Parallel Program Development

Abstract
This paper presents the Parallel program development process. Parallel computing is mostly used in modern scenario. The parallel computers perform the tasks parallel and in much lesser time as compared to the serial processing in the serial computers. The parallel execution is fast and saves time and money. We need to know why parallel computing is fast compared to serial computing and what is the need of performing this. This paper presents the introduction to parallel computing including the relevant examples, its need, major areas in which it is used and some parallel programming models.

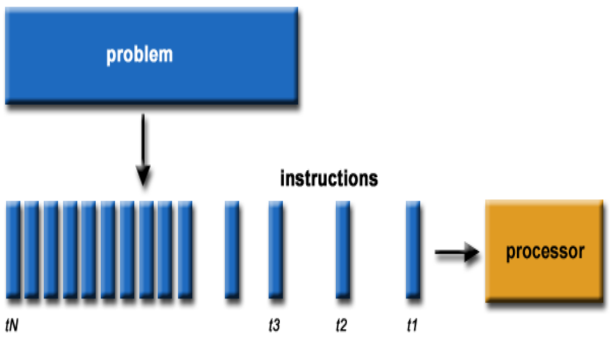
Index Terms
Parallel computers, parallel computing.

1. Introduction

1.1. The difference between serial computing and parallel computing

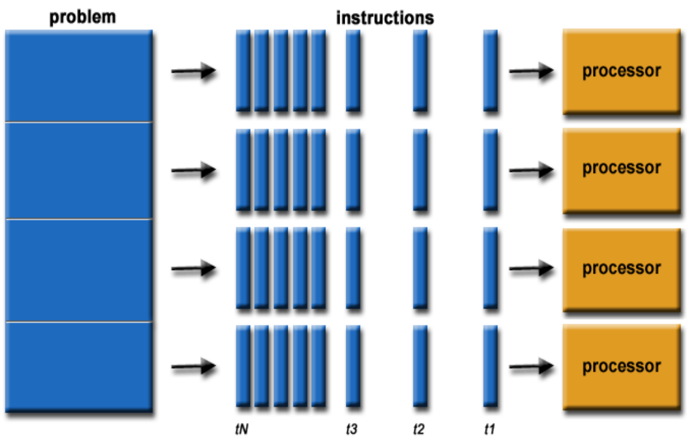
1.1.1.Serial Computing:

Traditionally, software has been written for serial computation:
A problem is broken into a discrete series of instructions and these instructions are executed sequentially one after another. The problem is executed on a single processor. So, only one instruction may execute at any moment in time.



1.1.2. Parallel Computing:

In the simplest sense, **parallel computing** is the simultaneous use of multiple compute resources to solve a computational problem:
A problem is broken into discrete parts that can be solved concurrently and each part is further broken down to a series of instructions. These instructions from each part execute simultaneously on different processors. An overall control/coordination mechanism is employed in parallel computing.



The computational problem should be able to:
a.) Be broken apart into discrete pieces of work that can be solved simultaneously.
b.) Execute multiple program instructions at any moment in time.

c.) Be solved in less time with multiple compute resources than with a single compute resource.
The computer resources are typically:
a.) A single computer with multiple processors/cores
b.) An arbitrary number of such computers connected by a network

1.2.**Parallel Computers:**
Virtually all stand-alone computers today are parallel from a hardware perspective:
a.) Multiple functional units (L1 cache, L2 cache, branch, prefetch, decode, floating-point, graphics processing (GPU), integer, etc.)
b.) Multiple execution units/cores
c.)Multiple hardware threads **[1]**

1.2.1.Reasons for using Parallel Computing:

1. **The Real World is Massively Parallel:**

In the natural world, many complex, interrelated events are happening at the same time, yet within a temporal sequence. Compared to serial computing, parallel computing is much better suited for modeling, simulating and understanding complex, real world phenomena.

2. **Save Time And Money:**

In theory, throwing more resources at a task will shorten its time to completion, with potential cost savings.
Parallel computers can be built from cheap, commodity components.

3. **Solve Larger / More Complex Problems:**

Many problems are so large and/or complex that it is impractical or impossible to solve them on a single computer, especially given limited computer memory.
Example: "Grand Challenge Problems"
Example: Web search engines/databases processing millions of transactions per second

4. **Provide Concurrency:**

A single compute resource can only do one thing at a time. Multiple compute resources can do many things simultaneously.

