in which its arguments are evaluated. Thus arguments in dynamically created structures of a functional program can be evaluated in parallel. Logical programming is based on predicate

logic and is suitable for knowledge processing dealing with large database. This model adopts an implicit search strategy and supports parallelism in the logic interface process. A question is answered if matching facts are found in the database. Two facts match if their predicates and associated arguments are the same. The process of matching and unification can be parallelised under certain conditions.

Both functional and logical programming models are used in artificial intelligence applications where parallel processing is very much in demand.

3.2.4 Operating System

Generally operating system manages the resources, but in multiprocessors it has to manage scheduling of resources across the competing tasks of the machine. The additional requirements of a multiprocessor operating system are as follows:

Load Balancing: The operating system must utilize the resources efficiently by achieving uniform balance of loads across the processors. The operating system should schedule the subtasks such that there are no idle resources including the processors.

Scheduling Cooperating Processes: Parallel programs, which consist of concurrently executing tasks, must be scheduled such that collectively they are able to use the resources in the machine required to solve the given problem. Scheduling of processes can be done such that the processes belonging to a single parallel program are scheduled for execution together.

Graceful Degradation in case of Failure of one of the Resources: To have fault tolerant system, failure of one of its resources should not result in a catastrophic system crash. The operating system should be able to reschedule the task running on failed resource and continue the parallel program.

Communication Schemes: To achieve effective cooperation among the subtasks of a parallel program, operating system must provide adequate facilities for communication between the tasks, to share data and intermediate results during the processing towards the solution of the problem.

Synchronization Mechanisms: Synchronization between tasks is required to ensure integrity of the shared data across the subtasks of a parallel program. The tasks may need to wait till some state is reached across all the tasks of parallel program. The operating systems need to provide signaling mechanism for such synchronization requirements.

Generally centralized operating system is used for shared memory multiprocessors and distributed operating system is used for distributed memory multiprocessors. Basically there are following three concepts in design of operating systems for multiprocessors:

- Master – slave configuration,
- Separate supervisor configuration, and
- Floating supervisor configuration.

In the master-slave mode, one processor, called as master, maintains the status of all the other processors in the system and distributes the work to all the slave processors. The operating system runs only on the master processor and all other processors are treated as schedulable resources. The separate supervisor system is running in each processors and the approach is similar to that in computer networks. Each processor contains a copy of or access to shared kernel and resource sharing occurs via shared memory blocks. Each processor services its own needs. The floating supervisor control treats all the processors as well as other resources symmetrically. This type of system can attain better load balancing over all types of resources and the most flexible but is the difficult mode of operation to implement. Generally the combination of above schemes is used for obtaining useful solution.

3.2.5 Software Tools

Many software tools are available for development of parallel programs for realistic applications. These tools are:

Parallelising Compilers: It compiles the sequential program and generate an object code that will optimally utilize the underlying computer architecture. Techniques like dependency analysis are used to detect those segments of a code which can be executed in paralleland then the code is generated to exploit

the parallelism in the architecture of the machine. The programmer is freed of the responsibility of detecting and analyzing the parallelism in the solution to the given problem and then coding accordingly.

UNIX IPC: The inter process communication (IPC) facilities in UNIX operating system lets the programmer develop programs which, when executed run as a collection of cooperating processes. The IPC facility provides a set of machines to share data, control access to shared data, synchronize process and dynamically create and destroy processes. The process can share certain block of memory for sharing and exchanging data while cooperating. Synchronization is achieved by using signaling mechanism supported by operating system.

Thread Model: Conceptually, multithreading is similar to multiprocess programming using IPC. The difference is that a multithreaded program has a single process that manages multiple threads of control executing asynchronously. The thread library provides functional calls to create threads, control threads, terminate threads, control access to shared data through locking mechanism, generate events and wait for events. The practical advantage of using threads is that threads are lightweight process. In multiprocess model multiple memory images of the program exist in the core while in multithreaded program this overhead is avoided by sharing the same core image of the process. Also context information for each thread is also maintained.

