Efficient Load Balancing Algorithms for Parallel and Distributed Systems

A Thesis

Submitted in the partial fulfillment of the
requirements for the award of the degree of

Doctor of Philosophy

Submitted By

Jahangir Alam

(Registration No. 90603504)

Under Supervision and Guidance of

Dr. Rajesh Kumar

(Professor, CSED, TU Patiala)

Department of Computer Science & Engineering

Thapar University,

Patiala - 147004 (Punjab), India.

July, 2016

Certificate

I hereby certify that the work which is being presented in this thesis entitled **EFFICIENT LOAD BALANCING ALGORITHMS FOR PARALLEL AND DISTRIBUTED SYSTEMS**, in fulfilment of the requirements for the award of degree of **DOCTOR OF PHILOSOPHY** submitted in the Department of Computer Science and Engineering, Thapar University, Patiala, is an authentic record of my own work carried out under the supervision and guidance of **Dr. Rajesh Kumar**, and refers the work of other researchers, which are duly listed in the reference section.

The matter presented in this thesis has not been submitted for the award of any other degree of this or any other university.

Jahangir 8/4/6/2016

Jahangir Alam

Registration No. 90603504

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge and belief.

4/06/2016

Dr. Rajesh Kumar

Professor

Department of Computer Science and Engineering

Thapar University, Patiala, Punjab - 147004 (INDIA)

Supervisor

Dedicated to

My Parents, Wife and Kids

Abstract

Parallel and Distributed computing is essential for modern research as the demand for more and more computing power is continuously increasing. A number of aspects within parallel and distributed computing have been explored in recent years, however two of the most pertinent issues relate to the design of scalable interconnection network topology and task allocation for load balancing. This work addresses these two problems.

The first part of the thesis presents the design and evaluation of a highly scalable and economical interconnection network topology known as the STH (Serially Twisted Hypercube). It begins with a brief survey of various interconnection network topologies proposed in literature followed by the design principles on which the proposed STH interconnection network will be based upon. Various properties of the proposed topology are derived and then compared with other topologies on a number of interconnection networks evaluation parameters.

The second part of the thesis deals with the task allocation problem taking into account load balancing. Exploiting the full potential of a parallel and distributed system requires efficient allocation of the program tasks to the diversely capable machines within the system. If the task allocation strategy is not properly framed, machines in the system may spend most of their time waiting for each other instead of performing useful computations. This part of the thesis focuses on static task allocation considering load balancing in heterogeneous distributed computing systems sometimes also referred to as heterogeneous multicomputer systems (HMS). The thesis first presents a brief literature survey on the proposed solutions and then classifies them according to the solution techniques. It then presents two mathematical models based on fuzzy logic to solve the task allocation problem. Finally based on the proposed models the the thesis presents two algorithms to solve the aforementioned problem. The algorithms are coded in C and the proposed models are verified by executing the corresponding programs with several sets of randomly generated data. Experimental results prove that the algorithms successfully allocate tasks to the machines whilst balancing the allocated task load.

Analysis of the proposed method also proves that its complexity is low compared to similar existing approaches.

Declaration

The work in this thesis is based on research jointly carried out at University Women's Polytechnic, Faculty of Engineering & Technology, Aligarh Muslim University, Aligarh (India) and Department of Computer Science & Engineering, Thapar University Patiala, Patiala (India). No part of this thesis has been submitted elsewhere for any other degree or qualification and it is all my own work unless referenced to the contrary in the text.

Acknowledgements

First of all, I thank ALLAH (SWT), the Lord Almighty, for giving me the health, strength, ability to complete this work and for blessing me with supportive supervisor, family and friends.

I wish to express my deepest appreciation to my supervisor, Dr. Rajesh Kumar for his idea, support, enthusiasm, and patience. I have learnt an enormous amount from working with him.

I would also like to thank the Hon'ble Director, Thapar University for giving me the opportunity to attend research programme at Thapar University, Patiala.

I would like to express my profound gratitude to my doctoral committee members: Prof. Rakesh Sharma, Dr. Maninder Singh, Dr. Deepak Garg and Dr. Rajesh Kumar for their enlightening suggestions from time to time.

I am forever indebted to my employer Aligarh Muslim University (India) and to Dr. Salma Shaheen, Principal, University Women's Polytechnic, AMU, Aligarh for extending me the facilities required to complete this work.

I express my sincere thanks to Dr. Mohhmad Athar Ali, Associate Professor, Dept. of Computer Engg., AMU Aligarh for English corrections, which helped improve all my writings.

Life would be harder without the support of many good relatives and friends. Thanks to Prof. Shamim Ahmad (for being such a wonderful mentor), Mr. Farrukh Shamim, Dr. Abdus Samad, Dr. Parveen Beg, Mr. M. Abdul Qadeer, Mr. Misbahurrahman Siddiqui, Mr. Tariq Ahmed, Mr. Wajid Ali, Mr. Rahat Mahmood, Mr. Gulsanover and Mr. Mohd. Hanzala.

Thank you to all my friends at AMU Aligarh and Thapar University, Patiala.

Finally, I thank to all my family members for their love, patience and uncountable supports.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

The speed of silicon based CPUs is reaching its physical limits as they are constrained by the speed of electricity, light and certain thermodynamics laws. To solve complex and large problems the viable solution to this limitation is to make use of multiple connected CPUs working in coordination with each other. Clearly high performance computing requires the use of Massively Parallel Systems containing thousands of powerful processors connected in some prescribed fashion.

This chapter introduces the context within which research was undertaken as part of this thesis. It starts with introducing the general idea of parallel processing. It then presents an overview of various parallel and distributed Systems. Next, it discusses the significance of interconnection network topologies, task allocation and load balancing within systems under consideration. The chapter then presents the primary contributions of this research and ends with a discussion on the organization of the rest of the thesis.

1.1 Parallel Processing

According to Buyya [60], processing of multiple tasks simultaneously on multiple processors is called parallel processing. A single large program is divided into multiple tasks (active processes) using a *divide-and-conquer* technique and each one of them is processed on a different CPU. Programming on multiprocessor systems

using a divide-and-conquer technique is referred to as *parallel programming*.

1.2 Need for Parallel Processing

Until only a few years ago, it was widely believed that single-CPU clock rates could be doubled every 18 months in accordance to Moore's Law [165]. However, recent advances in processor fabrication technology have begun to push the limit of this law to the extent that under certain conditions, Moores law has been violated [26]. Despite these advances, today's applications require more computing power than a sequential computer can offer. By adding more CPUs and an effective communication system between them, parallel processing provides a cost-effective solution to this problem. The evolution of parallel processing has been influenced by many factors. The most prominent among them include:

- Computational requirements are ever increasing, both in the area of scientific and business computing. Computing problems that require high speed computational power are extensively employed in areas of architecture, meteorology, life sciences and space etc.

- Sequential architectures are rapidly reaching their physical limit as they are constrained by the speed of light and thermodynamics laws. Hence, an alternative way to ensure high computational speed is to effectively connect multiple processors.

- Hardware innovations in pipelining, superscalar etc., are non-scalable and require sophisticated compiler technology. Developing such a compiler technology is a complex task.

- For certain kind of problems (e.g. massive matrix operations) vector processing works well, however, it is not suitable for generalized applications like databases.

- The technology that makes parallel processing possible has now matured and is commercially viable. A significant amount of research for development of tools and environment for parallel processing architectures has already been done.

- Advances in the area of networking is also paving the way for a more advanced form of parallel computing which is referred to as heterogeneous multicomputers.

1.3 Taxonomy of Computer Architectures

Flynn [109] defined the most popular classification of computer architecture. Quinn [179] provides a good review of Flynn's taxonomy.

On the basis of number of instruction and data streams that can be processed simultaneously, Flynn classified computer systems into following four categories:

1. Single Instruction Single Data (SISD)
2. Single Instruction Multiple Data (SIMD)
3. Multiple Instruction Single Data (MISD)
4. Multiple Instruction Multiple Data (MIMD)

Following subsection briefly discusses each of them:

1.3.1 Single Instruction Single Data (SISD)

Conventional uniprocessor Von Neumann computers are classified as SISD systems. Such machines are capable of executing a single instruction which operates on a single data stream. Figure 1.1 shows a typical SISD machine. In this architecture instructions are processed sequentially. Consequently, machines developed around this architecture are called sequential machines. Most of the machines which are used for general purpose computing are built on the SISD model. The

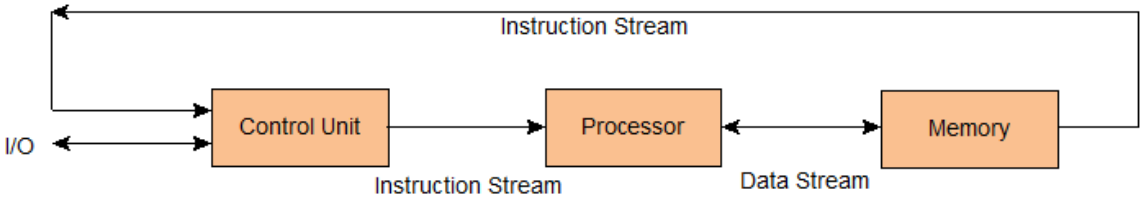

Figure 1.1: SISD architecture

data and instructions to be processed are first stored in primary memory. The speed of the processing element is governed by the rate at which the data transfer can take place internally. Examples of SISD architecture are IBM-PC, Apple Macintosh, general purpose workstations etc.

1.3.2 Single Instruction Multiple Data (SISD)

A multiprocessor machine which executes the same instruction on all processors but operate on different data streams is known as a SIMD machine. A computer

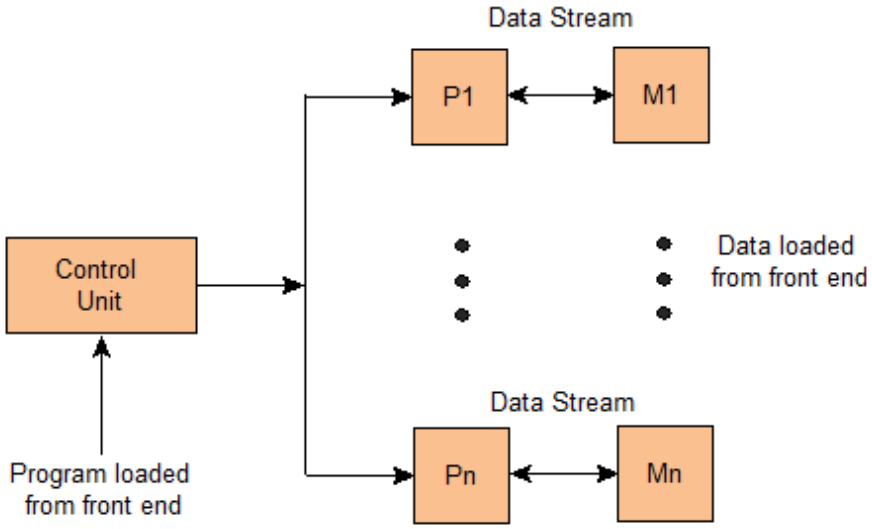

Figure 1.2: SIMD architecture

based on the SIMD model usually has two parts: a front-end computer of the usual Von Neumann style and a processor array as shown in Figure 1.3. The processor is a set of identical synchronized processing elements capable of simultaneously

performing the same operation on different data. Each processor in the array has a small amount of memory where data is stored while they are processed in parallel. Machines based on this architecture are extensively used in scientific computing applications since such applications involve lot of vector and matrix operations. Examples of SIMD based computers are CRAY's VPM (Vector Processing Machine), Thinking Machine's cm* etc.

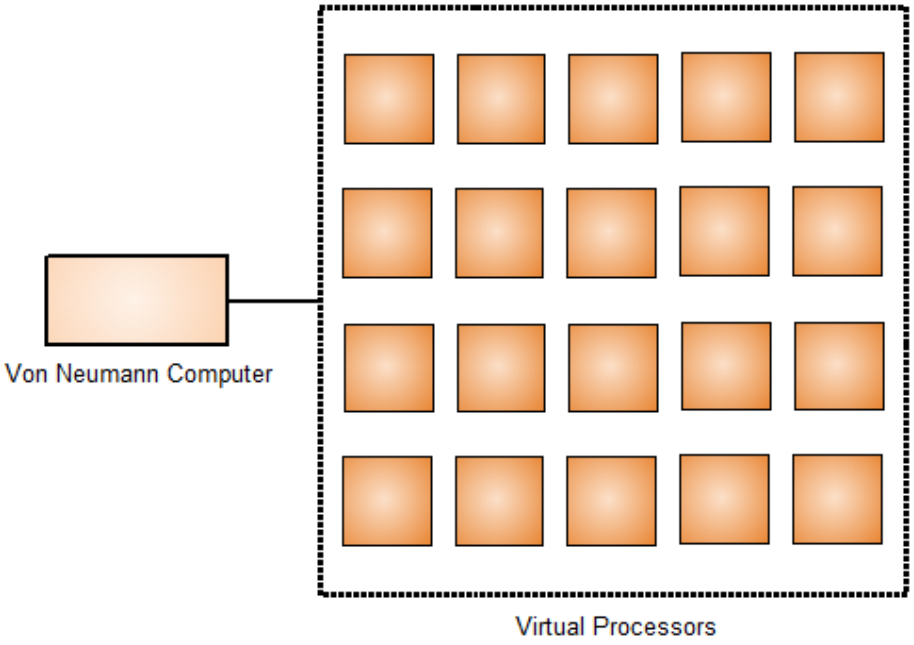

Figure 1.3: SIMD architecture components

1.3.3 Multiple Instruction Single Data (MISD)

A multiprocessor machine which executes different instructions on different processors but all of them operate on the same data set is classified as an MISD machine. Figure 1.4 shows a typical SIMD machine. Machines based on the MISD model are not useful for most applications, however few machines have been built around this architecture but none of them are commercially available. However, some authors like Roosta [183] have considered pipelined machines (and systolic-array computers) as examples of MISD architecture.

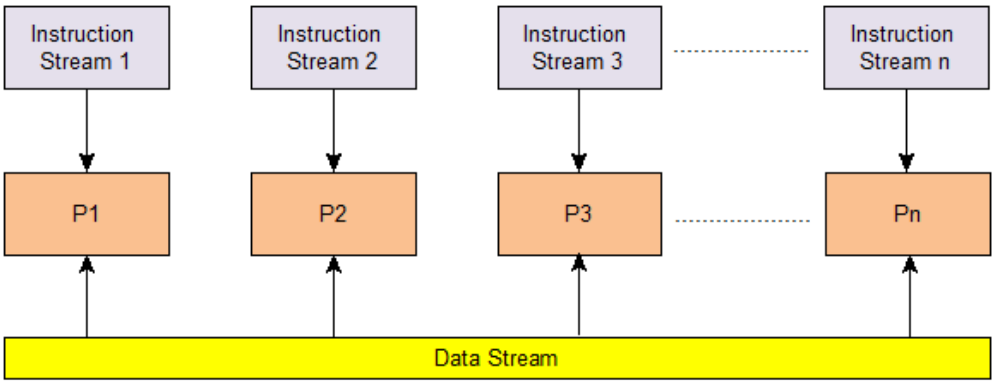

Figure 1.4: MISD architecture

1.3.4 Multiple Instruction Multiple Data (MIMD)

A multiprocessor machine which executes multiple instructions on multiple data sets is categorized as an MIMD machine. Figure 1.5 shows a typical MIMD machine. In this architecture, for each processor, there are separate instructions and

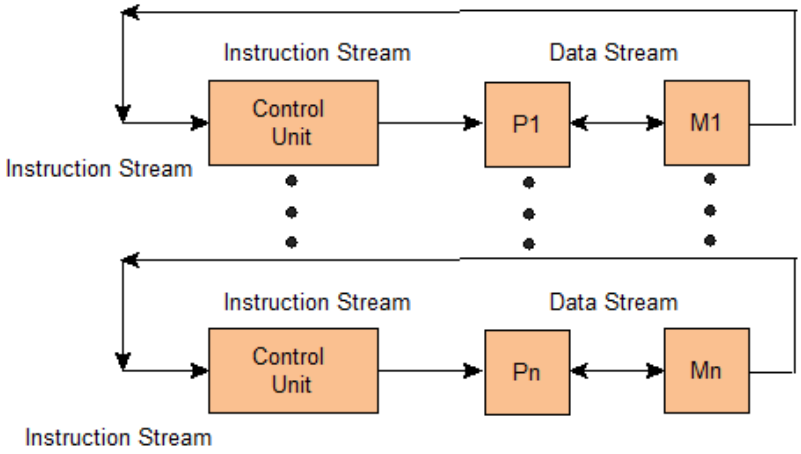

Figure 1.5: MIMD architecture

data stream and hence machines based on this architecture are useful for almost all applications [94] [103].

MIMD architectures are made up of multiple processors and multiple memory modules connected together through an interconnection network. They are classified into two categories: shared memory systems and message passing systems.

Figure 1.6 further elaborates the MMID architecture and Figure 1.7 highlights the distinctions between the two categories of this architecture.

Shared Memory Systems

A shared memory model is one in which processors communicate by reading and writing locations in a shared memory that is equally accessible by all processors. Each processor may have registers, buffers, caches, and local memory banks as additional memory resources [114]. The simplest shared memory system consists of one memory module that can be accessed from two processors. Requests arrive at the memory module through its two ports. An arbitration unit within the memory module passes requests through to a memory controller. If the memory module is not busy and a single request arrives, then the arbitration unit passes that request to the memory controller and the request is granted. The module is then placed in the busy state while a request is being serviced. If a new request arrives while the memory is busy servicing a previous request, the requesting processor may hold its request until the memory becomes free or it may repeat its request later. Depending on the interconnection network, a shared memory based system can be classified as: uniform memory access (UMA), non-uniform memory access (NUMA), and cache-only memory architecture (COMA). In the UMA system, a shared memory is accessible by all processors through an interconnection network in the same way a single processor accesses its memory. Therefore, all processors have an equal access time to any memory location. The interconnection network used in the UMA can be a single bus, multiple buses, a crossbar, or a multiport memory. In the NUMA system, each processor has part of the shared memory attached. The memory has a single address space. Therefore, any processor could access any memory location directly using its real address. However, the access time to modules depends on the distance to the processor. This results in a non-uniform memory access time. A number of architectures are used to interconnect processors to memory modules in a NUMA. Similar to the NUMA, each processor has part of the shared memory in the COMA. However, in this case the

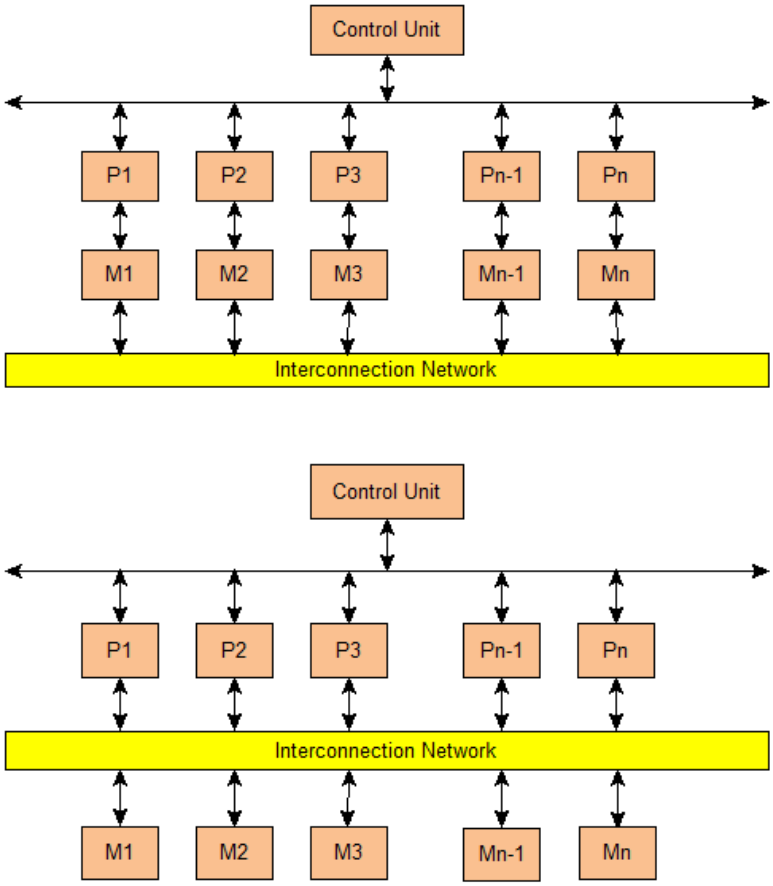

Figure 1.6: MIMD architecture categories -

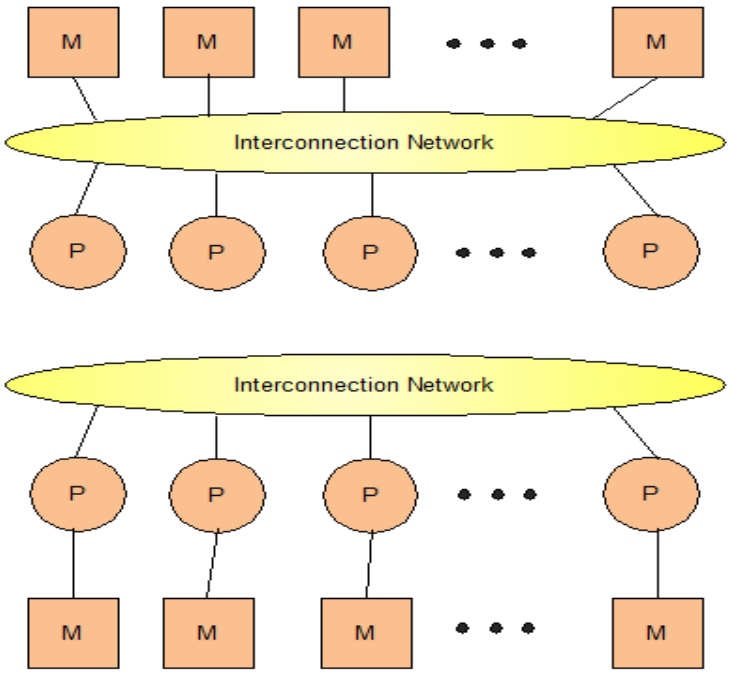

Figure 1.7: MIMD architecture - Shared Memory & Message Passing

shared memory consists of cache memory. A COMA system requires that data be migrated to the processor requesting it.

Commercial examples of shared memory systems are Sequent Computer's Balance and Symmetry, Sun Micro-systems multiprocessor servers, and Silicon Graphics Inc. multiprocessor servers.

Message Passing Systems

Message passing systems are a class of multiprocessors in which each processor has access to its own local memory. Unlike shared memory systems, communications in message passing systems are performed via send and receive operations. A node in such a system consists of a processor and its local memory. Nodes are typically able to store messages in buffers (temporary memory locations where messages wait until they can be sent or received), and perform send/receive operations at the same time as processing [121] [213]. Simultaneous message processing and problem calculation are handled by the underlying operating system. Processors

do not share a global memory and each processor has access to its own address space. The processing units of a message passing system may be connected in a variety of ways ranging from architecture-specific interconnection structures to geographically dispersed networks. The message passing approach is, in principle, scalable to large proportions. By scalable, it is meant that the number of processors can be increased without significant decrease in efficiency of operation.

Message passing multiprocessors employ a variety of static networks in local communication. Of importance are hypercube networks, which have received special attention for many years. The nearest neighbour two-dimensional and three-dimensional mesh networks have been used in message passing systems as well.

Commercial examples of message passing architectures were the nCUBE, iPSC/2 and various Transputer-based systems. These systems eventually gave way to Internet connected systems whereby the processor/memory nodes were either Internet servers or clients on an individual's desktop. These systems are also referred to as parallel and distributed systems (PDS).

An extension to Flynns taxonomy was introduced by Kuck [141]. In his classification, Kuck extended the instruction stream further to single (scalar and array) and multiple (scalar and array) streams. The data stream in Kucks classification is called the execution stream and is also extended to include single (scalar and array) and multiple (scalar and array) streams. The combination of these streams results in a total of 16 categories of architectures.

1.3.5 Hybrid Architecture

With regard to MIMD architecture, it was apparent that distributed memory was the only way to efficiently increase the number of processors managed by a parallel and distributed system. If scalability to larger and larger systems (as measured by the number of processors) was to continue, systems had to use distributed memory techniques. These two factors created a conflict: programming in the shared memory model was easier, and designing systems in the message passing model provided scalability. The distributed-shared memory (DSM) architecture

began to appear in systems like the SGI Origin2000 [101], and others. In such systems, memory is physically distributed; for example, the hardware architecture follows the message passing school of design, but the programming model follows the shared memory school of thought. In effect, software covers up the hardware. As far as a programmer is concerned, the architecture looks and behaves like a shared memory machine, but a message passing architecture lives underneath the software. Thus, the DSM machine is a hybrid that takes advantage of both design schools.

1.4 Parallel and Distributed Systems Addressed in This Thesis

This work proposes a specialized form of Parallel and Distributed Systems wherein several machines (computers) equipped with heterogeneous components (resources) are connected through a high speed message passing interconnection network. Such a system is usually referred to as a "Heterogeneous Multicomputer System (HMS)" [50]. Figure 1.8 depicts such a system. They are based on the hybrid architecture discussed above in Section 1.3.5.

Among several other problems of interest, two widely researched problems related to Parallel and Distributed Systems are the design of an economical and scalable interconnection network topology, and task allocation considering load balancing. This work addresses these two problems.

1.5 Motivations

This research is motivated by the need to design an economical and scalable interconnection network topology, and task allocation policies for load balancing to ensure efficient execution of parallel applications on parallel and distributed systems.The motivating factors behind this research are as follows:

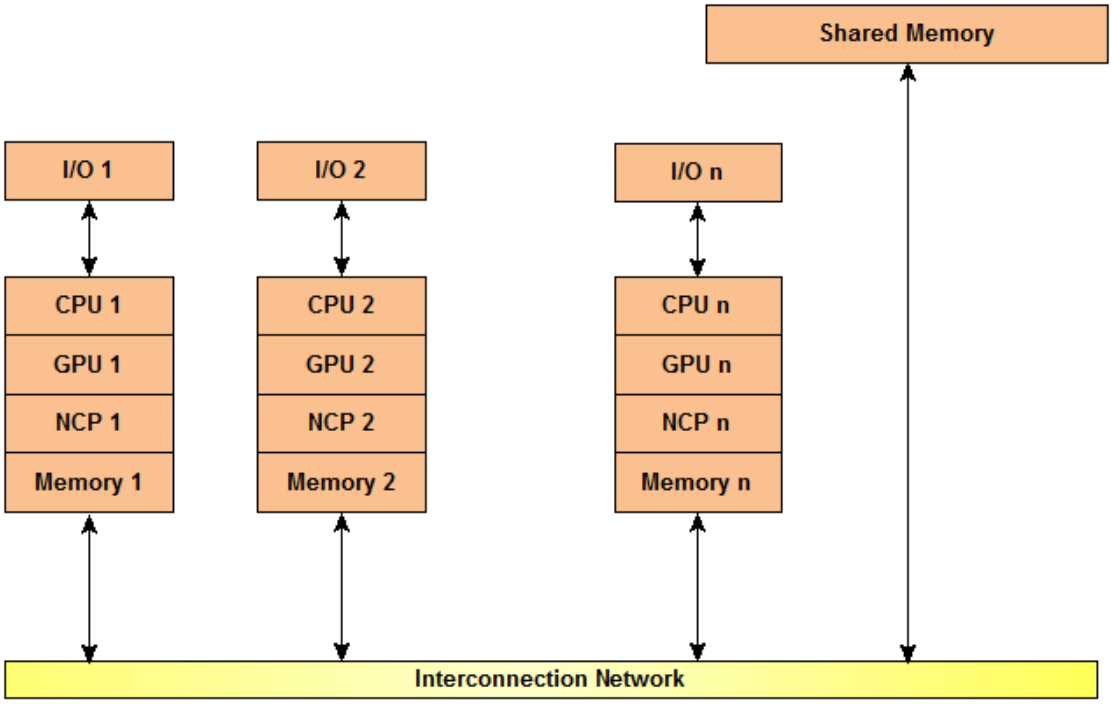

Figure 1.8: Heterogeneous Multicomputer System

1.5.1 Scalable and Economical Interconnection Network

In parallel and distributed systems, each processor has its own local memory and an application's task on separate processors coordinate their activities by sending messages through an interconnection network.

Communication efficiency is one of the most important factors which must be considered when designing a parallel and distributed architecture because it directly effects the efficiency of parallel algorithms [15]. When a message passes between a pair of nodes of a PDS it may be routed through a connected circuit in a number of hops. In addition to that, intermediate processors may be interrupted to store and forward the message, or the message may directly be transferred by the communication links through the connected circuit. Thus, the communication efficiency, apart from routing protocols, processor speed and data link speed greatly depend on the topology of the interconnection network. Moreover, Zhang [239] showed that varying physical parameters in an interconnection network topology significantly affect the performance of a load-balancing strategy, regardless of that

strategy's approach or the network load levels.

