Journal Of Harmonized Research (JOHR)

Journal Of Harmonized Research in Engineering
1(2), 2013, 54-64

ISSN 2347 – 7393

Original Research Article

High Performance Computing through Parallel and Distributed Processing

Shikha Yadav, Preeti Dhanda, Nisha Yadav

Department of Computer Science and Engineering,
Dronacharya College of Engineering, Khentawas,
Farukhnagar, Gurgaon, India

Abstract:

There is a very high need of High Performance Computing (HPC) in many applications like space science to Artificial Intelligence. HPC shall be attained through Parallel and Distributed Computing. In this paper, Parallel and Distributed algorithms are discussed based on Parallel and Distributed Processors to achieve HPC. The Programming concepts like threads, fork and sockets are discussed with some simple examples for HPC.

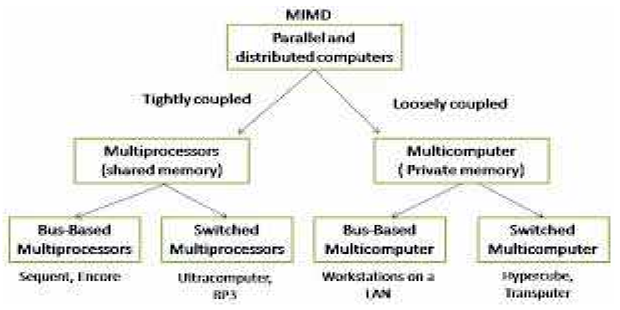
Keywords: High Performance Computing, Parallel and Distributed processing, Computer Architecture

Introduction

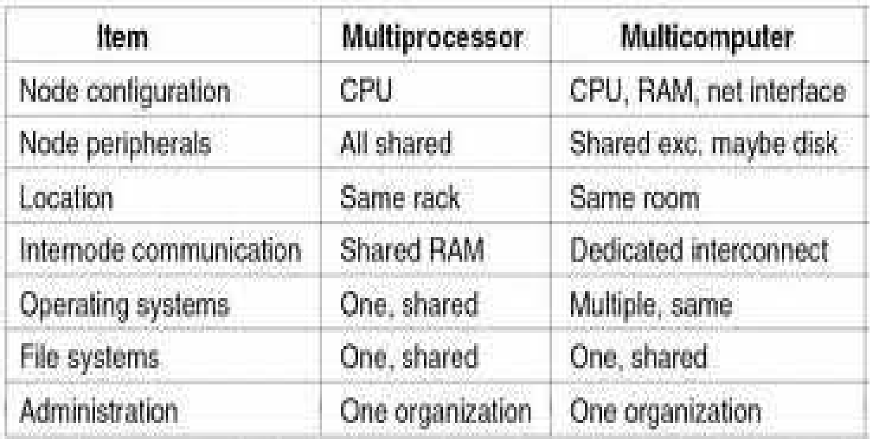
Computer Architecture and Programming play a significant role for High Performance computing (HPC) in large applications Space science to Artificial Intelligence. The Algorithms are problem solving procedures and later these algorithms transform in to particular Programming language for HPC. There is need to study algorithms for High Performance Computing. These Algorithms are to be designed to computer in reasonable time to solve large problems like weather forecasting, Tsunami, Remote Sensing, National calamities, Defence, Mineral exploration, Finite-element, Cloud Computing, and Expert Systems etc. The Algorithms are Non-Recursive Algorithms, Recursive Algorithms, Parallel Algorithms and Distributed Algorithms.

The Algorithms must be supported the Computer Architecture. The Computer Architecture is characterized with Flynn's Classification SISD, SIMD, MIMD, and MISD. Most of the Computer Architectures are supported with SIMD (Single Instruction Multiple Data Streams). The class of Computer Architecture is VLSI Processor, Multi-processor, Vector Processor and Multiple Processor.

