Supercomputers Today

Parallelism is the Name of the Game

V Rajaraman

V Rajaraman is with the Jawaharlal Nehru Centre for Advanced Scientific Research and the Indian Institute of Science, Bangalore. Several generations of scientists and engineers in India have learnt computer science using his lucidly written text books on programming and computer fundamentals.

In this article we review the evolution of supercomputers from vector supercomputers to massively parallel supercomputers and network of workstations. The thirst for higher speed is insatiable and petaflops computers are expected in the next decade.

Introduction

The word supercomputer was coined in the late 70s to describe a machine which was built by Seymour Cray while he was a designer with the Control Data Corporation and designed the CDC 7600 which was at that time one order of magnitude faster than the fastest computers of the day. At that time the two major companies competing for the market for high performance numerical computing for solving applications in science and engineering were IBM with their model 7090, 7094 and Control Data Corporation (CDC) with their models 3600 and 6600. IBM's market share was large and they were more interested in building general purpose computers with broader applications domain. CDC which was a new company (cofounded by Cray) concentrated on building computers for the scientific computing market and took away many university customers from IBM. An asset CDC had was Seymour Cray who led a team of designers and built the CDC 7600, at that time the fastest computer. Seymour Cray's idea was to build a 'balanced system', that is, a system whose parts were well matched in speed and to exploit parallelism in application programs. It was realised by Cray that very high performance computing was a 'niche market' and required special design efforts. He set up his own company in 1992 to build Cray-1 the first supercomputer which was carefully crafted to solve the most time-consuming applications of the day. The unique idea used by Cray in designing Cray-1 was an

extensive use of temporal parallelism which led to the idea of vector processing. He also realised the importance of large fast main memory from which operands can be retrieved as fast as they can be processed by the CPU. Innovative design of memory systems was pioneered by Cray to achieve this. Besides large fast main memory, the need for fast secondary memory which can continuously supply data to the main memory was also felt and high speed disks were designed. Integrated circuits had not yet been invented in those days and discrete transistor logic was used. The clock frequency of 80MHz used in that machine was the highest clock speed of the day and transistors switching at that speed needed to be carefully cooled. Early machines used ice cold water circulating in copper tubes over which circuits were mounted. Innovative cooling techniques was one of the challenges of those days.

The unique idea used by Cray in designing Cray-1 was an extensive use of temporal parallelism which led to the idea of vector processing.

We have come a long way since those days. Supercomputers have gone through three generations. Today it is very difficult to precisely define the term 'supercomputers'. It is also not easy to predict the direction which will be taken by this technology. The only point on which everyone would agree is the relentless thirst for speed demanded by scientists and engineers to solve problems of interest to them. Numerical simulation has become the most important tool in doing science. Models are becoming more sophisticated and realistic; higher and higher resolution is being demanded and there is an insatiable demand for speed. It is predicted that by the year 2010 we will need petaflop computers, i.e. computers whose arithmetic speed is 10^{15} floating point operations per second in order to solve problems such as simulating the performance of a human heart (a 3D model with details of every muscle and blood vessel), detailed global weather prediction, compensating for atmospheric turbulence in telescopic images in real-time and many other problems of similar nature. All supercomputers today use parallelism to achieve their speed. Two types of parallelism are used; temporal and data. We will explain this in what follows.

It is predicted that by the year 2010 we will need petaflop computers.

Early supercomputers used a technique called vector pipelined processing to attain high speeds.

Vector Supercomputers

Early supercomputers used a technique called vector pipelined processing to attain high speeds. This technique exploits temporal parallelism inherent in the problem being solved.

Consider a procedure for adding two floating point numbers x and y. A floating point number consists of two parts: a mantissa and an exponent. Let x be represented by the tuple (mant x, exp x) and y by the tuple (mant y, exp y) Let the result z be represented by the tuple (mant z, exp z). The job of adding x and y can be broken up into the following four tasks:

Task 1: Compute (exp x – exp y) = m
Task 2: If $m > 0$ shift mant y, m positions right and fill the leading bits of mant y with zeros. Set exp z = exp x
If $m < 0$ shift mant x, m positions right and fill the leading bits of mant x with zeros. Set exp z = exp y
If $m = 0$ do nothing. Set exp z = exp x.
Task 3: Add mant x and mant y. Let (mant x + mant y) = mant z
Task 4: If mant $z > 1$ then shift mant z right by 1 bit and add 1 to exp z. If one or more most significant bits of mant $z = 0$ shift mant z left until leading bit of mant z is non zero.
Let the number of shifts be p. Subtract p from exp z.