PVM and MPI: PVM and MPI are two competing and functionally equivalent tools for parallel programming using the message passing model. Both the libraries have been developed by academic institutions and are being used widely by the scientific community for improving efficiency of computationally intensive programs. They are discussed in detail later.

DEC-RPC Threads: Distributed computing environment (DCE) is a standard proposed by the Open Software Foundation (OSF) for design and implementation of distributed systems. The standard provides a specification of tools and services that implementor of DCE must provide for the development of distributed computing system. The major components of DCE are RPC (Remote Procedure Call). Time/directory/file services, access authorization features, threads package and interface definition facility. These components provide a

formwork for organizing distributed resources into a client server oriented network. RPC provides a procedural programming abstraction for design and implementation of distributed programs. The services available on geographically distributed machines can be accessed by calling a remote procedure. To bring this interface, the DCE framework requires daemons running on the distributed machines for routing and forwarding data. DEC provides a thread package for building servers having multiple threads of execution, serving multiple clients. The synchronization and concurrency issues involved in using threads interface must be handled by the application developer.

CORBA - Common Object Request Broker Architecture: CORBA is a set of standard mechanism for naming, locating, and defining objects in a distributed computing environment. The formwork provides an object-oriented abstraction for the design and implementation of a distributed computing system. The entire system consists of client server objects, which communicate through the object request broker (ORB). ORB is a set of distributed processes running on different machines and coordinating the communication between the objects.

3.3 LOOSELY COUPLED MULTI COMPUTER SYSTEMS

The shared memory multiprocessors or tightly coupled multiprocessing system is a specifically designed computer system. An alternative form of multiprocessor can be created by connecting complete computers through an interconnection network [2]. Each computer consists of a processor and a local memory that is not accessible by other processors. In a multicomputer, the memory is distributed among the computers and each computer has its own address space. A processor can only access a location in its own address space. The interconnection network is provided for processors to send messages to other processors. These messages can include data that other processor may require for their computation. Such multiprocessor systems are usually called message passing multiprocessors or simply multicomputers especially if they consists of self-contained computers that could operate independently. The messages in a message-passing multicomputer carry data from one processor to other as dictated by the computer.

Message passing paradigm is not as attractive for a programmer as the shared memory paradigm. It usually requires the programmer to provide explicit message passing calls in their code, which is very error prone and has been compared to low-level assembly language programming. Data cannot be shared, it must be copied. This may be problematic in applications that require multiple operations across large amount of data.

The message passing paradigm has the advantage that special mechanism are not necessary for controlling simultaneous access to data, which may significantly increase the execution time of a parallel program. Many custom designed shared memory multiprocessor systems have a very short life because of unending progression towards faster and faster single processors. It is better to use a new single processor than an old multiprocessor if new processor operates m times faster than each of m older processors. Using interconnected computers allows newer computers to be more easily incorporated into the system. The message-passing multicomputer will physically scale easier than a shared memory multiprocessors, that is it can more easily made larger.

Issues related to distributed computation over multicomputer are [10]:

- Identification and control of the separate subtasks in order to facilitate local and global management,
- Effective communication among the separate subtasks,
- Synchronization of subtask activities to ensure that the correct results are generated.

3.3.1 Operating System

In computing environment where many computers are linked, the operating system should provide the user with the facilities for distribution in a friendly manner. A desirable objective is to create an interface with the system which allows the user to feel that his / her interactions are with single computer.

The vehicle for user interaction with the computer system is the command processor or window manager. It may be helpful if the user does not have to specify in the command the separate processors that will be involved in handling the user's task.

In computer networks, each host computer runs its own operating system. This operating system can be:

(a) Distributed operating system in which each computer can be cooperating similar local edition of one large homogeneous operating system. Such systems are more likely to be found in networks, which use compatible machinery.

(b) Network operating system, in which each computer have distinct heterogeneous operating systems which cooperate through network-access agent processes. Such system can easily accommodate the differences in machine design prevalent in wide area network.