For Correspondence:
preeti.dhanda01ATgmail.com
Received on: October 2013
Accepted after revision: December 2013
Downloaded from: www.johronline.com



High-Performance Computing (HPC) is used to describe computing environments which utilize supercomputers and computer clusters to address complex computational requirements, support applications with significant processing time requirements, or require processing of significant amounts of data. Supercomputers have generally been associated with scientific research and compute-intensive types of problems, but more and more supercomputer technology is appropriate for both compute-intensive and data-intensive applications. A new trend in supercomputer design for high-performance computing is using clusters of independent processors connected in parallel. Many computing problems are suitable for parallelization, often problems can be divided in a manner so that each independent processing node can work on a portion of the problem in parallel by simply dividing the data to be processed, and then combining the final processing results for each portion. This type of parallelism is often referred to as data-parallelism, and data-parallel applications are a potential solution to petabyte scale data processing requirements. Data-parallelism can be defined as a computation applied independently to each data item of a set of data which allows the degree of parallelism to be scaled with the volume of data. The most important reason for developing data-parallel applications is the potential for scalable performance in high-performance computing, and may result in several orders of magnitude performance improvement.



Item	Multiprocessor	Multicomputer
Node configuration	CPU	CPU, RAM, net interface
Node peripherals	All shared	Shared exc. maybe disk
Location	Same rack	Same room
Internode communication	Shared RAM	Dedicated interconnect
Operating systems	One, shared	Multiple, same
File systems	One, shared	One, shared
Administration	One organization	One organization

The discussion below focuses on the case of multiple computers, although many of the issues are the same for concurrent processes running on a single computer.
Three view points are commonly used:
Parallel algorithms in shared-memory model-
All computers have access to a shared memory. The algorithm designer chooses the program executed by each computer.
One theoretical model is the parallel random access machines (PRAM) that are used. Though, the classical PRAM model assumes synchronous access to the shared memory.
A model that is closer to the behaviour of real-world multiprocessor machines and takes into account the use of machine instructions, such as Compare-and-swap (CAS), is that of asynchronous shared memory.
Parallel algorithms in message-passing model-
The algorithm designer chooses the structure of the network, as well as the program executed by each computer.

Models such as Boolean circuits and sorting networks are used. A Boolean circuit can be seen as a computer network: each gate is a computer that runs an extremely simple computer program. Similarly, a sorting network can be seen as a computer network, each comparator is a computer.

Distributed algorithms in message-passing model-

The algorithm designer only chooses the computer program. All computers run the same program. The system must work accurately regardless of the structure of the network.A commonly used model is a graph with one finite-state machine per node.

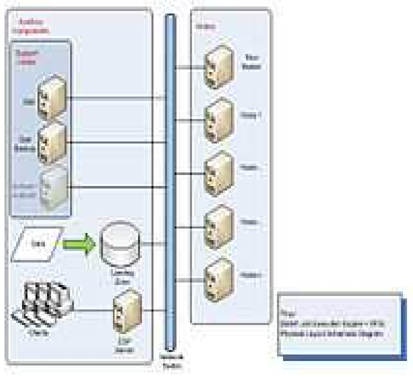
Architecture of high performance computers:

The Department of High Performance Computer Architecture is concerned with the architecture, the application and the continued development of high performance computers beneficial to the natural and life sciences. Our focus is on the selection and analysis of experimental data generated by accelerator facilities such as GSI ("Society for Heavy Ion Research" in Darmstadt, Germany) and CERN (the European Centre for Nuclear Research in Geneva, Switzerland). Both of these facilities employ shared, typically massive parallel systems and clusters operating under high-level, real-time and dependability standards. Our task is the research and development of new computer architectures and algorithms to achieve better energy-efficiency. Within the context of shared computing we implement both GRID and virtual technologies as well as cloud computing systems.

High Performance Computing Cluster (HPCC) System Architecture-

The HPCC system architecture includes two distinct cluster processing environments, each of which can be optimized independently for its parallel data processing purpose. The first of these platforms is called a Data Refinery whose overall purpose is the general processing of massive volumes of raw data of any type for any purpose but typically used for data cleansing and hygiene, ETL processing of the raw data, record linking and entity resolution, large-scale ad-hoc complex analytics, and creation of keyed data and indexes to support high-performance structured queries and data warehouse applications. The Data Refinery is also referred to as Thor, a reference to the mythical Norse god of thunder with the large hammer symbolic of crushing large amounts of raw data into useful information. A Thor cluster is similar in its function, execution environment, filesystem, and capabilities to the Google.

