CHAPTER 1

INTRODUCTION

Parallel processing is an integral part of everyday life. The concept is so inbuilt in our existence that we benefit from it without realizing. When faced with a tough problem, we involve others to solve it more easily. This co-operation of more than one worker to facilitate the solution of a particular problem may be termed as parallel processing. The goal of parallel processing is thus to solve a given problem more rapidly, or to enable the solution of a problem that would otherwise be impracticable by a single worker. The principles of parallel processing are, however, not new, as evidence suggests that the computational devices used over 2000 years ago by the Greeks recognized and exploited such concepts [4, 8, 10, 13].

In the Nineteenth Century, Babbage used parallel processing in order to improve the performance of his Analytical Engine. Indeed, the first general purpose electronic digital computer, the ENIAC[1] [2,10], was conceived as a highly parallel and decentralized machine with twenty-five independent computing units, co-operating towards the solution of a single problem. However, the early computer developers rapidly identified

[1] ENIAC:-Electronic Numerical Integrator and Calculator, in 1946 marked the beginning of first electronic digital computer

two obstacles restricting the widespread acceptance of parallel machines: the complexity of construction; and, the seemingly high programming effort required. As a result of these early set-backs, the developmental thrust shifted to computers with a single computing unit, to the detriment of parallel designs. Additionally, the availability of sequential machines resulted in the development of algorithms and techniques optimized for these particular architectures [5, 9, 18].

The evolution of serial computers may be finally reaching its peak due to the limitations imposed on the design by its physical implementation and inherent bottlenecks. As users continue to demand improved performance, computer designers have been looking increasingly at parallel approaches to overcome these limitations. All modern computer architectures incorporate a degree of parallelism. Improved hardware design and manufacture coupled with a growing understanding of how to tackle the difficulties of parallel programming has re-established parallel processing at the forefront of computer technology [8, 15].

1.1 Parallel Processing System (Parallel Computer)

A computer system is said to be Parallel Processing System or Parallel Computer if it provides facilities for simultaneous processing of various set of data or simultaneous execution of multiple instruction. On a computer with more than one processor each of several processes can be assigned to its own processor, to allow the processes to progress simultaneously. If only one processor is available the effect of parallel

processing can be simulated by having the processor run each process in turn for a short time. Parallel processing in multiprocessor computer is said to be true parallel processing and parallel processing in uniprocessor computer is said to simulated or virtual parallel processing [14, 16, 17]. We can easily understand the difference between true and virtual parallel processing by the following figures-

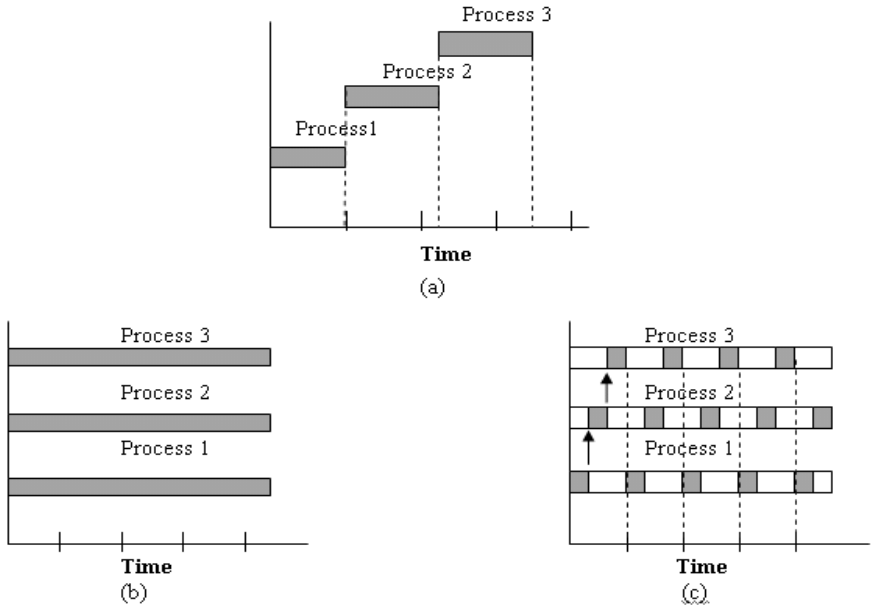

Figure 1.1 (a) Serial processing (b) True parallel processing with multiple processor (c) Parallel processing simulated by switching

Fig 1.1 (a) represents the serial processing means next processing is started when the previous process must be completed. In fig (b) all three process are running in one clock cycle of three processors. In fig (c) all three process are also running in one clock cycle but each process are getting only 1/3 of actual clock cycle on each clock cycle and the CPU is switching from on process to other in its clock cycle. When one process is running all other process must wait for their turn. If we see in fig (b) then we will find that at any particular point only one process is running and

other are waiting. But for one clock time all three process are running. So in uniprocessor system the parallel processing is called virtual parallel processing.