5. **Make better use of underlying parallel hardware:**

Modern computers, even laptops, are parallel in architecture with multiple processors/cores.
Parallel software is specifically intended for parallel hardware with multiple cores, threads, etc.
In most cases, serial programs run on modern computers "waste" potential computing power. **[2]**

1.2.2. Fields using Parallel Computing

1. **Science and Engineering**:

Historically, parallel computing has been considered to be "the high end of computing", and has been used to model difficult problems in many areas of science and engineering:

2. Atmosphere, Earth, Environment
3. Physics - applied, nuclear, particle, condensed matter, high pressure, fusion, photonics
4. Bioscience, Biotechnology, Genetics
5. Chemistry, Molecular Sciences
6. Geology, Seismology
7. Mechanical Engineering - from prosthetics to spacecraft
8. Electrical Engineering, Circuit Design, Microelectronics
9. Computer Science, Mathematics
10. Defense , Weapons
11. Computer simulations
12. **Industrial and Commercial**:

Today, commercial applications provide an equal or greater driving force in the development of faster computers. These applications require the processing of large amounts of data in sophisticated ways. For example:

13. Databases, data mining
14. Oil exploration
15. Web search engines, web based business services

16. Medical imaging and diagnosis
17. Pharmaceutical design
18. Financial and economic modelling
19. Management of national and multi-national corporations
20. Advanced graphics and virtual reality, particularly in the entertainment industry
21. Networked video and multi-media technologies
22. Collaborative work environments
23. Computer simulations
24. Global Applications:

Parallel computing is now being used extensively around the world, in a wide variety of applications. **[3]**

1.3. Flynn's Classical Taxonomy

There are different ways to classify parallel computers. One of the more widely used classifications, in use since 1966, is called Flynn's Taxonomy. Flynn's taxonomy distinguishes multi-processor computer architectures according to how they can be classified along the two independent dimensions of **Instruction Stream** and **Data Stream**. Each of these dimensions can have only one of two possible states: **Single** or **Multiple**. The matrix below defines the 4 possible classifications according to Flynn:

S I S D **Single Instruction Stream** **Single Data Stream**	**S I M D** **Single Instruction Stream** **Multiple Data Stream**
M I S D **Multiple Instruction Stream** **Single Data Stream**	**M I M D** **Multiple Instruction Stream** **Multiple Data Stream**

1.3.1.Single Instruction, Single Data (SISD):

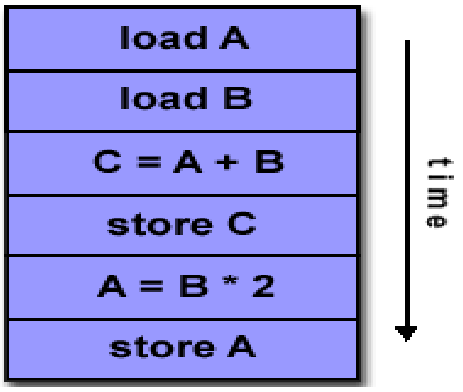
It is a serial (non-parallel) computer. There are two terms; Single instruction and single data.

Single Instruction: Only one instruction stream is being acted on by the CPU during any one clock cycle

Single Data: Only one data stream is being used as input during any one clock cycle

It is deterministic execution and is the oldest type of computer

Examples: older generation mainframes, minicomputers, workstations and single processor/core PCs.



1.3.2. Single Instruction, Multiple Data (SIMD):

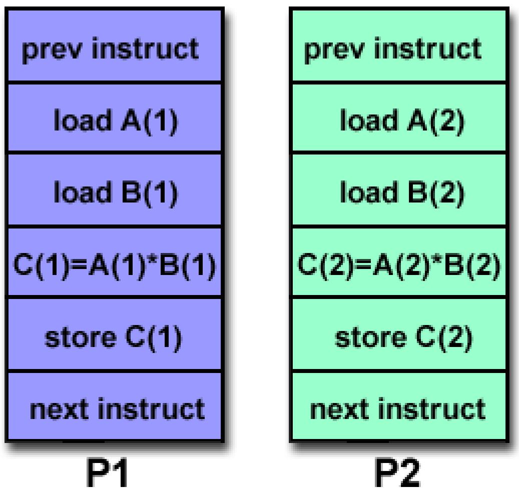
Single Instruction: All processing units execute the same instruction at any given clock cycle

Multiple Data: Each processing unit can operate on a different data element

This kind of parallel computers are best suited for specialized problems characterized by a high degree of regularity, such as graphics/image processing. It includes Synchronous (lockstep) and deterministic execution.