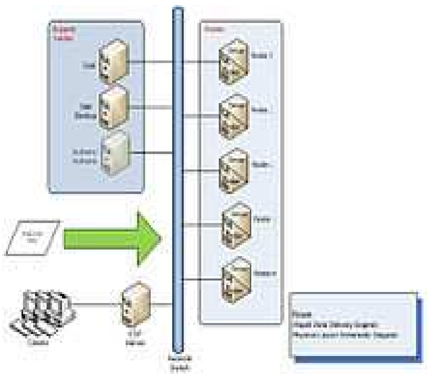
The figure given below shows a representation of a physical Thor processing cluster which functions as a batch job execution engine for scalable data-intensive computing applications. In addition to the Thor master and slave nodes, additional auxiliary and common components are needed to implement a complete HPCC processing environment.



Thor Processing Cluster

The second of the parallel data processing platforms is called Roxie and functions as a rapid data delivery engine. This platform is designed as an online high-performance structured query and analysis platform or data warehouse delivering the parallel data access processing requirements of online applications through Web services interfaces supporting thousands of simultaneous queries and users with sub-second response times. Roxie utilizes a distributed indexed file system to provide parallel processing of queries using an optimized execution environment and filesystem for high-performance online processing. Both Thor and Roxie clusters utilize the ECL programming language for implementing applications, increasing continuity and programmer productivity.

the figure given below shows a representation of a physical Roxie processing cluster which functions as an online query execution engine for high-performance query and data warehousing applications. A Roxie cluster includes multiple nodes with server and worker processes for processing queries; an additional auxiliary component called an ESP server which provides interfaces for external client access to the cluster; and additional common components which are shared with a Thor cluster in an HPCC environment. Although a Thor processing cluster can be implemented and used without a Roxie cluster, an HPCC environment which includes a Roxie cluster should also include a Thor cluster. The Thor cluster is used to build the distributed index files used by the Roxie cluster and to develop online queries which will be deployed with the index files to the Roxie cluster.



Roxie Processing Cluster

Distributed processing:

Distributed computing is a field of computer science that studies distributed systems. A distributed system is a software system in which components located on networked computers communicate and coordinate their actions by passing messages. The components interact with each other in order to achieve a common goal. There are many substitutes for the message passing mechanism, including RPC-like connectors and message queues. Three significant characteristics of distributed systems are: concurrency of components, lack of a global clock, and independent failure of components. An important aim and challenge of distributed systems is location transparency. Examples of distributed systems vary from SOA-based systems to massively multiplayer online games to peer-to-peer applications.

A computer program that runs in a distributed system is called a distributed program, and distributed programming is the process of writing such programs.

Distributed computing also refers to the use of distributed systems to solve computational problems. In distributed computing, a problem is divided into many tasks, each of which is solved by one or more computers which communicate with each other by message passing.

Properties of distributed systems-

So far the focus has been on designing a distributed system that solves a given problem. A complementary research problem is studying the properties of a given distributed system.

The halting problem is an analogous example from the field of centralised computation: we are given a computer program and the task is to decide whether it halts or runs forever. The halting problem is notdecidable in the general case, and naturally understanding the behaviour of a computer network is at least as hard as understanding the behaviour of one computer.

However, there are many interesting special cases that are decidable. In specific case, it is possible to provide reason about the behaviour of a network of finite-state machines. One example is telling whether a given network of interacting (asynchronous and non-deterministic) finite-state machines can reach a deadlock. This problem is PSPACE-complete, i.e., it is decidable, but it is not likely that there is an efficient (centralised, parallel or distributed) algorithm that solves the problem in the case of large networks.

Examples:

Examples of distributed systems and applications of distributed computing contain the following:

1.-Telecommunication networks:

Telephone networks and cellular networks

Computer networks such as the Internet

Wireless sensor networks

Routing algorithms

2.-Network applications:

Worldwide web and peer-to-peer networks

Massively multiplayer online games and virtual reality communities
Distributed databases and distributed database management systems
Network file systems
Distributed information processing systems such as banking systems and airline reservation systems
3.-Real-time process control:
Aircraft control systems
Industrial control systems
4.-Parallel computation:
Scientific computing, including cluster computing and grid computing and various volunteer computing projects; see the list of distributed computing projects rendering in computer graphics.
Applications:
Reasons for using distributed systems and distributed computing may include:
Parallel processing:

The very nature of an application may require the use of a communication network that connects several computers: for example, data produced in one physical location and required in another location.
There are many cases in which the use of a single computer would be possible in principle, but the use of a distributed system is beneficial for practical reasons. For example, it may be more cost-efficient to obtain the desired level of performance by using a cluster of several low-end computers, in comparison with a single high-end computer. A distributed system can provide more reliability than a non-distributed system, as there is no single point of failure. Moreover, a distributed system may be easier to expand and manage than a monolithic uniprocessor system.