1.1.1 Parallel Processing - Why

Traditionally, software has been written for serial computation: which is executed by a single computer having a single Central Processing Unit (CPU). Problems are solved by a series of instructions, executed one after the other by the CPU. Only one instruction may be executed at any moment in time. Parallel computing is an evolution of serial computing that attempts to copy what has always been the state of affairs in the natural world: many complex, interrelated events happening at the same time, yet within a sequence that require faster processing i.e. Image and Signal Processing, Weather and climate report, Database and Data Mining, Chemical and nuclear reactions, Mechanical devices - from prosthetics to spacecraft Electronic circuits.

A. There are two primary reasons for using parallel computing:

- Save time - wall clock time
- Solve larger problems

B. Other reasons might include:

- **Taking advantage of non-local resources -** using available compute resources on a wide area network, or even the Internet when local compute resources are scarce.
- **Cost savings -** using multiple "cheap" computing resources instead of paying for time on a supercomputer.

- **Overcoming memory constraints -** single computers have very finite memory resources. For large problems, using the memories of multiple computers may overcome this obstacle.

C. Limits to serial computing - both physical and practical reasons cause significant constraints to simply building ever faster serial computers:

- **Transmission speeds -** the speed of a serial computer is directly dependent upon how fast data can move through hardware. Absolute limits are the speed of light (30 cm/nanosecond) and the transmission limit of copper wire (9 cm/nanosecond). Increasing speeds necessitate increasing proximity of processing elements.
- **Limits to efficiency -** processor technology is allowing an increasing number of transistors to be placed on a chip. However, even with molecular or atomic-level components, a limit will be reached on how small components can be.
- **Economic limitations -** it is increasingly expensive to make a single processor faster. Using a larger number of moderately fast commodity processors to achieve the same (or better) performance is less expensive.

1.1.2 Hardware Architecture of Parallel Computer

The core element of parallel processing is CPUs. The essential computing process is the execution of sequence of instruction on asset of data. The term stream is used here to denote a sequence of items as executed by single processor or multiprocessor. Based on a number of

instruction and data streams can be processed simultaneously, Flyn's classified the parallel computer system into following categories [4, 7, 10, 14]:

- Single Instruction Single Data (SISD)
- Single Instruction Multiple Data (SIMD)
- Multiple Instruction Single Data (MISD)
- Multiple Instruction Multiple Data (MIMD)

Single Instruction Single Data (SISD)

The single processing element executes instructions sequentially on a single data stream. The operations are thus ordered in time and may be easily traced from start to finish. Modern adaptations of this uniprocessor use some form of pipelining technique to improve performance and, as demonstrated by the Cray supercomputers, minimise the length of the component interconnections to reduce signal propagation times

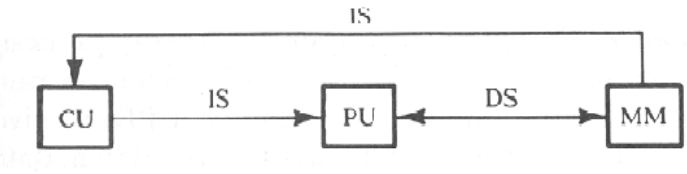

Figure 1.2 SISD Processor

Single Instruction Multiple Data (SIMD)

Machines apply a single instruction to a group of data items simultaneously. A master instruction is thus acting over a vector of

related operands. A number of processors, therefore, obey the same instruction in the same cycle and may be said to be executing in strict lock-step. Facilities exist to exclude particular processors from participating in a given instruction cycle.