An electronic adder can be designed with 4 stages, each stage doing one of the tasks explained above. Such an adder is called a *pipelined adder* and is shown in the block diagram of *Figure 1*. To use this adder a sequence of operand pairs to be added (a_1, b_1), (a_2, b_2), (a_3, b_3) ... (a_n, b_n) are fed to the adder (See *Figure 2*) and are shifted into the pipeline one pair at a time. Let T seconds be the time taken by each stage. One pair of operands is shifted into the pipeline from an input register every T seconds.

An electronic adder can be designed with 4 stages, each stage doing one of the tasks. Such an adder is called a *pipelined adder*

The shifting is controlled by a series of pulses called a *clock*. The time elapsed between successive pulses is the clock period which in this example is T seconds (See *Figure 2*). The sum c_1=

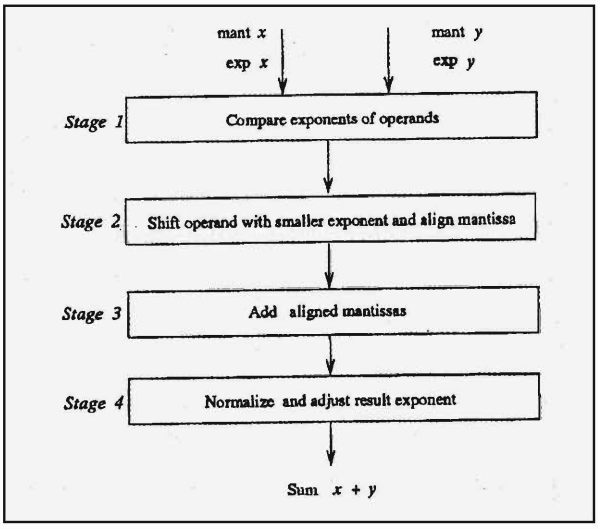

Figure 1. A pipelined adder unit with 4 stages.

$a_1 + b_1$ comes out of the pipeline after 4 clock periods which is $4T$ seconds. However, sums $c_2 = (a_2 + b_2)$, $c_3 = (a_3 + b_3)$ ... $c_n = (a_n + b_n)$ come out at time $5T, 6T$ ... $(4 + n-1)T$ seconds.

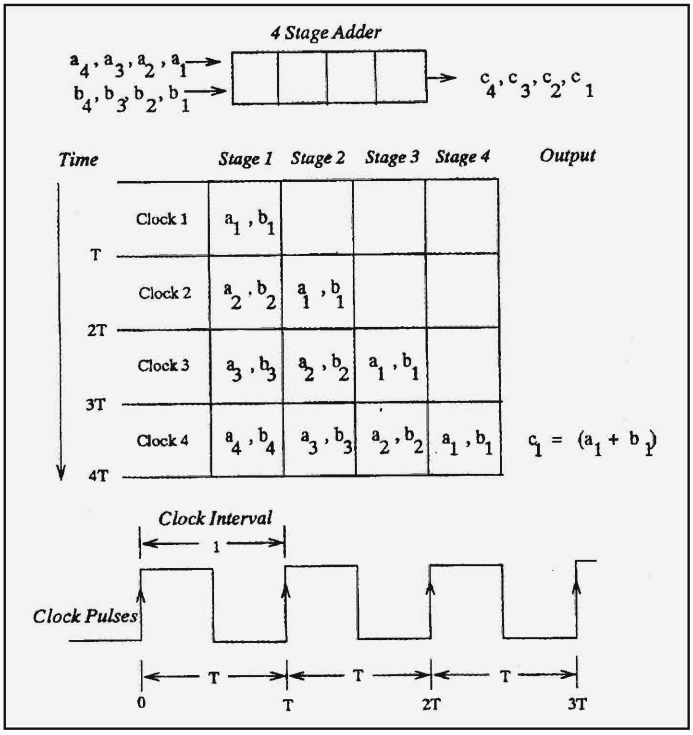

Figure 2. Adding vectors in a pipelined adder.

All vector supercomputers extensively use pipelined processing of vectors to attain high speeds.