Two varieties: Processor Arrays and Vector Pipelines

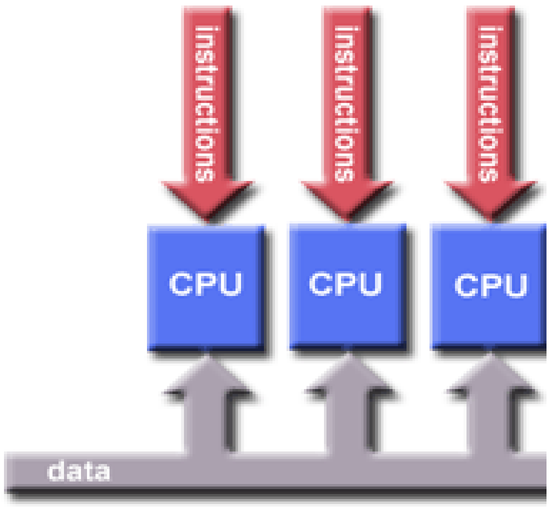
Examples:
Processor Arrays: Thinking Machines CM-2, MasPar MP-1 & MP-2, ILLIAC IV
Vector Pipelines: IBM 9000, Cray X-MP, Y-MP & C90, Fujitsu VP, NEC SX-2, Hitachi S820, ETA10
Most modern computers, particularly those with graphics processor units (GPUs) employ SIMD instructions and execution units.

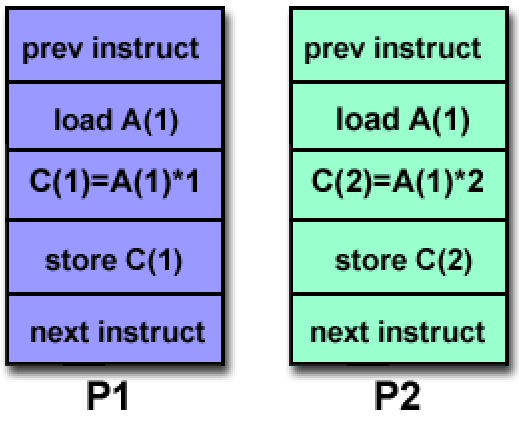


[1]

1.3.3. Multiple Instruction, Single Data (MISD):

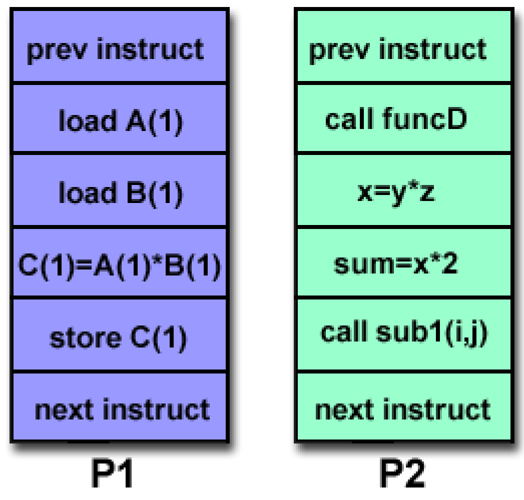
Multiple Instruction: Each processing unit operates on the data independently via separate instruction streams.
Single Data: A single data stream is fed into multiple processing units.
Examples: Few (if any) actual examples of this class of parallel computer have ever existed.
Some conceivable uses might be: multiple frequency filters operating on a single signal stream , multiple cryptography algorithms attempting to crack a single coded message.





1.3.4. Multiple Instruction, Multiple Data (MIMD):

Multiple Instruction: Every processor may be executing a different instruction stream.
Multiple Data: Every processor may be working with a different data stream
Execution can be synchronous or asynchronous, deterministic or non-deterministic Currently, the most common type of parallel computer - most modern supercomputers fall into this category.
Examples: most current supercomputers, networked parallel computer clusters and "grids", multi-processor SMP computers, multi-core PCs.
IBM POWER5, HP/Compaq Alphaserver, Intel IA32, AMD Opteron, Cray XT3, IBM BG/L
Many MIMD architectures also include SIMD execution sub-components **[1]**



[1]

1.4.Some General Parallel Terminology

Like everything else, parallel computing has its own "jargon". Some of the more commonly used terms associated with parallel computing are listed below.