Various interconnection network topologies have been proposed and studied in literature [172] [136]. The binary n-cube also known as hypercube (HC), is a well-known topology among researchers as it possesses a number of desirable features of an interconnection network which include high fault-tolerance, simple routing, logarithmic degree and logarithmic diameter. Yet, the major disadvantages of hypercube and its variants are, difficulty with its VLSI layout, Scalability and Modularity [174] [4] [110]. Numerous topologies have been proposed in literature which claim to solve problems associated with a hypercube architecture. They are either hypercube variants or hybrid architectures. Each of these topologies has some advantageous features as well as some inherent limitations. The demand for improving the scalability of the HC at an affordable cost motivates our research towards proposing a new interconnection network.

1.5.2 Static Load Balancing Task Allocation

Task allocation techniques for load balancing may be static or dynamic [101]. The first one static allocation techniques are used when the computational and communication requirements of a problem are known *a priori*. The problem is partitioned into tasks and the assignment of the task-processor is performed once before the parallel application initiates its execution. Clearly these techniques are intended to be executed off-line in a separate mapping step at compile time. The second approach, which is the dynamic allocation scheme, is applied in situations where no prior estimations of load distribution are possible. It is only during the actual program execution that it becomes apparent as to how much load is being assigned to the individual processor. In order to retain efficiency, the imbalance must be detected and an appropriate dynamic allocation strategy must be devised. Some dynamic strategies that use local information in a distributed architecture, have been proposed in literature. These strategies describe rules for migrating tasks from overloaded processors to under loaded processors in the network of a given topology. Dynamic allocation techniques are also referred to as Resource

Sharing, Resource Scheduling, Job Scheduling, Task Migration techniques etc.

In terms of advantages, static allocation techniques are advantageous compared to dynamic allocation techniques, as they have no run time overhead. The overhead of static techniques are incurred at compile time resulting in a more efficient execution time. Dynamic techniques are quite complex and the allocation process may create additional run time overhead that can adversely influence the system performance. In some situations for example those applications which can't be formulated in deterministic manner, dynamic allocation techniques first determine an initial assignment and then apply stepwise refinements to improve the solution. The overhead is multiplied proportionally. In such cases, an initial assignment may be found using a static allocation technique and then a dynamic allocation technique may be employed to improve the solution. The importance of using static allocation followed by a dynamic allocation has been demonstrated by Tindell et al. [207]. A similar approach has been used by Liang et al. [150] for reconfigurable systems. These obvious advantages of static allocation techniques was the motivation behind designing the proposed static allocation models.

1.5.3 Heterogeneous Multicomputer Systems

For simplifying the task allocation models and related algorithms [31], most of the techniques proposed in literature overlooked a number of characteristics that can be considered essential to both, parallel and distributed systems as well as the application. These characteristics are the heterogeneity of the machines, arbitrary structure of the application task graph, different requirements of the tasks with respect to available resources etc. Most of the algorithms and models proposed in literature deal with homogeneous distributed systems. In heterogeneous multicomputer systems, diversely capable machines are connected through an interconnection network. In such systems machines are constrained by factors like processor speed, memory capacity etc., hence application tasks can't be allocated in an arbitrary manner. A task must be assigned to a machine where it executes with the fastest possible speed. Moreover, the interconnection network has its own

characteristics like transmission rate and limited communication capacities. The tasks also communicate with each other at a given rate to exchange data during execution.

Given the above mentioned constraints related to heterogeneous multicomputer systems, conventional parallel algorithms and tools aimed at homogeneous systems can't be efficiently used for parallel computing. Due to the presence of diversely capable machines, task allocation should take place in such a way that more powerful machines do more work and less powerful machines do less work. Moreover, capable machines should not be overloaded. The allocation should be done not only to optimize a cost function but also to achieve the task requirements and validate system resources. If such limitations are not taken into account, a task may be assigned to a machine that is not the best suited to execute it. Such drawbacks have been noticed in a number of published techniques. Clearly, there is a need to develop new algorithms and tools to efficiently use the HMS type of architectures. The work addressed in this thesis considers various behavioural characteristics that are essential to both, parallel tasks and the HMS.

1.5.4 No Task Duplication

A scheduling technique that duplicates some tasks on more than one machine to reduce the intertask communication cost is known as scheduling with task duplication [200]. The rational behind task duplication scheduling is that some child task may wait for the output from a parent task running on the other machine. The output from the parent task can be locally fed to the child task without having any intertask communication, if the parent task and child task are allocated to the same machine. Though the technique seems effective as it reduces schedule length, one major drawback of this technique is that, duplicating tasks may require duplicating data among machines and thus may lead to wastage of storage space. Another drawback of this approach is that it reduces the effective number of machines in the system because the same task is executed by more than one machine. Moreover, it has been found that duplication based techniques have

high complexities. The load balancing task allocation techniques presented in this thesis avoid the concept of task duplication.

1.5.5 Problem Complexity

The task allocation problem is known to be NP-complete in most cases, where there are n^m possible ways for allocating m tasks to n machines. It has been shown that, an optimal assignment may be found efficiently by a polynomial time algorithm for two or three processors [202] [181] and for restricted cases such as tree structure of interconnection pattern of application tasks [108] [47], uniform communication cost between tasks [44] and linear array of processors [146] [145]. However, for an arbitrary number of machines the problem is known to be NP-Hard and the timing complexity of the proposed optimal algorithms increases rapidly with increase in number of tasks and machines. Due to the complexity of the problem several heuristics have been reported in literature which lead to near optimal solution. Although a near optimal solution can be found quickly using heuristics but most of these ignore machine and task heterogeneity while allocating tasks and under perform when exercised taking into account such details. Considering these key factors, two heuristics are presented in this thesis which lead to a near optimal solution with acceptable timing complexity $O(m(n+1))$.

1.6 Contributions

The major contributions of this thesis are as follows:

1.6.1 An Economical and scalable topology for PDS

A new interconnection network topology called Scalable Twisted Hypercube (STH) has been proposed in this work to counter the poor scalability of twisted hypercube. Its suitability for use as a multiprocessor interconnection networks has been explored by establishing various mathematical properties of the topology. The

topology has been analyzed and compared with some other topologies of interest on a number of interconnection networks evaluation parameters. The results obtained indicate that with reduced diameter, better average distance, low traffic density, low cost, maximum number of links, high bisection width and tremendous scalability, STH is comparatively more suitable for Massively Parallel Systems.

1.6.2 Routing and Broadcasting Procedures for STH Topology

An interconnection network topology offers no benefits until it is supported by an efficient routing algorithm. Procedures for routing and broadcasting on the proposed topology have also been discussed and a simple routing algorithm has been presented. In summary, the proposed interconnection network offers sound architectural support for parallel computing.

1.6.3 Load Balancing Task Allocation Models

A Heterogeneous Multicomputer System is characterized by machine and task heterogeneity. Processors centric allocation models reported in literature are therefore incapable of fair task allocation in such a vague environment. One of the possible solutions is the use of Fuzzy logic based techniques. Fuzzy logic is a powerful tool for designing intelligent systems. The ability of drawing conclusions and generating responses based on vague, ambiguous and imprecise data offers tremendous leverage to Fuzzy Logic. Consequently, mathematical problems that are difficult to model using conventional methods can easily be formulated using fuzzy principles.

To deal with the limitations of previously proposed processors centric models, this thesis presents two task allocation models based on fuzzy logic, which are suitable for allocating tasks on HMS fairly. The main objective of the proposed allocation models is to perform task allocation with load balancing. That is, to achieve a fairly distributed cumulative execution time on all machines under given task and machine characteristics.The proposed fuzzy models are novel and

efficient.

1.6.4 Fuzzy Expected Time to Compute $(FETC)$ Matrix Generation Algorithm

The task allocation models presented in this thesis assume that a parallel application can be divided into number of tasks which can execute on diversely capable machines in parallel. The machines are connected through an interconnection network and the execution times of tasks are unknown. To evaluate the execution times of tasks in a fuzzy environment, under a given task and machine heterogeneities in run time, a novel algorithm has been developed. The algorithm serves as the basis of two task allocation models developed in this thesis. The input to the algorithm is the task and machine heterogeneities and it generates the Fuzzy Expected Time to Compute $(FETC)$ Matrix.

1.6.5 Efficient Allocation Algorithms

Based on the proposed models two novel algorithms were developed to allocate tasks on HMS. To simulate these algorithms and to validate the proposed models, C programs are written using CodeBlocks 12.11 compiler under Ubuntu 12.0 LTS operating system and executed on a Dell Machine with an Intel Core i7 processor, 4GB RAM and 500GB Hard Disk. The files FLBTA.sh and FLBTAM.sh run the simulation. To obtain the test related data, random machine and task scenarios have been generated. Machine heterogeneity has been ascertained by consulting the components statistics available on various sites.

1.7 Thesis Organization

The remainder of the thesis consists of six chapters organized as shown in figure 1.9.

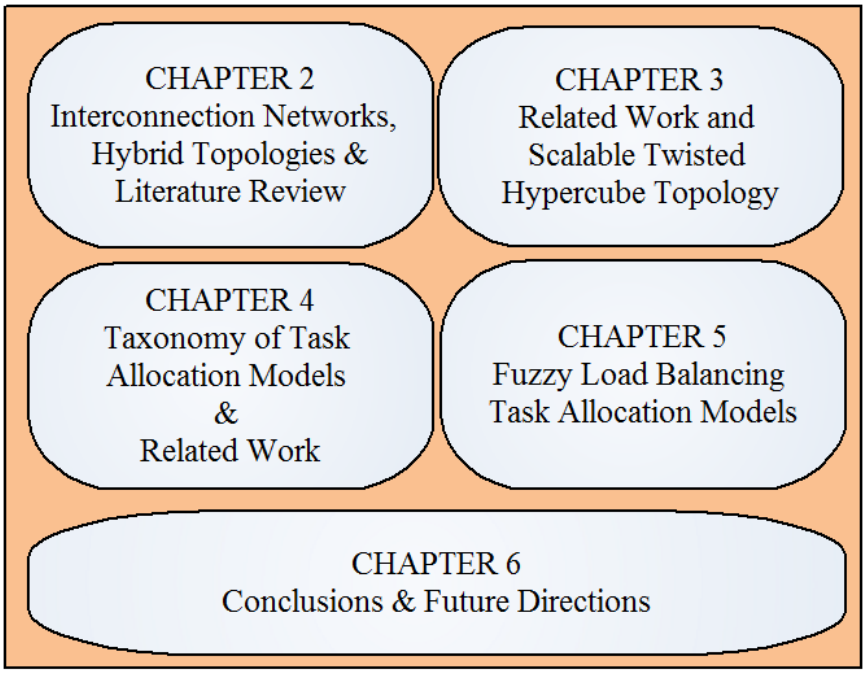

Figure 1.9: Thesis Organization

Chapter 2:

This chapter presents basic concepts related to the design and performance issues of multiprocessor interconnection networks and reviews the literature associated with them. It starts with an introduction to multiprocessor interconnection networks and briefly discusses static and dynamic interconnection networks while reviewing the literature related to them. Next, it introduces a new class of interconnection networks known as hybrid networks. Cross product as a network synthesizing operator is introduced and its significance is also briefly discussed. A good number of mathematical definitions used to describe interconnection networks are also introduced. Further, the chapter presents a section on performance evaluation parameters available for interconnection networks. Basic properties related to product networks are also explained. The chapter concludes with a discussion on how interconnection network topology effects load balancing in parallel and distributed systems.

Chapter 3:

This chapter begins with a literature survey on various interconnection network topologies proposed for parallel systems. It then introduces the design principles of a new interconnection network topology termed as STH (Scalable Twisted Hypercube). Suitability of STH for use as an interconnection network topology is explained. Mathematical expressions to represent various topological properties of the proposed STH topology such as degree, diameter, total number of nodes, total number of links, average distance and bisection width are derived. Further, the proposed topology is compared with some other topologies of interest on a number of interconnection networks evaluation parameters. Experimental results are presented to show that with a reduced diameter, better average distance, low traffic density, low cost, maximum number of links, high bisection width and tremendous scalability, STH is more suitable for massively parallel systems. Procedures for routing and broadcasting on the proposed topology are also discussed and a simple routing algorithm is also presented.

Chapter 4:

This chapter starts with an introduction to the Task Allocation Problem. It then classifies available solutions to this problem, on the basis of various techniques used to solve. Various solution techniques are discussed with a detailed literature survey on related task allocation models. The advantages and limitations of each technique are also pointed out. Further, the chapter considers a very important parameter for the task allocation models in parallel and distributed systems referred to as “load balancing”. Significance of load balancing as an allocation step is also explained.

Chapter 5:

The chapter begins with a brief introduction to Fuzzy Logic. Next, it presents the concept of fuzzy numbers, triangular membership, linguistic variables and de-

fuzzification. Significance and various operations on triangular fuzzy numbers is briefly presented. The chapter then focuses on the major contribution of the thesis. It first presents a brief literature review of related work and then proceeds to the principles of two fuzzy load balancing task allocation models. Various parameters on which the proposed Load Balancing Fuzzy Task Allocators are based, are discussed and argued. Three novel algorithms - One for determining Fuzzy Expected Time to Compute Matrix and two for the implementation of proposed fuzzy load balancing task allocation models are presented and discussed. Experimental data from random task scenarios and random machine scenarios are presented to test and verify the proposed models.

Chapter 6:

The thesis summarizes in Chapter 6. The contributions made include inferences drawn as a result of various experiments conducted within this work. From the experimental results obtained on Scalable Twisted Hypercube Topology, it can be concluded that with reduced diameter, better average distance, low traffic density, low cost, maximum number of links, high bisection width and tremendous scalability, STH is more suitable for Massively Parallel Systems. Further, from the experimental results obtained from the proposed Fuzzy Load Balancing Task Allocation Models it can be concluded that the models are simple and efficient. This chapter also suggests directions for future research in this area.

Chapter 2

Interconnection Networks and Hybrid Topologies

This chapter provides background knowledge about interconnection networks (INs) and hybrid topologies. Various properties associated with interconnection networks and metrics used to gauge the performance of various topologies are explained.

The major constituents of a multiprocessor system are multiple CPUs connected through an interconnection network and the software that enables the CPUs to work together. Communication in message passing systems is performed by send and receive commands, while in shared memory multiprocessor systems, communication is performed by writing to and reading from the global memory. In both cases, the interconnection network plays a vital role in determining the communication speed.

2.1 Criteria Used for Classification of INs

Several surveys related to interconnection networks have been reported in literature [164]
[197] [106] [206]. The needs of the communication industry particularly in the field of telephone switching, influenced the early work on interconnection net-

works. As the need for more computing power increased, applications for interconnection networks within computing machines began to become evident. Amongst the first of these were matrix manipulation, sorting etc., but as interest in parallel processing increased, numerous interconnection networks were proposed both for shared memory systems and for distributed memory systems [196]. With advances in fast packet switching, interest in interconnection networks which were originally proposed for parallel processing are now being considered for use in fast packet switch based systems. According to Bhuyan [42] multiprocessor interconnection networks (INs) can be classified based on the following criteria:

- Mode of operation (synchronous versus asynchronous)
- Control strategy (centralized versus decentralized)
- Switching techniques (circuit versus packet)
- Topology (static versus dynamic)

El-Rewini and Abd-El-Barr [101] and Bhuyan et al. [43] have also used the same criteria.

2.1.1 Mode of Operation

Based on the mode of operation INs are classified as synchronous or asynchronous. In synchronous mode of operation, a single global clock is used by all components within the system while the asynchronous mode of operation does not require a global clock; instead handshaking signals are used in order to coordinate system operation.

2.1.2 Control Strategy

Based on control strategy INs are classified as centralized or decentralized. In centralized control systems, a single central control unit is used to coordinate and control the operation of the components within the system. In decentralized

control, the control function is distributed among different components within the system.

2.1.3 Switching Techniques

This approach classifies the interconnection networks as packet switched or circuit switched. In the circuit switching mechanism, a complete path has to be established prior to the start of communication between a source and a destination while in a packet switching mechanism, communication between a source and destination takes place via messages that are divided into smaller entities, called packets.

2.1.4 Topology

An interconnection network topology is a mapping function from the set of processors and memories onto the same set of processors and memories.

In general, interconnection networks can be classified as static or dynamic. In static networks, direct fixed links are established among nodes to form a fixed network, while in dynamic networks, connections are established as needed. Switching elements are used to establish connections between inputs and outputs. Depending on the switch settings, different interconnections can be established. Nearly all multiprocessor systems can be distinguished by their interconnection network topology.

2.2 Topology Based Classification for INs

On the basis of topology, interconnection networks (INs) can be classified as static or dynamic. In static INs, fixed links are created at the time of fabrication while dynamic INs establish connections on the fly based upon connection requirements. In dynamic networks, switches are used to establish connections between inputs and outputs. By manipulating the switch settings, different interconnections can

be established. Figure 2.1 shows topology based classification of INs. The follow-

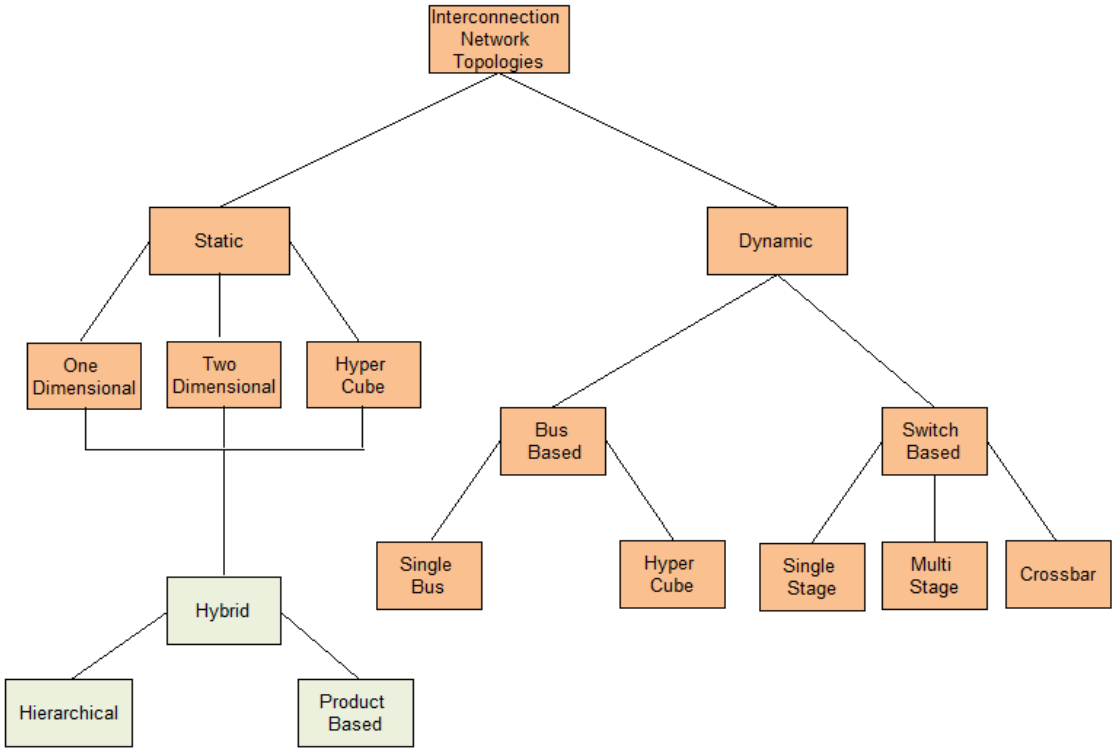

Figure 2.1: Topology based classification of INs

ing sections briefly discuss these topologies.

2.3 Dynamic Interconnection Networks

Dynamic interconnection networks can be either be bus based or switch based. The following subsections discuss them:

2.3.1 Bus Based Dynamic Interconnection Networks

Single Bus Systems

A single bus is considered the simplest way to connect multiprocessor systems. Figure 2.2 shows an illustration of a single bus system.

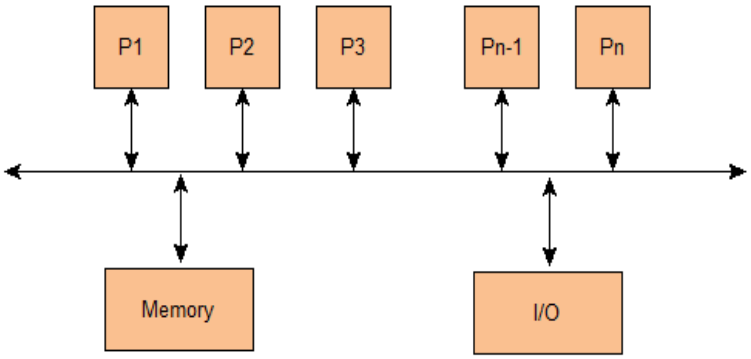

Figure 2.2: Topology based classification of INs

- In its general form, such a system consists of N processors, each having its own cache, connected by a shared bus.
- The use of local caches reduces the processormemory traffic. All processors communicate with a single shared memory.
- The typical size of such a system varies between 2 and 50 processors.
- System size is limited by the bandwidth of the bus and the fact that only one processor can access the bus, and in turn only one memory access can take place at any given time.

Multiple Bus Systems

The use of multiple buses to connect multiple processors is a natural extension to the single shared bus system.

- A multiple bus multiprocessor system uses several parallel buses to interconnect multiple processors and multiple memory modules.
- A number of connection schemes are possible in this case, namely multiple bus with full bus memory connection (MBFBMC), multiple bus with single bus memory connection (MBSBMC), multiple bus with partial busmemory connection (MBPBMC), and multiple bus with class-based memory connec-

tion (MBCBMC). Figure 2.3 and 2.4 illustrates these configurations respectively.

- The multiple bus with full busmemory connection has all memory modules connected to all buses. The multiple bus with single busmemory connection has each memory module connected to a specific bus. The multiple bus with partial busmemory connection has each memory module connected to a subset of buses. The multiple bus with class-based memory connection has memory modules grouped into classes whereby each class is connected to a specific subset of buses.

- In general, multiple bus multiprocessor organization offers a number of desirable features such as high reliability and ease of incremental growth.

- On the other hand, when the number of buses is less than the number of memory modules (or the number of processors), bus contention is expected to increase.

2.3.2 Switched Based Interconnection Networks

In this type of network, connections among processors and memory modules are made using simple switches. Three basic interconnection topologies exist: crossbar, single-stage, and multistage.

Crossbar

- Unlike the single bus which can provide only a single connection, a crossbar can provide simultaneous connections among all its inputs and outputs.

- The crossbar contains a switching element (SE) at the intersection of any two lines extended horizontally or vertically inside the switch.

- Figure 2.5 shows an 8×8 crossbar. A SE (also called a cross-point) is provided at each of the 64 intersection points.

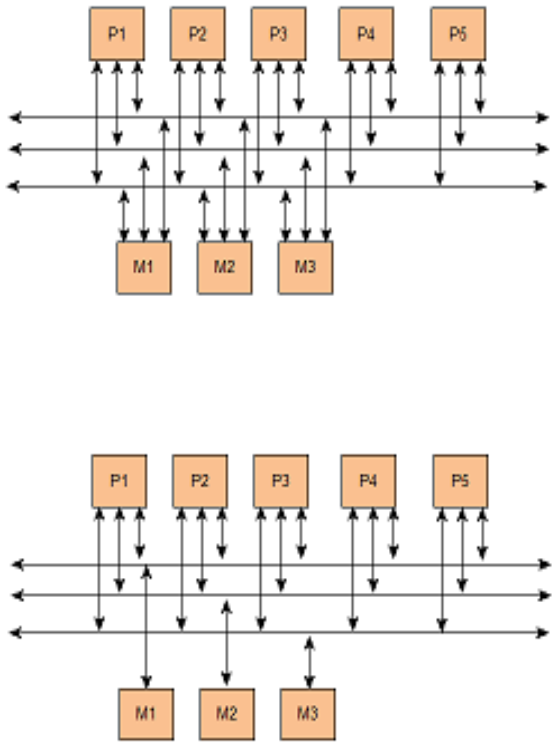

Figure 2.3: (i)MBFBMC (ii)MBSBMC

- Simultaneous connections between P_i and M_{8-i+1} for $1 \leq i \leq 8$ are made.
- The two possible settings of an SE in the crossbar (straight and diagonal) are also shown in the figure.
- As can be seen from the figure, the number of SEs (switching points) required is 64 and the message delay to traverse from the input to the output is constant, regardless of which input/output are communicating.
- In general for an $N \times N$ crossbar, the network complexity, measured in terms of the number of switching points, is $O(N^2)$ while the time complexity, measured in terms of the input to output delay, is $O(1)$.
- Crossbar is a nonblocking network that allows a multiple inputoutput connection pattern (permutation) to be achieved simultaneously.
- For a large multiprocessor system the complexity of the crossbar can become

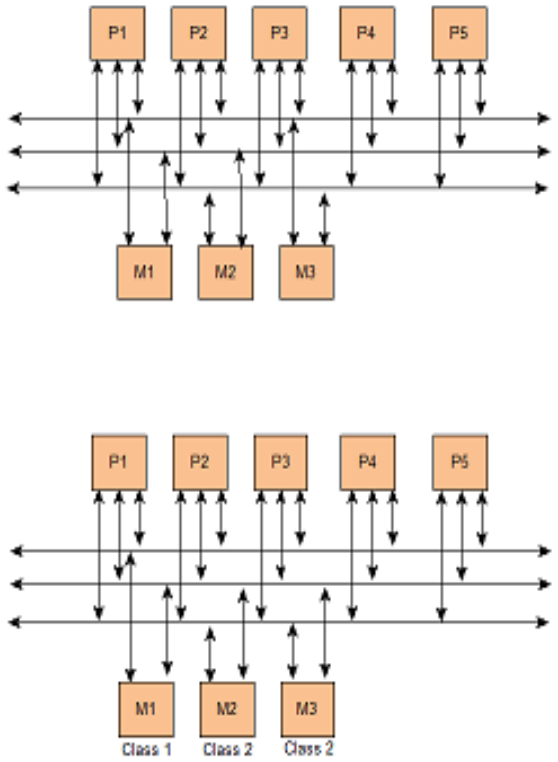

Figure 2.4: (i)MBPBMC (ii)MBCBMC

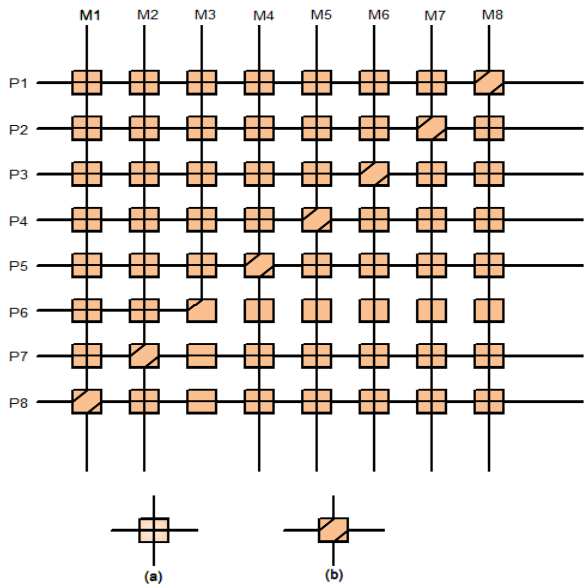

Figure 2.5: (i)MBFBMC (ii)MBSBMC

a dominant financial factor.

Single Stage Networks

In this case, a single stage of switching elements (SEs) exists between the inputs and the outputs of the network. The simplest switching element that can be used is the 2×2 switching element (SE). Figure 2.6 shows the four possible settings that a SE can assume. These settings are called straight, exchange, upper-broadcast, and lower-broadcast [204].

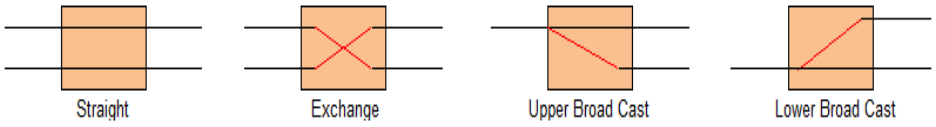

Figure 2.6: Different settings of the 2×2 SE

- In the straight setting, the upper input is transferred to the upper output and the lower input is transferred to the lower output.
- In the exchange setting, the upper input is transferred to the lower output and the lower input is transferred to the upper output.
- In the upper-broadcast setting, the upper input is broadcast to both the upper and the lower outputs.
- In the lower-broadcast setting, the lower input is broadcast to both the upper and the lower outputs.
- Following are some examples of Single Stage Networks:

Shuffle Exchange Network

- To establish communication between a given input (source) to a given output (destination), data has to be circulated a number of times around the network.