An ordered sequence of operands $(a_1, a_2, a_3, ..., a_n)$ is known as a *vector*. In the above example we add vectors $(a_1, a_2, a_3, ..., a_n)$ and $(b_1, b_2, b_3, ..., b_n)$ to obtain the sum vector $(c_1, c_2, c_3, ..., c_n)$. In order to obtain the potential speedup of a pipelined arithmetic unit, it is necessary to add vectors of length n much larger than 4, where 4 is the number of stages in the pipeline.

All vector supercomputers extensively use pipelined processing of vectors to attain high speeds. A typical vector pipeline unit of supercomputers manufactured by Cray Research Inc., USA, is shown in *Figure 3*. Here pipelined units consisting of a number of stages work synchronously for arithmetic computations. The computer has a centralized clock which generates a pulse once every T seconds (T is around 10^{-9}sec). In a pipelined addition unit one addition is carried out every T seconds by the system (after an initial time required to fill the pipeline). In a model known as Cray T90 series system the value of T is 2 nano seconds (nano$=10^{-9}$). Thus one floating point arithmetic operation is carried out in 2 nanoseconds giving a peak speed of $(1/2 \times 10^{-9})$flops which is 500 megaflops. There are several pipeline arithmetic units in a vector computer giving aggregate speed of a few gigaflops.

Technology of Vector Supercomputers

Vector supercomputers are very expensive costing millions of

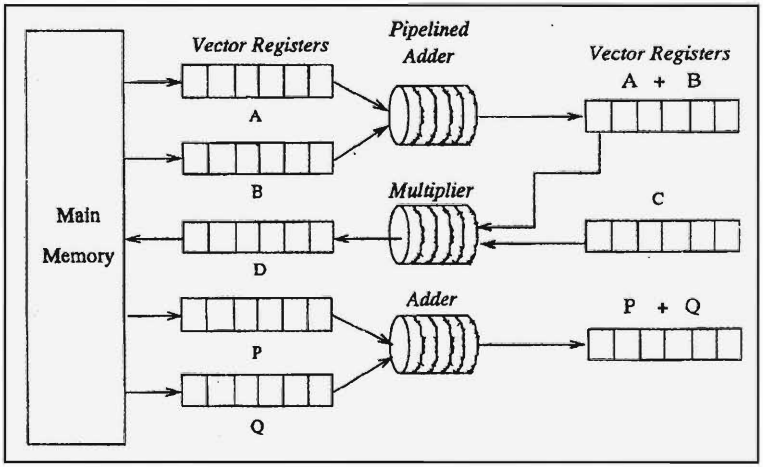

Figure 3. Pipelines in vector supercomputer.

dollars and are manufactured only by four companies in the world today – SGI which took over Cray in USA, Fujitsu, Hitachi and NEC in Japan. One of the main reasons for their high cost is the difficulty in fabricating electronic circuits to the exacting standards and high speeds demanded by vector supercomputers.

Vector machines will be built and used till important application programs are effectively ported to other cheaper supercomputers.

They are built as there is still a market for them. Organizations which do complex simulation such as weather modelling and nuclear explosion simulation use vector supercomputers as these computers deliver high effective speed in the gigaflop range on efficiently written vectorized code. Such sustained speed is not delivered by recently developed parallel supercomputers. Thus vector machines will be built and used till important application programs are effectively ported to other cheaper supercomputers.

Shared Memory Multiprocessors

In this type of computers around 8 to 32 processors are connected to a main memory using a communication network (see *Figure 4*). This memory can be accessed by any processor. Thus the address space of the memory is common to all processors. This is called a *uniform address space* and the time to access a location in memory is the same for all processors. This type of parallel computer is also known in the literature as *symmetric multiprocessor system* (SMP System).

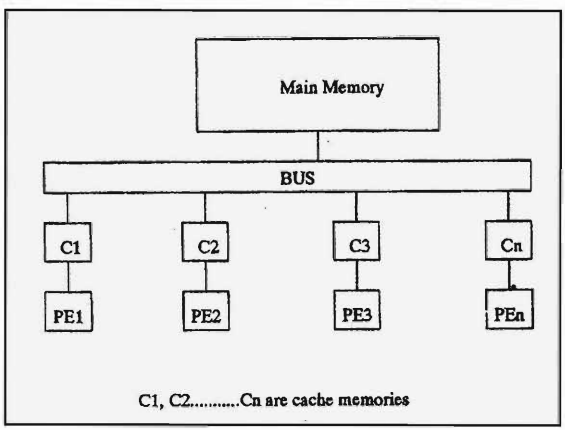

Figure 4. A shared memory multiprocessor on a bus (also known as symmetric multiprocessor).