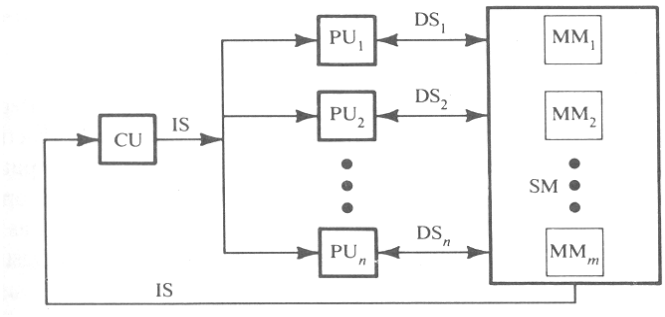

Figure 1.3 SIMD Processor

Multiple Instruction Single Data (MISD)

In this system there are n processor units, each receiving distinct instructions operating over the same data stream. The result of one processor becomes the input of the next processor. One the closest architecture to this concept is a pipelined computer. This structure has received much less attention and has no real implementation.

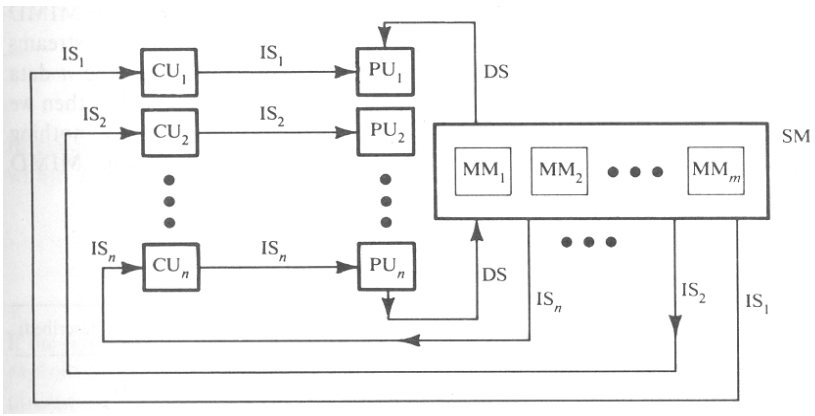

Figure 1.4 MISD Processor

Multiple Instruction Multiple Data (MIMD)

MIMD systems provide a separate set of instructions for each processor. This allows the processors to work on different parts of a problem asynchronously and independently. Such systems may consist of a number of interconnected, dedicated processor and memory nodes, or interconnected "stand-alone" workstations. The processors within the MIMD classification autonomously obey their own instruction sequence and apply these instructions to their own data. By providing these processors with the ability to communicate with each other, they may interact and therefore, co-operate in the solution of a single problem.

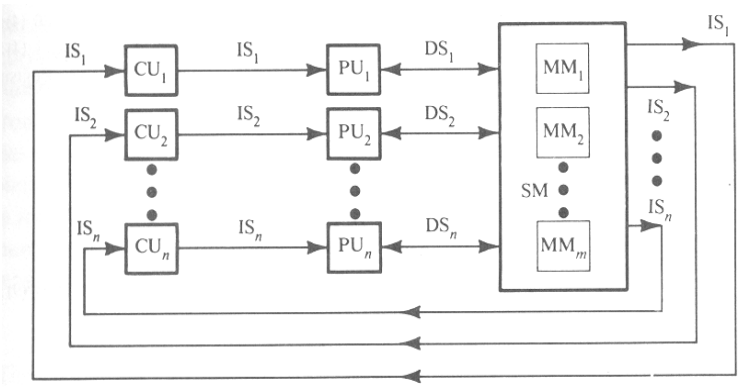

Figure 1.5 MIMD Processor

1.1.3 Memory Architecture of Parallel Computer

The primary memory architectures are:

- Shared memory
- Distributed memory
- Hybrid Distributed-Shared memory

1.1.3.1 Shared memory

In shared memory architecture multiple processors operate independently but share the same memory resources. Only one processor can access the shared memory location at a time.

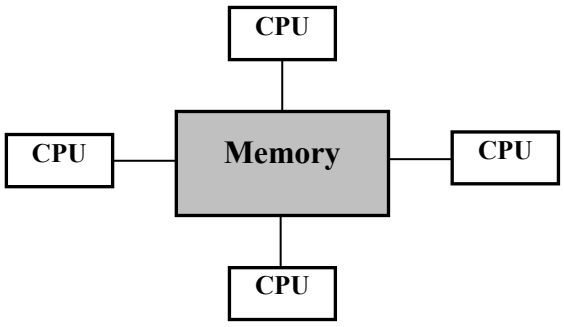

Figure 1.6 Shared Memory Architecture

Changes in a memory location effected by one processor are visible to all other processors. Synchronization is achieved by controlling tasks reading from and writing to the shared memory. Shared memory machines can be divided into two main classes based upon memory access times: **UMA** and **NUMA**.