- A well-known connection pattern for interconnecting the inputs and the outputs of a single-stage network is the ShuffleExchange.
- Two operations are used. These can be defined using an m bit-wise address pattern of the inputs $p_{m-1}p_{m-2}p_{m-3}\ldots p_1p_0$ as follows:

$$S(p_{m-1}p_{m-2}p_{m-3}\ldots p_1p_0) = p_{m-2}p_{m-3}p_1p_0p_{m-1}$$

$$E(p_{m-1}p_{m-2}p_{m-3}\ldots p_1p_0) = p_{m-1}p_{m-2}p_{m-3}\ldots p_1\overline{p}_0$$

- With shuffle (S) and exchange (E) operations, data is circulated from input to output until it reaches its destination.
- To complete the network every output is buffered and fed back to its corresponding input. Packets of data therefore circulate through the structure until they exit at the desired output [68].
- For example, if the number of inputs (processors), in a single-stage IN is N and the number of outputs (memories), is N, the number of SEs in a stage is $N/2$.
- The maximum length of a path from an input to an output in the network, measured by the number of SEs along the path, is $\log_2 N$.
- Stone [205] introduced the perfect shuffle as a pattern of interconnection links of some interest. The goal was to solve a number of classes of computational problem via a tightly coupled parallel processor.

The Cube Network

- The interconnection pattern used in the cube network is defined as follows:

$$C_i(p_{m-1}p_{m-2}p_{m-3}\ldots p_{i+1}p_ip_{i-1}p_1p_0) = p_{m-1}p_{m-2}p_{m-3}\ldots p_{i+1}\overline{p}_ip_{i-1}\ldots p_1p_0$$

- Considering a 3-bit address $(N = 8)$, then $C_2(6) = 2$, $C_1(7) = 5$ and $C_0(4) = 5$. Figure 2.7 shows the cube interconnection patterns for a network with $N = 8$.

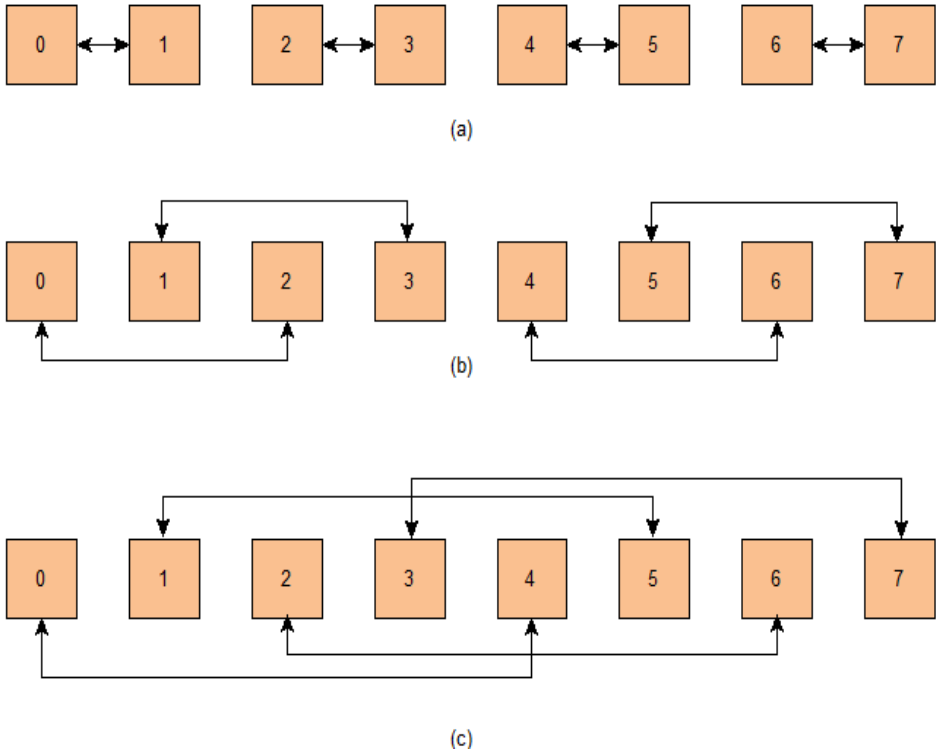

Figure 2.7: The cube network for$N = 8$ (a) C_0; (b) C_1; and (c) C_2

- The network is called the cube network due to the fact that it resembles the interconnection among the corners of an n-dimensional cube $(n = \log_2 N)$.

The Butterfly Network

- The interconnection pattern of Butterfly Network uses the butterfly function, which is defined as:

$$B(p_{m-1}p_{m-2}p_{m-3}\dots p_1p_0) = p_0p_{m-2}p_{m-3}\dots p_1p_{m-1}$$

- For a 3-bit address (N = 8), the following is the butterfly mapping:

  $B(000) = 000$

  $B(001) = 100$

  $B(010) = 010$

  $B(011) = 110$

  $B(100) = 001$

$B(101) = 101$

$B(110) = 011$

$B(111) = 111$

Multistage Networks

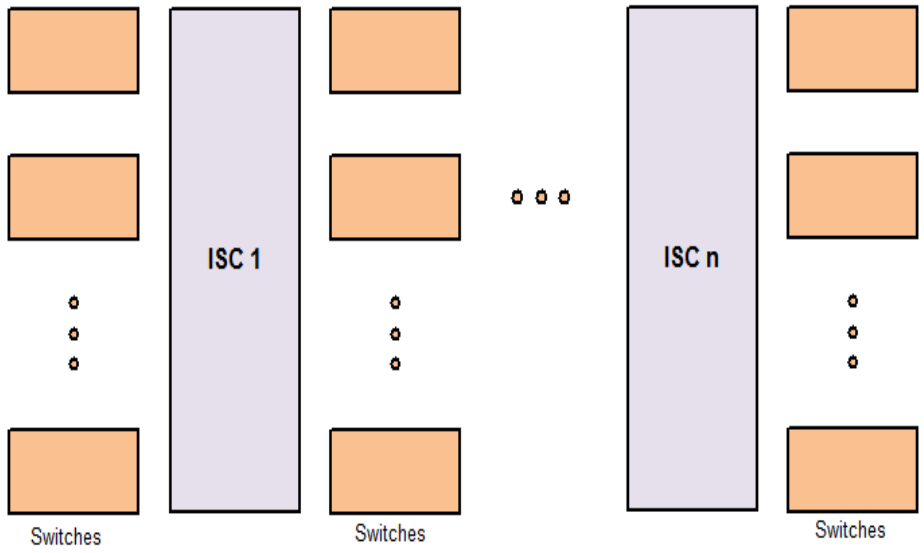

Figure 2.8: Multistage interconnection network

- If multiple copies of the single stage shuffle exchange are cascaded, a multi-stage interconnection network (MIN) results and is called a multi-stage shuffle exchange. Data is no longer required to circulate through the network but passes through the structure from input to output.
- The most undesirable single bus limitation that MINs are set to improve is the availability of only one single path between the processors and the memory modules. Thus, MINs provide a number of simultaneous paths between the processors and the memory modules.
- As shown in Figure 2.8, a general MIN consists of a number of stages each consisting of a set of 2×2 switching elements. Stages are connected to each other using Inter-stage Connection (ISC) Pattern.

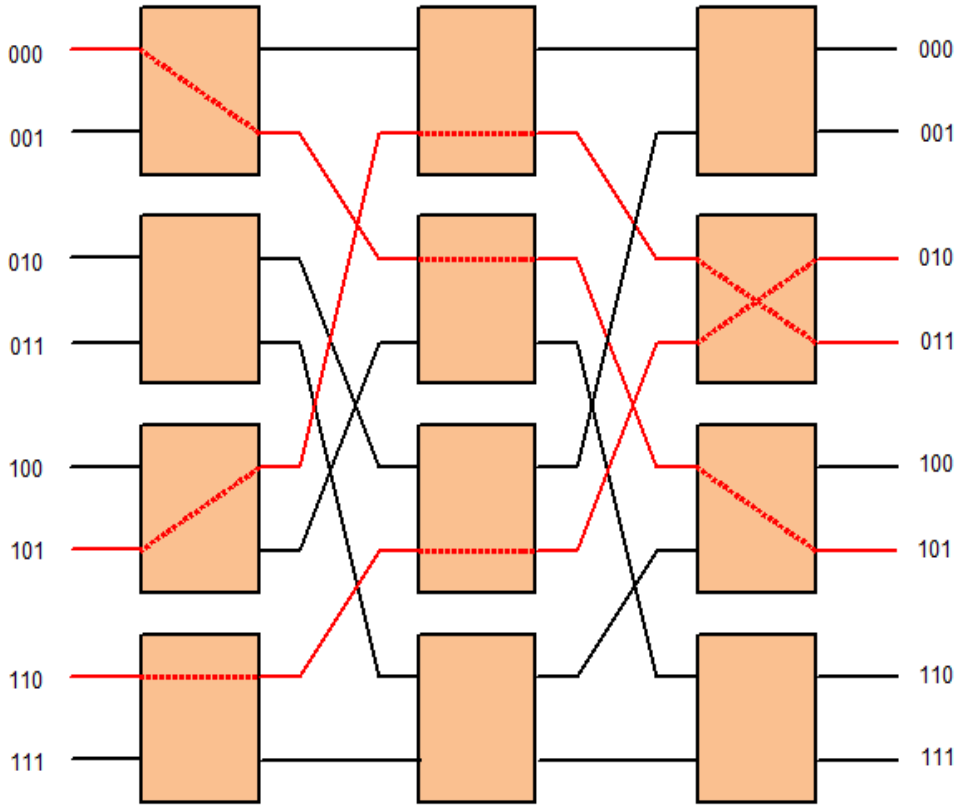

Figure 2.9: An example of 8×8 ShuffleExchange network (SEN).

- Figure 2.9 shows an example of an 8×8 MIN that uses the 2×2 SEs described before. This network is known as the ShuffleExchange network (SEN).
- There exist $\log_2 N$ stages in a $N \times N$ MIN.
- Other examples of multi-stage interconnection networks include:

The Banyan Network

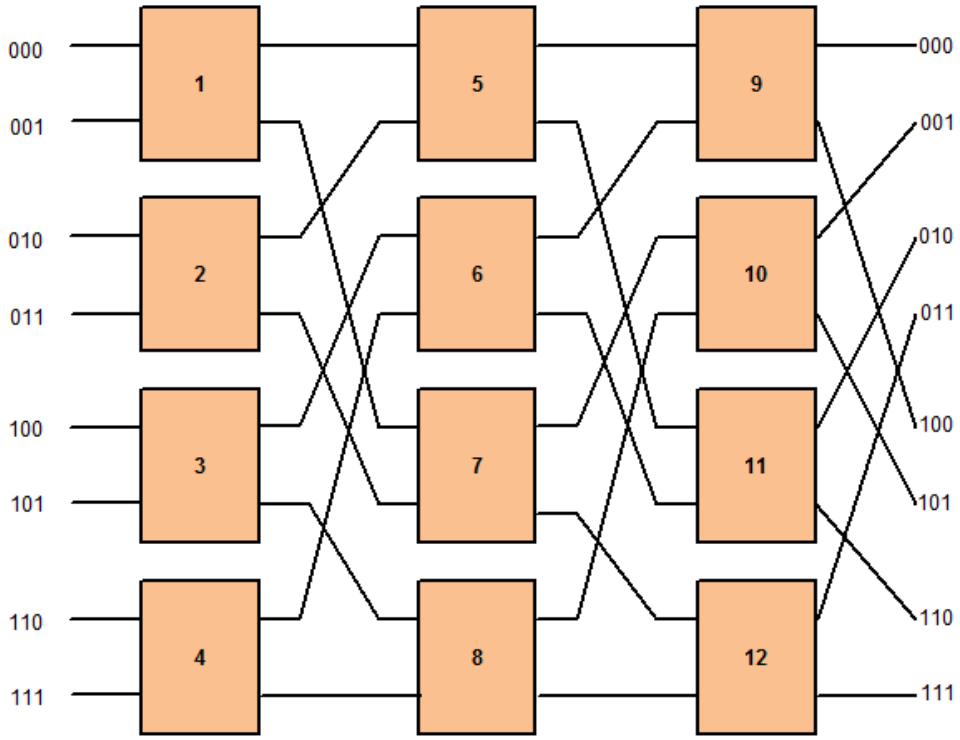

Figure 2.10: An example of 8 x 8 Banyan Network.

- Figure 2.10 shows an example of an 8 x 8 Banyan network.
- The design principle of banyan network is that: if the number of inputs (processors) is N and the number of outputs (memory), is also N, the number of MIN stages is $\log_2 N$ and the number of SEs per stage is $N/2$. Hence, the network complexity, measured in terms of the total number of SEs is $O(N \log_2 N)$.

The Omega Network

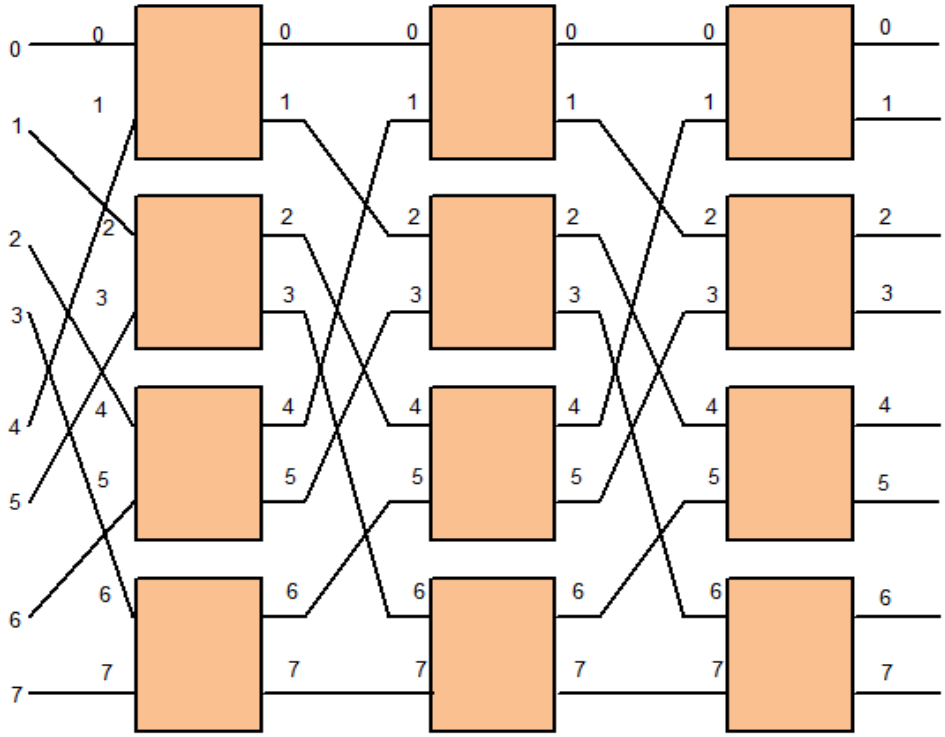

Figure 2.11: An example of 8×8 Omega Network.

- Figure 2.11 shows an example of an 8×8 Omega network.
- A size N omega network consists of $log_2 N$ single-stage shuffle exchange networks [143].
- Each stage consists of a column of $N/2$, two-input switching elements whose input is a shuffle connection.

There is a long list of multistage interconnection networks reported in literature. Some better known examples include The Delta Network [173], The Clos [78], The Benes [38] and The Batcher Sorting Network [36]

2.3.3 Blocking and Non-Blocking Networks

- Multi-stage interconnection networks may be further classified according to the blocking characteristics they present which is reflected in the throughput they offer to traffic with a random distribution of packet destinations.
- Blocking networks possess the property that in the presence of a currently established interconnection between a pair of input/output, the arrival of a request for a new interconnection between two arbitrary unused input and output may or may not be possible. Examples of blocking networks include Omega, Banyan, ShuffleExchange, and Baseline.
- Nonblocking networks are characterized by the property that in the presence of a currently established connection between any pair of input/output, it will always be possible to establish a connection between any arbitrary unused pair of input/output.
- Re-arrangeable networks are characterized by the property that it is always possible to rearrange already established connections in order to make allowance for other connections to be established simultaneously. The Benes is a well-known example of a re-arrangeable network.

2.4 Static Interconnection Networks

There are two types of static interconnection networks:

- Completely Connected Networks (CCNs)
- Limited Connection Networks (LCNs)

Before proceeding further with discussing static interconnection networks, it is necessary to introduce some important topological definitions which play a major role in grading of these networks.

Degree(d)

The Degree (d) of a node in a network is defined as the number of channels incident on the node. The number of channels into the node is termed as the in-degree, d_{in}. The number of channels out of a node is termed as the out-degree, d_{out}. The total degree, d, is the sum, $d = d_{in} + d_{out}$. For an undirected network, the number of edges incident on a node is called the degree of that node. The node degree reflects the number of I/O ports required per node, and thus the cost of the network. Therefore, the node degree should be kept constant and as small as possible in order to reduce cost. A constant node degree is very much desired to achieve modularity in building blocks for scalable systems.

Diameter(D)

The Diameter (D) of a network having N nodes is defined as the longest path, l, of the shortest paths between any two nodes $D = \max(\min_{l \in l_{i,j}}(length(l))$. In this equation, l_{ij} is the length of the path between nodes i and j and $length(l)$ is a procedure that returns the length of the path, l. In simple words diameter of a network is the maximum shortest path between any two nodes. The path length is measured by the number of links traversed. The network diameter indicates the maximum number of distinct hops between any two nodes, thus provides a figure of communication merit for the network [129]. It should be obvious that the diameter should be as small as possible from a communication point of view.

Bisection Width (ω)

Bisection Width (b) of a network is the minimum number of edges that must be removed in order to divide the network into two halves. In case of a communication network each edge corresponds to a channel with w bit wires. Then the wire bisection width is $B = bw$. The parameter B reflects the wiring density of a network. When B is fixed, the channel width (in bits) $w = B/b$. Thus the bisection width is a good indicator of the maximum communication bandwidth along the

bisection of a network.

Symmetry

A network is said to be symmetric if it is isomorphic to itself with any node labeled as the origin; that is, the network looks the same from any node. Rings and Tori networks are symmetric while linear arrays and mesh networks are not.

Network Throughput

Network throughput is an indicative measure of the message carrying capacity of a network. It is defined as the total number of messages the network can transfer per unit time. To estimate the throughput, the capacity of the network and the messages number of actually carried by the network are calculated. Practically the throughput is only a fraction of its capacity.

Latency

Latency is the delay in transferring the message between two nodes.

Data Routing Functions

These are the functions which when executed, establish the path between the source and the destination. In dynamic interconnection networks, there can be various interconnection patterns that can be generated from a single network. This is done by executing various data routing functions. Thus data routing operations are used for routing the data between various processors.

Dimensionality of Interconnection Network

Dimensionality indicates the arrangement of nodes or processing elements in an interconnection network. In a single dimensional or linear network, nodes are

connected in a linear fashion; in a two dimensional network the processing elements (PEs) are arranged in a grid while in a cube network they are arranged in a three dimensional network.

Hardware Cost

It refers to the cost involved in the implementation of an interconnection network. It includes the cost of switches, arbiter unit, connectors, arbitration unit, and interface logic.

2.4.1 Completely Connected Networks

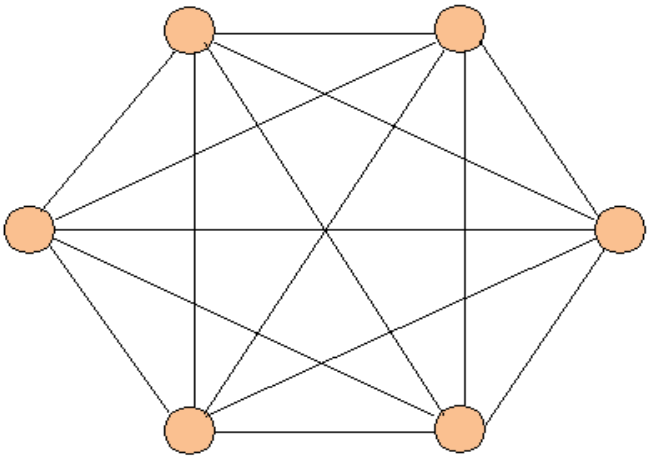

Figure 2.12: An example of six node CCN

- In a completely connected network (CCN), each node is connected to all other nodes in the network.
- Completely connected networks guarantee fast delivery of messages from any source node to any destination node (only one link has to be traversed).
- Since every node is connected to every other node in the network, routing of messages between nodes becomes a straightforward task.

- Completely connected networks are, however, expensive in terms of the number of links needed for their construction. This disadvantage becomes more and more apparent for higher values of N.
- The number of links in a completely connected network is given by $N(N-1)/2$, that is, $O(N^2)$.
- The delay complexity of CCNs, measured in terms of the number of links traversed as messages are routed from any source to any destination is constant, that is, $O(1)$.
- An example having $N = 6$ nodes is shown in Figure 2.12. A total of 15 links are required in order to satisfy the complete interconnectivity of the network.

2.5 Limited Connection Networks

Limited connection networks (LCNs) do not provide a direct link from every node to every other node in the network. Instead, communications between some nodes have to be routed through other nodes in the network. The length of the path between nodes, measured in terms of the number of links that have to be traversed, is expected to be longer compared to the case of CCNs. Two other conditions seem to have been imposed due to limited interconnectivity in LCNs. These are: the need for a pattern of interconnection among nodes and the need for a mechanism for routing messages around the network until they reach their destinations. A number of regular interconnection patterns have evolved over the years for LCNs. These patterns include:

Linear Array

- This is a one dimensional network in which N nodes are connected by $N-1$ links in a line.

- Internal nodes have degree 2 and the terminal nodes have degree 1. The diameter is $N-1$, which is rather long for large N. The Bisection Width is $b=1$.
- Linear arrays are the simplest connection topology. However, the structure is asymmetric and communication is quite inefficient when N becomes very large.
- As the diameter increases linearly with respect to N, linear arrays are not suited for large N. For very small N, it is rather economical to implement a linear array.

Ring and Chordal Ring

- A ring is obtained by connecting the two terminal nodes of a linear array with one extra link.
- A ring can be unidirectional or bidirectional. It is symmetric with a constant node degree of 2.
- The diameter is $\lfloor N \rfloor/2$ for a bidirectional ring and N for unidirectional ring.
- By increasing the node degree from 2 to 4 we obtain a chordal ring.
- Two or more links may be added to produce other chordal rings. In general, the more links added, the higher the node degree and the shorter the network diameter.

Tree and Star

- A binary tree network consists of nodes connected in a binary tree arrangement.
- In general a k-level, completely balanced binary tree should have $N=2k-1$ nodes. The maximum mode degree is 3 and the diameter is $2(k-1)$.

- With a constant node degree the binary tree is a scalable architecture. However, the diameter is rather long.
- The Star is a two level tree with a high node degree of $d = N - 1$ and a small constant diameter of 2.

Fat Trees

- The channel width of a fat tree increases as we ascend from levels to root [148].
- A fat tree is more like a real tree in that branches get thicker toward the root.
- One of the major problems in using the conventional binary tree is the bottleneck problem toward the root, since traffic toward the root becomes heavier. The fat tree was proposed to alleviate the problem.
- The idea of fat tree has been applied in the Connection Machine CM-5. The idea of binary trees can also be extended to multiway fat trees.

Mesh and Torus

- A 3×3 mesh network consists of 9 nodes.
- In general, a k-dimensional mesh with $N = nk$ nodes has an interior node degree of $2k$ and the network diameter is $k(n - 1)$. The node degrees at the boundary and corner nodes are 3 and 2 respectively.
- Pure mesh is not symmetric. However, some of its variants may be symmetric.
- The torus can be viewed as another variant of the mesh with an even shorter diameter.
- This topology combines the ring and mesh and extends to higher dimensions. The torus has ring connections across each row and along each column of the array.

- In general an $n \times n$ torus has a node degree of 4 and a diameter of $\lfloor n \rfloor / 2$.
- The torus is a symmetric topology. All added wraparound connections help reduce the diameter by one-half from that of mesh.

Systolic Arrays

- This is a class of multidimensional pipelined array architectures designed for implementing fixed algorithms.
- A systolic array has been specially designed for performing matrix-matrix multiplication.
- The systolic array has become a popular research area ever since its introduction. With fixed interconnection and synchronous operation, a systolic array matches the communication structure of the algorithm.
- For special applications like image/ signal processing, systolic arrays may offer a better performance/ cost ratio.

Hypercube

- This is a binary n-cube architecture which has been implemented in the iPSC, nCUBE and CM-2 systems.
- In general, an n-cube consists of $N = 2n$ nodes spanning along n dimensions, with two nodes per dimension.
- The node degree of an n-cube equals n and so does the network diameter.
- In fact, the node degree increases linearly with respect to the dimension, making it difficult to consider the hypercube a scalable architecture.
- Binary hypercube has been a very popular architecture for research and development. Both Intel iPSC/1, iPSC/2 and nCUBE machines were built based upon the hypercube architecture. The architecture has a dense connection.

- With poor scalability and difficulty in packaging higher-dimensional hypercubes, the architecture has been replaced by other architecture. For example the CM-5 chooses the fat tree over the hypercube implemented in CM-2. The Intel Paragon chooses a two dimensional mesh over its hypercube predecessors.

Besides above popular networks, numerous limited connection networks like Cube Connected Cycles(CCC) [176], Barrel Shifter etc. have also been reported in literature. Table 2.5 lists topological properties of some important limited connection networks for network size of N:

Network	**Degree**	**Diameter**	**Bisection Width**	**Symmetry**
Linear Array	2	$N-1$	1	No
Ring	2	$\lfloor N \rfloor/2$	2	Yes
Completely Connected	2	1	$(N/2)^2$	Yes
Binary Tree	3	$2(\log_2 N - 1)$	1	No
Star	$N-1$	2	$\lfloor N \rfloor/2$	No
2D-Mesh	4	$2(\sqrt{N}-1)$	$\sqrt{N}$	No
2D-Torus	4	$2\lfloor\sqrt{N}/2\rfloor$	$2\sqrt{N}$	Yes
k-Hypercube	$\log_k N$	$\log_k N$	$N/2$	Yes
CCC(k), $k = log_2(N/k), k \geq 3$	3	$(2k-1+\lfloor k/2 \rfloor)$	$N/(2k)$	Yes

Table 2.1: Topological properties of limited connection networks

2.6 Hybrid Interconnection Networks

Hybrid topology, as the name suggests, is a combination of two or more topologies. Two types of hybrid interconnection networks have been proposed in literature:

1. Hierarchical Interconnection Networks: Hierarchical interconnection networks (HINs) provide a framework for designing networks with reduced link cost by taking advantage of the locality of communication that exists in parallel applications [2]. Some well known networks in this category are Hierarchical n-Hypercube [161], Extended Hypercube [142], dBCube [67], HFCube [195] etc. Abd-El-Barr and Al-Somani [2] present a good review of various hierarchical networks and their topological properties.

2. Product Networks: There are several graph-theoretic operators which enable us to derive large networks from small networks. Cross product or Cartesian product is one of them.

   Cross product of undirected graphs as a means of combining two topologies with known properties has been extensively exploited in literature. The resultant graph retains the properties of both the graphs. Example includes multidimensional meshes, tori, ω-graphs [190], Banyan-Hypercubes [235], mesh connected trees [98] etc.

   The next chapter presents a highly scalable and economical topology for parallel and distributed systems. The proposed topology is based on the cross product of two topology graphs. Hence, to facilitate ease of understanding of the next chapter, the next section is devoted to Product Networks.

2.7 Cross Product as Net. Synthesizing Operator

Let $X_1 = (V_1, E_1)$ and $X_2 = (V_2, E_2)$ be two undirected graphs representing two interconnection topologies, where V_1 and V_2 are sets of vertices (processors) and E_1 and E_2 are sets of edges (interconnections between processors). Then the cross product of the two topologies is defined as below:

Definition 2.7.1 Cross Product: The cross product $X = X_1 \otimes X_2$ of two undirected connected graphs $X_1 = (V_1, E_1)$ and $X_2 = (V_2, E_2)$ is the undirected graph $X = (V, E)$, where V and E are given by:

1. $V = < x_1, x_2 > | x_1 \in V_1 \quad and \quad x_2 \in V_2$, and

2. for any $u =< x_1, x_2 >$ and $v =< y_1, y_2 >$ in $V, (u, v)$ is an edge in E, iff either (x_1, y_1) is an edge in E_1 and $x_2 = y_2$, or (x_2, y_2) is an edge in E_2 and $x_1 = y_1$.