It has been found experimentally that references to instructions and data tend to be sequential and local.

The speed of accessing data stored in the shared main memory is several times slower than the processing speed of the CPU. This is due to the different semiconductor technologies used in CPU and main memory. It is expensive to make a large main memory using the same technology used to fabricate CPU. In order to alleviate the speed mismatch between CPU and main memory all computers have a smaller fast memory called a *cache memory* placed between the main memory and the CPU (see *Figure 4*). Many computers have two caches – one which stores instructions to be executed called the *instruction cache* and the other which stores data called the *data cache*. The size of these caches is around 512 Kbytes as compared to main memory size of 512Mbytes. The computer places the instructions currently being executed and the data currently being processed respectively in the instruction cache and the data cache. It has been found experimentally that references to instructions and data tend to be sequential and local. Thus if a sequence of instructions and data are placed in cache they can be retrieved at high speed and executed. As soon as it is completed, the next sequence of instructions can be copied into the cache. A similar procedure is adopted for data. Data required currently is placed in the data cache and CPU fetches the requisite data from the cache and processes it. This processed data is then written back into the main memory and the next chunk of data is brought to the cache. Note that the data in the cache may be modified by a program and thus it should be written back in the main memory.

In order to maintain cache coherence, whenever a processor modifies a data in its own cache it should broadcast this to all the other processors.

In a shared memory multiprocessor each processor has its own cache memory. Thus each processor can retrieve instructions from its own cache and also write the results in its own cache. One major problem is encountered as the main memory is globally addressed and shared by all processors. If any processor changes the value of a variable in its cache it should also be changed in all the other caches which may happen to have this variable. In other words, data in a specified address must be identical in all the caches. This is called *cache coherence*. In order to maintain cache coherence, whenever a processor modifies a

data in its own cache it should broadcast this to all the other processors. Whichever processor happens to have a copy of this data in its cache should immediately update it. The value of the data in the shared main memory should also be updated. This operation can slow down the entire computer particularly if the number of processors in a shared memory system is large.

It has been found in practice that placing more than 16 processors on a single shared bus will seriously degrade the performance of the multiprocessor.

There are two types of communication systems used to connect processors to the main memory in a symmetric multiprocessor. One of them is a common shared bus. (A bus is a collection of wires interconnecting processors and memory. The number of wires equals the number of memory address bits, the number of bits per word of memory and several control signal wires). As the bus is a shared resource only one transaction can be carried out at a time using the bus. Thus if one processor is reading from main memory, other processors have to wait till this transaction is over before they can access memory. This severely limits the number of processors which can share a bus in a practical multiprocessor. It has been found in practice that placing more than 16 processors on a single shared bus will seriously degrade the performance of the multiprocessor. The major advantages of a shared bus are :

- It is easy to make a shared bus system as the bus is merely a set of parallel wires. It is reliable and inexpensive.

- Maintaining cache coherence is relatively simple in a shared bus system as it is easy to broadcast on the bus any change of data in a cache. All other caches holding this data can simultaneously update their respective caches. This is known as the *snoopy cache protocol.*

The other type of communication system used to connect processors to the main memory is called an *interconnection network* or *switch* (see *Figure 5*). The advantage of such networks is that they allow several processors to access memory simultaneously. Cache coherence is however, more difficult to maintain as

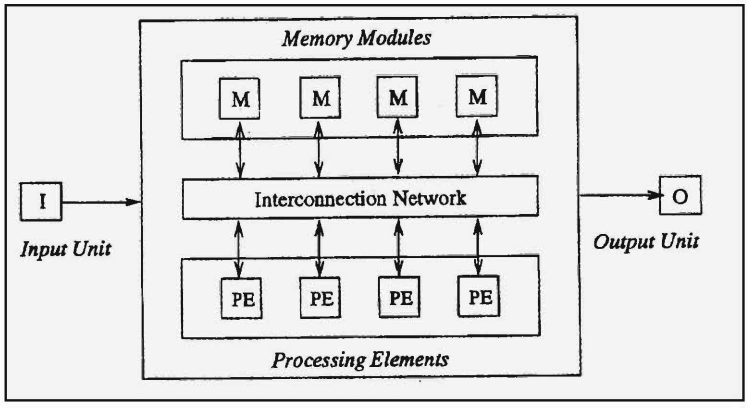

Figure 5. General structure of a shared memory multiprocessor using an interconnection network.

broadcasting in such a network is difficult. Most systems have a directory which has the status of shared variables stored in caches and they refer to this directory to maintain cache coherence. This slows down the overall system performance when the number of processors is increased. Thus such shared memory systems also limit the processors to around 16 to 32.