The creation and management of names are key issues in the design of operating systems. Names must be given to the users, to the processes, which they generate on the computer, and to the resources, which they access. The presence of the network increases the volume of names, and presents a distributed context in which the names can emerge and in which these names must be managed.

Another major issue in operating system design that is particularly relevant to distributed computing is the implementation of a communication mechanism for process-to-process interaction. The system should be easy extension from purely local interaction to interaction with remote sites, which led development of interprocessor communication schemes. A communication system can also provide synchronization of activity among processes.

The control access to shared resources is also of major importance. The linkage of a facility to a communication network exposes the facility to a large user base and there is probability of existence of malicious intruder.

The operating system must manage efficiently the many processing units and other resources available. The parallel strands within the program have to be identified and allocated to available units and their interaction ought to be adequately controlled.

3.3.2 Client-Server Model

One of the advantages for linking computers in network is the opportunity to share the services located at single site to widely available. In many cases the cost of expensive resources like large software package, laser printer can be justified by sharing. The client-server model of distributed computing identifies the service and provides a software system to administer that services. This software system includes server, which produces the service and runs on a dedicated computer and client, which consume the service and run on user machine.

Due to inherent modularity in client-server architecture, it provides an effective strategy for making computer resources very widely available. New servers and clients can be added on demand to allow smooth incremental development of information systems. The innumerable services can be offered via the client-server technology. File server system for shared file storage system is the most popular of such system due to considerable significance of file storage and retrieval facilities.

To ensure only authorized use and to increase the availability of resource security consideration arises. Individual request for the service must be identifiable. The server must be able to determine about handling of a particular request, and client should be able to repeat a request if earlier one was not received.

The server should have facility to regulate flow of traffic to handle heavy demands by clients. To improve throughput times, concurrent access should be allowed. To ensure that data area, when changed, preserve the integrity of information stored some mechanism must be employed.

3.3.3 Communication and Computer Network

In order to communicate a signal, three elements are required: a transmitter to send, medium to convey and receiver to receive. The digital signaling of computer uses binary coded forms and the representation of a sting of binary digit in computer communication approximates a square wave. In order to transmit such a signal two values of voltage are used, zero and one. In many

cases installing a new network to transmit digital signals over a long distance is expensive and so existing telephone lines are used. As telephone lines are used to carry analog signals, suitable modulation techniques are used to convert digital signal to analog signal and vise versa. The device used for the same is called modem (modulator / demodulator). Some of the mediums used in communication are twisted pair, coaxial cable, optical fibers, radio frequencies and communication satellites.

A computer network is built upon a communication base using layered architecture [10] in order to facilitate the design, construction and maintenance. It is a seven layer model called the reference model of open system interconnection (OSI). These seven layers range from the hardware dependent layer at the bottom of the hierarchy to the user level applications environment at the top. These layers beginning at the lowest layer are as:

1. The Physical layer which is responsible for the transmission of the raw bits from one host to other over communication channel. It is necessary to choose suitable signaling technique, transmission medium and related equipment to provide an acceptable communication channel. Other aspects to be considered are multiplexing to provide better utilization of available bandwidth, making choice between circuit-switching and packet-switching techniques, the handling of terminals and the management of errors.

2. The Data link layer which transforms the raw bit stream into a string of bits which is free of transmission errors. Link protocols are concerned with efficient and reliable transmission of information from one node (host or IMP) in the network to a neighboring node. In order for a host to communicate with another, the message will have to pass through the subnet, stopping several IMPs along the way. Therefore each node should shoulder the responsibility of error-free transmission to neighboring nodes. This layer addresses link-level responsibility which involves the IMP-to-IMP link as well as host-to-IMP link.