An Example

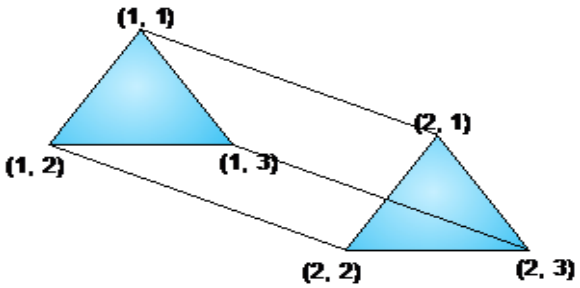

Figure 2.13: An example of Cross Product

Consider $X_1 = (V_1, E_1)$, where, $V_1 = \{1, 2\}$ and $E_1 = \{(1, 2)\}$

$X_2 = (V_2, E_2)$, where, $V_2 = \{1, 2, 3\}$ and $E_2 = \{(1, 2), (2, 3), (3, 1)\}$

The product graph $X = X_1 \otimes X_2$ is shown in Figure 2.13.

As can be seen from the above example $X = X_1 \otimes X_2$ can be constructed by replacing each vertices of X_1 by X_2 (or by repeating X_2, $|V_1|$ times) and setting up the appropriate links as per Definition 2.7.1. Here $|V_1|$ means the cardinality of set V_1. Clearly, the Cartesian Product Operator is a network synthesizer. It takes two networks X_1 and X_2 and synthesizes a third network $X = X_1 \otimes X_2$.

2.7.1 Topological Properties of Product Networks

Topological properties of product networks have been described in [234]. From the point of view of this thesis the following properties are important, hence are being briefly discussed below. Throughout the discussion the following notion have been adopted:

- $deg_X(x)$ = the degree of node x in graph X

- $d_X(x, y)$ = the distance from x to y in X
- D_X = the diameter of graph X
- $\overline{d}_X$ the average distance of X

Definition 2.7.2 The Degree Formula: If X_1 and X_2 are two graphs of degrees deg_{X1} and deg_{X2} respectively then the degree of $X_1 \otimes X_2$ i.e. $deg_{(X_1 \otimes X_2)}$ is given by:

$$deg_{(X_1 \otimes X_2)} = deg_{X1} + deg_{X2}$$

Definition 2.7.3 Size and Diameter Formula: if X_1 and X_2 are two undirected connected graphs with diameters D_{X_1} and D_{X_2} and of sizes N_1 and N_2 respectively then:

- $X_1 \otimes X_2$ is connected
- the size N and the diameter $D_{(X_1 \otimes X_2)}$ of $X_1 \otimes X_2$ are given by $N = N_1 \times N_2$ and $D_{(X_1 \otimes X_2)} = D_{X_1} + D_{X_2}$

Corollary 2.7.4 It is obvious that $D_X(n) = nD_X$

Definition 2.7.5 Shortest Path Formula: If $x, x_1, x_2, \ldots, x_l, y$ is a shortest path from x to y in X_1 and if $x', x'_1, x'_2, \ldots, x'_r, y'$ a shortest path from x to y in X_2, then $xx', x_1x', x_2x', \ldots, x_lx', yx'_r, yy'$ is a shortest path from xx' to yy' in $X_1 \otimes X_2$.

Definition 2.7.6 Average Distance Formula: if X_1 and X_2 are two undirected connected graphs then

$$\overline{d}_{X_1 \otimes X_2} = \overline{d}_{X_1} + \overline{d}_{X_2}$$

Corollary 2.7.7 It is obvious that $\overline{d}_X(n) = n\overline{d}_X$

2.7.2 Significance of Cross Product

The following example illustrates the significance of Product Networks.

Example

Let $SC(n,h)$ be the cross product of the n-Star Graph $S(n)$ and the h-Hypercube $C(h)$. $SC(n,h)$ offers a higher flexibility in choosing the network size and adjusting the degree and diameter parameters. Table 2.2 on page 50 illustrates this feature by showing the size, degree and diameter of $S(n)$, $C(h)$ and $SC(n,h)$. From this table, it can be observed that in all the cases considered, $SC(n,h)$ has a better fit to the desired size than $S(n)$ and a lower degree and diameter than $C(h)$.

2.7.3 Routing on Product Networks:

A distributed routing algorithm for a network $X = (V,E)$ is a function R from $V \times V$ to V that associates for each pair of nodes(current, destination) a node next, where current is the node at which a given message is currently stored, destination is the destination node for that message, and next is the next node to be visited by the message in its way to the destination node. Let R_1 and R_2 be two distributed routing algorithms for X_1 and X_2, respectively then, a distributed routing algorithm R for $X_1 \otimes X_2$ is given by:

$$R[(cur_1, cur_2),(dest_1, dest_2)] = \begin{cases} [R_1(cur_1, dest_1), cur_2] & if \quad cur_1 \neq cur_2 \\ [cur_1, R_2(cur_2, dest_2)] & if \quad cur_1 = cur_2 \end{cases}$$

The above routing rules route a message along X_1-edges until the X_2-component of the current node is equated to the X_1-component of the destination node. Once that is achieved, the routing continues along X_2 edges. The relative order of these two routing stages may be reversed or interleaved.

2.7.4 Broadcasting on Product Networks

Many parallel and distributed applications require an efficient broadcast algorithm. Given a broadcast algorithm for each X_1 and X_2, a broadcast algorithm for $X_1 \otimes X_2$, can be obtained as follows. Assume that a message M, originally at

Desired Size	**Star Graph,** $S(n)$				**Hypercube,** $C(h)$			**Star Cube,** $SC(n,h)$			
	n	Size	Degree	Diameter	h	Size	Degree	Diameter	(n,h)	Size	Degree	Diameter
50	5	120	4	6	6	64	6	6	(4,1)	48	4	5
100	5	120	4	6	7	128	7	7	(4,2)	96	5	6
500	6	720	5	7	9	512	9	9	(5,2)	480	6	8
1000	6	720	5	7	10	1024	10	10	(6,1)	1440	6	8
10,000	7	50,40	6	9	13	8192	13	13	(7,1)	10080	6	10
100,000	9	362880	8	12	17	131072	17	17	(8,2)	161280	9	12

Table 2.2: Significance of product networks

source node (a, b) of $X_1 \otimes X_2$, is to be broadcast to all other nodes. In the first stage, a known broadcast algorithm of X_1 can be used to broadcast M in X_1 subgraph of $X_1 \otimes X_2$ that contains all the nodes of the form (c, b) for any node c in X_1. In other words in the first stage, a copy of M is delivered to each node that can be reached from the source using only X_1 edges. In the second stage, all nodes of the form (c, b) have received a copy of M during the first stage, apply broadcast algorithm of X_2 to initiate parallel broadcasts in different X_2 subgraphs of $X_1 \otimes X_2$. During the second stage, communication is performed along X_2 edges only.

2.7.5 Performance Issues of Product Networks

As explained earlier, the cross product of two interconnection networks yields a third interconnection network, so ultimately a product network is also a multiprocessor interconnection network. It is therefore obvious that the factors which are used to determine the performance of any multiprocessor also apply to a product network. They have already been described in Section 2.4.

2.8 Network Topology and Load Balancing

The effect of topology on the performance of load balancing schemes have been studied by few researchers including Loh et al. [155]. The following paragraphs outline their observations.

- Loh et al. [155] studied four topologies namely 4×4 mesh, 4d-Hypercube, Fibonacci Cube and Linear array under five load balancing schemes. Their performance parameters were normalized performance and stabilization time. The researchers concluded that during load balancing process, the topologies which possessed larger average processor distances and lower average node connectivity introduced significant communication overheads.

- All load balancing schemes performed best with hypercube and Fibonacci cube topologies (because of better node connectivity and better internode

average distances).

- They further showed that varying physical parameters in an interconnection network topology can significantly effect the performance of a load balancing scheme irrespective of the load conditions of the system.

- Zhang [239] showed that in the absence of wormhole routing, communication latencies can be a serious problem in parallel systems as not all parallel systems support wormhole routing.

Above findings justify the research undertaken within this thesis for a better topology for parallel and distributed systems.

Chapter Summary

In this chapter we have discussed different interconnection networks used for interconnecting multiprocessors. The design issues of interconnection networks, types of interconnection network, permutation network and performance metrics of the interconnection networks are discussed. Classification of interconnection networks based on topology has been reviewed and a new addition namely 'Hybrid Interconnection Networks' has been introduced to the conventional topology based classification to accommodate hierarchical and product networks. Dynamic and static interconnection networks have also been reviewed. In the dynamic interconnection scheme, three main mechanisms have been covered. These are the bus topology, the crossbar topology, and the multistage topology. In static interconnection networks several limited connection interconnection network topologies including hypercube, mesh, ring etc. have been reviewed. Principles of designing product networks have been briefed. The discussion on the contemporary interconnection network is related to the design of limited connection hybrid interconnection networks.The focus of this research is to design a cost-effective and highly scalable topology for massively parallel systems.

The next chapter presents the design and analysis of a new interconnection network called “Scalable Twisted Hypercube (STH)”.

Chapter 3

The Design and Analysis of STH Interconnection Network

This chapter presents the design principles of a new processor interconnection topology called STH (Scalable Twisted Hypercube) to counter the poor scalability of twisted hypercube. Its suitability for use as multiprocessor interconnection network has also been explored. Various properties of the proposed topology have been derived, analyzed and compared with some other topologies on a number of standardized evaluation parameters. solution is given is the final section.

3.1 Related Work

High speed parallel computing is essential for modern research as the demand for more and more computing power is continuously increasing. In the recent past several high performance parallel computing platforms, consisting of more than 10,000 processing elements, have been installed. Examples of such installations are BlueGene/L at Lawrence Livermore National Laboratory, Blue Gene Watson at IBM and Columbia at NASA. Research projects in varied application areas such as environmental simulation, astronomy and engineering design have been undertaken to utilize this tremendous amount of computing power. Massively parallel systems are placing a major emphasis on scalable processor topologies with low

degree and diameter [92]. Many processor topologies such as Hyperstar [35], Hyper-Mesh [7] and Hex-Cell [192] have been proposed in literature capable of connecting hundreds or thousands of processors. Each of these topologies has its own inherent advantages as well as some limitations. The hypercube graph is a topology with a logarithmic diameter, simple node designation scheme, good connectivity, fault tolerance, vertex/ edge symmetry, partitionability, simple routing and existence of node disjoint parallel paths [185] [125]. From a research perspective, hypercube and its variants like twisted hypercube [6] have always been the popular choice of researchers and numerous processor interconnection topologies and effective computation algorithms based on them have been reported. Day and Tripathi [88] compared the topological properties of hypercube with Star Graph. Al-Sadi et al. [14] proposed a fault tolerant routing algorithm while Chiu and Chon [71] (1998) proposed efficient multicasting procedure for binary hypercube. Klasing [137] discussed efficient comparison of CCC network and a dynamic interconnection network named as "Scalable Optical Hypercube" which was reported by Louri and Sung [158]. Consequently, numerous commercial installations of parallel computers like Intel's iPSC Series (up to 128 processors), NCUBE/ 10 (up to 1024 processors) and FPSs T Series (up to 4096 processors) and nCube exist. All of these are based on the hypercube architecture. A parallel architecture is said to be scalable if it can be expanded (reduced) to a larger (smaller) system with a linear increase in its performance [156] [101]. This general rule indicates the desirability for providing equal chance for scaling up a system for improved performance and for scaling down a system for greater cost-effectiveness and/ or affordability. In spite of their claimed superiority the hypercube and its variant architectures like twisted hypercube have a major disadvantage with regard to scalability. It grows to its next higher dimension by a factor of 2. For example the nine-dimensional hypercube (HC(9)) has $2^9 = 512$ nodes, whereas a ten-dimensional hypercube (HC(10)) has $2^{10} = 1024$ nodes. This significant gap between the two consecutive sizes of the Hypercube is considered a major drawback of this topology and needs further attention for its improvement.

Moreover, in a hypercube like architecture moving from a particular dimension to the next higher dimension, the number of communication paths (wires) and the number of ports per processor increase significantly.Considering the above mentioned problems with the Hypercube like architectures, a number of topologies like Cross Cube [97], dBCube [67], Hierarchical Hypercube [161] , Cube Connected Cycles (CCC) [176] have been proposed. These topologies are essentially the modifications of the hypercube architecture and suffer from similar problems. For example, Cube Connected Cycles (CCC) topology, is obtained by replacing each node of an n-dimensional hypercube with a ring of size n. The result of this replacement is a constant degree (three) topology (i.e., CCC). But, such a replacement introduces certain unwanted features in the Hypercube architecture such as a large diameter and more complex routing. A recently proposed topology [192], that is not based on the hypercube architecture has been termed as the Hex-Cell. The maximum degree of Hex-Cell is 3 (constant) and it offers simpler routing. But, the diameter of a Hex-Cell consisting of N nodes is given by $4\sqrt{N/6} - 1$ which is significantly large compared to the diameter of hypercube. Also, it is irregular, vertex asymmetric and its bisection width is too low. So, Hex-Cell is not a suitable topology to design massively parallel systems. Other attempts for scaling the hypercube architecture have been made by means of hybrid topologies. Hyper-Star [35], Hype-Mesh [7], Arrangement Star Network [33] and Double-Loop Hypercube [236] are some examples of such attempts. A hybrid topology is derived from two or more existing topologies using a graph theoretic operator. Researchers have extensively used the Cartesian Product operator while designing hybrid topologies. Star-Hypercube Hybrid Interconnection Networks [240], Hyper-Mesh Multicomputers [5], Banyan-Hypercube Networks [235] and Hyper Petersen Network [86] are some examples of topologies based on Cartesian product of graphs. Generalized results for the Cartesian Product of topology graphs were first derived by Youssef [234] and later extended by Day and Ayyoub [87]. The most recently reported topology in this category has been termed as the Double Loop Hypercube (DLH) [236]. It has been derived as the Cartesian Product of

double loop topology with hypercube.

3.2 Preliminaries and Graph Theoretic Definitions

This design principles of STH addressed in this thesis use the standard graph theoretic terminology defined by Chartrand and Lesniak [65] and the definitions used by Alam and Kumar [16]. The important definitions related to the design of serially twisted hypercube topology are as given below:

Definition 3.2.1 The interconnection network topology is a finite undirected graph $G = (V, E)$, where $V = v_1, v_2, \ldots, v_n$ is the set of nodes (vertices) and $E = e_1, e_2, \ldots, e_m$ is the set of edges. Each vertex represents a processor and each edge a communication link between processors.

Definition 3.2.2 The degree of a vertex v in G, denoted as d_v is the number of edges incident on v. The minimum degree of a graph G, $\min\{d_v(G)|v \in V\}$ is denoted by $\delta(G)$. The maximum degree of G, $\max\{d_v(G)|v \in V\}$ is denoted by $\Delta(G)$. The degree of the graph is the maximum of the degrees of all vertices in the graph. Moreover, a graph is called regular if all of its vertices have the same degree.

Definition 3.2.3 The distance between two nodes u and v of a graph G denoted by $s_G(u, v)$ is the number of edges in G on the shortest path connecting i and j.

Definition 3.2.4 The diameter of a graph $G(V, E)$ denoted by D_G, is the maximum distance between any two nodes in G. The diameter provides a bound on communication between any two nodes. Mathematically: $D_G = \max\{s_G(u, v)|u, v \in V\}$

Definition 3.2.5 A graph $G(V, E)$ is vertex-symmetric, if for every pair of vertices u and v, $u, v \in V$, there exists an automorphism of the graph that maps u into v.

Definition 3.2.6 A graph G is said to be κ-connected (or κ-vertex connected, or κ-point connected) if there does not exist a set of $\kappa - 1$ vertices whose removal disconnects the graph, i.e., the vertex connectivity of G, $\kappa(G) \geq k$.

Definition 3.2.7 A vertex-induced subgraph (or simply an induced subgraph) is a subset of the vertices of a graph G together with any edges whose endpoints are both in this subset.

Definition 3.2.8 The $(\kappa - 1)$ - fault diameter of a κ-connected topology graph $G(V, E)$, $D_\kappa(G)$, is defined as:

$$D_\kappa(G) = \{\max\{D_{G-F}\} | F \subset V, |F| = \kappa - 1\}$$

where F is an induced subgraph of G.

Definition 3.2.9 An n-dimensional binary hypercube, Q_n, consists of 2^n nodes. Each node v, for $0 \leq v \leq 2n-1$, is labeled as n-bit binary string, $L(v)$. There is an edge between two nodes, u and v, if, and only if, their labels differ in exactly one bit position.

Definition 3.2.10 An n-dimensional twisted hypercube, TQ_n, is constructed from a Q_n as follows. Two distinct edges, say (a, b) and (c, d), which have no nodes in common, in a 4-cycle of the hypercube are selected. Now the two new edges (a, d) and (b, c) are created while the original edges (a, b) and (c, d) are removed. Such a twist does not effect the degree of a node (d) but the diameter of TQ_n is $\left\lceil \frac{(n+1)}{2} \right\rceil$ [6] [83]

Figure 3.1 shows Q_8 and TQ_8.

3.3 Design of Scalable Twisted Hypercube

The proposed $STH(m, n)$ network is based on two topologies - an m level Linearly Scalable Topology (LST), which is an improved version of the Linearly Extendible

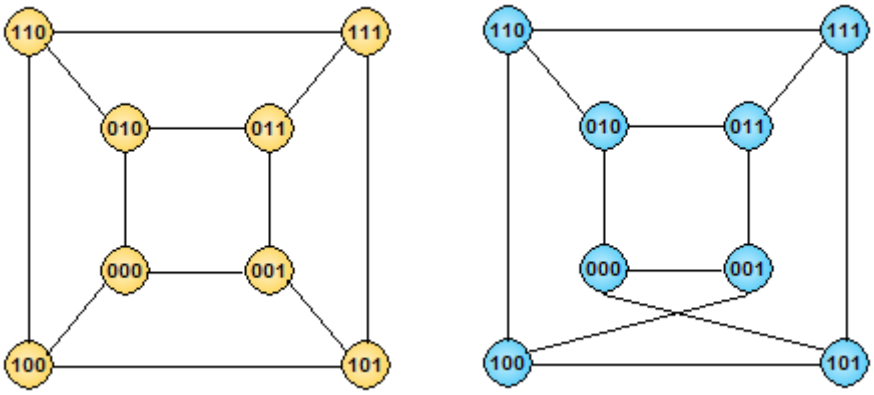

Figure 3.1: 8-node hypercube and twisted hypercube

Arm Topology [16], and the well known n-dimensional twisted hypercube topology. Whereas the level m LEA topology consists of $6m$ processors, the $LST(m)$ consists of $8m$ processors and an altered connectivity formula. Formally it is defined as follows:

Let m be a positive integer such that $m \geq 2$, then level m LST, denoted as $LST(m)$, is an undirected graph consisting of $N(= 8m)$ processing elements (vertices) labeled as $PE_0, PE_1, PE_2, \ldots PE_{(N-1)}$ for $0 \leq i < N$ and arranged in m columns, each of which consists of exactly eight processors. A link (edge) exists between processors PE_i and PE_j, iff , $(i+1) \quad modulo \quad N = j \quad or \quad (i+4) \quad modulo \quad N = j$. Alternately, for $m \geq 2$, $LST(m) = G(V, E)$, where,

- $V = \{PE_i | i \in I^+ \quad and \quad 0 \leq i < N\}$
- $E = \{(PE_i, PE_j) | j = ((i+1) \quad modulo \quad N) \quad OR \quad j = ((i+4) modulo N)\}$.

From the definition of LST it is obvious that each processor on a LST network provides two links to two distinct processors and receives two links from two distinct processors. An example of a $LST(m)$ is shown in Figure 3.2, where m equals to 3, and hence it consists of $3x8 = 24$ nodes.

The topology of LST network is simple, symmetric and scalable in architecture and it is 4-regular vertex (node) symmetric graph.

$STH(m, n)$ topology is obtained as a Cartesian product of level m-LST topology

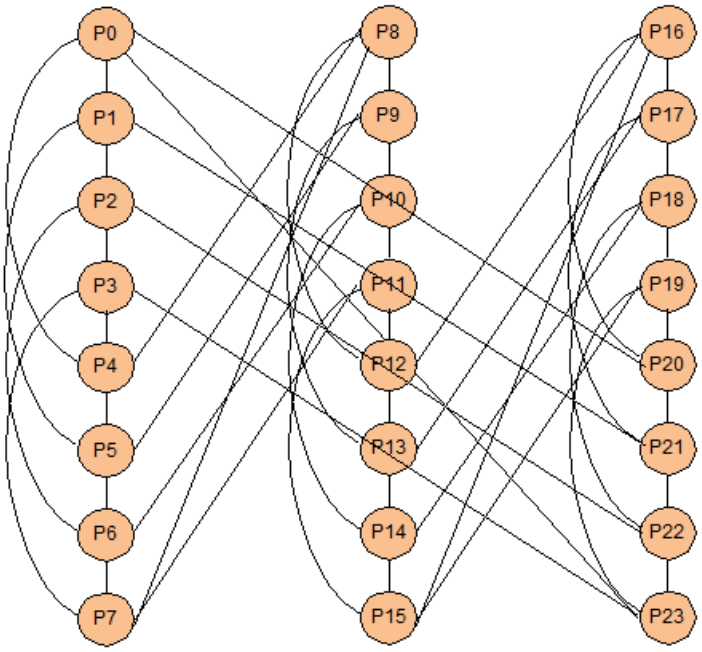

Figure 3.2: 24-node linearly scalable topology

and n-dimensional twisted hypercube topology, so it can formally be defined as follows: Let m and n be two positive integers such that $m \geq 2$. The level $mLST$ topology is represented by the undirected graph, $X_1(V_1, E_1)$ and the n-dimensional twisted hypercube topology is represented by the undirected graph, $X_2(V_2, E_2)$, then the STH topology, is an undirected graph, $X(V, E)$, where V and E are given by:

- $V = \{(a, b)|a \in V_1 \quad and \quad b \in V_2\}$, and
- For any $x = (a, b)$ and $y = (c, d)$ in V, (x, y) is an edge in E if, and only if, (a, c) is an edge in E_1 and $b = d$ or (b, d) is an edge in E_2 and $a = c$.

Figure 3.3 on page 61 shows STH (2, 3), which has been obtained as a Cartesian Product of LST (2) and TQ3. From Figure 3.3 it is fairly obvious that $STH(m, n)$ can be obtained from an n-dimensional twisted hypercube such that each of its node is replaced by $LST(m)$ and connections are established according to the above definition. For the sake of simplicity, each node in $STH(m, n)$ is represented as two tuples, (u, v), where PE_u is a node in LST network and PE_v is a node in

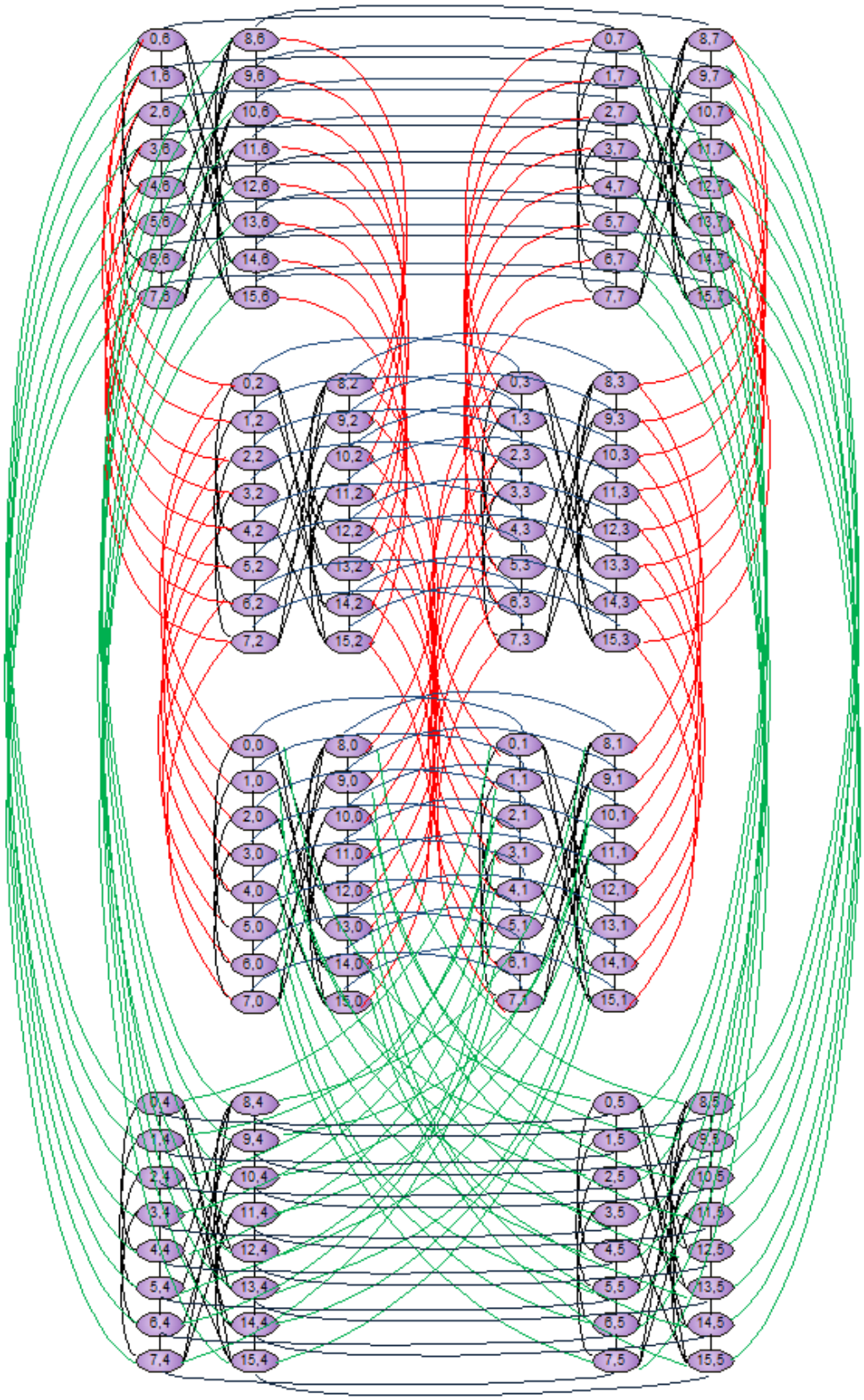

Figure 3.3: 128-node serially twisted hypercube topology

TQ_n network. More precisely, in an $STH(m,n)$ network we define the following two kinds of connections:

LST Connection: A node $(u,v) \in STH$, is adjacent to another node $(u',v) \in STH$, if and only if, $u' = ((i+1) \quad modulo \quad N)$ OR $u' = ((i+4) \quad modulo \quad N)$.

Twisted Hypercube Connection: A node $(u,v) \in STH$, is adjacent to another node $(u,v') \in STH$ if, and only if, $L(v)$ and $L(v')$, where $L(v)$ and $L(v')$ denote the n-bit binary strings corresponding to v and v' respectively following definition 3.2.10.

3.3.1 Topological Properties of STH

In this section we derive various topological properties of the $STH(m,n)$ network.

Theorem 3.3.1 Node Formula: The total number of nodes in $STH(m,n)$ network is given by $m.2^{n+3}$.

Proof: From the construction method of $STH(m,n)$, we know that $STH(m,n)$ can be obtained by replacing each of the n nodes of an n-dimensional twisted hypercube network (TQ_n) by an m level LST network consisting of $8m$ nodes. So, replacement of one node of TQ_n, introduces $8m$ new nodes in the network. It implies that replacement of all n nodes will introduce $8m.2^n = m.2^{n+3}$ nodes in the network.

Theorem 3.3.2 Degree Formula: The degree of $STH(m,n)$ is $(n+4)$.

Proof: It follows directly from the definition of $STH(m,n)$. Two components of $STH(m,n)$ are $LST(m)$. LST is a is 4-regular vertex (node) symmetric graph and the degree of TQ_n is n. Hence the degree of $STH(m,n)$ is $(4+n)$. Fig.2 further confirms this expression. In $STH(2,3)$, the degree of each node is 7.

Theorem 3.3.3 Diameter Formula: Diameter of $STH(m,n)$ is given by $(m+1) + \left\lceil \frac{(n+1)}{2} \right\rceil$.

Proof: It is easy to analyze that:

Diameter of 16 node LST, i.e., $D_{LST}(2) = 3$.

Diameter of 24 node LST, i.e., $D_{LST}(3) = 4$.

Diameter of 32 node LST, i.e., $D_{LST}(4) = 5$.

So, the Diameter of $8m$ node (level m) LST i.e. $D_{LST}(m) = m + 1$

The diameter of an n-dimensional twisted hypercube is $\left\lceil \frac{(n+1)}{2} \right\rceil$. So, the diameter of STH(m, n) is given by the following expression [234] [87]:

$$D_{STH}(m, n) = (m + 1) + \left\lceil \frac{(n + 1)}{2} \right\rceil$$

The proof can also be obtained using the analytical method as follows. Let $x = (p, q)$ and $y = (r, s)$ be two vertices of STH graph. x would be at a maximum distance from y in STH if, and only if, p is at a maximum distance i.e. $(m+1)$ from r in LST and q is at a maximum distance from s in TQ_n. Clearly, the maximum distance (diameter) of $STH(m, n)$ is $(m + 1) + \left\lceil \frac{(n+1)}{2} \right\rceil$.