Ideally the speed of a n processor system should be n times the speed of a single processor system. A parallel computer architecture is said to be scalable if this ideal is attained. In practice it is attainable only when a problem being solved is naturally parallel and a single program multiple data (SPMD) algorithm can be formulated for the problem. In general it is not so and processors have to interact and exchange intermediate results. Shared memory multiprocessors are in general not scalable. As we already pointed out the value of n is limited to about 16 to 32 in practical systems.

The main advantage of shared memory multiprocessors is ease of programming as there is a single uniform address space and parallelising programs is relatively easy.

The main advantage of shared memory multiprocessors is ease of programming as there is a single uniform address space and parallelising programs is relatively easy. If the speed of individual processors is high, 16 processor system can give a supercomputer performance. This is the approach used by Cray in Cray YMP design in the 80s and by Cray T90 systems now. In these computers the individual computers are very powerful vector computers and the shared memory parallel computer architecture leads to much higher speeds.

Message Passing Parallel Computers

Another method of designing a parallel supercomputer is to interconnect a set of independent computers (called computing element – CE) using communication lines (see *Figure 6*). Each CE has its own main memory and cache. The CEs can be interconnected in many ways, for example, as a tree, a mesh, a ring etc. A popular interconnection method is called a *hypercube interconnection* and is shown in *Figure 6*. This parallel computer architecture is called a *message passing* multicomputer as individual CEs coordinate their work by interchanging messages. For a message passing multicomputer to work efficiently it is necessary to:

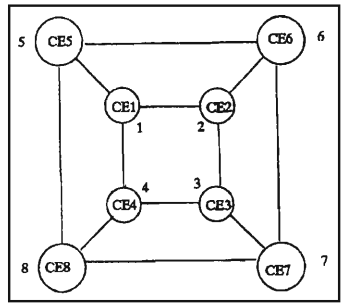

Figure 6. A 3D hypercube connection of CEs.

- Partition the program and schedule tasks to the CEs in such a way that the data required to carry out the tasks are in the local memory of the respective CEs. This will avoid accessing the memory of a remote CE through the communication network.
- Minimize the number of messages transmitted between CEs.
- Minimize the number of inter-CE links the messages have to traverse from source CE to destination CE.
- Minimize the time taken to communicate messages between 9200 CEs. This must be much less than that taken to compute.

Unlike a shared memory computer a message passing computer does not share a common global memory. It is thus scalable. In other words the number of CEs can be increased provided a proper task allocation is possible. Message passing machines with 9200 CEs have been built and used for solving problems.

Programming message passing multicomputers is more difficult than programming shared memory machines.

Programming message passing multicomputers is more difficult than programming shared memory machines. The major difficulty in programming is the allocation of tasks to CEs and managing message routing between CEs. As the time taken for messages to travel between CEs is normally much

Distributed shared memory parallel computers try to combine the advantages of both these architectures.

higher than the computing time, a program with a poor task allocation which requires many messages to be transmitted to distant CEs will be executed very slowly. Some automated tools to optimize task allocation to reduce execution time have been proposed. The problem is, however, difficult to solve as task graphs vary from one program to another, whereas the CE interconnection structure is fixed. Another approach is to have a dynamically reconfigurable interconnection between CEs. The reconfiguration is program controlled and is dependent on the task graph. (See also the article 'An Introduction to Parallel Computing' by Abhiram Ranade in the January 1998 issue of *Resonance* [6]).