Supercomputing / High Performance Computing (HPC)
Using the world's fastest and largest computers to solve large problems.

Node
A standalone "computer in a box". Usually comprised of multiple CPUs/processors/cores, memory, network interfaces, etc. Nodes are networked together to comprise a supercomputer.

CPU / Socket / Processor / Core
This varies, depending upon who you talk to. In the past, a CPU (Central Processing Unit) was a singular execution component for a computer. Then, multiple CPUs were incorporated into a node. Then, individual CPUs were subdivided into multiple "cores", each being a unique execution unit. CPUs with multiple cores are sometimes called "sockets" - vendor dependent. The result is a node with multiple CPUs, each containing multiple cores

Task
A logically discrete section of computational work. A task is typically a program or program-like set of instructions that is executed by a processor. A parallel program consists of multiple tasks running on multiple processors.

Pipelining
Breaking a task into steps performed by different processor units, with inputs streaming through, much like an assembly line; a type of parallel computing.

Shared Memory
From a strictly hardware point of view, describes a computer architecture where all processors have direct (usually bus based) access to common physical memory. In a programming sense, it describes a model where parallel tasks all have the same "picture" of memory and can directly address and access the same logical memory locations regardless of where the physical memory actually exists.

Symmetric Multi-Processor (SMP)
Hardware architecture where multiple processors share a single address space and access to all resources; shared memory computing.

Distributed Memory
In hardware, refers to network based memory access for physical memory that is not common. As a programming model, tasks can only logically "see" local machine memory and must use communications to access memory on other machines where other tasks are executing.

Communications
Parallel tasks typically need to exchange data. There are several ways this can be accomplished, such as through a shared memory bus or over a network, however the actual event of data exchange is commonly referred to as communications regardless of the method employed.

Synchronization
The coordination of parallel tasks in real

time, very often associated with communications. Often implemented by establishing a synchronization point within an application where a task may not proceed further until another task(s) reaches the same or logically equivalent point Synchronization usually involves waiting by at least one task, and can therefore cause a parallel application's wall clock execution time to increase.

Granularity
In parallel computing, granularity is a qualitative measure of the ratio of computation to communication.
a.) Coarse: relatively large amounts of computational work are done between communication events
b.)Fine: relatively small amounts of computational work are done between communication events

Observed Speedup
Observed speedup of a code which has been parallelized, defined as:

wall-clock time of serial execution
----
wall-clock time of parallel execution

One of the simplest and most widely used indicators for a parallel program's performance.

Parallel Overhead
The amount of time required to coordinate parallel tasks, as opposed to doing useful work. Parallel overhead can include factors such as:

a.) Task start-up time
b.) Synchronizations
c.) Data communications
d.) Software overhead imposed by parallel languages, libraries, operating system, etc.
e.) Task termination time

Massively Parallel
Refers to the hardware that comprises a given parallel system - having many processors. The meaning of "many" keeps increasing, but currently, the largest parallel computers can be comprised of processors numbering in the hundreds of thousands.

Embarrassingly Parallel
Solving many similar, but independent tasks simultaneously; little to no need for coordination between the tasks.

Scalability
Refers to a parallel system's (hardware and/or software) ability to demonstrate a proportionate increase in parallel speedup with the addition of more resources. Factors that contribute to scalability include:

a.) Hardware - particularly memory-cpu bandwidths and network communication properties
b.) Application algorithm
c.) Parallel overhead related
d.) Characteristics of your specific application

Complexity:
In general, parallel applications are much more complex than corresponding serial applications, perhaps an order of magnitude. Not only do you have multiple instruction streams executing at the same time, but you also have data flowing between them.
The costs of complexity are measured in programmer time in virtually every aspect of the software development cycle:

a.) Design
b.) Coding
c.) Debugging
d.) Tuning
e.) Maintenance

Adhering to "good" software development practices is essential when when working with parallel applications - especially if somebody besides you will have to work with the software.