Theorem 3.3.4 Symmetry: $STH(m, n)$ is a regular vertex (node) symmetric graph.

Proof: Each node of $STH(m, n)$ has the degree $4+n$ so, its obvious that $STH(m, n)$ is a regular graph.

Let $LST(m)$ network is represented by a graph $X(V, E)$, and ϕ be a permutation of the vertex set V, of $LST(m)$. Then, for any edge $x = (a, b), x \in E$ and $a, b \in V$, we have $\phi(x) = (\phi(a), \phi(b))$, is also an edge. It shows that graph X (representing $LST(m)$ is isomorphic to itself (automorphism). Clearly LST is a vertex symmetric graph. Twisted Hypercube belongs to the family of graphs known as Cayley Graphs and every Cayley Graph is vertex symmetric [87]. Hence, Twisted Hypercube graph is also vertex symmetric. If two given graphs X and Y are vertex symmetric then $X \otimes Y$ is also vertex symmetric [234]. From this discussion, it is obvious that $STH(m, n)$ is vertex symmetric.

Theorem 3.3.5 Bisection Width (ω): The Bisection Width (ω) of $STH(m, n)$ is given by $(3m - 1).2^{n+1}$ *or* $(3N - 8).2^{n-2}$, where $N = 8m$.

Proof: The Bisection Width of an interconnection network is defined as the minimum number of edges (wires) cut to split a network into two parts each having the same number of nodes. Bisection Width of $LST(m)$ can be calculated as shown in Table 3.1:

m	N = 8m	No. of Wires to Cut
2	16	20 = (12*2-4)
3	24	32 = (12*3 -4)
4	32	44 = (12*4-4)
5	40	56 = (12*5-4)
⋮	⋮	⋮
⋮	⋮	⋮
m	8m	(12*m-4)

Table 3.1: Bisection Width of $LST(m)$

$STH(m, n)$ is obtained by replacing each of the $8m$ nodes of $LST(m)$ by an n-dimensional twisted hypercube. We have already seen that the bisection width of $LST(m)$ is $(12m - 4)$. The bisection width of TQ_n is 2^{n-1}.Hence, the Bisection Width of STH(m, n) is $(12m - 4).2^{n-1}$, i.e.

$$\omega = (12m - 4).2^{n-1} = (3m - 1).2^{n+1} = (3N - 8).2^{n-2}$$

Theorem 3.3.6 Average Internode Distance: The average distance of STH(m, n) is given by:

$$\overline{d}_{STH} = \frac{m(4m + 1)}{(8m - 1)^2} + \frac{n.2^{n-1} - 1}{2^n - 1}$$

Proof: The average distance $\overline{d}_X$ of a graph X, consisting of N nodes is given by the following equation:

$$\overline{d}_X = \frac{\sum_{i=1}^{N}\sum_{j=1}^{N} d(i, j)}{N(N - 1)}, \quad where \quad i \neq j$$

The average distance of a node v denoted as $\overline{d}_v$ is obtained from the following equation:

$$\overline{d}_v = \frac{\sum_{j=1}^{N} d(i, j)}{(N - 1)}, \quad where \quad i \neq j$$

Average Distance of LST can be calculated as follows. Average distances for node 1 in different level LST networks are shown in the Table 3.2: Clearly, the average

m	**No. of Nodes**	$\overline{\mathbf{d}}_{\mathbf{1}}$
2	16	$(1+1+1+1+2+2+2+2+2+2+2+2+3+3+3+3)/15 = (4\times 1+7\times 2+4\times 3)/15 = 30/15$
3	24	$(1+1+1+1+2+2+2+2+2+2+2+2+2+3+3+3+3+3+3+3+4+4+4+4)/23 = (4\times 1+8\times 2+7\times 3+4\times 4)/23 = 57/23$
4	32	$(4\times 1+8\times 2+8\times 3+7\times 4+4\times 5)/31 = 92/31$
$\vdots$	$\vdots$	$\vdots$
$\vdots$	$\vdots$	$\vdots$
m	$8k$	$(4\times 1+8\times 2+8\times 3+\cdots+8\times(m-1)+7\times m+4\times(m+1)/(8m-1)$

Table 3.2: Bisection Width of $LST(m)$

distance of first node in $LST(m)$ is be given by:

$$\overline{d}_1 = \frac{(4\times 1+8\times 2+8\times 3+\cdots+8\times(m-1)+7\times m+4\times(m+1)}{(8m-1)} = \frac{m(4m+7)}{(8m-1)}$$

As LST is a 4-regular, node symmetric graph, average distance for each node is same and can be given by the above expression. Thus, the average distance of LST graph i.e. $\overline{d}_{LST}$ can be calculated as follows:

$$\overline{d}_{LST} = \frac{\sum_{i=1}^{N}\sum_{j=1}^{N} d(i,j)}{N(N-1)} = \frac{m(4m+7)}{(8m-1)^2}$$

We know that if X_1 and X_2 are two graphs with average distances $\overline{d}_{X_1}$ and $\overline{d}_{X_2}$ respectively, then the average distance $\overline{d}_X$ of $X_1 \otimes X_2$ is given by :

$$\overline{d}_X = \overline{d}_{X_1} + \overline{d}_{X_2}$$

So, for $STH(m,n)$ we have:

$$\overline{d}_{STH} = \frac{m(4m+1)}{(8m-1)^2} + \frac{n.2^{n-1}-1}{2^n-1}$$

Theorem 3.3.7 Fault Diameter: The $(n+3)$-fault diameter of $STH(m,n)$ is bounded by the following inequality:

$$D_{n+4}(STH) \leq (m+n+2)$$

Proof: It has been shown by Xu et al. [225] that if X_1 and X_2 are two undirected graphs such that X_1 is κ_1 connected with $D_{\kappa_1}(X_1)$ as $(\kappa_1 - 1)$ fault diameter and X_2 is κ_2 connected with $D_{\kappa_2}(X_2)$ as $(\kappa_2 - 1)$ fault diameter then if, the product graph $X_1 \otimes X_2$ is $(\kappa_1 + \kappa_2)$ connected, its diameter is bounded by the following inequality:

$$D_{\kappa_1+\kappa_2}(X_1 \otimes X_2) \leq D_{\kappa_1}(X_1) + D_{\kappa_2}(X_2) + 1$$

$STH(m, n)$ is a product graph of $LST(m)$ and TQ_n. Using Whitneys Theorem [217], it is easy to prove that LST is 4-connected i.e. $\kappa_{LST} = 4$ and its 3-fault diameter i.e. $D_4(LST)$ is $(m + 2)$. The connectivity of n-dimensional Twisted Hypercube is n [6] i.e. $\kappa_{TQ_n} = n$ and its $(n - 1)$ fault diameter is $(n - 1)$ i.e. $D_n(TQ_n) = n - 1$. Also,

$$\kappa_{STH} = \kappa_{LST} + \kappa_{TQ} = n + 4$$

Hence, using the results of Xu et al. [225] it is easy to prove that $(n + 3)$- fault diameter of $STH(m, n)$ is bounded by the following inequality:

$$D_{n+4}(STH) \leq (m + n + 2)$$

3.3.2 Routing on STH Network

Routing Algorithm is an important factor which affects the performance of an interconnection network. Routing involves the process of identification of a set of permissible paths that may be used by a message to reach its destination, and a function that selects one path from the set of permissible paths. We have two approaches to develop the routing algorithms for interconnection networks [101] Adaptive Routing and Deterministic or Oblivious Routing. This work favours the deterministic routing techniques due to their simplicity.

The following sub-sections describe the unicast and broadcast routing algorithms for STH interconnection networks.

Unicast Routing Algorithm for $STH(m, n)$

Assuming that on $STH(m, n)$ network a node $u = (a, b)$ intends to send a message to another node $v = (a, b)$. We present the following optimal $O(m + n)$ (the worst case complexity) algorithm for the required routing.

Variables and Procedures:

u, v: Source and Destination Nodes

m : Level of LST Network

n: Dimension of Twisted Hypercube

R_i: Variable to record movements along path i in LST network.

K_i: List of nodes traversed along path i.

$add(K, p)$: A function that inserts a node p in the list L.

$follow(K)$: A function that forwards a message from source to destination along the path K.

$spath$: Shortest Path

$splength$: Length of the Shortest Path

$l(K)$: Last node in the list K.

$procedureroute()$: Main Routing Procedure

$subprocedureroutLST()$: Sub procedure for dealing with routing on LST part of STH Network.

$subprocedureroutTH()$: Sub procedure for dealing with routing on TQ part of STH Network.

The main routing procedure is listed below:

Twisted hypercube routing has been discussed extensively in literature [9] [6] and the algorithm for $subprocedureroutLST()$ is listed below. The algorithm presented here makes good use of the topological properties of STH architecture and is straightforward. Given a source processor PE_s and a destination processor PE_d, the algorithm attempts to find the shortest path along all the four LST nodes connected to PE_s, using four different variables. When the destination is reached along a particular path, the algorithm terminates declaring the path in question as shortest path and avoiding rest of the paths. The message is forwarded along the

Algorithm 1 Routing on $STH(m, n)$ Network

$procedurerout(u, v, m, n)$

begin

if $(a \neq a'$ **and** $b = b')$ **then**

call $subprocedureroutLST(a, a', m)$

$exit()$

end if

if $(a = a'$ **and** $b \neq b')$ **then**

call $subprocedureroutTH(b, b', n)$

$exit()$

end if

if $(a \neq a'$ **and** $b \neq b')$ **then**

call $subprocedureroutLST(a, a', m)$

call $subprocedureroutTH(b, b', n)$

end if

end

path returned by the algorithm.

$subprocedureroutLST(x, y, p)$
begin
$splength \leftarrow 1$
$m \leftarrow 8 \times p$
for i =1 to 4 **do**
$add(K_i, PE_x)$
end for
$R_1 \leftarrow x - 4$
if $(R_1 < 0)$ **then**
$R_1 \leftarrow m + R_1$
end if
$R_2 \leftarrow x + 4$
if $(R_2 \geq m)$ **then**
$R_2 \leftarrow |m - R_2|$
end if
$R_3 \leftarrow x + 1$
if $(R_3 \geq m)$ **then**
$R_3 \leftarrow |m - R_3|$
end if
$R_4 \leftarrow x - 1$
if $(R_4 < 0)$ **then**
$R_4 \leftarrow m + R_4$
end if
for i =1 to 4 **do**
$add(K_i, PE_{R_i})$
end for
while $true$ **do**
if $(R_1 = y$ **or** $R_2 = y$ **or** $R_3 = y$ **or** $R_4 = y)$ **then**
$break$

end if

$hopsreq \leftarrow (y - X_1)$

if $(hopsreq \geq 4)$ **then**

$R_1 \leftarrow R_1 - 4$

else

if $(hopsreq > 0)$ **then**

$R_1 \leftarrow R_1 + 1$

else

$R_1 \leftarrow R_1 - 1$

end if

end if

end

if $(R_1 < 0)$ **then**

$R_1 \leftarrow m + R_1$

end if

$hopsreq \leftarrow (y - R_2)$

if $(hopsreq \geq 4)$ **then**

$R_2 \leftarrow R_2 + 4$

else

if $(hopsreq > 0)$ **then**

$R_2 \leftarrow R_2 + 1$

else

$R_2 \leftarrow R_2 - 1$

end if

end if

if $(R_2 \geq m)$ **then**

$R_2 \leftarrow |m - R_2|$

end if

$R_3 \leftarrow R_3 + 1$

if $(R_3 \geq m)$ **then**

$R_3 \leftarrow |m - R_3|$

end if

$R_4 \leftarrow R_4 - 1$

if $(R_4 < 0)$ **then**

$R_4 \leftarrow m + R_4$

end if

for i =1 to 4 **do**

$add(K_i, PE_{R_i})$

end for

$splengthsplength + 1$

end while

for i =1 to 4 **do**

if $(l(K_i) = PE_y)$ **then**

$spath \leftarrow K_i$

end if

end for

$follow(spath)$

end

Execution: An Example

We illustrate the execution of algorithm 1 with the help of an example. Assume that in a 192-node STH network $(STH(3,3))$, node $u(2,6)$ has a message to send to node v$(19,3)$. Then according to the above algorithm the message is routed as follows:

1. Here, $a = 2, b = 6, a' = 19 \quad and \quad b' = 3$, therefore the procedure

$$subprocedureroutLST(2,19,3)$$

is executed as follows:

Steps 1 to 12:

$splength = 1, m = 8 * 3 = 24, K_1 = K_2 = K_3 = K_4 = PE_2, R_1 = 22, R_2 = 6, R_3 = 3, R_4 = 1, K_1 = PE_2.PE_{22}, K_2 = PE_2.PE_6, K_3 = PE_2.PE_3, K_4 = PE_2.PE_1$.

Table 3.3 on page 73 illustrate the variable updates as the algorithm proceeds:

Steps 13-38: Clearly, List K_1 is selected as it contains the shortest path. The message is now routed along the path Contained in K_1. And, now the message is available at node $(19, 6)$.

2. From node $(19, 6)$ the message is sent to node $(19, 3)$ using

$$subprocedureroutTH(6, 3, 3)$$

3.3.3 Broadcasting On STH Network

Assuming that node X intends to broadcast a message on a $STH(m, n)$ network, then such an action will be performed as follows:

1. X will send message to all the nodes of that $LST(m)$ network on which X itself resides..

2. Then on every twisted hypercube of $STH(m, n)$, the node that receives the message forwards it to all other nodes of that twisted hypercube.

3.4 STH Analysis and Comparative Study

Designing a high performance interconnection network is the most challenging task in the field of parallel computers. It is essentially impossible to fairly compare interconnection networks, simply because there are too many parameters and topological properties. The most suitable way for evaluating a new topology is to conduct a comparative study between the topological properties of both; the

R_1	R_2	R_3	R_4	K_1	K_2	K_3	K_4	$splength$
21	10	4	0	$PE_2PE_{22}PE_{21}$	$PE_2PE_6PE_{10}$	$PE_2PE_3PE_4$	$PE_2PE_1PE_0$	2
20	14	5	23	$PE_2PE_{22}PE_{21}PE_{20}$	$PE_2PE_6PE_{10}PE_{14}$	$PE_2PE_3PE4_PE_5$	$PE_2PE_1PE_0PE_{23}$	3
19	18	6	22	$PE_2PE_{22}PE_{21}PE_{20}PE_{19}$	$PE_2PE_6PE_{10}PE_{14}PE_{18}$	$PE_2PE_3PE_4PE_5PE_6$	$PE_2PE_1PE_0PE_{23}PE_{22}$	4

Table 3.3: LST Shortest Path Route Calculations

proposed topology and the other topologies that are familiar by their appealing topological properties. Our study is based on the most common criteria used for evaluating interconnection networks such as degree, diameter, cost, cost factor, bisection width, cost of one-to-all broadcasts, cost of all-to-all broadcasts, average node distances and message traffic density. The results show that the proposed STH network exhibits the appealing properties of its constituent interconnection networks and more importantly eliminates their drawbacks.

The following sub sections present comparative study of the STH network with some well known topologies namely Hypercube (HC) [84], 2D Mesh (ME) [1], Hex-Cell (HX) [192], Hyper-mesh (HM) [5] and Double Loop Hypercube (DLH) [236].

In topologies such as hyperstar or hyperpeterson, for obtaining the desired network size or a value close to desired network size, the only way is to increase the dimension of the hypercube or star graph. Double Loop Hypercube can be scaled up to almost any network size without increasing the dimension of hypercube. As pointed out earlier Hex-Cell is not a product graph. It is obtained by fabricating several Hex-Cells together. The topology is highly scalable and offers simple routing. Our objective in this thesis is to design a highly scalable and cost effective topology, so after considering the scalability merits of Hex-Cell it has been chosen for comparative study.

Table 3.4 on page 75 summarizes topological properties of STH and other topologies. For a fair comparison we have chosen a hypercube of fixed degree (7). Tables 3.5 to 3.9 on pages 76-80 present calculated values of several parameters related to the topologies under consideration.

Diameter (D) Comparison

Figure 3.4 on page 81 compares the diameters of all the topologies of interest. It has been observed that the diameter of STH is smallest among all hybrid topologies by up to more than 10^6 nodes, when we scale up the network to a fixed hypercube dimension. The diameter may be further reduced by choosing larger

Topological Property	$DLH(m,n)$	$HX(i)$	$STH(m,n)$	$HC(n)$	$ME(r,c)$	$HM(n,r,c)$
No. of Nodes (N)	$m.2^{n+2}$	$6(2i-1)$	$m.2^{n+3}$	2^n	$r \times c$	$2^n(r \times c)$
Degree (d)	$3+n$	≤ 3	$4+n$	n	4	$n+4$
Diameter (D)	$m+n+1$	$4\sqrt{2i-1}-1$	$(m+1)+\lceil\frac{n+1}{2}\rceil$	n	$(r+c-2)$	$n+r+c-2$
No. of Links (L)	$m.2^{n+1}.(3+n)$	$3(3i^2-i)$	$m.2^{n+2}.(4+n)$	$n.2^{n-1}$	$2(r \times c)$	$2^{n-1}(r \times c)(n+)$
Cost Factor ($d*D$)	$(3+n)(m+n+1)$	$3.(4\sqrt{2i-1}-1)$	$(4+n).[(m+1)+\lceil\frac{n+1}{2}\rceil]$	n^2	$4(r+c-2)$	$(n+4)(n+r+c-2)$
Cost ($L*D$)	$(m+n+1).m.2^{n+1}.(3+n)$	$(4\sqrt{2i-1}-1).(9i^2-3i)$	$((m+1)+\lceil\frac{n+1}{2}\rceil).(m.2^{n+2}.(4+n)$	$n^2.2^{n-1}$	$2(r+c-2)(r \times c)$	$(n+r+c-2).2^{n-1}(r \times c)(n+4)$
Bisection Width (ω)	$m.2^n$	$2.i$	$(3m-1).2^{n+1}$	2^{n-1}	$\sqrt{(r \times c)}$	$2^{n-1}\sqrt{r.c}$

Table 3.4: Topological Properties - STH and Other Topologies

No. of Nodes	Diameter						No. of Links					
	$DLH(m,7)$	HX	$STH(m,7)$	$HM(r,c,7)$	HC	$ME(r,c)$	$DLH(m,7)$	HX	$STH(m,7)$	$HM(r,c,7)$	HC	$ME(r,c)$
1024	10	51	7	11	10	62	5120	1497	5120	5632	11264	2048
2048	12	72	7	13	11	94	10240	3017	11264	11264	24576	4096
4096	16	103	9	17	12	126	20480	6066	22528	22528	53248	8192
8192	24	146	13	25	13	190	40960	12178	45056	45056	114688	16384
16384	40	208	21	41	14	254	81920	24420	90112	90112	245760	32768
32768	72	294	37	73	15	382	163840	48931	180224	180224	524288	65536
65536	136	417	69	137	16	510	327680	97991	360448	360448	1114112	131072
131072	264	590	133	265	17	1022	655360	196165	720896	720896	2359296	524288

Table 3.5: Diameter and No. of Links comparison

No. of Nodes	Cost						Cost Factor					
	$DLH(m,7)$	HX	$STH(m,7)$	$HM(r,c,7)$	HC	$ME(r,c)$	$DLH(m,7)$	HX	$STH(m,7)$	$HM(r,c,7)$	HC	$ME(r,c)$
1024	100	153	70	121	100	248	51200	76337	35840	61952	51200	126976
2048	120	216	77	143	121	376	122880	217193	78848	146432	123904	385024
4096	160	309	99	187	144	504	327680	624758	202752	382976	294912	1032192
8192	240	438	143	275	169	760	983040	1777863	585728	1126400	692224	3112960
16384	400	624	231	451	196	1016	3276800	5079200	1892352	3694592	1605632	8323072
32768	720	882	407	803	225	1528	11796480	14385507	6668288	13156352	3686400	25034752
65536	1360	1251	759	1507	256	2040	44564480	40862024	24870912	49381376	8388608	66846720
131072	2640	1770	1463	2915	289	4088	1.73E+08	115737111	95879168	191037440	18939904	535822336

Table 3.6: Cost and Cost Factor comparison

No. of Nodes	Cost of One-to-All Broadcast						Cost of All-to-All Broadcast					
	$DLH(m,T)$	HX	$STH(m,T)$	$HM(r,c,T)$	HC	$ME(r,c)$	$DLH(m,T)$	HX	$STH(m,T)$	$HM(r,c,T)$	HC	$ME(r,c)$
1024	11022.4	58603.698	7670.0734	10000	10000	69159.416983	10102.3	51341	7102.30000	11093	11186.00	62255.750000
2048	13225.04	81187.077	7587.8089	12000	11000	103014.688436	12204.7	72682.333	7186.0909	13186	12341.00	94511.750000
4096	17581.109	114142.35	9819.0426	16000	12000	136569.708499	16409.5	104365	9372.2727	17372	13630.00	127023.750000
8192	26171.732	159411.63	14206.941	24000	13000	203167.721676	24819.1	148730.33	13744.636	25744	15170.00	192047.750000
16384	43099.199	224152.76	22822.06	40000	14000	269351.713715	41638.3	213461	22489.364	42489	17184.00	258095.750000
32768	76495.136	313342.4	39754.387	72000	15000	401008.06318	75276.7	304922.33	39978.818	75978	20095.00	390191.750000
65536	142538.53	440173.64	73125.061	136000	16000	532086.155497	142553.5	438845	74957.727	142957	24710.00	526383.750000
131072	273481.46	617699.42	139103.94	264000	17000	1053590.254282	277107.1	633690.33	144915.55	276915	32563.00	1087535.750000

Table 3.7: Cost of One-to-All and All-to-All Broadcast

No. of Nodes	Average Distance				Message Traffic Density			
	$DLH(m,7)$	HX	$STH(m,7)$	$HM(r,c,7)$	$DLH(m,7)$	HX	$STH(m,7)$	$HM(r,c,7)$
1024	3.48932	5.004888	3.165079	3.314567	0.697864	1.000978	0.633016	0.602649
2048	3.485549	5.502687	3.653018	3.321011	0.69711	1.000489	0.664185	0.60382
4096	3.523719	6.001465	3.615419	3.321167	0.704744	1.000244	0.657349	0.603849
8192	3.773481	6.500794	3.598294	3.514567	0.754696	1.000122	0.654235	0.639012
16384	4.89913	7.000427	3.590117	4.788671	0.979826	1.000061	0.652749	0.870667
32768	9.215962	7.500229	3.586121	8.778891	1.843192	1.000031	0.652022	1.596162
65536	23.69883	8.000122	3.584146	21.99017	4.739765	1.000015	0.651663	3.998213
131072	65.81525	8.500065	3.583163	44.77694	13.16305	1.000008	0.651484	8.141262

Table 3.8: Average Internode Distance and Message Traffic Density Comparison

No. of Nodes	Bisection Width					
	$DLH(m,7)$	HX	$STH(m,7)$	$HM(r,c,7)$	HC	$ME(r,c)$
1024	256	26.127891	640	181	512	32.000000
2048	512	36.950417	1280	256	1024	45.254834
4096	1024	52.255781	2816	362	2048	64.000000
8192	2048	73.900834	5888	512	4096	90.509668
16384	4096	104.51156	12032	724	8192	128.000000
32768	8192	147.80167	24320	1024	16384	181.019336
65536	16384	209.02313	48896	1448	32768	256.000000
131072	32768	295.60334	98048	2048	65536	512.000000

Table 3.9: Bisection Width Comparison

dimensions of hypercube part.

Degree (d) Comparison

For network of any size, the node degree of $STH(m,n)$ is equal to the degree of Hypermesh which is admittedly larger than the degree of any other networks under consideration. For a fixed value of hypercube degree (7), the degree of DLH is 10 whereas the degree of STH and HM is 11.

Cost Factor (ξ) and Cost (ζ) Comparison

A network with high node degree is expensive and a network with large diameter suffers form high latency. It is always desirable to have a topology with both small degree (low cost) and small diameter (low latency). Thus, for an interconnection network, the product of degree (d) and diameter (D) is defined as the cost factor (ξ) [19]. Other metrics used to describe the cost (ζ) of an interconnection network topology is the product of the number of Links (L) and the diameter [157]. Figures 3.5 to 3.6 on page 82 compare the cost factor and cost respectively, for the topologies under consideration. From these figures, it is evident that STH outperforms other hybrid topologies when they are either evaluated using Cost Factor or

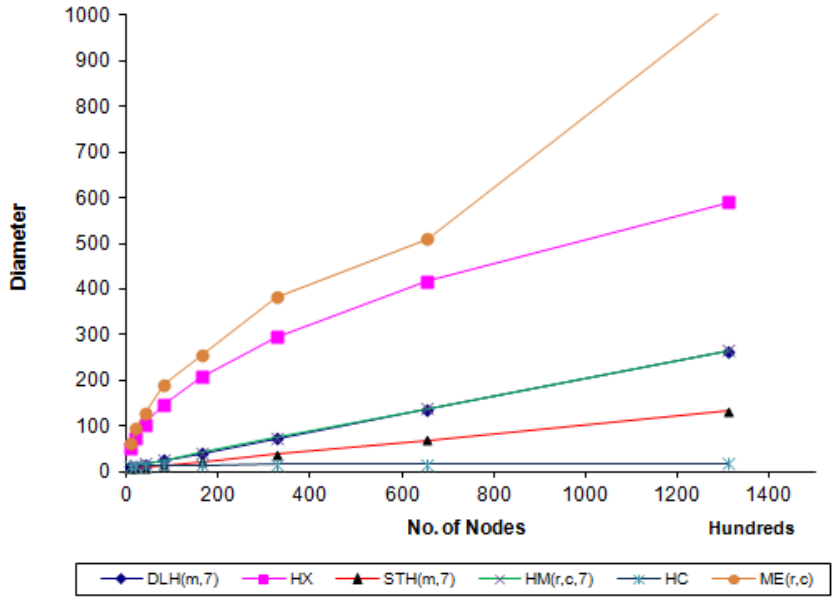

Figure 3.4: Diameter (D) Comparison

Cost.

No. of Links (L) Comparison

Figure 3.7 compares the number of links offered by the STH and other topologies of interest. It confirms that, in spite of its lowest cost, among all hybrid topologies, STH has maximum number of links for a given network size. In other words it can also be said that among all hybrid topologies STH is the most fault tolerant.