Distributed Shared Memory Parallel Computers

We saw that shared memory multiprocessors are easy to program (due to their global address space) but not scalable. Message passing multicomputers, on the other hand, are scalable but difficult to program. Distributed shared memory parallel computers try to combine the advantages of both these architectures. In *Figure 7* we give the architecture of such a machine. Observe that a small number of processors are connected as a shared memory computer. A set of these shared memory systems are connected to a communication

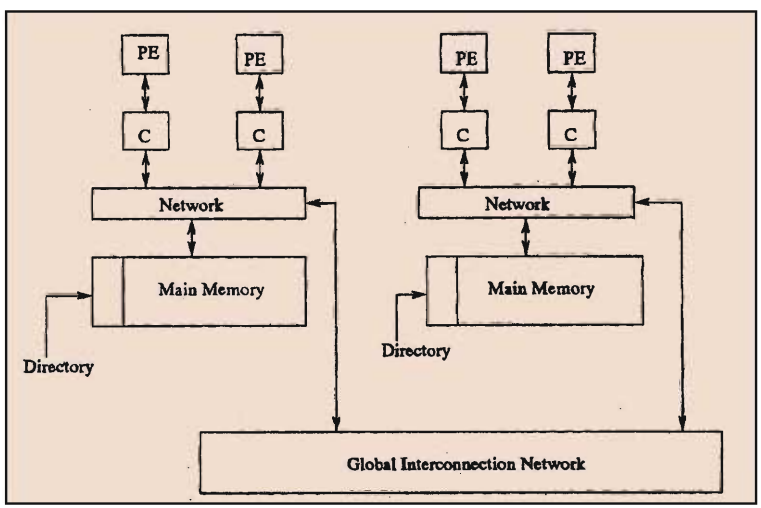

Figure 7. Distributed shared memory system.

system. From a programmer's point of view the system has one globally addressed memory. Thus one may program the system just as one would a shared memory computer. The programmer does not have to explicitly place data in a memory closely coupled to a processor which uses it. The operating system of the computer tries to minimize memory references to non-local memory. In such a machine, even though it has a shared globally addressed memory, the access time to memory is non-uniform as the physical memory is distributed. In the literature such a parallel computer is also known as *non-uniform memory access* (NUMA) computers. Some recent supercomputers built by Silicon Graphics use this design.

Cluster Architecture

Another type of parallel computer which is of the message passing type is to connect full fledged computers such as powerful workstations on a high speed local area network such as the one which uses the *asynchronous transfer mode* (ATM) protocol or the recently developed gigabit ethernet connection. Local networks using this protocol have a raw communication speed of 1 Giga bits/second and can thus support message passing between computers at high speed. Building the hardware of such a computer is easy as one can buy workstations off-the-shelf with network connectivity. Primarily the designer's job would be to develop a good software environment to allow easy use of computers in the cluster. Another advantage of such a system is that it allows use of individual workstations in a stand-alone mode and the group of computers connected to the network, as a parallel computer. Such a machine can approach supercomputer performance when several hundred workstations are connected to the network, particularly on single program multiple data type applications. This structure is also known as a *network of workstations* (NOW). In *Box 1* we list two of the fastest supercomputers in the world today which are both parallel computers. The Intel parallel supercomputer is rated at 1.34 teraflops.

Box 1. Fastest Computers in the World

The world's fastest supercomputer sites are maintained at the web address http://www.top500.org/. The June 98 list (the next list will be published in November 1998) lists Intel's 9200 Pentium Pro processors computer called ASCI Red as the fastest machine and Silicon Graphics Cray T3E1200 as the second fastest machine.

One of the major shortcomings of parallel computers is a lack of uniform programming model to program them.

Programming Parallel Computers

One of the major shortcomings of parallel computers is a lack of uniform programming model to program them. One of the greatest advantages of a uniprocessor computer is the standardization of programming languages. A program in a high level language written for a uniprocessor sequential computer can be executed without any change on any other uniprocessor computer. This allows development of program libraries for a large class of applications which can be used on any sequential uniprocessor machine. Parallel computer designers have been examining various approaches to programming which would allow a uniform programming model independent of the architecture of the parallel computer. We have seen that parallel computer architecture varies widely ranging from shared memory computers to a network of workstations. The challenge is to develop a uniform method of programming independent of the architecture. The first major attempt at creating a common programming method is known as the *parallel virtual machine* (PVM) model. Parallel virtual machine is a software system that makes a variety of architectures such as symmetric multiprocessor, message passing machine, network of workstations, distributed shared memory machine look like one large distributed memory computer called a *virtual machine*. In PVM a task is defined as a unit of computation. PVM software supplies the routines to automatically start up tasks on the virtual machine and allows the tasks to communicate and synchronize with each other.