Portability:
Thanks to standardization in several APIs, such as MPI, POSIX threads, and Open MP, portability issues with parallel programs are not as serious as in years past. However, all of the usual portability issues associated with serial programs apply to parallel programs. For example, if you use vendor "enhancements" to Fortran, C or C++, portability will be a problem.Even though standards exist for several APIs, implementations will differ in a number of details, sometimes to the point of requiring code modifications in order to effect portability. Operating systems can play a key role in code portability issues. Hardware architectures are characteristically highly variable and can affect portability.

Resource Requirements:

a.) The primary intent of parallel programming is to decrease execution wall clock time, however in order to accomplish this, more CPU time is required. For example, a parallel code that runs in 1 hour on 8 processors actually uses 8 hours of CPU time.
b.) The amount of memory required can be greater for parallel codes than serial codes, due to the need to replicate data and for overheads associated with parallel support libraries and subsystems.
c.) For short running parallel programs, there can actually be a decrease in performance compared to a similar serial implementation. The overhead costs associated with setting up the parallel environment, task creation, communications and task termination can comprise a significant portion of the total execution time for short runs.

Scalability:
Two types of scaling based on time to solution:
Strong scaling: The total problem size stays fixed as more processors are added.
Weak scaling: The problem size per processor stays fixed as more processors are added.
The ability of a parallel program's performance to scale is a result of a number of interrelated factors. Simply adding more processors is rarely the answer.The algorithm may have inherent limits to scalability. At some point, adding more resources causes performance to decrease. Many parallel solutions demonstrate this characteristic at some point. Hardware factors play a significant role in scalability.
Examples:

.) Memory-cpu bus bandwidth on an SMP machine
a.) Communications network bandwidth
b.) Amount of memory available on any given machine or set of machines
c.) Processor clock speed

Parallel support libraries and subsystems software can limit scalability independent of your application. **[1]**

1.5. Parallel Computer Memory Architectures

1.5.1. Shared Memory

General Characteristics:
1. Shared memory parallel computers vary
for all processors to access all memory as g

2. Multiple processors can operate indepe
3. Changes in a memory location effec processors.
4. Historically, shared memory machines upon memory access times.

1.5.1.1. **Uniform Memory Access (UMA)**
General Characteristics

0. Most commonly represented today b
1. Identical processors
2. Equal access and access times to me
3. Sometimes called CC-UMA - Cach processor updates a location in sha the update. Cache coherency is acco

1.5.1.2 Non-Uniform Memory Access (NUMA):

- Often made by physically linking two or more SMPs
- One SMP can directly access memory of another SMP
- Not all processors have equal access time to all memories
- Memory access across link is slower
- If cache coherency is maintained, then may also be called CC-NUMA - Cache Coherent NUMA

Advantages:

- Global address space provides a user-friendly programming perspective to memory
- Data sharing between tasks is both fast and uniform due to the proximity of memory to CPUs

Disadvantages:

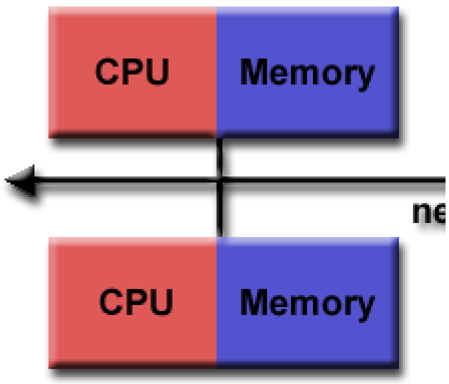
- Primary disadvantage is the lack of scalability between memory and CPUs. Adding more CPUs can geometrically increases traffic on the shared memory-CPU path, and for cache coherent systems, geometrically increase traffic associated with cache/memory management.
- Programmer responsibility for synchronization constructs that ensure "correct" access of global memory.

1.5.2. **Distributed Memory**

General Characteristics:

- Like shared memory systems, distributed memory systems vary widely but share a common characteristic. Distributed memory systems require a communication network to connect inter-processor memory.

- Processors have their own local memory. Memory addresses in one processor do not map to another processor, so there is no concept of global address space across all processors.
- Because each processor has its own local memory, it operates independently. Changes it makes to its local memory have no effect on the memory of other processors. Hence, the concept of cache coherency does not apply.
- When a processor needs access to data in another processor, it is usually the task of the programmer to explicitly define how and when data is communicated. Synchronization between tasks is likewise the programmer's responsibility.
- The network "fabric" used for data transfer varies widely, though it can be as simple as Ethernet.