Non-Uniform Memory Access (NUMA)

Non-Uniform Memory Access (NUMA) is designed to take the best attributes of MPP and SMP systems. NUMA based machines can be extremely cost effective and scalable while preserving the semantics of a shared memory Symmetric Multiprocessor: the NUMA architecture enables users to preserve their investments in SMP applications. NUMA is a means of implementing a

distributed, shared memory system that can make processor/memory interaction appear transparent to application software [2, 11, 19].

In a NUMA machine, physical memory is distributed amongst the computing nodes so that a small portion of the total machine memory is local to each. Since this is shared memory architecture, the remaining machine memory may be remotely accessed across an interconnection network. Local accesses have roughly the same latency as accesses in a SMP system, while remote data accesses can be orders of magnitude slower; hence the term "Non-Uniform Memory Access".

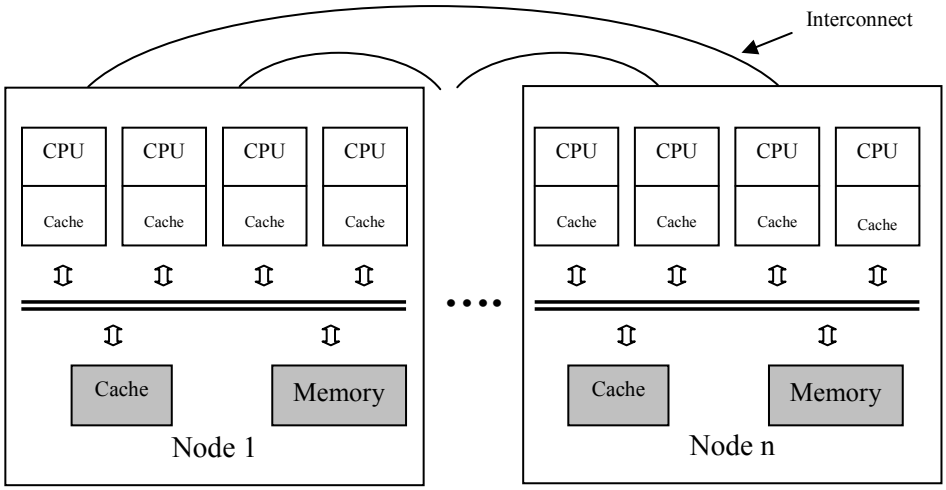

Figure 1.7 NUMA architecture

NUMA system is composed of nodes that are linked together via a fast interconnect, with each node containing small number of processors, its own bus, and its own physical memory. In most NUMA architectures each node has not only its own local memory, but also a local cache for storing local copies of recently accessed data that natively resides on other nodes (Figure 1.7). At the

hardware level, NUMA introduces the concept of “local memory” (which is memory that physically resides on that node) and “remote memory” (which is memory that physically resides on other nodes). To reduce the latency of accesses to both local and remotely held data, caches are used. However, this introduces the problem of keeping cached copies of data coherent when one node in the system modifies a data item. This task may fall to the memory system hardware: “Cache Coherent”-NUMA (cc-NUMA).

Uniform Memory Access (UMA)

Uniform Memory Access (UMA) is a shared memory architecture used in parallel computers. All the processors in the UMA model share the physical memory uniformly.