Bisection Width (ω) Comparison

The Bisection Width of an interconnection network is defined as the minimum number of edges (wires) cut to split the network into two parts each having the same number of nodes. Bisection width reflects the wiring density of an interconnection network and provides a good indicator of the maximum communication bandwidth along the bisection of an interconnection network. Interconnection network designers strive for high bisection width. Figure 3.8 on page 84 shows that the bisection width of STH is best among all the hybrid topologies. Hex-Cell

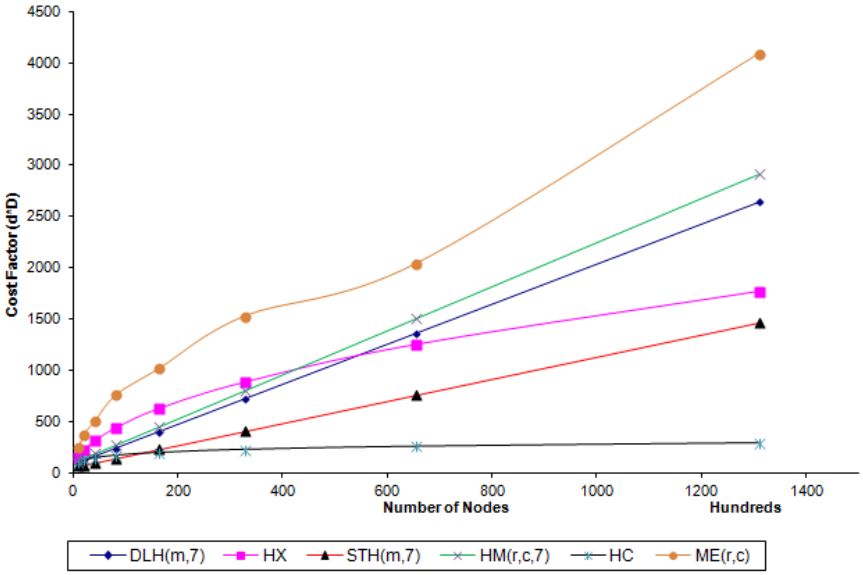

Figure 3.5: Cost Factor

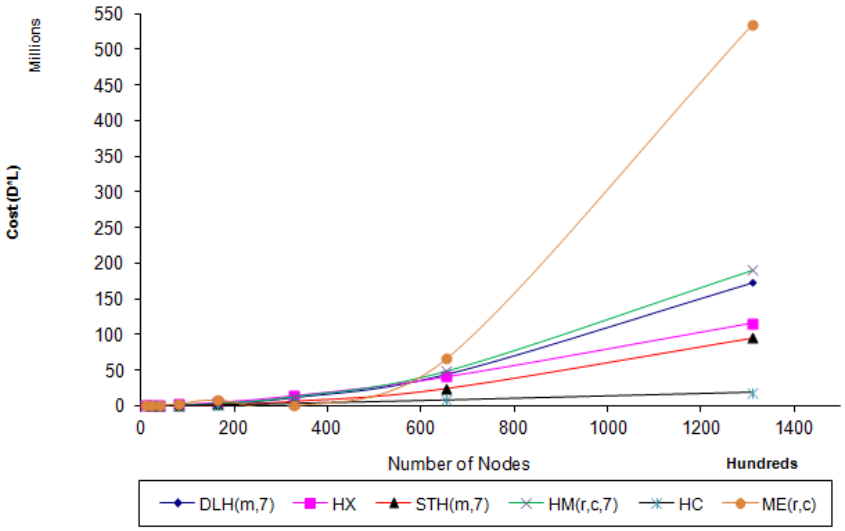

Figure 3.6: Cost

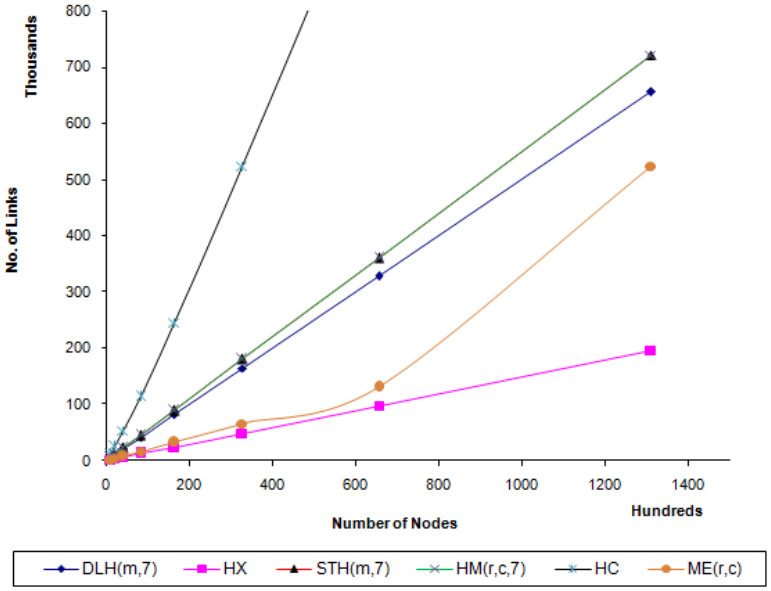

Figure 3.7: Number of Links Comparison

(HX) and Mesh have very low bisection widths hence they are not suitable for designing massively parallel systems.

Scalability Comparison

To compare the network size support capability of all topologies under consideration, the percentage of deviation allowed in network size representation (ψ), is defined as follows:

$$\psi = \frac{|\alpha - \beta|}{\alpha} \times 100$$

Where α is the actual requested network size and β is the available network size. Starting with $\psi = 2$ (i.e. deviation allowed from actual network size = 2%), for each topology, the number and percentage of network sizes available (in the range 1 to 50000) with flexibility of ψ are found. Then, ψ is gradually increased up to 20. The results are presented in Table 3.10 and Figure 3.9 on page 85.

From Figure 3.9 and Table 3.10, it is clear that the scalability of the STH network is at par with Hypermesh, DLH and Hex-Cell. The STH and other hybrid

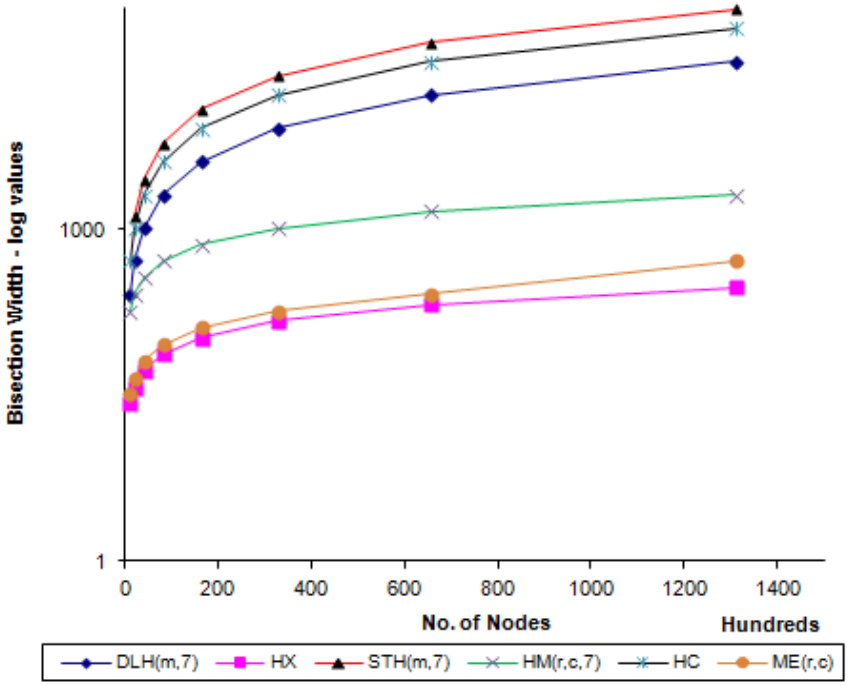

Figure 3.8: Bisection Width Comparison

ψ	$DLH(m,7)$		HX		$STH(m,7)$		TQ		$HM(r,c,7)$	
	No.	%age	No.	%age	No.	%age	No.	%age	No.	%age
2	49898	99.83	49851	99.70	49797	99.59	2627	5.25	48924	97.84
5	49959	99.91	49939	99.87	49919	99.83	6572	13.14	49825	99.64
8	49974	99.94	49962	99.92	49947	99.89	10554	21.1	49933	99.86
10	49978	99.95	49968	99.93	49957	99.91	13239	26.47	49955	99.90
15	49985	99.96	49981	99.96	49969	99.93	20115	40.22	49984	99.96
20	49986	99.97	49984	99.96	49973	99.94	27299	54.59	49988	99.97

Table 3.10: Scalability Comparison

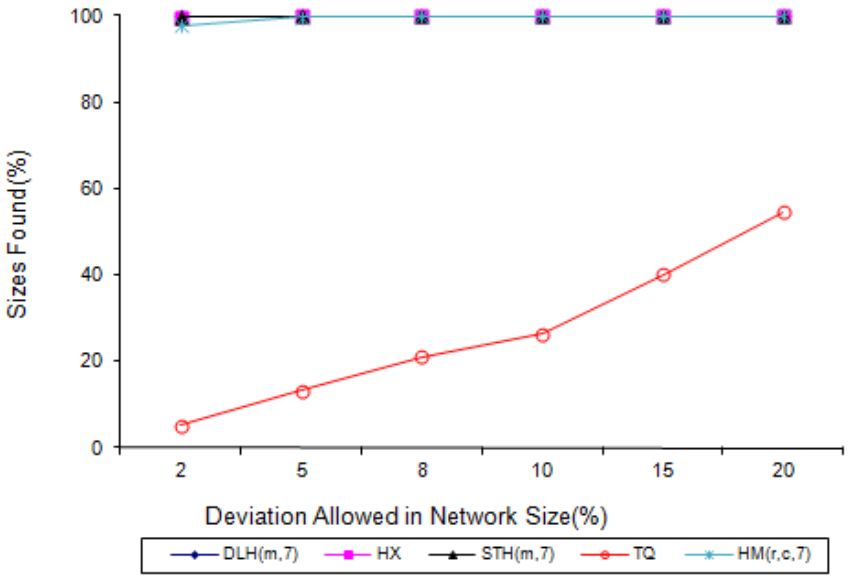

Figure 3.9: Scalability Comparison

topologies are capable of representing more than 99% of the network sizes in the range of 1 to 50000, even with a very low percentage in deviation $(\psi = 2)$ allowed in the network size representation.

Scalability of Twisted Hypercube (TQ) on the other hand is really poor. TQ is capable of representing at the most 55% (approx.) network sizes when maximum percentage deviation in network size representation is allowed $(\psi = 20)$.

Costs of One-to-All and All-to-all Broadcasts

The lower bound on the cost of one-to-all broadcast on a d-port network (L_G^d), is given by the following equation [115]:

$$L_G^d = (\sqrt{Ma/bd} - \sqrt{D-1})^2$$

and, the lower bound (U_G^d) on the cost of all-to-all broadcast is given by the following equation:

$$U_G^d = \frac{(N-1)a}{d} + Db$$

Where, N= No. of nodes in the network, d = Degree of the network, D = Diameter of the network, M = Length of the message, a= Unit transmission cost, b = Network latency.

Figures 3.10 to 3.11 show that the cost of one-to-all broadcasts and the cost of all-to-all broadcasts are lowest for the STH, compared to other hybrid topologies. All the calculations have been made assuming $M = 1024$, $a = 1\mu s$, $b = 1000\mu s$ [115].

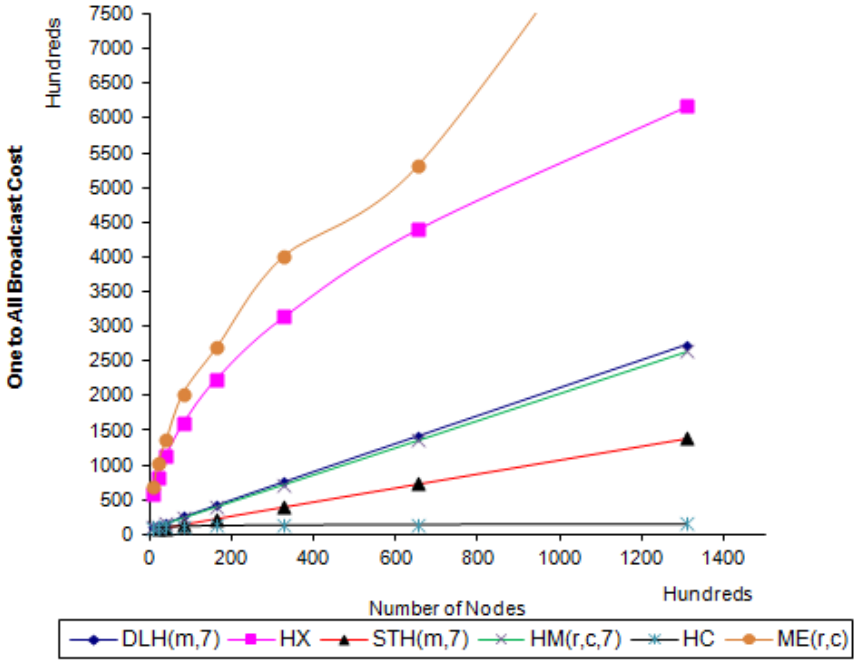

Figure 3.10: Cost of One-to-all-broadcast

Average Internode Distance $(\overline{d})$ Comparison

Figure 3.12 on page 87, compares the average internode distance of DLH, HM and STH with that of Hypercube (HC). From Figure 3.12, it is clear that average internode distance in DLH and HM increases rapidly with the increase in network size. In case of Hypercube, it increases gradually when the network is scaled up. However, in case of the STH, the average node distance remains almost constant, close to 3, even for very large network size.

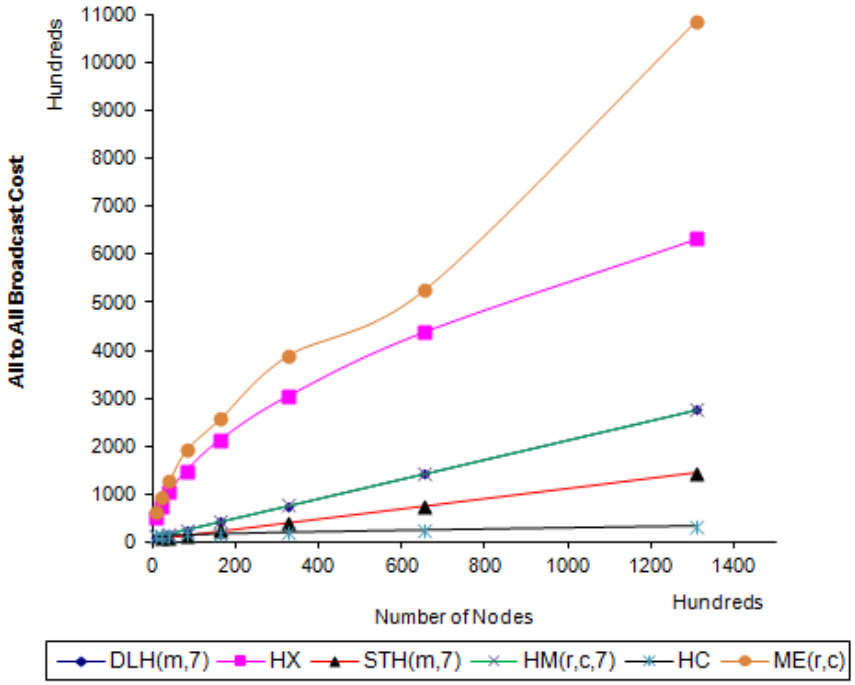

Figure 3.11: Cost of All-to-all-broadcast

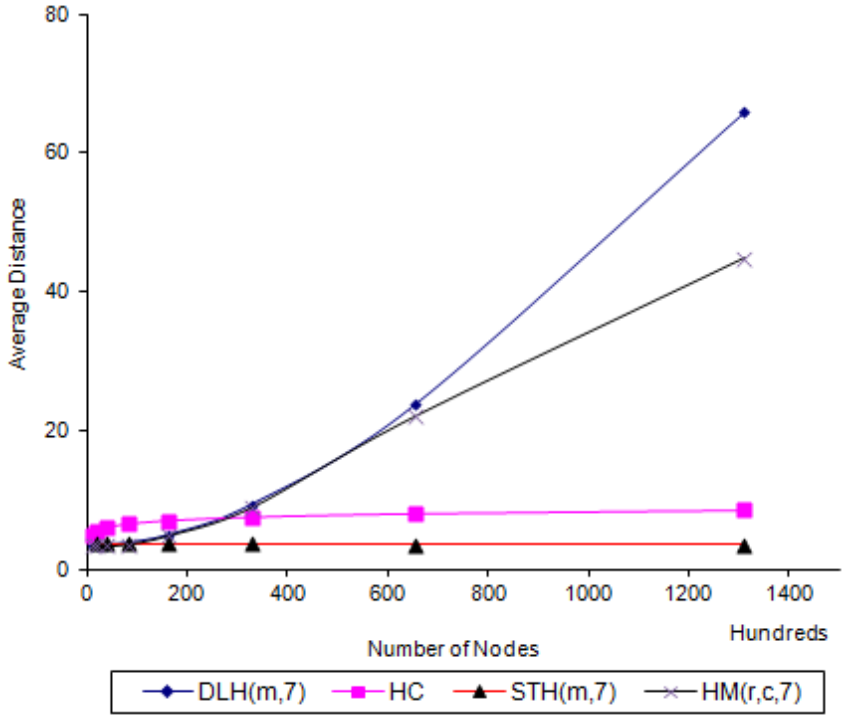

Figure 3.12: Average Internode Distance Comparison

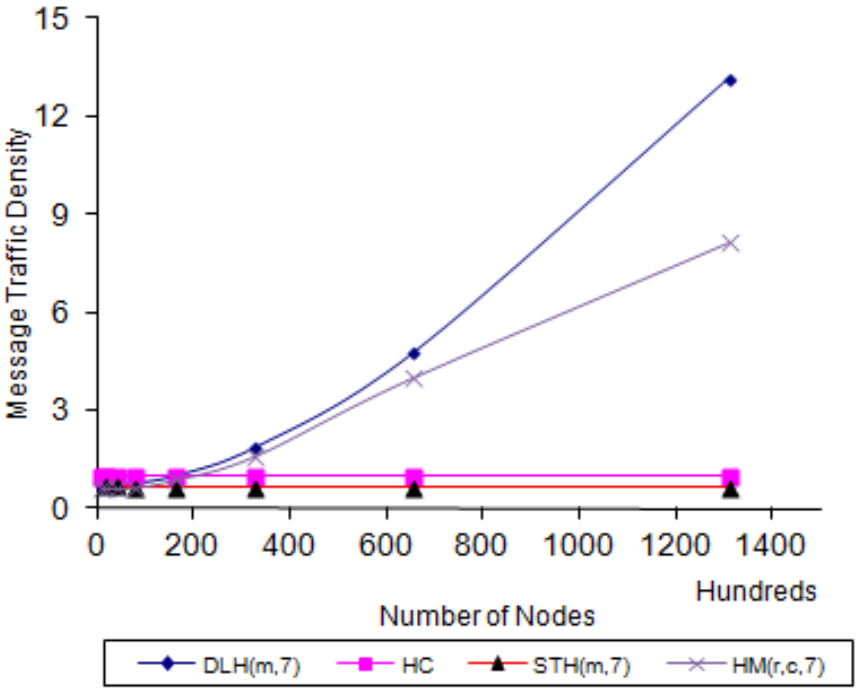

Figure 3.13: Message Traffic Density Comparison

Message Traffic Density (ρ) Comparison

The message traffic in a network can be estimated by calculating the message traffic density (ρ). Assuming that in a network of N nodes and L links each node is sending one message to a node at an average distance , the message traffic density is given by:

$$\rho = \frac{(\overline{d} \times N)}{L}$$

Figure 3.13 shows that for the DLH and HM networks, message traffic density grows rapidly when the network is scaled up. For Hypercube the message traffic density is almost constant (close to 1) and for the STH network it is also constant (close to 0.65). Clearly, STH has the lowest value of ρ.

Chapter Summary

In this chapter we have presented a new interconnection network topology called, Scalable Twisted Hypercube (STH), to counter the poor scalability of twisted hy-

percube. Its suitability for use as a multiprocessor interconnection network has been explored. Various properties of the proposed topology have been analyzed and compared with some other topologies on a number of standardized interconnection networks evaluation parameters. With reduced diameter, better average distance, low traffic density, low cost, maximum number of links, high bisection width and tremendous scalability, STH is highly suitable for massively parallel systems. Procedures for routing and broadcasting on the proposed topology have also been discussed and a simple routing algorithm has been presented. In summary, the new interconnection network offers strong architectural support for parallel computing due to the concurrent existence of multiple $LST(m)$ and TQ_n.

With this chapter the first part of the thesis has come to an end. The next chapter is related to second part of the thesis within which another important issue related to parallel and distributed systems namely task allocation or scheduling is dealt with.

Chapter 4

Taxonomy of Task Allocation Models and Related Work

Over the past two decades, a number of studies in optimization techniques and their applications to parallel and distributed applications have led to the identification of several challenging problems. One of these problems is optimally assigning the tasks of an application to the machines within the parallel and distributed system. This problem is known as task allocation problem or task assignment problem or scheduling. The Task Allocation Problem is NP-complete for most of its variants except for a few highly simplified cases [73] [79] [80] [102] [111]. As a result, it has attracted the attention of many researchers and has been extensively studied. Consequently, numerous approaches based on various constraints and assumptions have been reported to solve this problem. In a parallel and distributed system, it is essential to assign each task to the machine whose characteristics are most suitable to execute that task.

This chapter begins with a precise introduction to the task allocation problem. This is followed by an overview on the known task allocation models and their taxonomy based on the solution techniques used. Basic techniques used for task allocation have been reviewed with their essential characteristics and constraints.

4.1 The Task Allocation Problem

The problem being addressed in this thesis is concerned with allocating the tasks of a parallel program among machines of a parallel and distributed system in such a way that load on each machine remains balanced and some objectives under defined constraints are achieved.

A task allocation system consists of the following:

- Program Tasks
- Target System
- Allocation Schedule in which specific performance criterion is optimized.

Program Tasks

The characteristics of a parallel program can be defined as the system (τ, Δ, ITC, EA) as follows:

- $\tau = \tau_1, \tau_2, \ldots, \tau_n$ is a set of tasks to be executed.
- Δ, is partial order defined on τ, which specifies operational precedence constraints. That is $\tau_i < \tau_j$ means that task τ_i must be completed before task τ_j can start execution.
- ITC, is an $n \times n$ matrix of communication data, where $ITC_{ij} \geq 0$ is the amount of data required to be transmitted from task τ_i to task τ_j, $1 \leq i, j \leq n$.
- EA, is an n vector of the computations i.e. $EA_i \geq 0$ is a measure of the amount of computation at task τ_i, $1 \leq i \leq n$.

The relationship among tasks in parallel and distributed system may or may not include precedence constraints. When some precedence constraints need to be enforced, the partial order Δ is conveniently represented as a directed acyclic graph (DAG), called a task graph. In this case allocating these tasks is usually referred to as Precedence Constraints Allocation/ Scheduling. A task graph $G = (\tau, E)$ has

a set of nodes τ and a set of directed edges E. A directed edge (i, j) between two tasks τ_i and τ_j specifies that τ_i must be completed before τ_i can begin. Associated with each node τ_i is its computational needs EA_i (how many instructions or operations, for example). Associated with each edge (i, j) connecting tasks τ_i and τ_j is the data size ITC_{ij} , that is, the size of a message from τ_i to τ_j. Figure 4.1 shows a typical DAG.

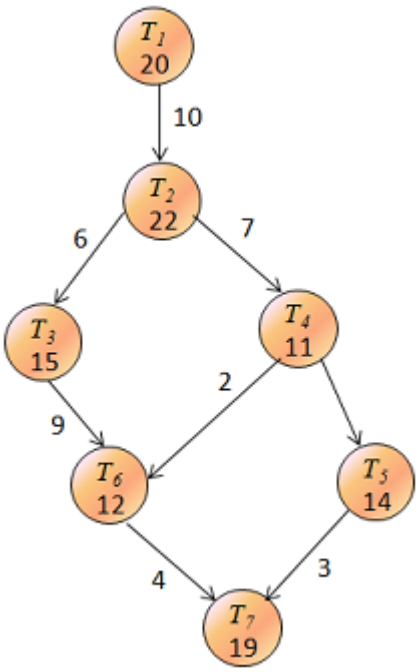

Figure 4.1: Directed Acyclic Graph

When there are no precedence constraints among the tasks, the relationships are only communication among tasks, which are represented by a undirected graph called a Task Interaction Graph (TIG). Figure 4.2 shows a TIG. This work addresses the allocation of Directed Acyclic Graphs (DAGs) on parallel and distributed systems.

Target System

The target system consists of a set M of m heterogeneous machines connected using a high-speed message passing interconnection network. The connectivity of

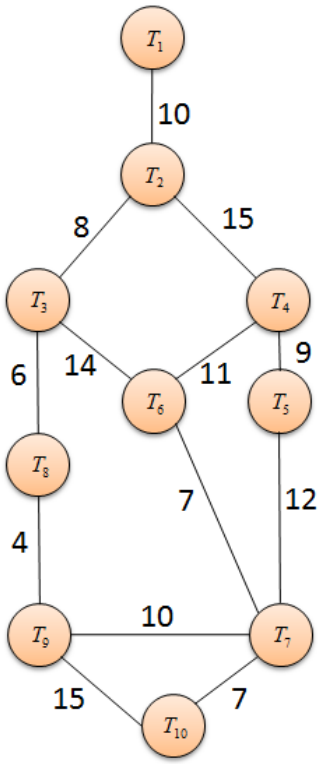

Figure 4.2: Task Interaction Graph

the machines can be represented using an undirected graph called the network graph. Associated with each edge (i, j) connecting two processing elements M_i and M_j in the network graph is the transfer rate R_{ij} , that is, how many units of data can be transmitted per unit of time over the link. As pointed out in Chapter 1, the target system addressed in this work is a heterogeneous multicomputer system.

The Allocation Schedule

An allocation schedule of the task graph $G = (\tau, E)$ on a system of m machines is a function f that maps each task to a machine. Formally, $f : \tau \rightarrow \{1, 2, \ldots, m\}$. If $f(v) = i$, for some $v \in \tau$, we say that task v is scheduled to be processed by machine M_i.

The goal of any allocation schedule is to minimize the total completion time of a parallel program. This performance measure is known as the schedule length or

the maximum finishing time of any task.

Execution and Communication Time

Once the parameters of the task graph and the target system are known, the execution and communication times can be obtained as follows.

- The execution time, e_{ij}, of task τ_i when executed on machine M_j whose computational speed is s_j is given by:

$$e_{ij} = \frac{EA_i}{s_j}$$

  The work addressed in this thesis assumes that task execution times are unknown.

- The communication delay (over a free link), c_{ij}, between tasks τ_i and τ_j when they are executed on adjacent machines M_i and M_j is given by:

$$c_{ij} = \frac{ITC_{ij}}{r_{kl}}$$

  where r_{kl} represents the bandwidth of like (k, l).

4.2 Task Allocation and Load Balancing

In parallel and distributed systems, an application is divided into a fixed number of tasks that are to be executed in parallel. If these tasks are simply allocated to the available machines without any consideration of the types of processing elements and their speeds, there is a distinct possibility that some processing elements will complete their tasks before others and will become idle since the tasks are unevenly distributed or some processing elements may operate faster than others (or a combination of both situations). To achieve minimum execution time, all processing elements must operate continuously on the tasks allocated to them. When tasks are divided among the processing elements evenly with the goal of minimizing the application's execution time, this is termed as load balancing [220].

Figures 4.3 and 4.4 explain how load imbalance leads to a larger application execution time. Two types of load balancing schemes have been reported in literature [220]:

Static Load Balancing

Static load balancing attempts to allocate tasks to the machines before execution begins. All parameters required to perform load balancing i.e. the characteristics of tasks, machines and interconnection network are known *a priori* and remain constant throughout the allocation process. Load balancing decisions are made at compile time either deterministically or probabilistically. This form of load balancing is usually referred to as mapping [47] or scheduling. These schemes are simple to implement and has less runtime overhead. Some well known static load balancing techniques are as follows [220]:

- **Round Robin Technique:** To distribute the load evenly on available machines, tasks are allocated to them in a round robin order i.e. in circular order without any priority. The scheduler assigns the $(i+1)^{th}$ task to the first node if the i^{th} task is allocated to last node. Round robin techniques are simple and straightforward but under performs when tasks are of unequal processing time.

- **Randomization Techniques:** In these techniques, the task and machine are selected randomly, and then the selected task is allocated to the selected machine. Once again, the approach is simple but suffers from the same drawback as the round robin technique.

- **Partitioning Techniques:** In these techniques, the parallel application is divided into tasks of equal computational load whilst minimizing inter-task communication.

  Partitioning techniques to map a two dimensional data matrix onto heterogeneous resources have been investigated by Crandall and Quinn [82] and

Kaddoura et al. [131]. The two papers are all based on Recursive Bisection algorithm.

- **Heuristics:** Heuristics make use of different approaches for allocating tasks of a parallel program to machines. Section 4.3.6 on page 110 presents various heuristic approaches used for task allocation.

The work addressed in this part of this thesis is related to task allocation with load balancing i.e to implement load balancing whilst allocating tasks. So it is essential to have a taxonomy and literature review on various approaches used for task allocation. Section 4.3 on page 98 covers the same. Further the terms task allocation, task assignment, scheduling and mapping have been assumed to be synonyms within this thesis.

Dynamic Load Balancing

Dynamic load balancing schemes implement load balancing by transferring the load from heavily loaded machines to lightly loaded machines during program execution. They need not be aware about run time parameters of a program i.e. the characteristics of tasks, machines and interconnection network may not be known *a priori* and do not remain constant during program execution. Information policy, location policy and transfer policy are key terms used to describe a dynamic load balancing scheme. Information policy is used to update the machines on the number of tasks waiting in the queue; transfer policy determines whether a task should be processed on the same machine or should be transferred to some other machine to improve the performance whilst the location policy identifies the machine to which the task should be transferred. They have some advantages over static load balancing schemes but generate more runtime overhead compared to them.

Dynamic load balancing is beyond the scope this thesis hence details regarding this scheme is not discussed further.