IBM's Blue Gene massively parallel supercomputer

Parallel computing is a form of computation in which many calculations are carried out concurrently, operating on the principle that large problems can frequently be divided into smaller ones, which are then solved simultaneously ("in parallel"). There are several different forms of parallel computing: bit-level, instruction level, data, and task parallelism. Parallelism has been employed for many years, primarily in high-performance computing, but interest in it has grown lately due to the physical constraints preventing frequency scaling. As power consumption (and consequently heat generation) by computers has become a concern in recent years, parallel computing has become the dominant paradigm in computer architecture, mainly in the form of multicore processors.
Parallel computers can be roughly classified according to the level at which the hardware supports parallelism, with multi-core and multi-processor computers having multiple processing elements within a single machine, while clusters, MPPs, and grids use multiple computers to work on the same task. Specialized parallel computer architectures are

sometimes used alongside traditional processors, for accelerating specific tasks.

Parallel computer programs are more difficult to write than sequential ones, because concurrency introduces several new classes of potential software bugs, of which race conditions are the most common. Communication and synchronization between the different subtasks are typically some of the greatest complications to getting good parallel program performance.

Parallel computing is the simultaneous use of more than one CPU or processor core to execute a program or multiple computational threads. Ideally, parallel processing makes programs run faster because there are more engines (CPUs or Cores) running it. In practice, it is often tough to divide a program in such a way that separate CPUs or cores can execute different portions without interfering with each other. Most computers have just one CPU, but some models have several, and multi-core processor chips are becoming the norm. There are even computers with thousands of CPUs.

With single-CPU, single-core computers, it is possible to perform parallel processing by connecting the computers in a network. However, this type of parallel processing requires very sophisticated software called distributed processing software.

Note that parallelism differs from concurrency. Concurrency is a term used in the operating systems and databases communities which refers to the property of a system in which multiple tasks remain logically active and make progress at the same time by interleaving the execution order of the tasks and thereby creating an illusion of simultaneously executing instructions. Parallelism, on the other hand, is a term typically used by the supercomputing community to describe executions that physically execute simultaneously with the goal of solving a problem in less time or solving a larger problem in the same time. Parallelism exploits concurrency.

Parallel processing is also called parallel computing. In the quest of cheaper computing alternatives parallel processing provides a viable option. The idle time of processor cycles across network can be used effectively by sophisticated distributed computing software. The term parallel processing is used to represent a large class of techniques which are used to provide simultaneous data processing tasks for the purpose of increasing the computational speed of a computer system.

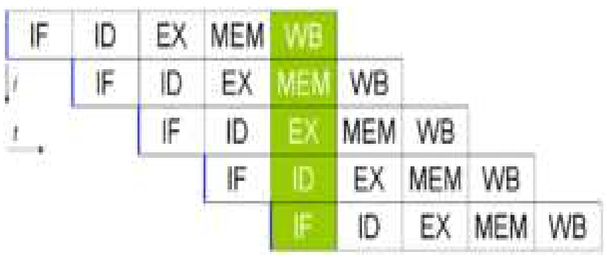
<u>Types of parallelism-</u>

➢ Bit-level parallelism-

From the advent of very-large-scale integration (VLSI) computer-chip fabrication technology in the 1970s until about 1986, speed-up in computer architecture was driven by doubling computer word size—the amount of information the processor can manipulate per cycle.Increasing the word size reduces the number of instructions the processor must execute to perform an operation on variables whose sizes are greater than the length of the word. For example, where an 8-bit processor must add two 16-bit integers, the processor must first add the 8 lower-order bits from each integer using the standard addition instruction, then add the 8 higher-order bits using an add-with-carry instruction and the carry bit from the lower order addition; thus, a 8-bit processor requires two instructions to complete a single operation, where a 16-bit processor would be able to complete the operation with a single instruction.

Historically, 4-bit microprocessors were replaced with 8-bit, then 16-bit, then 32-bit microprocessors. This trend generally came to an end with the introduction of 32-bit processors, which has been a standard in general-purpose computing for two decades. Not until recently (c. 2003–2004), with the advent of x86-64 architectures, have 64-bit processors become commonplace.