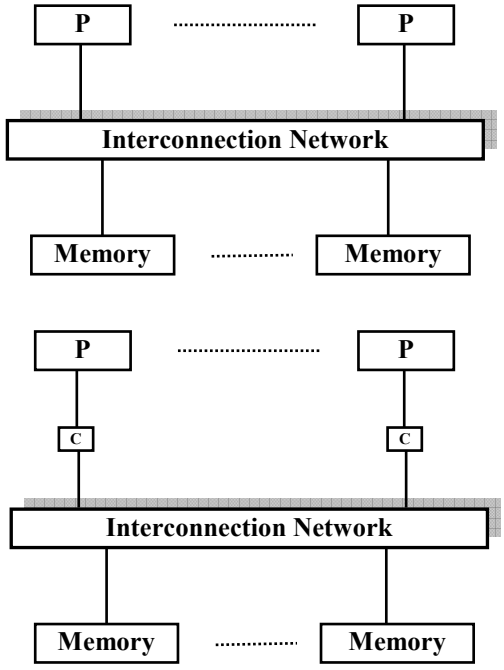

Figure 1.8 UMA architecture

In UMA architecture, access time to a memory location is independent of which processor makes the request or which memory chip contains the transferred data [2, 14, 15]. In the UMA architecture, each processor may use a private cache. Peripherals are also shared in some fashion; The UMA model is suitable for general purpose and time sharing applications by multiple users. It can be used to speed up the execution of a single large program in time critical applications.

Cache only Memory Architecture (COMA)

It is Just like as NUMA but distributed memory is converted to caches. There is no memory hierarchy at each processor node. All the caches form a global address space. In NUMA, each address in the global address space is typically assigned a fixed home node. When processors access some data, a copy is made in their local cache, but space remains allocated in the home node. Instead, with COMA, there is no home [2, 11, 14]. An access from a remote node may cause that data to migrate.

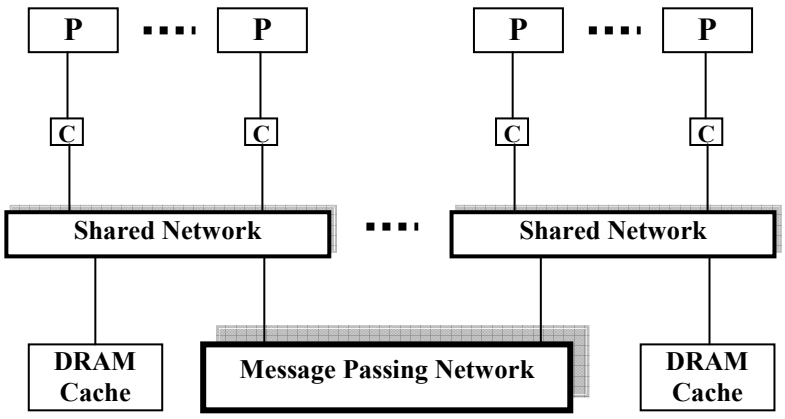

Figure 1.9 COMA architecture

Compared to NUMA, this reduces the number of redundant copies and may allow more efficient use of the memory resources. On the other hand, it raises problems of how to find a particular data and what to do if a local memory fills up. Hardware memory coherence mechanisms are typically used to implement the migration. When all copies are kept consistent in COMA model then it is **called Cache-Coherent COMA.**

1.1.3.2 Distributed 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.

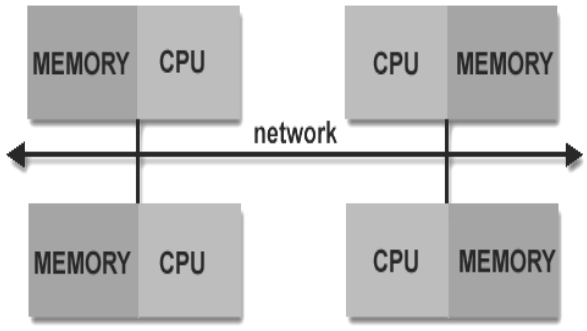

Figure 1.10 Distributed Memory Architecture

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.

1.1.3.3 Hybrid Distributed-Shared Memory

The largest and fastest computers in the world today employ both shared and distributed memory architectures. The shared memory component is usually a cache coherent SMP[2] machine. Processors on a given SMP can address that machine's memory as global. The distributed memory component is the networking of multiple SMPs. SMPs know only about their own memory - not the memory on another SMP. Therefore, network communications are required to move data from one SMP to another.

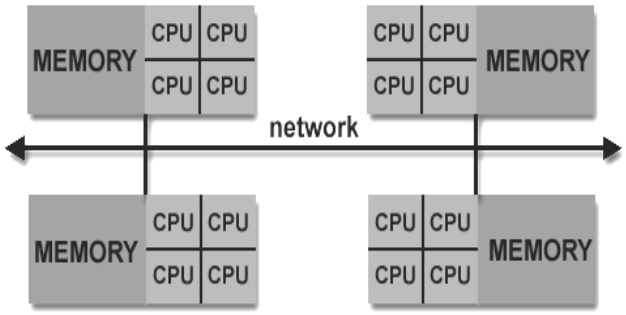

Figure 1.11 Hybrid Distributed-Shared Memory Architecture

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.