Advantages:

- Memory is scalable with the number of processors. Increase the number of processors and the size of memory increases proportionately.
- Each processor can rapidly access its own memory without interference and without the overhead incurred with trying to maintain global cache coherency.
- Cost effectiveness: can use commodity, off-the-shelf processors and networking.

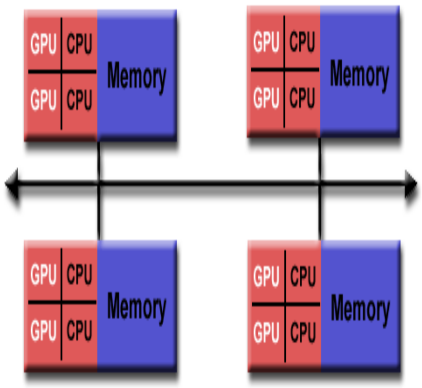
Disadvantages:

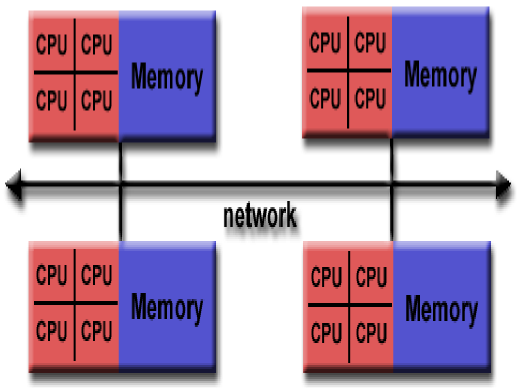
- The programmer is responsible for many of the details associated with data communication between processors.
- It may be difficult to map existing data structures, based on global memory, to this memory organization.
- Non-uniform memory access times - data residing on a remote node takes longer to access than node local data.

1.5.3. Hybrid Distributed-Shared Memory

General Characteristics:

- The largest and fastest computers in the world today employ both shared and distributed memory architectures.
- The shared memory component can be a shared memory machine and/or graphics processing units (GPU).
- The distributed memory component is the networking of multiple shared memory/GPU machines, which know only about their own memory - not the memory on another machine. Therefore, network communications are required to move data from one machine to another.
- Current trends seem to indicate that this type of memory architecture will continue to prevail and increase at the high end of computing for the foreseeable future.





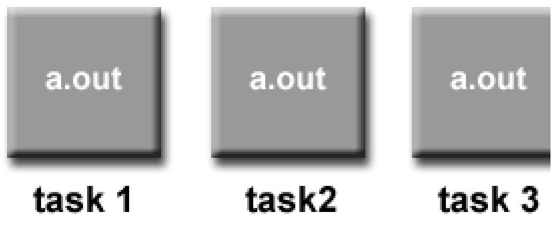
Advantages and Disadvantages:

- Whatever is common to both shared and distributed memory architectures.
- Increased scalability is an important advantage
- Increased programmer complexity is an important disadvantage **[1]**

1.6.Parallel Programming Models

1.6.1.Single Program Multiple Data (SPMD):

- SPMD is actually a "high level" programming model that can be built upon any combination of the

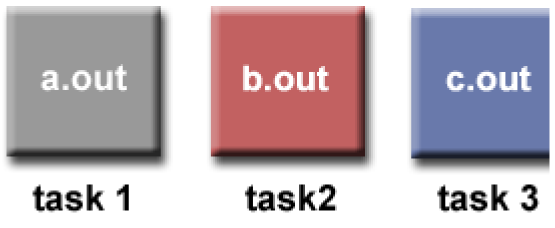


previously mentioned parallel programming models.

- **SINGLE PROGRAM**: All tasks execute their copy of the same program simultaneously. This program can be threads, message passing, data parallel or hybrid.
- **MULTIPLE DATA:** All tasks may use different data
- SPMD programs usually have the necessary logic programmed into them to allow different tasks to branch or conditionally execute only those parts of the program they are designed to execute. That is, tasks do not necessarily have to execute the entire program - perhaps only a portion of it.
- The SPMD model, using message passing or hybrid programming, is probably the most commonly used parallel programming model for multi-node clusters.

1.6.2. **Multiple Program Multiple Data (MPMD):**

- Like SPMD, MPMD is actually a "high level" programming model that can be built upon any combination of the previously mentioned parallel programming models.