3. The Network layer, which handles the routing within the subnet and determines the interface between the host and IMP (Interface Message Processor). At this layer the host receives a packet of information from the

layer above it, adds its protocol header to the packet and uses data link layer to transmit it to the IMP. At that IMP, network layer selects a route to another IMP and then uses the data link layer to handle link transmission. This process is repeated until the packet is delivered to the destination host. The protocol header information will depend on type of service, i.e. virtual call to support connection oriented services or datagram which supports a connectionless service.

4. The Transport layer, which is responsible for the safe transfer of messages from one application process at a host to another. The transport layer is designed to support interprocess communication and the protocols in this layer are implemented only at host and not in the IMPs.

5. The Session layer, in which connection is initiated for a communication exercise. The user interacts with the network to obtain services which are made available via sessions which have identifiable begin and end points and must be carefully managed.

6. The Presentation layer, which resolves differences in formats among the various hosts. The major concerns are to ensure that information remains semantically sound when transmitted to a remote host and to provide an acceptable level of security and privacy to user of the network. It handles the form in which data are presented to the network by the users and delivered to the user by the network.

7. The Application layer where the functions or applications that user can run are created. The network provides several services to the user designed to meet the users' application needs, which may vary with users. A few relatively popular applications include electronic mail, electronic fund transfer, file and job transfer systems, remote job entry, public bulletin board, advertising system etc.

The topology of the network involves the location of IMPs, hosts and terminals and the existence of physical links between these devices. Two widely adopted topologies are Wide Area Network (WAN) and Local Area Network (LAN).

(A)Wide Area Network: The aim of WAN is to provide services to users over a wide geographical area and major concerns are location of IMPs (nodes) and lines so as to provide reliable services. The extent for demand for network services will differ with hosts and therefore volume of information traffic generated in the segment of the network will vary. Some hosts and terminals can be connected to concentrators rather than directly to IMP. A concentrator accepts input from several lines and output information onto a single line and in addition it can feed several lines off the input from that single line. The topology usually involves a hierarchical approach with a backbone system of IMPs and large capacity lines to which are connected clusters of concentrators serving the hosts or terminals. One IMP can therefore serve a relatively large user population, which will then be divided, into smaller areas each served by concentrators. To provide acceptable level of reliability some redundancy must be built into the network.

(B) Local Area Network: LAN occupy a limited physical range, usually an office building, manufacturing plant, university campus and similar single organizations. The computing elements are connected directly to each other via the communication link. The location of computing units and their interconnection depends on needs of various departments of the organization. A few patterns have emerged offering different levels of performance and reliability with the tradeoffs being simplicity and inexpensiveness. The degree of reliability can be achieved by upgrading the quality of the physical medium and employing an efficient method for sharing the use of the medium. Some of the commonly used patterns are Bus, Ring, Star, Tree, and Mesh.

In order to extend the opportunities for information exchange, computer networks can be interconnected. To connect two networks, a computer is dedicated to the handling of the interface between the two networks, which is known as Gateway. The function of gateway is to perform protocol conversion i.e. packets addressed to the other network go to the gateway which converts them from protocol of sender's net to that of the receiver's. After the conversion the packet will be relayed.

The Internet uses the transport layer protocol, TCP (Transmission Control Protocol) and the network layer protocol and IP (Internet Protocol) to link

together a vast number of local area and wide area networks. TCP is connection-oriented protocol and IP is a connectionless protocol to transport layer. TCP/IP is widely used in many systems.

3.3.4 Software Tools

PVM and MPI are two competing and functionally equivalent tools for parallel programming using message passing.

3.3.4.1 PVM (Parallel Virtual Machine)

PVM is a portable message passing programming system designed to link separate Unix host machines to create a virtual machine which is a single computing resource [96]. The PVM system is composed of two parts. The first is a daemon program, which runs on all machines that are part of the virtual machine. This permits the user to run a PVM application at a Unix prompt from any of the machines. The second part of the system is a library of PVM interface routines. This library contains user callable routines for passing messages, spawning processes, coordinating tasks and modifying the virtual machine. Application programs can be written in a mixture of C, C++ and FORTRAN but must be linked with the PVM library.