➢ Instruction-level parallelism-



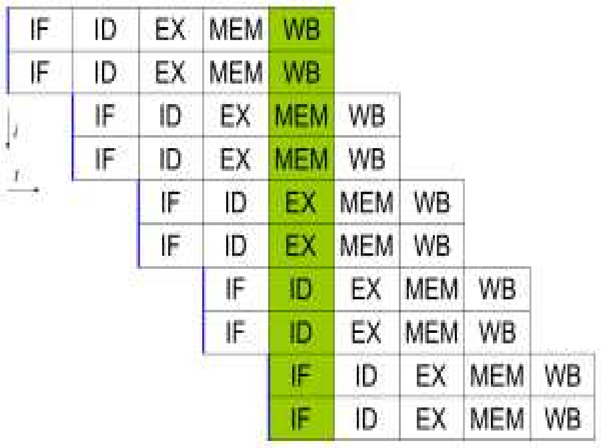
A canonical five-stage pipeline in a RISC machine (IF = Instruction Fetch, ID = Instruction Decode, EX = Execute, MEM = Memory access, WB = Register write back)

A computer program is in essence, a stream of instructions executed by a processor. These instructions can be re-ordered and combined into groups which are then executed in parallel without changing the result of the program. This is known as instruction-level parallelism. Advances in instruction-level parallelism dominated computer architecture from the mid-1980s until the mid-1990s.

Modern processors have multi-stage instruction pipelines. Each stage in the pipeline corresponds to a different action the processor performs on that instruction in that stage; a processor with an N-stage pipeline can have up to N different instructions at different stages of completion. The canonical example of a pipelined processor is a RISC processor, with five stages: instruction fetch, decode, execute, memory access, and write back. The Pentium 4 processor had a 35-stage pipeline.

In addition to instruction-level parallelism from pipelining, some processors can issue more than one instruction at a time. These are known as superscalar processors. Instructions can be grouped together only if there is no data dependency between them. Scoreboarding and the Tomasulo algorithm (which is similar to scoreboarding but makes use of register renaming) are two of the most common techniques for implementing out-of-order execution and instruction-level parallelism.

➢ Task parallelism-



IF	ID	EX	MEM	WB				
IF	ID	EX	MEM	WB				
	IF	ID	EX	MEM	WB			
	IF	ID	EX	MEM	WB			
		IF	ID	EX	MEM	WB		
		IF	ID	EX	MEM	WB		
			IF	ID	EX	MEM	WB	
			IF	ID	EX	MEM	WB	
				IF	ID	EX	MEM	WB
				IF	ID	EX	MEM	WB

Figure-A five-stage pipelined superscalar processor, capable of issuing two instructions percycle. It can have two instructions in each stage of the pipeline, for a total of up to 10instructions (shown in green) being simultaneously executed.

Task parallelism is the characteristic of a parallel program that "entirely different calculations can be performed on either the same or different sets of data". This contrasts with data parallelism, where the same calculation is performed on the same or different sets of data. Task parallelism does not usually scale with the size of a problem.

Classes of parallel computers-

Parallel computers can be roughly classified according to the level at which the hardware supports parallelism. This classification is broadly analogous to the distance between basic computing nodes. These are not mutually exclusive; for example, clusters of symmetric multiprocessors are relatively common.

➢ Multicore computing-

A multicore processor is a processor that includes multiple execution units ("cores") on the same chip. These processors differ from superscalar processors, which can issue multiple instructions per cycle from one instruction stream (thread); in contrast, a multicore processor can issue multiple instructions per cycle from multiple instruction streams. IBM's Cell microprocessor, designed for use in the Sony PlayStation 3, is another prominent multicore processor.

Each core in a multicore processor can potentially be superscalar as well—that is, on every cycle, each core can issue multiple instructions from one instruction stream. Simultaneous multithreading (of which Intel's Hyper Threading is the best known) was an early form of pseudo-multi coreism. A processor capable of simultaneous multithreading has only one execution unit ("core"), but when that execution unit is idling (such as during a cache miss), it uses that execution unit to process a second thread.

➢ Symmetric multiprocessing-

A symmetric multiprocessor (SMP) is a computer system with multiple identical processors that share memory and connect via a bus. Bus contention prevents bus architectures from scaling. As a result, SMPs generally do not comprise more than 32 processors. "Because of the small size of the processors and the significant reduction in the requirements for bus bandwidth achieved by large caches, such symmetric multiprocessors are extremely cost-effective, provided that a sufficient amount of memory bandwidth exists."