- MULTIPLE PROGRAM: Tasks may execute different programs simultaneously. The programs can be threads, message passing, data parallel or hybrid.
- MULTIPLE DATA: All tasks may use different data
- MPMD applications are not as common as SPMD applications, but may be better suited for certain types of problems, particularly those that lend themselves better to functional decomposition than domain decomposition **[4]**

2.Literature Survey

2.2.Guy E. Blelloch and Bruce M. Maggs ' Parallel Algorithms'

Most of today's algorithms are sequential, that is, they specify a sequence of steps in which each step consists of a single operation. These algorithms are well suited to today's computers, which basically perform operations in a sequential fashion. Although the speed at which sequential computers operate has been improving at an exponential rate for many years, the improvement is now coming at greater and greater cost. As a consequence, researchers have sought more cost-effective improvements by building "parallel" computers – computers that perform multiple operations in a single step. In order to solve a problem efficiently on a parallel machine, it is usually necessary to design an algorithm that specifies multiple operations on each step, i.e., a parallel algorithm.

2.3.Ananth Grama, Anshul Gupta, George Karypis, and Vipin Kumar, 'Principles of Parallel Algorithm Design' , 2003.

The first step in developing a parallel algorithm is to decompose the problem into tasks that can be executed concurrently. A given problem may be decomposed into tasks in many different ways. Tasks may be of same, different, or even interminate sizes. A decomposition can be illustrated in the form of a directed graph with nodes corresponding to tasks and edges indicating that the result of one task is required for processing the next. Such a graph is called a task dependency graph.

2.4. Dr.N.Sairam & Dr.R.Seethalakshmi School of Computing , Need for Parallel Computers

Parallel processing is much faster than sequential processing when it comes to doing repetitive calculations on vast amount of data. This is because a parallel processor is capable of multithreading on a large scale, and can therefore simultaneously process several streams of data. The one advantage of parallel processing is that it is much faster for simple, repetitive calculations on vast amounts of similar data. If a difficult computational problem needs to be attacked, where the execution time of program code must be reduced, parallel processing may be useful. Some (not all) problems may be subdivided into pieces. If so, the pieces may be simultaneously processed by multiple processing units.

2.5.Developing Parallel Programs —A Discussion of Popular Models, 2012

It discusses emerging issues and key trends in the area of parallel software development, and examines several common approaches to the process of parallel programming. Common industry frameworks, APIs, and standards, such as Open MultiProcessing (OpenMP), Intel® Threading Building Blocks, Apache Hadoop, Message Passing Interface (MPI), and Apple's Grand Central Dispatch, are described and illustrated using a typical programming example. This white paper explores the outlook for parallel software development and focuses in particular on application development for shared memory multicore systems. Two real-world application case studies are introduced to highlight some challenging parallel program design issues. Important tools critical to parallel software design phases also are discussed along with the pros and cons of each tool.

2.6. Justin R. Smith, The Design and Analysis of Parallel Algorithms

Parallel processing algorithms is a very broad field — in this introduction we will try to give some kind of overview. Certain applications of computers require much more processing power than can be provided by today's machines. These applications include solving differential equations and some areas of artificial intelligence like image processing. Efforts to increase the power of sequential computers by making circuit elements smaller and faster have approached basic physical limits. Consequently, it appears that substantial increases in processing power can only come about by somehow breaking up a task and having processors work on the parts independently. The parallel approach to problem-solving is sometimes very different from the sequential one — we will occasionally give examples of parallel algorithms that convey the flavor of this approach.

2.7. DEWAYNE E. PERRY, HARVEY P. SIY, LAWRENCE G. VOTTA, ''Parallel Changes in Large-Scale Software Development: An Observational Case Study"

An essential characteristic of large-scale software development is parallel development by teams of developers. How this parallel development is structured and supported has a profound effect on both the quality and timeliness of the product. We conduct an observational case study in which we collect and analyze the change and configuration management history of a legacy system to delineate the boundaries of, and to understand the nature of, the problems encountered in parallel development. The results of our studies are (1) that the degree of parallelism is very high—higher than considered by tool builders; (2) there are multiple levels of parallelism, and the data for some important aspects are uniform and consistent for all levels; (3) the tails of the distributions are long, indicating the tail, rather than the mean, must receive serious attention in providing solutions for these problems; and (4) there is a significant correlation between the degree of parallel work on a given component and the number of quality problems it has. Thus, the results of this study are important both for tool builders and for process and project engineers.