Terminologies related to PVM

Host: A networked computer that can participate in the formation of a parallel multi computer is referred to as a host. A host has attributes such as name, architecture type and a relative speed rating.

Virtual machine: A meta machine formed with heterogeneous collection of networked computers. A collection of programs running on these machines cooperates to realize a virtual machine. This machine can be accessed from any of the host through a console program that gives an interface to the virtual machine. The virtual machine is configured by user through software and no hardware reconfiguration is required.

Task: the process running on virtual machine is called task.

TID: Each task has a unique task-id (TID) per virtual machine. TID is used to refer to the task for sending messages, synchronizations and control.

Pvmd (PVM daemon): This process runs on every host that forms a part of the PVM configuration. These daemons collectively support message routing across task in virtual machine. Tasks running on a host communicate with the other tasks via pvmd running on that host. This pvmd process keeps communicating with the other pvmds to monitor the status of other hosts.

Message: An ordered list of data elements sent between tasks. A message can be composed using elements of various data types. The sender composed a message before sending it and the receiver must decompose it exactly in the same order as it was composed.

Group: An ordered list of tasks is assigned a symbolic name. Each task has a unique index in a group. Any task may be in zero or more groups.

Architecture of PVM

A virtual machine across a networked collection of computers is realized through a layer of PVM daemons, running on each host. These daemons provide certain functionality such as addressing hosts, addressing of tasks, mapping of tasks to hosts, routing of messages, scheduling tasks on hosts for execution etc. Fig 3.6 shows the architecture of a PVM.

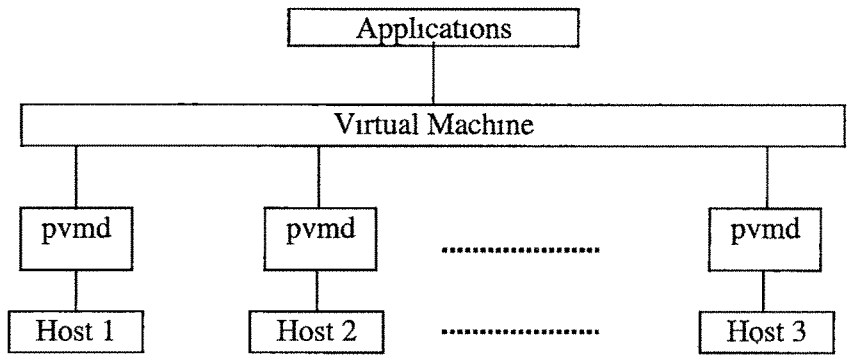

FIG. 3.6 PARALLEL VIRTUAL MACHINE ARCHITECTURE

3.3.4.2 MPI (Message Passing Library)

In past, as different hardware manufacturers developed their own distributed memory parallel computers, they also developed and implemented their own message passing libraries. So, the programmer was tied to particular computer

architecture. This lack of portability lead to a number of public domain message passing libraries being developed, each with their own software implementation. The natural progression from this therefore was the definition of a standard message passing library.

MPI (Message Passing Interface) provides a standard set of definitions, which allow parallel programs to be written under distributed memory paradigm [97, 98]. These definitions describe a library of over 100 C and FORTRAN subprograms, which are now supported by almost all parallel computer manufacturers. In addition, numerous commercial and public domain implementations of MPI exist, which allow clusters of workstations to be used as a single parallel computing system. The main purpose behind writing such specifications is to develop some software, for the first time since the advent of parallel computers, which is truly portable between different parallel architectures.

When working with a distributed memory computer, it is necessary to ensure that each processor has its own copy of each data item that is required for each computation, performed by that processor. Often, some or all of this data will depend upon the result of a previous computation, which may have been made on one of the other processors. Therefore, some mechanism is required for the transfer of copies of data between processors. This is achieved through the message passing whereby each processor is given the ability to send and receive copies of data to and from other processors. This requires the cooperation of each processor that is involved in communication, whether as a sender or receiver or both. This system provides an API (Application Programming Interface), which permits a standard to develop parallel programs with standard message passing. A carrier such as PVM is required for distinct implementations if the operating system does not support the message passing.