1.1.4 Approaches to Parallel Programming

[2] Systematic Multiprocessor

A sequential programming is that which runs on a single processor and has a single line of control. To make many processors collectively work on a single process, the program must be divided in to smaller independent parts so that each processor can work on each part of the problem [3, 5]. Generally a parallel programming involves:

- Decomposing an algorithm or data into parts
- Distributing the parts as tasks which are worked on by multiple processors simultaneously
- Coordinating work and communications of those processors

In case of decomposition of algorithm divide and conquer technique is used to split the data into multiple sets and each set are processed on different processor. This technique is used in SIMD processor. The second technique is used in MIMD processor. A parallel programming can be considered by

- Type of parallel architecture being used
- Type of processor communications used

1.2 Motivation

With the growing demand for performance and speed, parallel processing has become a gradually more powerful technology, exploited both in hardware and software. One of the most important classes of parallel computers is based on the shared memory abstraction, which consists of

DSM[3], VSM[4], and SVM[5]. It employ a single address space, shared memory is easier to use in most applications in comparison to other communication mechanisms such as message passing. In recent years, many shared memory multiprocessors with aggressive memory architectures have been developed.

1.2.1 Perfection of DSM

One of the solutions to parallel systems is Distributed Shared Memory (DSM) whose memory is physically distributed but logically shared. DSM appears as shared memory to the applications programmer, but relies on message passing between independent CPUs to access the global virtual address space. The advantages of DSM programming model are well known. Firstly, DSM programs are usually shorter and easier to understand than equivalent message passing programs [6, 7, 18]. Secondly, DSM gives transparent communication between processes to process.

1.2.2 Need and Importance of Structural Memory Model

The consistency model of a DSM system specifies the ordering constraints on concurrent memory accesses by multiple processors. Whereas memory coherency of DSM system define the location based objective. It gives the way by which a processor becomes not able to read a wrong value which is written by another processor at the same

[3] Distributed Shared Memory
[4] Virtual Shared Memory
[5] Shared Virtual Memory

location [6, 12, 18]. A memory consistency model (MCM[6]) can be defined as the combination of an event ordering mechanism and coherency protocol.

The memory consistency model of DSM system can be combined in two ways; one Uniform memory model[7] and the other is Hybrid Model[8]. Uniform memory model is the group of memory consistency models which depends upon only read and write operation. Where as the Hybrid memory model not only depend upon read and write operation but also depend upon synchronization of process. In comparison to Hybrid Model, the Uniform memory model is stricter and follows the strong ordering of operation. As the uniform and hybrid memory model are different entity and defining different sets of memory model, so a structural memory model is necessary to define the entire consistency models on a unified framework.

1.2.3 Memory Consistency Maintenance

Lots of memory models for DSM architecture are designed so for, thus for designing a good DSM architecture; selection of perfect memory model as well as its proper maintenance is required. The maintenance of memory consistency can be done with the help of coherency protocol.

1.2.4 Improvisation of Parallelism

[6] Memory Consistency Model or simply Memory Model
[7] Group of Uniform Memory Consistency Model
[8] Group of Hybrid Memory Consistency Model

The performance of parallel computer is measured by the degree of parallelism. The parallelism in DSM architecture can be improved by replicating the data. The more replication of data increases more parallelism. But for the replication of data maintenance of memory consistency and proper data replication algorithm is required.

1.2.5 Memory Management Technique

Implementing DSM system with paging scheme leads to false sharing and high cost associated with virtual memory operation. If homogeneous page system is taken then there is no problem of false-sharing but heterogeneous page system which supports different virtual memory page size may cause false sharing and thrashing. So choosing a perfect page size becomes impotent issue.