2.8. Luis Moura e Silvay and Rajkumar Buyyaz, 'Parallel Programming Models and Paradigms'

In the 1980s it was believed computer performance was best improved by creating faster and more efficient processors. This idea was challenged by parallel processing, which in essence means linking together two or more computers to jointly solve a computational problem. Since the early 1990s there has been an increasing trend to move away from expensive and specialized proprietary parallel supercomputers (vector-supercomputers and massively parallel processors) towards networks of computers (PCs/Workstations/SMPs). Among the driving forces that have enabled this transition has been the rapid improvement in the availability of commodity high- performance components for PCs/workstations and networks. These technologies are making a network/cluster of computers an appealing vehicle for cost-effective parallel processing and this is consequently leading to low-cost commodity super- computing. Scalable computing clusters, ranging from a cluster of (homogeneous or heterogeneous) PCs or workstations, to SMPs, are rapidly becoming the standard platforms for high-performance and large-scale computing. The main attractiveness of such systems is that they are built using affordable, low-cost, commodity hardware parallel programming environments. These systems are scalable, i.e., they can be tuned to available budget and computational needs and allow efficient execution of both demanding sequential and parallel applications.

2.9. Konrad Hinsen, Rue Charles Sadron , High-Level Parallel Software “Development with Python and BSP

One of the main obstacles to a more widespread use of parallel computing in science is the difficulty of implementing, testing, and maintaining parallel programs. The combination of a simple parallel computation model, BSP, and a high-level programming language, Python, simplifies these tasks significantly. It allows the rapid development facilities of Python to be applied to parallel programs.

2.10. Parallel Development Strategies for Software Configuration Management

Software project managers routinely face the challenge of developing parallel configurations of software assets. If not identified and planned for in advance, the complexity introduced by parallel development can derail even an otherwise well-managed project. This article describes business situations where parallel development is necessary and examines strategies for configuration management in each situation. "Patterns" are identified which allow different projects with similar characteristics to be managed in a repeatable way, and make it possible to carry forward a body of knowledge and experience from one project to another. Patterns may be combined or nested to deal with complex scenarios. Parallel development potentially needs to be managed at multiple levels within a software asset repository, recognising that different products and components may have their own independent development lifecycles.

2.11. Parallel Programming Environments

To implement a parallel algorithm you need to construct a parallel program. The environment within which parallel programs are constructed is called the parallel programming environment. Programming environments correspond roughly to languages and libraries, as the examples below illustrate -- for example, HPF is a set of extensions to Fortran 90 (a "parallel language", so to speak), while MPI is a library of function calls. There are hundreds of parallel programming environments. To understand them and organize them in a meaningful way, we need to sort them with regard to a classification scheme. In this note, we organize programming environments in terms of their core programming models. This is a complicated way to sort parallel programming environments, since a single programming environment can be classified under more than one programming model (for example, the Linda coordination language can be thought of in terms of a distributed-data-structure model or a coordination model). In this note, the classifications are given, and the programming environments in each class are described in general terms. We then give a very small sampling of the most important programming environments for each category.

2.12. David Rodriguez-Velazquez, Dr. Elise de Doncker " Designing parallel programs", 2009

This paper gave the conclusions that parallel computing is the most common type of tool used to automatically parallelize a serial program into parallel programs. The parallelizing Compiler works in 2 different ways: Fully Automatic and Programmer Directed. The compiler analyzes the source code and identifies opportunities for parallelism. The analysis includes: Identifying inhibitors to parallelism, possibly a cost weighting on whether or not the parallelism would actually improve performance and loops (do, for) loops are the most frequent target for automatic parallelization.

3.Conclusion

The parallel computers perform the tasks parallel and in much lesser time as compared to the serial processing in the serial computers. The parallel execution is fast and saves time and money. There are many different approaches and models of parallel computing. Parallel computing is the future of computing.

References

[1] Blaise Barney , "Introduction to Parallel Computing".

[2] Dr. Umesh Sehgal, Mr. Manoj Kumar "Parallel computing use and their application"

[3] Plamen Krastev, "Introduction to Parallel Computing".

[4] Luis Moura e Silvay and Rajkumar Buyyaz, 'Parallel Programming Models and Paradigms'