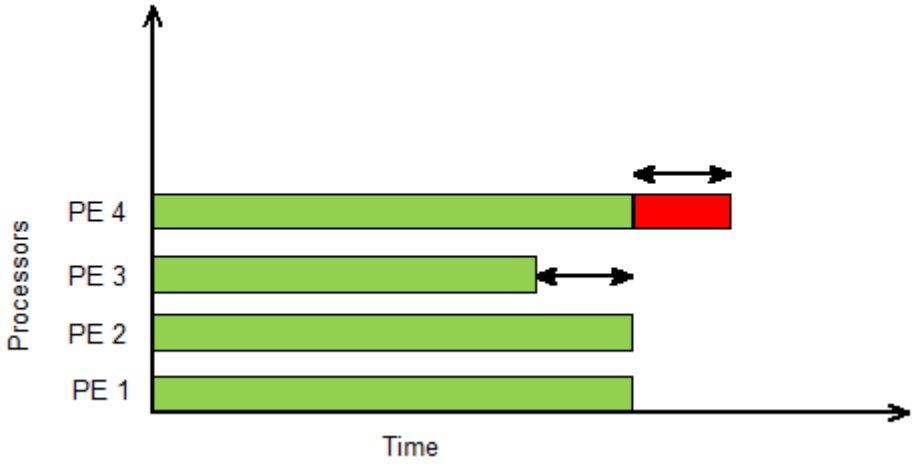

Figure 4.3: Load Imbalance: Larger Execution Time

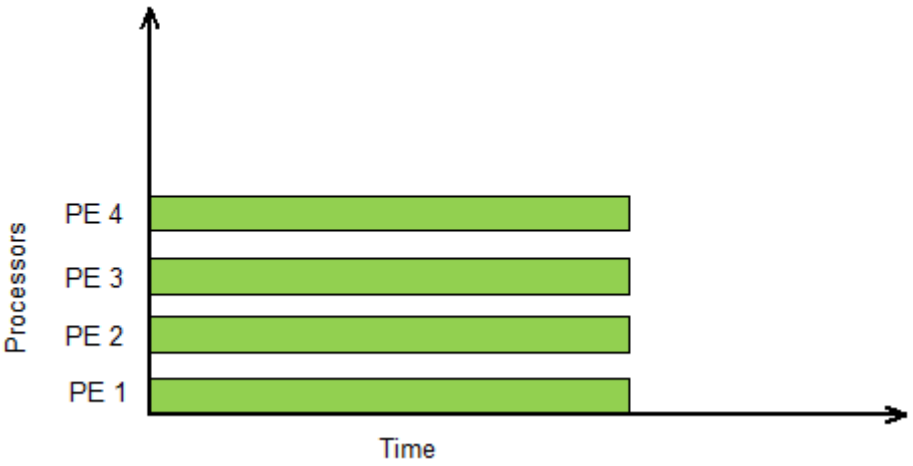

Figure 4.4: Perfect Load Balancing: Less Execution Time

4.3 Taxonomy of Task Allocation Models

Several strategies [202] [181] [47] [211] [108] [44] [146] [175] [63] [218] [208] [187] [53] [54] [233] [28] [30] [232] have been reported in literature to solve the task allocation problem both in the field of optimization and computer science. These strategies make use of numerous methods such as A^*-Algorithm, min-cut max flow, clustering, greedy approach, simulated annealing, tabu search etc. Though no general classification of these approaches exist, yet they may be grouped at different levels of hierarchy as shown in Figure 4.5.

At the first level of hierarchy these algorithms can be roughly classified into two categories namely, exact algorithms and approximate algorithms. [202] [181] [47] [211] [108] [44] [233] [28] [30] [232]. In the following subsections we briefly discuss each of them.

4.3.1 Exact Algorithms

Two categories of exact algorithms are - Restricted Exact Algorithms and Non-Restricted Exact algorithms. Restricted exact algorithms place some restrictions on the parallel program structure or on the interconnection network or on both and then lead to an exact solution in polynomial time.

Non-Restricted Exact algorithms on the other hand place no restrictions, either on the program structure or on the interconnection network and lead to an optimal solution but not in polynomial time.

4.3.2 Approximate Algorithms

Approximate algorithms may further be classifies as - Random Optimization and Heuristics. Random optimization algorithms use the same techniques as that used by the exact optimization algorithms to solve the problem. But, they restrict the solution search criteria by employing certain metrics which decide ”how good the solution is”. When a solution of certain satisfactory level is achieved the algorithm

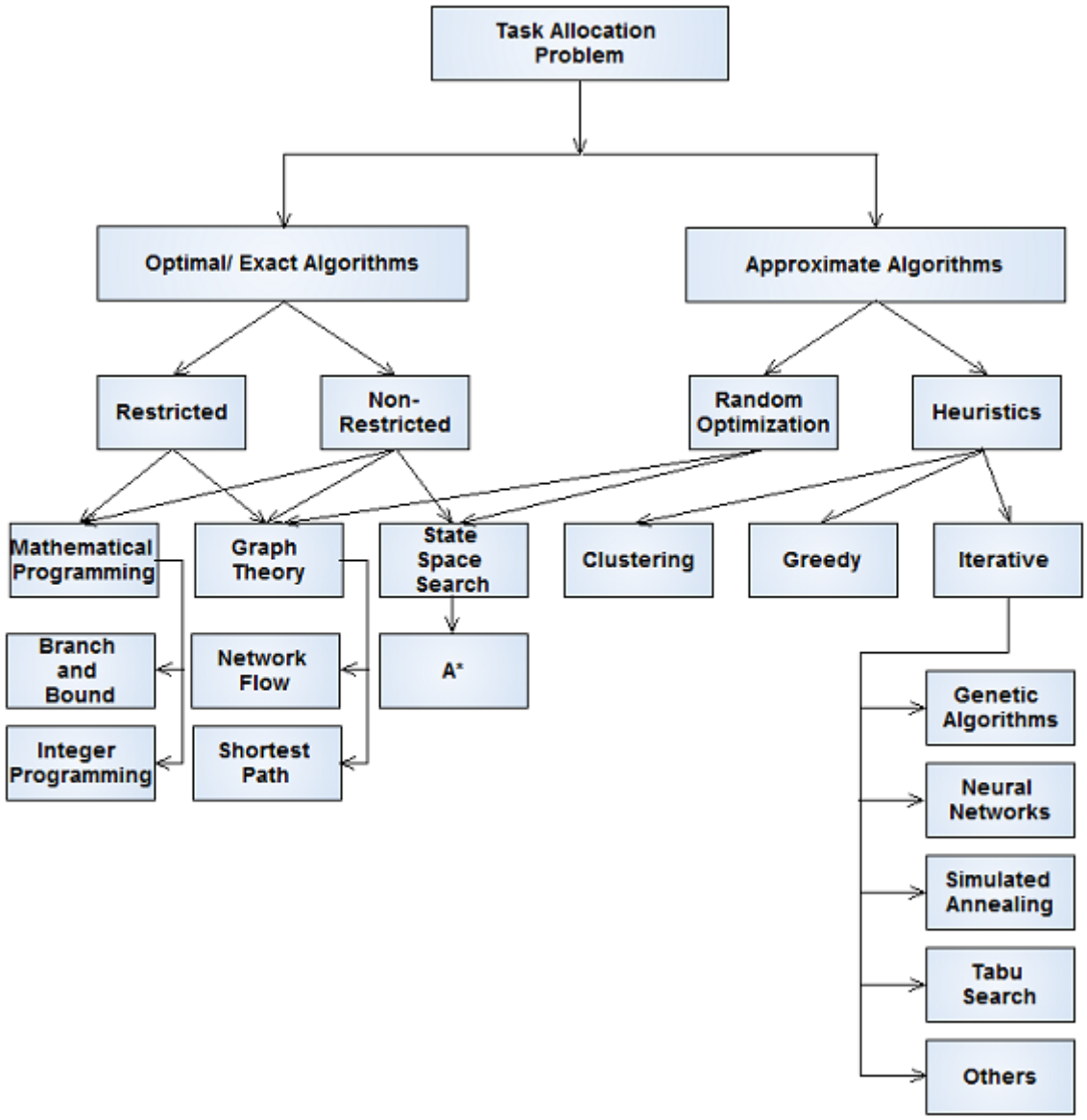

Figure 4.5: Task Allocation Taxonomy

terminates, declaring it as a good solution.

Heuristics on the other hand make use of techniques which effect the entire process in an indirect fashion. For example, in a clustering technique, tasks with extremely high inter-task communication are clustered and then allocated to the same machine in order to reduce the communication overhead. In a greedy approach, some tasks from the task set are allocated in the first step and then each subsequent step allocates only one task without backtracking until the entire task set is allocated. In iterative approaches, the entire task set is allocated in the first step and then the allocation is improved by checking certain system parameters in subsequent steps. These steps may involve techniques like task migration (moving task from one machine to another) and pairwise exchange etc.

Looking down in the hierarchy in Figure 4.5, and based on the above discussion, it may be concluded that allocation algorithms may be broadly classified into following four categories:

- Mathematical Programming
- Graph Theoretic
- State Space Search
- Heuristics.

In the following subsections, each of them is reviewed along with related literature.

4.3.3 Mathematical Programming Techniques

Mathematical programming approach has been recognized as the most powerful approach for modeling and analyzing several kinds of problems. This approach formulates task allocation problem as an optimization problem and solves it with mathematical programming techniques. From this approach following techniques have been developed for the task allocation problem:

- Branch and Bound Technique
- Integer Programming Technique
- Dynamic Programming Technique

Branch and Bound Technique

One of the most popular techniques among researchers and widely investigated in distributing systems for task allocation is the branch and bound (BB) technique. This technique is an exhaustive search approach. Using this technique, numerous algorithms for task allocation in distributed system have been reported.

An optimal task allocation strategy with the goal of maximizing the reliability whilst considering communication cost as the constraint function for a distributed database management system has been reported by Verma [214]. The model is converted into a state space search tree and the Branch and Bound technique is used to achieve optimum results. This approach has been adopted for optimizing the distributed execution of join queries as reported by Reid [182]. Kartik and Murthy [133] used the idea of branch and bound technique and have presented efficient algorithms to maximize the system reliability in both redundant and non-redundant systems. Magirou and Milis [160] have presented an algorithm based on this technique for the problem of assignment of tasks to processors in a distributed processing system so that the sum of execution and communication cost is minimized.

Billionnet et al. [45] proposed another algorithm based on the BB technique to minimize the sum of inter-task communication and execution costs. In their approach the processors are heterogeneous while the communication links are identical. The use of identical links ensures that identical transmission times are generated when identical messages are transmitted through different links. The capacities of processors and links are assumed to be unlimited (uncapacited network). The problem is first formulated to measure the inter-task communication and processing costs under an uncapacited network. Then, the problem is refor-

mulated as a quadratic Boolean function with linear constraints. Finally, the BB technique is applied to search for an optimal allocation.

Chang et al. [64] have proposed two algorithms based on the BB technique to allocate files into a distributed system whilst minimizing the data transfer rate. The first algorithm called OFA (Optimal File Allocation), uses the BB technique with an evaluation function derived from critical cut concepts, and leads to an optimal solution. In second algorithm which is termed as HFA (Heuristic File Allocation), the sub-optimal solution is obtained by terminating the OFA search as soon as the first complete allocation is found.

Hagin [117] considered the allocation problem for assigning distributed multimedia applications into distributed computer systems. He has proposed an algorithm based on the BB technique to solve two types of mapping problems.

Integer Programming Techniques

Integer programming is a mathematical programming technique in which some of all the variables are restricted to be integers. It is directly applicable to the problem of task allocation in a distributed system. A task allocation model becomes more realistic when it incorporates real time constraints such as inter-processor communication, memory limitations of each processor etc. In the past, a significant number of studies have been devoted to the optimization in distributed systems using integer programming technique. For the task allocation problem, integer programming is a useful and exhaustive technique, as it is capable of reflecting real life situations of distributed processing and it is simple as well.

A number of researchers have worked on task allocation problem using integer programming technique to determine the optimal solution under given constraints [70] [75] [99]. A model based on this is developed by Chu [77] for optimum file allocation in a distributed system. A similar approach for data file allocation has also been proposed by Marcogliese and Novarese [163]. Lisper and Mellgren [152] have discussed various integer programming methods for the allocation in distributed real time systems. Using integer programming some scheduling tech-

niques have been developed by Tompkins [209] for task allocation in distributed systems.

Dynamic Programming Technique

Richard Bellman, first used the term dynamic programming to describe the solution procedure of the problems where one needs to find the best solution among a number of solutions. A dynamic programming approach examines the previously solved subproblems and combines their solutions to give the best solution for the given problem. It is both a mathematical optimization method and a computer programming technique. In terms of mathematical optimization, dynamic programming is a procedure to design algorithms for the problems wherein the solution is a result of a sequence of decisions.

A number of papers are available in literature proposing the solution to the task allocation problem using dynamic programming. Using this technique Bokhari [47] formulated a shortest tree algorithm which runs in $O(mn^2)$ time for an optimal assignment of m program modules to n processors. Rosenthal [184] established a relationship between module allocation and non-serial dynamic programming. Dynamic programming technique was applied by Fernandez-Baca and Medepalli [108] for obtaining the optimal assignment through local search. Berman and Ashrafi [41] have developed optimization models for measuring the reliability of modular software systems using this technique.

Pros. and Cons. of Mathematical Programming Techniques

- These algorithms are more flexible compared to other allocation techniques.
- An optimal solution is guaranteed.
- System constraints like inter-task communication, network delay, memory limit, processor speed limit etc. can easily be formulated in terms of mathematical programming problem.

- These algorithms are time and space hungry. Their complexity grows exponentially as the parallel and distributed system is scaled up. So they are not suitable for large systems until some constraints are applied on them which in turn lead to randomized optimization.

4.3.4 Graph Theoretic Techniques

Graph theoretic approach is one of the most important mathematical techniques used in optimization. It has been extensively used in computer science research particularly in distributed systems, data mining and networking. This approach allows the use of graphical methods to represent and allocate program tasks to various processors in distributed processing system.

In most of the graph theoretical approaches, the solution begins with the abstraction of tasks and inter-task communication cost through graph model in which tasks are represented by nodes and inter-task communication cost as weights on bi-directional edges connecting these nodes. These approaches do consider load balancing, resource limitations etc. without giving much attention to the timing complexity. Graph theoretic approaches used in literature in the context of parallel and distributed systems can be further categorized as follows

- Network Flow Techniques
- Shortest Path Techniques

Network Flow Techniques

In this approach the problem of allocating m tasks to n processors proceeds as follows:

- Denote the program by a graph consisting of a set of m nodes (tasks) and link these nodes with a set of edges where each edge represents communication among the tasks it is linking. The nodes of the graph are referred to as ordinary nodes.

- Convert the graph into commodity flow graph by adding n distinguished nodes in the graph, where each distinguished node corresponds to a processor.

- Connect every ordinary node with every distinguished node. The edge connecting ordinary node t_i with distinguished node P_s is given the weight w_{is}, where $w_{is} = \frac{1}{n}\sum_j e_{ij} - e_{is}$, for $1 \leq i \leq m$ and $1 \leq j \leq m$. Here e_{ik} denotes the execution cost of task t_i on processor P_k.

- Apply an n-way cut on the graph. An n-way cut is defined as the set of edges which partition the graph into n disjoint subsets of nodes, each subset containing only one processor.

- Every subset obtained in previous step actually indicates an assignment. Sum of the weights on the edges in the cut is called the cost of the n-way cut and it is equal to the total sum of inter-task communication costs and execution costs [153].

Numerous techniques have been proposed in literature for solving the task allocation problem using network flow techniques. Stone [202] showed that the total sum of inter-task communication costs and execution costs can be minimized when program modules are assigned to a two processor distributed system. The complexity of this algorithm is $O(m+2)^3$. He further extended his network flow techniques [203] and proved the existence of a critical load factor for each program task. However, this problem proved to be computationally intractable in the general case and thus for distributed systems.

To obtain a task allocation for an n-processor system where $n \geq 2$, the commodity flow graph is to be partitioned into n disjoint subsets of nodes, each subset containing only one processor. n-way partitioning is not possible with network flow algorithm alone. It has been attempted by repeatedly applying the network flow algorithm by Wu and Liu [223] [224] and also in combination with heuristics by Arora and Rana [23] for suboptimal solutions.

Rao et al. [181] again attempted to solve the same problem with an alteration in the system model within which one processor had limited memory while the other had unlimited memory. Using network flow techniques the authors demonstrated the method of constructing a GH-Graph, by finding the maximum flow between every pair of nodes of the original task interaction graph. In searching for a minimum cost assignment, they only produce the minimum cut and then reassign some subset of the tasks from one processor to another in order to satisfy the memory constraints. The algorithm had the complexity $O(n^3)+O(m-1)$ for m tasks and n machines configuration. They further showed that the algorithm was NP-Hard for $n \geq 3$, so can't be applied to distributed system allocation problem in general case.

Lee et al. [146] also considered the same problem using network flow techniques. They extended Ston's work and generalized it from a two processor to a network of n processors connected as a linear array. In their approach, the task allocation problem is first converted into a two terminal network flow problem and then solved using network flow technique in $O(n^2m^3 \log n)$ time. Again Lee and Shin [145] considered the allocation problem on a network of n-homogeneous processors. Each of them had its own memory. They first developed a modeling technique that transformed the assignment problem in a tree into a minimum-cut maximum flow problem and then solved the problem in $O(\sum_i (n_i - 1)m^3)$ time.

The allocation problem has also been considered by Hui and Chanson [124] for assigning n tasks without precedence constraints (usually represented by a TIG) to a set of m homogeneous and heterogeneous system connected through shared media, assuming access time to the shared media as zero. Initially they presented an algorithm for homogeneous system which had the complexity $O(n^2(n + E)log\frac{n^2}{n+E})$. They then extended the algorithm for heterogeneous processors and presented an algorithm with complexity $O((n + E)n \log n)$

Shortest Path Techniques

In a distributed processing system the task allocation problem can also be solved using the shortest path techniques. This approach has been widely investigated by the several researchers [47] [178] [211] to obtain an optimal solution.

The shortest path algorithm evaluates the set of nodes in the assignment graph corresponding to a program graph to select the best possible allocation of task to a processor. A shortest tree algorithm described by Bokhari [47] minimizes the total sum of the execution and communication costs for arbitrarily connected distributed systems with arbitrary number of processors, provided that the interconnection pattern of modules form a tree. The timing complexity of this algorithm is $O(mn^2)$.

Price and Pooch [178] claim that their shortest path method is applicable to all cases but, an optimal solution is possible in certain cases only. The shortest path algorithm proposed by them evaluates the set of nodes in the assignment graph corresponding to a program graph to select the best possible allocation of a task to a processor. A modification to the shortest path method is made using the non-backtracking branch and bound method and the complexity of algorithm is known to be roughly $O(mn)$.

The critical path method [91] and the ideas from the renewal theory along with the theory of large deviations [140] are also used for designing shortest path task allocation techniques.

Towsley [211] work considers special cases where the inter-task communication pattern is series parallel in nature. A search is made in the assignment graph for the parts of the program graph where inter-task communication patterns are series parallel or tree in nature. For such inter-task communication patterns, the shortest paths are derived and these shortest paths are combined to get the overall shortest path of the assignment graph. The timing complexity of the algorithm developed on this principle is $O(mn^3)$.

The problem of assigning the modules of chain structured programs and tree structured programs is presented in [48] [127] [138] [49] [128]. These programs

are considered for allocation on chain structured computer systems and single host, multiple satellite computer systems.

To derive an optimal allocation, an assignment graph is constructed which contains as many layers as the number of processors. The calculation of weights of edges depends on the nature of the program graph and the processor system. Each edge of the assignment graph is given two weights: sum weight and bottleneck weight. The sum and bottleneck weights of the path from source to terminal node indicate the time required by corresponding assignment. The optimal path should have the minimum sum bottleneck weight.

Pros. and Cons. of Graph Theoretic Allocation Methods

- These methods have been widely researched due to their simplicity.
- They are not meant for a generalized allocation of m tasks to n processors since such configurations have exponentially high complexities.
- Formulating resource constraints in graph theoretic approaches is a challenge.

4.3.5 State Space Search Techniques

State space search is a well-known approach to achieve optimal cost of the task allocation in a distributed system. In this approach, first the task allocation problem is converted in terms of a state space search tree and then a cost function is defined which is then used to guide the search. The search space tree is drawn as follows. For the task allocation in a distributed processing system, each state description is denoted by a node. Operators applicable to nodes are defined for generating successors of the nodes called node expansions. A solution path of a search problem is the path defined by a sequence of operators which leads a start node (i.e. initial node) to one of the goal nodes (i.e leaf nodes or external nodes). All the internal nodes in this state space search tree correspond to incomplete task allocations and all external (leaf) nodes correspond to complete allocations [193].

In this search tree, the job is to find the goal node i.e. a leaf node corresponding to the optimal task allocation. The well known A^*-algorithm is known to be the best tool for searching optimal assignment in state space tree.

A number of researchers have used the A^*-Algorithm to find the optimal allocations in distributed systems. Shen and Tasi [194] proposed the first A^* based solution. They first translated the task assignment problem into a special graph known as weak homomorphism in graph matching. Then on this weak homomorphism they applied the A^*-algorithm to find the optimal solution. However, their solution was not designed for a generalized parallel and distributed system. They considered a point-to-point interconnection network in which tasks with inter-task communications were required to reside either on the same processor or on two directly connected processors.

Sinclair [199] has developed a method to reduce the size of the search tree. This method is termed as the reduction method. Two criteria proposed by the authors to implement the reduction method are known as Minimum Processor Cost Underestimate (MPCU) function and Minimum Independent Assignment Cost Underestimate (MIACU) function. The allocation algorithm developed based on reduction method is called "Branch and Bound with Underestimate (BBU)". BBU attempts to minimize the total sum of execution costs and inter-task communication costs while allocating the program modules to processors of a distributed system. The authors further proved that Minimum Independent Assignment Cost Underestimate when used with A^*, leads to less space and timing complexities. The solution proposed by Sinclair [199] does not take into account the task precedence constraints. The method was further improved by Wang and Tasi [218], by considering task precedence constraints. Tom and Murthy [208] improved the models proposed by Shen and Tasi [194] and Wang and Tasi [218]. They altered the order of nodes in the search tree in such a way that the independent tasks were restricted to be assigned last.

Kafil and Ahmed [132] have proposed a two phase algorithm for finding optimal solution of task allocation problem using A^*. The first phase of the algo-

rithm is known as "Optimal Assignment Sequential Search (OASS)" and is meant to produce an initial solution. The purpose of this phase is to reduce all those nodes which have higher costs than the resulting solution cost during search operation. The second phase of the algorithm is known as "Optimal Assignment Parallel Search (OAPS)" and its purpose is to accelerate the search process.

Pros. and Cons. of State Space Search Techniques

- They are flexible compared to their graph theoretic counterparts.
- System constraints like inter-task communication costs, network delay etc. are easy to formulate using these methods.
- For the task allocation problem an optimal solution is guaranteed by A^*.
- Like mathematical programming technique, these techniques are also constrained by the time and space complexities and, algorithms based on them require searching for an optimal solution.

4.3.6 Heuristic Techniques

These techniques have been developed to obtain the near optimal solution of task allocation problem in acceptable time. Although an optimal solution to task allocation problem can be obtained using exact algorithm, the general n-processor task allocation problem is NP-Complete. Therefore, finding an optimal solution with exact algorithm to large scaled task allocation problems is computationally prohibitive [132] [212] [233]. Therefore, the development of effective heuristics is gaining importance among researchers. Heuristics provide fast and effective alternatives for obtaining near optimal solution of large scaled task allocation problems. During the past two decades numerous heuristics have been reported in literature for the task allocation problem. They may be further classified as follows:

- Clustering Heuristics

- Greedy Heuristics
- Iterative Heuristics.

In the following subsection we briefly discuss some popular heuristics from each category.

Clustering Based Heuristics

Much of the focus in heuristic algorithms has been devoted to the minimization of inter-task communication cost. Task clustering technique is one of the heuristic approaches to reduce the total inter-task communication cost. In this technique, a set of communicating tasks is fused together to form a task cluster. If the number of created clusters is greater than the number of available processors then clusters are fused in such a manner that the number of clusters becomes equal to the number of processor. It may drastically reduce the size of the search space [216]. Finally, task clusters are allocated to the available processors.

Efe [96] presented a task-clustering algorithm called "two module clustering" that forces task pairs to be allocated to the same processor. This procedure is run until all the candidate task pairs are grouped together. Arora and Rana [22] proposed module assignment in two-processor distributed system. The authors later [24] extended the idea of task clustering and proposed the concept of clustering the tasks which exhibited certain particular behaviour to reduce the problem size. Sagar and Sarje [188] used the clustering technique to propose other models for task allocation in distributed systems. Bowen et al. [55] proposed a clustering algorithm for assignment problem of arbitrary process systems to heterogeneous distributed computing system.

The clustering technique used by Kim and Browne [135] iteratively applies a critical path algorithm to transform the graph into a virtual architecture graph which consists of a set of linear clusters and the interconnection between them. A different heuristic approach based on the concept of clustering to allocate the tasks for maximizing reliability has been proposed by Srinivasan and Jha [201].

Based on the clustering approach, a multiprocessor scheduling technique named "de-clustering" has been formulated by Sih and Lee [198]. The authors claimed that their de-clustering approach not only retains the clustering advantages but at the same time overcomes its drawbacks. Palis [171] presented task clustering and scheduling algorithm for distributed memory parallel architecture.

Abdelzaher and Shin [3] developed a graph-based model which involved recursive invocation of two stages: clustering and assignment. The clustering stage partitions tasks and processors into clusters while the assignment stage maps task clusters to processor clusters. The problem of minimizing the cost by clustering heavily communicating tasks and assigning the clustered tasks to appropriate processors has also been researched by [221] [177] [219].

Greedy Heuristics

A partial solution is essential to initiate a greedy heuristic. These approaches begin with a partial solution and repeatedly improve this solution until the complete allocation is done. At each stage one task allocation is made and this decision remains unchanged throughout all the subsequent stages. The allocation is done in a manner that a choice once made can never be reconsidered i.e. without any backtracking. The allocation of next task to be allocated say, τ_k depends on the criteria chosen to allocate the previous $k-1$ tasks. Generally these approaches are easy to implement and lead to near optimal solution in polynomial time, less than $O(m^3)$, for m tasks in most of the cases.

Numerous solutions for task allocation problem using greedy approaches have been reported in literature [153] [144] [120] [162] [126]. Lo [153] merged a greedy algorithm with Stone [202] and thereby minimized the total execution and inter-task communication costs. The algorithm proposed by the authors consists of three phases namely $Grub$, $Lump$ and $Greedy$. The first phase $Grub$, produces the initial partial solution required to run the greedy heuristics. The second phase $Lump$, is executed when the allocation is not completed by $Grub$. Similarly, the third phase is executed when the allocation is not completed by the second phase.

Another parameter used by this approach is interference cost. It is the measure of the amount of incompatibility among tasks.

The approach used by Lee [144] implements load balancing aspect through the greedy heuristic. The algorithm is termed as Largest Processing Time First $(LPFT)$. The tasks are first arranged in descending order of communication cost and then allocated to the less loaded processors as per the order already defined.

Hluchy et al. [120] proposed static and dynamic greedy heuristics for multicomputer systems. An objective function was formulated by the authors to check the optimality of the solution in a static heuristic approach. A semi-distributed approach has been developed for dynamic allocations. For distributing programs and data files to networked multicomputers whilst maximizing the system reliability Hwang and Tseng [126] have proposed greedy heuristics based algorithms.

Iterative Heuristics

Like greedy heuristics, these approaches also start with an initial allocation and try to refine it further in subsequent steps. The initial allocation may be obtained using some faster heuristic or some random optimization technique. However, to improve the initial allocation they use techniques such as task migration or pairwise tasks exchange, which are generally not used by greedy approaches.

The following subsection presents some popular iterative heuristics found in literature:

Genetic Algorithms (GA)

It is a meta-heuristic optimization technique based on Darwin's evolutionary theory of the survival of the fittest. It emulates the behaviour of reproduction in nature [215] [212]. A genetic algorithm can be applied to search large multi-objective and complex problem spaces in a distributed system. Genetic algorithms attempt to search a near optimal solution by fostering a population of strings (called chromosomes) using predefined genetic operators [151]. The main steps of a genetic algorithm are reproduction, selection, crossover and mutation. The

selection, crossover and mutation processes are repeated until the termination condition is satisfied [151].

In the last two decades a lot of work on the application of GA on a distributed processing system has been done by the researchers which is mostly related to the task allocation problem [12] [10] [212] [215] [123] [122] [231] [227] and task scheduling problem [222] [85] [167]. Task allocation in a distributed system is a challenging issue as it requires the system performance to be optimized. A technique based on problem-space genetic algorithm (PSGA) for static task allocation in heterogeneous distributed system has been proposed by Ahmad et al. [12], which combines the power of GA with the problem specific heuristic to search the best possible solution. For multiple task allocation, Tripathi et al. [212] developed a GA based method which is memory efficient and gave an optimal solution of the problem. Vidyarthi et al. [215] also used GA to maximize the reliability of a distributed computing system.

In distributed computing systems the hardware redundancy policy is of equal importance as the task allocation policy because it has direct impact on system cost and system reliability. In this context, researchers have addressed some algorithms based on GA [122] [227]. Levitin et al. [149] discussed a redundancy optimization problem for multistate systems. Deeter and Smith [89] used the similar approach to solve the all-terminal network design problem considering cost and reliability.