➢ Distributed computing-

A distributed computer (also known as a distributed memory multiprocessor) is a distributed memory computer system in which the processing elements are connected by a network. Distributed computers are highly scalable.

➢ Cluster computing-

A cluster is a group of loosely coupled computers that work together closely, so that in some respects they can be regarded as a single computer. Clusters are composed of multiple standalone machines connected by a network. While machines in a cluster do not have to be symmetric, load balancing is more difficult if they are not. The most common type of cluster is the Beowulf cluster, which is a cluster implemented on multiple identical commercial off-the-shelf computers connected with a TCP/IP Ethernet local area network. Beowulf technology was originally developed by Thomas Sterling and Donald Becker. The vast majority of the TOP500 supercomputers are clusters.

➢ Massive parallel processing-

A massively parallel processor (MPP) is a single computer with many networked processors. MPPs have many of the same characteristics as clusters, but MPPs have specialized interconnect networks (whereas clusters use commodity hardware for networking). MPPs also tend to be larger than clusters, typically having "far more" than 100 processors. In a MPP, "each CPU contains its own memory and copy of the operating system and application. Each subsystem communicates with the others via a high-speed interconnect."

➢ Grid computing-

Distributed computing is the most distributed form of parallel computing. It makes use of computers communicating over the Internet to work on a given problem. Because of the low bandwidth and extremely high latency available on the internet ,distributed computing typically deals only with embarrassingly parallel problems. Many distributed computing applications have been created, of which SETI@home and Folding@home are the best-known examples.

Most grid computing applications use middleware, software that sits between the operating system and the application to manage network resources and standardize the software interface. The most common distributed computing middleware is the Berkeley Open Infrastructure for Network Computing (BOINC). Often, distributed computing software makes use of "spare cycles", performing computations at times when a computer is idling.

➢ Specialized parallel computers-

Within parallel computing, there are specialized parallel devices that remain niche areas of interest. While not domain-specific, they tend to be applicable to only a few classes of parallel problems.

Reconfigurable computing with field-programmable gate arrays-

Reconfigurable computing is the use of a field-programmable gate array (FPGA) as a co-processor to a general-purpose computer. An FPGA is, in essence, a computer chip that can rewire itself for a given task.

FPGAs can be programmed with hardware description languages such as VHDL or Verilog. However, programming in these languages can be tedious. Several vendors have created C to HDL languages that attempt to emulate the syntax and/or semantics of the C programming language, with which most programmers are familiar.

General-purpose computing on graphics processing units (GPGPU)-

General-purpose computing on graphics processing units (GPGPU) is a fairly recent

trend in computer engineering research.
GPUs are co-processors that have been heavily optimized for computer graphics processing.
Computer graphics processing is a field dominated by data parallel operations—particularly linear algebra matrix operations.
In the early days, GPGPU programs used the normal graphics APIs for executing programs. However, several new programming languages and platforms have been built to do general purpose computation on GPUs with both Nvidia and AMD releasing programming environments with CUDA and Stream SDK respectively. Other GPU programming languages include Brook GPU, PeakStream, and Rapid Mind. Nvidia has also released specific products for computation in their Tesla series. The technology consortium Khronos Group has released the OpenCL specification, which is a framework for writing programs that execute across platforms consisting of CPUs and GPUs. AMD, Apple, Intel, Nvidia and others are supporting OpenCL.

➢ Application-specific integrated circuits-

Several application-specific integrated circuit (ASIC) approaches have been devised for dealing with parallel application because an ASIC is (by definition) specific to a given application, it can be fully optimized for that application. As a result, for a given application, an ASIC tends to outperform a general-purpose computer. However, ASICs are created by X-ray lithography. This process requires a mask, which can be extremely expensive. A single mask can cost over a million US dollars. (The smaller the transistors required for the chip, the more expensive the mask will be.) Meanwhile, performance increases in general-purpose computing over time (as described by Moore's Law) tend to wipe out these gains in only one or two chip generations. High initial cost, and the tendency to be overtaken by Moore's-law-driven general-purpose computing, has rendered ASICs unfeasible for most parallel computing applications. However, some have been built. One example is the peta-flop RIKEN MDGRAPE-3 machine which uses custom ASICs for molecular dynamics simulation.