Parallel computer designers have been examining various approaches to programming which would allow a uniform programming model independent of the architecture of the parallel computer.

PVM system is composed of two parts. The first part is a program which resides on all the computers making up the virtual machine. (It is called a daemon abbreviated pvmd). The second part of the system is a library of PVM interface routines which contains user callable routines for message passing, spawning processes, coordinating tasks, and modifying the virtual machine. Application programs must be linked with the library to use PVM.

The major problem in using PVM is that it is a low level aid to parallel programming. In other words the entire work of dividing up a large job into tasks, parallelizing, allocating tasks and coordinating tasks is the users' responsibility. The major contribution of PVM is easing the portability of parallel programs from one parallel architecture to another. PVM started as a project at the Oak Ridge National Laboratories, USA, in 1989. It was widely distributed as free software and gained popularity. As it was the effort of a small group, efforts were initiated to get a widely acceptable standard for programming message passing computers using the ideas of PVM and improving it. This resulted in a standard called *message passing interface* (MPI) standard which is now supported by many vendors of parallel computers.

The major problem in using PVM is that it is a low level aid to parallel programming.

MPI is a portable message passing standard that facilitates development of parallel applications and libraries. It defines the syntax and semantics of a core of library routines useful to a large class of users who write portable message passing programs in C or Fortran 90. Many vendors have implemented MPI on their parallel machines.

Conclusions

We have seen in the last few years a rapid increase in the power of personal computers (PCs). The speed has been doubling almost every 18 months. A similar trend is also observed in the capacity and speed of hard disks. The latest PCs have a clock speed of over 466 MHz, main memory of 256MB and a secondary memory of 10GB. The increase in speed seems relentless and expected to continue for the next few years. Thus it seems worthwhile to ask the question 'Is it possible to build a supercomputer using PCs and interconnecting them using a standard interconnection such as gigabit Ethernet?' The answer is yes. In fact, the network of workstations architecture for parallel computers follows this idea. (see *Box 2*). The major advantage of such a system is easy expandability as new PCs/ workstations become available. In fact the new PARAM 10000

MPI is a portable message passing standard that facilitates development of parallel applications and libraries.

Box 2 Build Your Own Supecomputer

We have seen a phenomenal increase in the speed and capacity of Pentium based personal computers. A PC running at 466 MHz with a main memory of 256 MB and a 10GB disk is now easily available. Interconnection of PCs is now standardized with 100 Mbit/sec. Ethernets and gigabit ethernets are on the horizon. It is thus possible to build parallel machines by interconnecting a group of headless PCs (i.e. PCs without a VDU and a keyboard) all running LINUX (a version of Unix available in the open domain) with an inter-computer communication using a simplified version of TCP/IP protocol and MPI standard for parallel programming. A system with 16 PCs can give a peak speed in the gigaflops range. This class of machines is now being used by scientists in many laboratories. They are called 'Beowulf class cluster computers'.

architecture of CDAC follows this general idea. Programming such a system is not as simple as programming vector supercomputers or shared memory computers. However, systems such as MPI and PVM are getting standardized and are being improved. The trend is thus away from large, expensive vector type supercomputers towards computers connected to a high speed network. A separate high speed internet is being planned in many countries to provide interconnectivity among geographically distributed high performance computers within the country or even across the world. With programming systems getting standardised we are getting into an era of distributed supercomputing where owning a supercomputer is not as important as having an easy access to supercomputing.

Suggested Reading

[1] V Rajaraman. *Supercomputers*. Wiley Eastern. New Delhi. 1993 (New edition to be published by Universities Press, Hyderabad 1998).

[2] P R Woodward. Perspectives in Supercomputing: Three Decades of Change. *IEEE Computer*. 29. 10, October 1996.

[3] K Kennedy and others. A Nationwide Parallel Computing Environment. *Communications of ACM*. 40. 11. Nov.1997.

[4] D J Kuck. *High Performance Computing*. Oxford University Press. UK, 1996.

[5] Grupp W and others. *Using MPI: Portable Parallel Programming with Message Passing Interface*. MIT Press. Cambridge. M.A. USA, 1994.

[6] Abhiram Ranade. An Introduction to Parallel Computing. *Resonance*. Vol. 3.No. 1, 1998.

Address for Correspondence
V Rajaraman
Supercomputer Education and Research Centre
Indian Institute of Science
Bangalore 560 012, India.