1.3 Context

After analyzing the memory architecture of parallel computer we found that the DSM architecture is well suitable and acceptable because of its simple architecture and ease of programming. So we analyze the DSM architecture in detail and found that it depends upon two things:

- Memory Consistency and
- Memory Coherency

As a lot of memory consistency is designed so for. Some are classified under Uniform memory model and some are in Hybrid Model. So we study both the models in details. Apart from memory consistency

we study memory coherency in details. The thesis is designed in context with the following problems arises during our analysis of DSM architecture:

- ✓ The memory consistency models for uniform as well as hybrid model are not defined on a unified framework they are defined in individual manner.
- ✓ The models of memory consistency are also not structured.
- ✓ How the memory consistency and the coherency can be related to each other.
- ✓ Which consistency model is prefect for DSM system?
- ✓ Verification of consistency model
- ✓ How can the memory consistency maintained in DSM system with memory coherence.
- ✓ Is there is any possibility of data replication in DSM system if memory consistency is maintained.
- ✓ If the data replication is possible in DSM system then there is any possibility of improvement of parallelism
- ✓ And last the pages-size analysis

The selection of Distributed shared memory systems for our research work is due to three reasons. First, they provide for the exploitation of locality; the distribution of memory allows for the distinction between local and remote data. Second, they provide a convenient programming model; the presence of a shared name space simplifies programming. Finally, many of them provide a portable

interface for parallel programming. The contents of the thesis describe the identification and solution of the above problem. For analyzing the relationship among all memory consistency models a Venn-diagram like representation is designed. For maintenance of memory consistency a framework is designed which is based on memory coherency protocol.

1.4 Thesis Organization and Overview

The brief description of our thesis in chapter wise is as follows:-

Chapter 2

Chapter 2 overviews the basic concept of Distributed Shared Memory. It defines Implementation, Classification and challenges of Distributed shared memory. It also defines the consistency and coherency issues of DSM system. Finally it describes current implementation of DSM system and brief description of our approach.

Chapter 3

Chapter 3 proposes the Structural Uniform memory model. It investigates the issue involved in the different models of uniform memory models. It relates and unifies all the uniform memory models with rectangular representation. All the models are defined and verified on the basis of execution history.

Chapter 4

Chapter 4 proposes the design and development of structural memory model of Hybrid memory model and investigates the issue involved in the different models of hybrid memory model. Finally it relates and unifies the entire hybrid memory model with Venn-like representation. All the models are defined and verified on the basis of execution history.

Chapter 5

Chapter 5 proposes a framework for maintaining the memory consistency with the help of memory coherence. The chapter describes the designing issue of framework in detail. The working and development of framework is also discussed. Finally it discuss about the implementation of framework.

Chapter 6

Chapter 6 extends the framework described in chapter 5. It describes how the parallelism of proposed DSM system can be improved. It discusses the various algorithms developed for parallelism improvement. Finally it discusses the degree of parallelism and performance in detail.

Chapter 7

Chapter 7 describes paging technique in details. It describes granularity, thrashing and false sharing in detail. It also gives the solution of false sharing and thrashing. The chapter gives a complete analysis of different overheads of the proposed DSM framework and

finally it discusses about the calculation the page size based on network communication overhead.

Chapter 8

Chapter 8 concludes the thesis with summary, observation of work, limitation, and suggest about the future research.

1.5 Summary of Contribution

The thesis analyzes both hardware and software implementations of DSM. A comparative analysis of existing DSM architecture is done; on a result similar point and distinct points of all architectures come to one platform. The thesis presents a framework that implement the memory model as well as memory coherence. The major contributions of the thesis are following:

- Using structural memory model of uniform/hybrid memory model the memory model can be defined on the basis of execution history. They can also be defined on unified framework.

- Comparative analyses of all the uniforms models and hybrid memory model have been done so that they can be related with each other. A Venn-diagram relation for uniform and hybrid model is also developed to analysis the relationship between all the models. This analysis gives the idea for selecting the perfect memory consistency model with respect to different DSM architecture.

- A framework is developed for DSM architecture for the proper maintenance of memory consistency. The framework is designed on the basis of memory coherency protocol. For memory coherence protocol Write-update coherence protocol is taken.

- The framework helps in replicating the data in DSM architecture with the help of Multiple Read Multiple Write (MRMW). Due to this the parallelism of DSM architecture can be improved.

- The thesis also analysis the performance cost of DSM architecture with different parameter like number of read/write operation number of processors.

- The thesis gives a memory management technique for paging system of DSM architecture based N/W communication overhead. The thesis tells the solution of false sharing and granularity. At last a technique for calculating page-size for DSM architecture is also given.