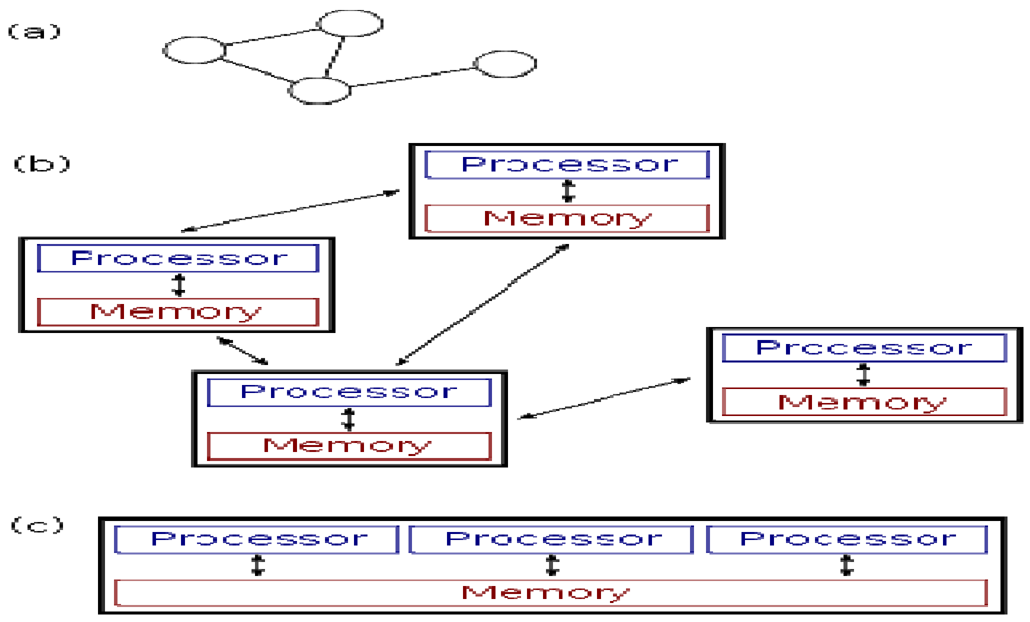
➢ Vector processors-

A vector processor is a CPU or computer system that can execute the same instruction on large sets of data. "Vector processors have high-level operations that work on linear arrays of numbers or vectors. An example vector operation is A = B × C, where A, B, and C are each 64-element vectors of 64-bit floating-point numbers." They are closely related to Flynn's SIMD classification.
Cray computers became famous for their vector-processing computers in the 1970s and 1980s. However, vector processors—both as CPUs and as full computer systems—have generally disappeared. Modern processor instruction sets do include some vector processing instructions, such as with AltiVecand Streaming SIMD Extensions (SSE).

Advantages-
Faster execution time so, higher throughput.

Disadvantages-
More hardware required, also more power requirements. Not good for low power and mobile devices.

Parallel and distributed computing:
Distributed systems are groups of networked computers, which have the similar goal for their work. The terms "concurrent computing", "parallel computing", and "distributed computing" have a lot of overlap, and no clear distinction exists between them. The same system may be characterised both as "parallel" and "distributed"; the processors in a typical distributed system run concurrently in parallel. Parallel computing may be seen as a particular tightly coupled form of distributed computing, and distributed computing may be seen as a loosely coupled form of parallel computing. Still, it is possible to roughly classify concurrent systems as "parallel" or "distributed" using the following criteria:



(a),(b)-A Distributed system (c)-A Parallel system

In parallel computing, all processors may have access to a shared memory to exchange information between processors. In distributed computing, each processor has its own private memory (distributed memory). Information is exchanged by passing messages between the processors.
The figure above illustrates the difference between distributed and parallel systems. Figure (a) is a schematic view of a typical distributed system; as usual, the system is represented as a network topology in which each node is a computer and each line connecting the nodes is a communication link. Figure (b) shows the same distributed system in more detail: each computer has its own local memory, and information can be exchanged only by passing messages from one node to another by using the available communication links. Figure (c) shows a parallel system in which each processor has a direct access to a shared memory.
The situation is further complicated by the traditional uses of the terms parallel and distributed algorithm that do not quite match the above definitions of parallel and distributed systems. Nevertheless, as a rule of thumb, high-performance parallel computation in a shared-memory multiprocessor uses parallel algorithms while the coordination of a large-scale distributed system uses distributed algorithms.