3.4 PERFORMANCE MEASUREMENT OF PARALLEL PROCESSING

The objective of parallel or distributed processing must be to execute a problem at greater speed and to solve large problems with greater or more dedicated idealization. The performance measurement of parallel processing and other factors affecting its performance are discussed in following sections.

3.4.1 Granularity

To achieve improvement in speedup through the use of parallelism, it is necessary to divide the computation into tasks or processes that can be executed simultaneously. The size of a process can be described by its granularity. In coarse granularity, each process contains a large number of sequential instructions and takes a substantial time to execute. In fine granularity, a process might consist of a few or perhaps single instruction. The granularity must be increased to reduce the cost of process creation and inter process communication. For message passing, it is desirable to reduce the communication overhead because of the significant time taken by inter computer communication. This is especially true for the network of workstations.

Granularity = Computation time / Communication time

It is very important to maximize the computation / communication ratio while maintaining sufficient parallelism.

3.4.2 Speedup

A measure of relative performance between a multiprocessor system and a single processor system is the speed up and is defined as:

$$S = T_{seq} / T_N$$

where T_{seq} is the time taken by the code to execute on a single processor and T_N is the time taken for the same code to execute on a parallel system having N processors.

Theoretically a problem should run N times faster on a network containing N processors than on a single processor. In reality, the speed up achieved are less than the theoretical values owing to the increase in the communication time with the increase in the number of processors. In addition any load imbalance in the problem will result in less than optimum performance.

3.4.3 Efficiency

The efficiency of the parallel system for a given algorithm is defined as:

$$E = S / N$$

where S is the speed up and N is the number of processors.

Efficiency is given as percentage. Efficiency gives the fraction of the time that the processors are being used on the computation. If E = 50%, the processors are being used half the time on the actual computation, on average. The maximum efficiency of 100% occurs when all the processors are being used on the computation at all times and the speed up factor would be N.

3.4.4 Overhead

There are several factors that will appear as overhead in the parallel version and limit the speed up. These factors are:

1. Periods when not all the processors can be performing useful work and are simply idle (load imbalance).
2. Extra computations in the parallel version not appearing in the sequential version
3. Communication time for sending message

3.4.5 Cost

The processor-time product or cost of computation can be defined as,

Cost = Execution time × Total number of processors used

The cost of a sequential computation is simply its execution time T_{seq}. The cost of parallel computation is $T_N \times N$. A cost optimal parallel algorithm is one in which the cost to solve a problem on a multiprocessor is proportional to the cost on a single processor system.

3.4.6 Scalability

It is used to indicate a hardware design that allows the system to the increased in size and in doing so to obtain increased performance. This could be described as architecture or hardware scalability. Scalability is also used to indicate that a parallel algorithm can accommodate increased data items with a low and

bounded increase in computational steps. This could be described as algorithmic scalability. Combined architectural / algorithmic scalability suggests that increased problem size can be accommodated with increased system size for a particular architecture and algorithm.

3.5 CHOICE OF DISTRIBUTED COMPUTING ENVIRONMENT

Developments in microelectronics and networking increased popularity of multi computer system for distributed processing which is cost effective and attractive alternative for high performance computing. To utilize network of computer as distributed computing environment message passing libraries like PVM and MPI can be used. However its use requires background of operating system, networking for communication of data, various constructs of message passing libraries and concept of parallel processing. These message passing libraries are available freely for UNIX or LINUX operating system but for WINDOWS operating system commercial products are available. The user familiar with WINDOWS operating system may find it difficult to use UNIX or LINUX. In such situation Client-Server approach is very useful which can work on WINDOWS operating system also. WebDedip is one such tool developed using JAVA which relieves application developer from using message passing functions in program. The overview of WebDedip environment and application development is given in detail in Chapter 4.