1.6 Research Methodology

The whole research is divided in to five parts:

1. Problem identification
2. Analysis of problems
3. Selecting the appropriate problem
4. Finding the solution

5. Verification and implementation of solution

The above five organization includes a general definition or some type of overview of the approach we used in conducting our research. Next, we needed to present a thorough description of how we will go about collecting the necessary data as well as the analytical procedure; we will use to draw conclusions based on this information. In this regard conference paper, journal paper and magazines were very useful to know about the new things. Statistical method such as correlation is also used for correlation of data. Technology such as Parallel Computing toolbox in MATLab is also used for simulating the framework proposed. Apart from this Internet was very useful source for our research work.

REFERENCE

[1]. Babak Falsafi and David A. Wood (1997), ***"Reactive NUMA: A Design for Unifying S-COMA and CC-NUMA"***, International Symposium on Computer Architecture, **ISCA-97**, pp 2-6.

[2]. Arthur w. Burks (1947), ***"Electronic Computing Circuit of the ENIAC"***, Proceeding of the institute of Radio Engineers Vol. 35 No. 8, pp 1-3.

[3]. Barry Wilkinson and Michael Allen (2009), ***"Parallel Programming: Techniques and Application using Networked Workstations and Parallel Computer"***, Second Edition, Pearson Education South Asia, pp 3-37.

[4]. Behrooz Parhami (2002), ***"Introduction to Parallel Processing Algorithms and Architectures"***, Kluwer Academic Publishers.

[5]. Bernd Mohr (2006), ***"Introduction to Parallel Computing"***, Computational Nan science: Do It Yourself, NIC Series, Vol. 31, pp 491-505.

[6]. Bill Nitzherg and Virginia Lo (1991), ***"Distributed Shared Memory: A Survey of Issues and Algorithm"***, 0018-9162/91/0806-0052501 IBM, **IEEE**, pp 52-58.

[7]. Changhun Lee (2002), ***"Distributed Shared Memory"*** Proceedings on the 15th CISL Winter Workshop, Kushu, Japan, February.

[8]. David A. Wood and Mark D. Hill (1995), ***"Cost Effective Parallel Computing"***, Computer Practices, **IEEE**, pp 69-72.

[9]. Harry F. Jordan and Gita Alaghband (2007), ***"Fundamental of Parallel Processing"***, First Impression, Pearson Education pp 8-13.

[10]. http://en.wikipedia.org/wiki/History_of_computing_hardware, ***"First Generation Machine"***.

[11]. https://computing.llnl.gov/tutorials/parallel_comp, ***"Introduction to Parallel Computing"*** Blaise Barney, Lawrence Livermore National Laboratory.

[12]. Jelica Protic, Milo Tomasevic and Veljko Milartinovic (1996), ***"Distributed Shared Memory: Concepts and System***", Summer **IEEE** Parallel and Distributed Technologies, pp 63-79.

[13]. Jenny Mankin (2007), ***"Parallel Computing Memory Consistency Models: A Survey in Past and Present Research"***, CSG'07, pp 2-7.

[14]. John E. Bentley (2000), ***"An introduction to Parallel Computing"***, System Architecture.

[15]. Kai Hwang (2008), ***"Advance Computer Architecture: Parallelism, Scalability, Programmability"***, Third Edition, Tata McGrawhill, pp 9-14.

[16]. Ralph Dunca (1990), ***"A Survey of Parallel Computer Architectures"***, Survey & Tutorial Series, **IEEE** Feb, pp 5-15.

[17]. Roberto Hoyos (2006), ***"Inside Parallel Computing: Present and Future"***, Emerging Information Technologies, **ITSEM,** pp 1-8.

[18]. S. K. Ghosal (2000), ***"A Practical Approach to Parallel Computing"***, First Edition, University Press Hyderabad, pp 1-26.

[19]. Varun Chandola (1999), ***"Design Issues in Implementation of Distributed Shared Memory in User Space"***.

[20]. Z. Vranesic, S. Brown, M. Stumm, S. Caranci, A. Grbic, R. Grindley, M. Gusat, O. Krieger, G. Lemieux, K. Loveless, N. Manjikian Z. Zilic, T. Abdelrahman, B. Gamsa, P. Pereira, K. Sevcik, A. Elkateeb and S. Srbljic (1995), ***"The NUMAchine Multiprocessor"***, Technical Report, , pp 4 - 8.