Conclusion

High Performance Computing is required when large computations of the problems. HPC shall be performed through the Parallel and distributed processing. The Parallel and Distributed are discussed based on Computer Architecture. The Class of Algorithms and Class of Computer Architecture are discussed. The Programming concepts like threads, fork and sockets are discussed for HPC. This shall be extending to large problems likeGrid Computing and Cloud Computing. Usually Fortran is used for HPC. The Perl and Java Programming are also useful for HPC.

References:

1. Bader, David; Robert Pennington (June 1996).”Cluster computing: Applications”. Georgia Tech College of computing. Retrieved 2007-07-13.
2. **“**Nuclear weapons supercomputer reclaims world speed record for US**”**. The Telegraph. 18 Jun 2012. Retrieved 18 Jun 2012.

3. *Network-Based Information Systems: First International Conference, NBIS 2007* ISBN 3-540-74572-6 page 375
4. William W. Hargrove, Forrest M. Hoffman and Thomas Sterling (August 16, 2001). “Do-It-Yourself Supercomputer” .*Scientific American* **265** (2). pp. 72–79. Retrieved October 18, 2011.
5. William W. Hargrove and Forrest M. Hoffman (1999). “Cluster Computing: LinuxTaken to the Extreme”. *Linux magazine*. Retrieved October 18, 2011.
6. TOP500 list to view all clusters on the TOP500 select "cluster" as architecture from the sub list menu.
7. M. Yokokawa et al *The K Computer*, in "International Symposium on Low Power Electronics and Design" (ISLPED) 1-3 Aug. 2011, pages 371-372
8. Pfister, Gregory (1998). *In Search of Clusters* (2nd ed.). Upper Saddle River, NJ: Prentice Hall PTR. p. 36. ISBN 0-13-899709-8.
9. *Readings in computer architecture* by Mark Donald Hill, Norman Paul Jouppi, GurindarSohi1999 ISBN 978-1-55860-539-8 page 41-48
10. *High Performance Linux Clusters* by Joseph D. Sloan 2004 ISBN 0-596-00570-9 page
11. *High Performance Computing for Computational Science - VECPAR 2004* by Michel Daydé, Jack Dongarra 2005ISBN 3-540-25424-2 pages 120-121
12. Hamada T. *et al.* (2009) A novel multiple-walk parallel algorithm for the Barnes–Hut treecode on GPUs – towards cost effective, high performance N-body simulation. *Comput. Sci. Res. Development*24:21-31.doi:10.1007/s00450-009-0089-1.
13. Maurer, Ryan: Xen Virtualization and Linux Clustering.
14. *Distributed services with OpenAFS: for enterprise and education* by Franco Milicchio, Wolfgang Alexander Gehrke 2007, ISBN pages 339-341[1].
15. Grid and Cluster Computing by Parbhu 2008 8120334280 pages 109-112.
16. Gropp, William; Lusk, Ewing; Skjellum, Anthony (1996). "A High-Performance, Portable Implementation of the MPI Message Passing Interface". *Parallel Computing*. CiteSeerX: 10.1.1.102.9485.
17. *Computer Organization and Design* by David A. Patterson and John L. Hennessy 2011 ISBN 0-12-374750-3 pages 641-642.
18. K. Shirahata, et al *Hybrid Map Task Scheduling for GPU-Based Heterogeneous Clusters* in: Cloud Computing Technology and Science (CloudCom), 2010 Nov. 30 2010-Dec. 3 2010 pages 733-740 ISBN 978-1-4244-9405[2].
19. Alan Robertson *Resource fencing using STONITH*. IBM Linux Research Center, 2010 [3].
20. *Sun Cluster environment: Sun Cluster 2.2* by Enrique Vargas, Joseph Bianco, David Deeths 2001 ISBN page 58
21. *Computer Science: The Hardware, Software and Heart of It* by Alfred V. Aho, Edward K. Blum 2011 ISBN 1-4614-1167-X pages 156-166 [4].
22. *Parallel Programming: For Multicore and Cluster Systems* by Thomas Rauber, GudulaRünger 2010ISBN3-642-04817-Xpages 94–95 [5].
23. *A debugging standard for high-performance computing* by Joan M. Francioni and Cherri Pancake, in the "Journal of Scientific Programming" Volume 8 Issue 2, April 2000 [6].
24. *Computational Science-- ICCS 2003: International Conference* edited by Peter Sloot 2003 ISBN 3-540-40195-4 pages 291-292.
25. Top500 List-June2006 (1-100)|TOP500 Super Computing Sites.