Applications of GA have revealed that they generate more efficient solutions than other heuristic approaches, which are applied to task scheduling in heterogeneous system. Page et al. [170] presented a GA based scheduling strategy to dynamically schedule a set of heterogeneous task onto a set of heterogeneous processors to minimize the total execution time.

Hill-Climbing (HC)

In the field of computer science, Hill-Climbing is an iterative technique for solving computationally hard problems. It starts with a sub-optimal solution to the

problem (i.e start at the base of a hill) and iteratively improves the solution (climb up the hill) [130] until some criterion is maximized (top of the hill is reached). The improvement to the current state (solution) is made gradually in small steps. Associated with each random move is a cost function which is evaluated after each step. If the change in the cost function is positive the move is accepted and a new solution is generated otherwise no change is made to the current state. The process is repeated until there are no changes to the current state which lead to the reduction in the value of the cost function. When this condition occurs, it indicates that a local optimum has reached, instead of the required global optimum.

The problem with this algorithm is that it guarantees an optimal solution in case of convex problems and it is considered to be a good algorithm for finding a local optimum (a solution that can't be improved by considering a neighbouring state). However, it doesn't guarantee to find the best possible solution (the global optimum) out of all possible solutions (the search space).

Yanping and Haijiang [229] have developed a method of allocating tasks to welding robots using the hill climbing algorithm. Fattah et al. [105] have used the hill climbing techniques to rapidly find the appropriate start node in the application mapping of network based many-core systems.

Simulated Annealing (SA)

Simulated Annealing is a global optimization technique which attempts to find the lowest point in an energy landscape. It emulates the physical concepts of temperature and energy to present and solve the optimization problems [30]. The objective function of the optimization problem is treated as the energy of a dynamic system while the temperature is introduced to randomize the search for a solution.

Over past few years SA, has been used by many researchers for solving the task allocation problem in a distributed system. In 2004, Attiya and Hamam [27] proposed a simulated annealing based optimal two-phase algorithm for task allocation problem in a heterogeneous distributed computing system with the goal of

maximizing the system reliability. The authors also proposed [30] another simulated annealing based optimal algorithm for the same problem and with same goal in 2006. Their hybrid algorithm first finds a sub-optimal allocation by applying the well known SA algorithm and then finds an optimal allocation by applying the branch and bound (BB) technique by considering the solution provided by SA, as the initial solution. Further they used the same algorithm for load balancing in heterogeneous distributed computing system [29].

With the goal of minimizing the inter-task communication delays, Bollinger and Midkiff [52] have proposed a SA based task allocation technique for a multicomputer network system. Hamam and Hindi [118] have used a similar approach for allocating program modules to a distributed system connected through a general purpose interconnection network.

Mean Field Annealing (MFA)

The MFA technique was developed to solve combinatorial optimization problems. In this approach discrete variables, termed as *spins*, are used for encoding the combinatorial optimization problem. An *energy function* written in terms of *spins* is used as cost function. Then, using the expected values of *spins*, a gradient descent type relaxation scheme is used to find a configuration of the *spins* which minimizes the cost function.

Based on MFA approach, Bultan and Aykanat [59] have developed a heuristic approach to solve the task allocation problem. They have shown that, the algorithms based on MFA are faster than those based on the SA but the later produces better solutions than the former.

Aykanat and Haritaoglu [34] have used the MFA techniques mapping unstructured domain to a hypercube connected distributed memory architecture. Their goal is to find a mapping which minimizes the communication overhead while maintaining the same workload on processors (load balancing).

Other Heuristic Approaches

Heuristic approaches discussed above are not the only ones but are the most present in literature. [66] [191] [72] [61] [139] [18] describe some other heuristic approaches.

Chaudhary and Aggarwal [66] proposed a general algorithm for allocating a Task Interaction Graph (TIG) as well as a Directed Acyclic Graph (DAG) on a distributed system. The algorithm first finds an initial assignment and then improves it by pairwise exchange. The algorithm is very general so the complexity is high.

Selvakumar and Murthy [191] have proposed a heuristic algorithm for allocating programs into a distributed system with heterogeneous machines. An algorithm based on recursive divide-and-conquer was first developed for the case of two machines and then extended to the case of n machines.

Choi et al. [72] proposed three heuristic algorithms for allocating tasks of a linear task graph into a parallel system of heterogeneous processors. The complexities of first and second algorithms are $O(n^2m)$ and $O(n^4m^4)$ respectively. The third algorithm is based on greedy heuristic.

Cai and Lee [61] have proposed a three phase algorithm to distribute data files among heterogeneous server machines. The goal is to balance the response time between the servers i.e., minimize the difference in the response time. The first phase, *selection phase*, selects the set of servers that can participate in file allocation by eliminating those servers that have limit lower than a predefined value. The second phase, *allocation phase*, sorts the selected servers in increasing orders of their service rates, allocates files with higher access rate to a server until reaching its allocation limits and then the allocation moves onto a next server. The third phase, *completion phase*, allocates the remaining files into servers with higher capacity until all files are assigned.

Koziris et al. [139] presented a two phase heuristic algorithm to map tasks of a TIG into multiprocessor architecture to minimize the communication time. The algorithm first assigns the highly communicating tasks on adjacent nodes in

the processor's network. Then, without backtracking it maps the remaining tasks beginning from those close to the assigned tasks.

Altenbernd and Hansson [18] proposed an algorithm which is termed as *slack method*. This heuristic allocates communicating periodic hard real-time tasks onto a multiprocessor system.

Pros. and Cons. of Heuristic Techniques

- Heuristics provide efficient and effective methods of obtaining near optimal solutions.
- Their time and space complexities are less compared to exact techniques.
- It is very difficult to choose best values for the parameters required by them.
- Sometimes, they lead to a suboptimal solution that may be far away from the exact solution.
- Braun et al. [56] have proved that if they are used to get a solution close to optimal solution, their timing complexity may increase exponentially. They have further concluded that it is almost impossible to select a single heuristic technique as the best general method of solving different types of applications.

Chapter Summary

In this chapter a precise introduction to the task allocation problem is presented. Relation between task allocation and static load balancing schemes is explained. In literature survey various techniques available for task allocation problem are classified based on the solution techniques used: mathematical programming, graph theoretic, state space search and heuristics. It has been established that graph theoretic schemes are simple to implement and are acceptable as long as the number of processing elements in a distributed processing system are less than four. They

can not be used for the generalized problem of allocating m tasks to n machines due to their high computational time and inability of handling various system constraints. The mathematical programming and state space search techniques are found to be more suitable compared to graph theoretic techniques for the generalized problem of task allocation. They lead to an exact solution and can easily handle the system constraints but once again their time and space complexities grow exponentially as the system is scaled up. Heuristic approaches unlike the other three, are simpler and flexible. They are suitable for the applications where an exact solution is not possible in a reasonable time. However, solutions obtained by them may be far away from the exact solution. Their timing complexity may be problematic if a solution close to the exact solution is required. Moreover, it is not easy to find the best values for the parameters required by them and hence none of the heuristics can be designated as good approach for solving the task allocation problem.

The focus of next chapter is our contribution to the task allocation problem considering load balancing.

Chapter 5

Fuzzy Load Balancing Task Allocation Models

The task allocation problem is a well-known problem in the field of computer science and therefore has been extensively studied. Numerous algorithms, mostly for homogeneous systems and relatively few for heterogeneous systems have been proposed to solve this problem. The algorithms meant for heterogeneous systems require high allocation cost and may not produce the right allocations while lowering costs. Moreover, in a heterogeneous multicomputer system (HMS), efficient use of available resources requires that the tasks be allocated to available machines intelligently. In this context "intelligently" means that a given task is assigned to a machine where it executes with the fastest speed and the load on various machines remains balanced (load balancing).

In this chapter, two novel task allocation models called Fuzzy Load Balancing Task Allocation (FLBTA) have been proposed with an objective to simultaneously meet high performance and fast allocation time while balancing various allocation parameters. The proposed models have been verified by implementing related algorithms in C and executing them on several sets of randomly generated input data. It has been observed that the models successfully achieve their goal i.e. task allocation with load balancing in all cases. Moreover, the algorithms outperform previous approaches of interest on metrics such as speedup, efficiency and execu-

tion time.

5.1 Related Work

A Heterogeneous Multicomputer System (HMS) consists of diversely capable machines harnessed through an interconnection network [17]. Exploiting an HMS to its fullest extent requires the proper assignment (matching) of each task to a machine in such a way that loads on individual machines remain balanced. This assignment (matching) is generally referred to in literature as mapping, resource allocation, resource management or task allocation. It is a restricted case of task scheduling wherein the orders of execution i.e. precedence relations between tasks are not considered [101]. This process is named as task allocation within the scope of this thesis. The task allocation problem is NP-complete in its general form [112], and polynomial-time solutions are known only for a few restricted cases. [80] [107] [46] [147] [186].

During last two decades many researchers have studied this problem. Two types of solutions have been proposed in literature: Optimal solutions (Physical Optimizations) for some restricted cases and for most of the cases heuristic techniques have been developed which led to near optimal solutions. Early heuristic techniques used graph theoretic approaches such as network flow and/or enumeration techniques [100] [159] [202] . Research then evolved to approximate heuristic algorithms [189] [96] [153] [159] [74] [76]. Unfortunately, most of these algorithms deal with homogeneous systems and the complexity of these algorithms increases rapidly as the problem size increases. Exact algorithms using physical optimization have been proposed by many researchers [25] [51] [169] [81]. Though optimization algorithms produce exact solution (better than heuristics) they tend to be very slow. In realistic scenarios, for large data sets, their execution times are unacceptable when compared to the task execution times. A good review of models and algorithms based on them can be found in [11] [69] [166] [154].

Almost all approaches of interest are processors centric and undermine other

vital machine resources. HMS is characterized by machine and task heterogeneity and hence processors centric allocation models are not suitable for fair task allocation in such a vague environment. Fuzzy logic is a powerful tool for designing intelligent systems. The ability of drawing conclusions and generating responses based on vague, ambiguous and imprecise data give tremendous power to Fuzzy Logic. Consequently, mathematical problems that are difficult to model using conventional methods can easily be formulated using fuzzy principles [119] [20]. To deal with the limitations of previously proposed processors centric models this work presents two task allocation models based on fuzzy logic, which are suitable for allocating tasks on HMS in such a way that the load on various machines remains balanced. When allocating tasks, the proposed models assess the suitability of candidate machines based on various machine, task and underlying interconnection network characteristics in a fuzzy environment. For a given task, machines are graded according to their suitability of executing the task and the task is allocated to the most suitable machine. The tasks are assumed to be non-preemptable. All proposed models were tested by coding and executing the related algorithms in C on several sets of input data.

5.2 Problem Statement

The task allocation problem addressed in this work is formally defined as follows: Let $M = M_1, M_2, M_3, \ldots, M_n$ be the set of n heterogeneous machines of a HMS which are connected together through an interconnection network. Let $T = \tau_1, \tau_2, \tau_3, \ldots, \tau_m$ be the set of m heterogeneous tasks to be allocated to the machines under consideration and $ETC = [e_{ij}]_{m \times n}$ be the expected time to compute matrix where e_{ij} denotes the execution time of task τ_i on machine M_j. Associated with each machine M_i is a list L_i which keeps track of the load allocated to machine. Let $G = (X, E)$ be the Directed Acyclic Graph (DAG), where the set of edges E represents the communication cost c_{ij}, which reflects when tasks τ_i and τ_j are not allocated to the same machine. The reliability of each machine and link

of the system is also known.

Given the above scenario, the objective is to find an allocation $FA : T \rightarrow M$, that minimizes the total execution time (makespan), total inter-task communication and maximizes the system reliability while keeping the load on various machines balanced.

5.3 Brief Overview of Fuzzy Logic

This section presents a basic overview of fuzzy logic based on which two load balancing task allocations models have been developed.

5.3.1 Fuzzy Sets

Zadeh [237] proposed the Fuzzy set theory in 1965. It is based on multivalued logic that allows intermediate values to be defined between conventional evaluations like true/false, yes/no, high/low etc. Notions like "superb" or "too slow" can be formulated mathematically and processed by computer programs to apply a more human-like way of thinking in computer based decision making. Let A be a classic set of the universe U. Then a fuzzy set $\tilde{A}$ is defined by a set or ordered pair as follows:

$$\tilde{A} = \{(x, \mu_{\tilde{A}}(x)) | x \in A, \mu_{\tilde{A}}(x) \in [0, 1]\}$$

where $\mu_{\tilde{A}}(x)$ is a function called membership function. Membership function specifies the grade or degree to which any element x in A belongs to the fuzzy set $\tilde{A}$.

5.3.2 Fuzzy Numbers

Fuzzy numbers are special classes of fuzzy quantities. Dubis and Prade [93] noted that a fuzzy number is a fuzzy set defined on real line $\Re$. In principle a fuzzy set $\tilde{A}$ with membership function $\mu_{\tilde{A}}(x) : \Re \rightarrow [0, 1]$, is a fuzzy number if it possesses the following three properties:

1. It is a normalized fuzzy set, i.e. for some $z \in \Re$, $\mu_{\tilde{A}}(z) = 1$.

2. It is a convex fuzzy set i.e. it holds:

$$\mu_{\tilde{A}}(\lambda z_1 + (1-\lambda)z_2) \geq \mu_{\tilde{A}}(z_1) \wedge \mu_{\tilde{A}}(z_2)$$

for any $z_1, z_2 \in \Re$, and $\lambda \in [0,1]$.

3. The support of $\tilde{A}$ is bounded.

5.3.3 Triangular Fuzzy Numbers

Fuzzy numbers can be used to represent vague inputs (imprecise numerical values and linguistic terms). Considering their simplicity and solid theoretical base, a specialized form of fuzzy numbers called Triangular Fuzzy Numbers are mostly used for this purpose [93] [20]. A triangular fuzzy number $\widetilde{W}$, is a fuzzy number represented with three points as $\widetilde{W} = (a, b, c)$, and its membership function $\mu_{\widetilde{W}}(x) : \Re \rightarrow [0,1]$ is defined as follows:

$$\mu_{\widetilde{W}}(x) = \begin{cases} (x-a)/(b-a), & a \leq x \leq b, \\ (c-x)/(c-b), & b \leq x \leq c, \\ 0 \quad Otherwise \end{cases}$$

where, $-\infty < a \leq c \leq b < \infty$.

Let $\widetilde{W_1} = (a_1, b_1, c_1)$ and $\widetilde{W_2} = (a_2, b_2, c_2)$ be two fuzzy triangular numbers and $\kappa \in \Re^+$ be a constant then according to Zadeh [237] basic operations possible on triangular numbers fuzzy are performed according to Table 5.1. Whereas the addition and subtraction of and are exact triangular, their multiplication and division are approximate triangular [119] [93].

5.3.4 Linguistic Variables

Zadeh [238] realized that conventional quantification methods had difficulty in properly expressing the situations that were explicitly complex or hard to define. The concept of a linguistic variable is necessary in such situations. A linguistic variable is a variable whose values are words or sentences in a natural or artificial

Operation	**Expression**
Addition	$\widetilde{W_1} \oplus \widetilde{W_2} = (a_1, b_1, c_1) \oplus (a_2, b_2, c_2) = (a_1 + a_2, b_1 + b_2, c_1 + c_2)$
Subtraction	$\widetilde{W_1} \ominus \widetilde{W_2} = (a_1, b_1, c_1) \ominus (a_2, b_2, c_2) = (a_1 - a_2, b_1 - b_2, c_1 - c_2)$
Multiplication	$\widetilde{W_1} \otimes \widetilde{W_2} = (\min(a_1a_2, a_1c_2, c_1a_2, c_1c_2), b_1b_2, \max(a_1a_2, a_1c_2, c_1a_2, c_1c_2))$
Division	$\widetilde{W_1} \oslash \widetilde{W_2} = (\min(a_1/a_2, a_1/c_2, c_1/a_2, c_1/c_2), b_1/b_2, \max(a_1/a_2, a_1/c_2, c_1/a_2, c_1/c_2))$
Scalar Product	$\kappa(\widetilde{W_1}) = \kappa(a_1, b_1, c_1) = (\kappa a_1, \kappa b_1, \kappa c_1)$
Symmetric Image	$\Theta(\widetilde{W_1}) = (-a_1, -b_1, -c_1)$
Inverse	$\overline{(\widetilde{W_1})} = \frac{1}{\widetilde{W_1}} = (a_1, b_1, c_1)^{-1} = (1/c_1, 1/b_1, 1/a_1)$

Table 5.1: Basic Operations on Triangular Fuzzy Numbers

language. For example the values of the linguistic variable "performance" can be described using five linguistic terms: very low, low, medium, high, and very high. Triangular fuzzy numbers can be used to represent these linguistic terms. Linguistic variables and fuzzy numbers provide the opportunity of using linguistic terms instead of crisp values to the decision makers. In this work fuzzy triangular numbers and fuzzy linguistic variables have been used to model the task requirements and machine heterogeneities.

5.3.5 Defuzzification

The proposed task allocation model evaluates suitability score of a machine as a fuzzy number. Therefore, it is necessary that the non-fuzzy ranking method for fuzzy numbers be employed for comparison of various machine scores. Defuzzification is a technique to convert the fuzzy numbers into crisp real numbers. The goal of defuzzification procedure is to locate the Best Non-fuzzy Performance (BNP) value. Methods of such defuzzified fuzzy ranking generally include three kinds of methods - mean of maximal, centroid and α-cut [168]. Among these the centroid method is simple and practical and has been extensively used by researchers to determine BNP values of fuzzy triangular numbers. Yao and Chaing [230] noted that the signed distance method produces better results for defuzzification of triangular fuzzy numbers, so this work uses signed distance method. The BNP value of a fuzzy triangular number $\widetilde{W} = (z_l, z_m, z_r)$, from signed distance method is obtained as follows:

$$BNP_{\widetilde{W}} = \frac{z_l + 2z_m + z_r}{4.0}$$

Bellman and Zadeh [37] developed the theory of decision behaviour in a fuzzy environment. A large number of models have been developed based on it, and have been applied to a variety of fields such as control engineering, artificial intelligence, management science, and Multiple Criteria Decision Making (MCDM) among others. The concept of combining the fuzzy theory and multiple criteria decision making is referred to as Fuzzy Multi-criteria Decision Making (FMCDM).

5.4 Fuzzy Load Balancing Task Allocation Model - I

This section presents the assumptions and concepts related to the development of Fuzzy Task Allocation Model - I $(FLBTA - I)$, being addressed in this chapter.

5.4.1 Assumptions

Given the problem statement in Section 5.2, the objective is to find an allocation that minimizes the total execution time (makespan), total inter-task communication and maximizes the system reliability while keeping the load on various machines balanced. $FLBTA-I$, considers the task allocation problem with following assumptions:

- The machines involved in the HMS are heterogeneous and are connected through a high-speed interconnection network. They may have different processing speeds and failure rates. Moreover, the communication links may have different speeds and failure rates.
- Each machine is capable of executing any task. Two tasks, if executed on different machines, may communicate with each other and incur a specific amount of inter-task communication (ITC), cost. A task may experience different execution times when executed on different machines.
- ITC cost may differ if data is transmitted through different communication links.
- The state of machines and communication links is either operational or down.
- There is high task and high machine heterogeneity [17] in the system.
- Tasks are assumed to be non-preemptable, have known inter-task communications, no execution deadline, their execution times are unknown and they have precedence relations with one another.

The main objective of the proposed allocation model is to perform task allocation with load balancing. That is, to achieve a fairly distributed cumulative execution time on all machines under known task and machine heterogeneities. Machines communicate with each other through an interconnection network using a shared-memory area. The model assumes that the shared-memory area stores the task allocator (scheduler) which manages all task allocations. The task allocator serves as a fuzzy allocator and, therefore, keeps all the required information about the incoming tasks.

The allocation policy of the allocator is fairly simple. To allocate a new task, it evaluates the suitability score of each machine with regards to various parameters (explained below) in a fuzzy environment. Machines are ranked according to their suitability scores and the task is awarded to the machine with the highest ranking.

5.4.2 Task Execution Time and Execution Cost

The execution times of tasks are unknown and are estimated in a fuzzy environment. The core of this estimation is the range based Expected Time to Compute (ETC) matrix generation method proposed by Ali et al. [17] and by Braun et al. [56]. An algorithm that uses this method as its basis is developed to generate fuzzy expected time to compute matrix $(FETC = [\tilde{e}_{ij}]_{m \times n})$.

One of the contributions of this thesis is that the proposed algorithm fuzzifies the concept presented by Ali et al. [17] using a random fuzzy vector and random fuzzy triangular numbers. The algorithm proceeds as follows. First of all a random fuzzy vector $\tilde{Z}_i = (z_{i1}, z_{i2}, z_{i3}), i = 1, 2, 3, \ldots, m$, is generated such that $\tilde{Z}_i \in [1, R_{task}) \quad \forall \quad i$. Several methods have been proposed in literature to generate random fuzzy vectors and random triangular fuzzy numbers in $[a, b)$ [57] [58]. This work uses the method proposed by Buckley and Jowers [58], in which a random fuzzy vector is obtained by three consecutively generated real random numbers and then ordering them. This way a sequence of random numbers is

made:

$$v_i = (x_{i1}, x_{i2}, x_{i3}), x_{i1} < x_{i2} < x_{i3}, i = 1, 2, 3, \ldots, m \quad in \quad [0, 1]^3$$

The random vector $\tilde{Z}_i$ is obtained from v_i by setting:

$$z_{ij} = (b - a)x_{ij} + a, j = 1, 2, 3 \quad and \quad i = 1, 2, 3, \ldots, m$$

. Clearly following is the expression for $\tilde{Z}_i$:

$$\tilde{Z}_i = ((b - a)x_{i1} + a, (b - a)x_{i2} + a, (b - a)x_{i3} + a), i = 1, 2, 3, \ldots, m$$

After generating $\tilde{Z}_i$, the rows of $FETC$ matrix are produced. Let $\tilde{e}_{ij}$ be an arbitrary element of $FETC$, then it is obtained as follows:

$$\tilde{e}_{ij} = \tilde{Z}_i \otimes \tilde{y}_j, j = 1, 2, 3, \ldots, n \quad and \quad i = 1, 2, 3, \ldots, m$$

where, $\tilde{y}_j$ is a random fuzzy triangular number in the range $[1, R_{mach}) \quad \forall \quad j$. It is obvious that all elements of $FETC$ are within the range $[1, R_{task} \times R_{mach})$. The variables R_{task} and R_{mach} represent task and machine heterogeneity [17] respectively. Algorithm 2 computes $FETC$ matrix: The execution cost of task τ_i on machine M_j is nothing but its execution time on M_j. So, the execution cost for task τ_i can be evaluated using equation:

$$\widetilde{\bigtriangledown}_{EC}(M_j) = \tilde{e}_{ij}$$

Machines with higher execution cost should have less likelihoods of executing τ_i , so above equation is revised and normalized as follows:

$$\widetilde{\bigtriangledown}_{EC}(M_j) = \frac{\max\limits_{i,j}(\tilde{e}_{ij}) - \tilde{e}_{ij}}{\max\limits_{i,j}(\tilde{e}_{ij})} \tag{1}$$

In equation (1) the term $\max\limits_{i,j}(\tilde{e}_{ij})$ corresponds to the largest element of $FETC$ matrix.

Algorithm 2 Generating $FETC$ Matrix

begin
Data: Values of R_{task} and R_{mach}
Result: $FETC$ Matrix
Generate three real random numbers in $[0, 1]$ and sort them in ascending order. $(x_{i1} < x_{i2} < x_{i3})$
for $i = 0$ **to** $m - 1$ **do**
    $\tilde{Z}_i \leftarrow ((R_{task} - 1)x_{i1} + 1, (R_{task} - 1)x_{i2} + 1, (R_{task} - 1)x_{i3} + 1)$
    for $j = 0$ **to** $n - 1$ **do**
        $\tilde{e}_{ij} \leftarrow \tilde{Z}_i \otimes \tilde{y}_j$
    end for
end for
end

5.4.3 Task Precedence Constraints and Priorities

Most task allocation algorithms are based on the list scheduling technique [8] [62] [79] [102] [104] [113] [134] [228]. The basic idea of list scheduling is to make a scheduling list (a sequence of tasks for scheduling) by assigning them some priorities, and then repeatedly execute the following two steps until all the tasks in the graph are allocated:

1. Select the first task from scheduling list;

2. Allocate the task to the machine which is most suitable to execute it.

Moreover, task allocation must be done in such a manner that the precedence constraints among the program tasks are preserved. Three frequently used attributes for assigning priorities to tasks (which lead to the preservation of the precedence constraints) are the *t-level* (top level), *b-level* (bottom level) and *static b-level* or simply *static level (sl)* [8] [13] [113]. The *t-level*, *b-level* and *static level* of a node (task) in a DAG are defined as follows:

Definition 5.4.1 The *t-level* of a task (node) τ_i is the length of a longest path

(there can be more than one longest path) from an entry task (node) to τ_i (excluding τ_i). Here, the length of a path is the sum of all the task and edge weights along the path.

Definition 5.4.2 The *b-level* of a task (node) τ_i is the length of a longest path from τ_i to an exit task (node).

Definition 5.4.3 Some allocation algorithms do not take into account the DAG edge weights in computing the *b-level*. To distinguish such definition of *b-level* from the one given in 5.4.2, it is called the *static b-level* or simply *static level (sl)*.

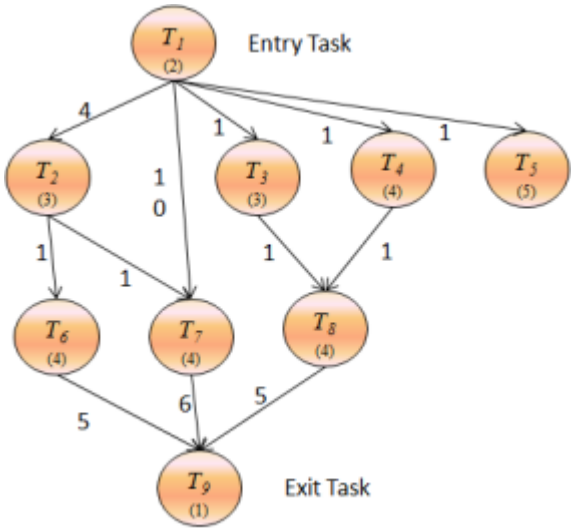

Figure 5.1: A Sample DAG

Consider the DAG shown in Figure 5.1. The node and edge weights indicate task execution times and inter-task communication costs respectively. This task graph is an example of DAGs addressed in this thesis. Table 5.2 shows the calculated values for the *t-level*, the *b-level* and the *static level (sl)*.

During the allocation process the *t-level* and *b-level* of a task frequently vary because the weight of an edge may be zeroed when the two incident tasks are scheduled to the same machine. On the other hand, the static-level does not change throughout the allocation process. An allocation algorithm which makes use of the dynamic *t-level* or *b-level* for assigning priorities to tasks is considered

Task	$t-level$	$b-level$	$static-level$
T_1	0	23	11
T_2	6	15	8
T_3	3	14	8
T_4	3	15	9
T_5	3	5	5
T_6	10	10	5
T_7	12	11	5
T_8	8	10	5
T_9	22	1	1

Table 5.2: b-level, t-level and static level Values

as dynamic allocation algorithm [13]. In contrast, if an algorithm uses a static attribute, such as *static-level*, to order nodes for allocation, it is called a static allocation algorithm [13]. The focus of this thesis is static task allocation with load balancing. So *static-level* is used to assign the priorities to tasks.

Algorithm 3 Task Priorities Assignment

1: **Data:** Directed Acyclic Graph
2: **Result:** Task Priorities List
3: **begin**
4: $priority(\tau) \leftarrow \phi$
5: **for** i =1 to m **do**
6: $\quad priority(\tau_i) \leftarrow \frac{\bar{e}_i}{\min\limits_k\{\bar{e}_k\}} + lp_i$
7: $\quad priority(\tau) \leftarrow priority(\tau) \| priority(\tau_i)$
8: **end for**
9: **end**

As pointed out earlier, the execution times of tasks are unknown and they are estimated using the range based ETC matrix generation method proposed by Ali et al. [17]. So instead of using the actual execution times of tasks the following

approach, formulated for assigning priorities to tasks, uses normalized average execution times of tasks. In this approach, the priority of a task τ_i is calculated as follows:

$$priority(\tau_i) = \left\lceil \frac{\overline{e}_i}{\min\limits_k \{\overline{e}_k\}} \right\rceil + lp_i$$

Where, $\overline{e}_i$ denotes the average execution time of task τ_i, $\min\limits_k \{\overline{e}_k\}$ denotes the minimum value of average execution time and lp_i corresponds to the length of a longest path from τ_i to an exit task. Algorithm 3 assigns priorities to tasks. Two or more tasks may have same priorities.

The task with a higher priority is selected for allocation before a task with a lower priority; if more than one task has the same priority, ties are broken by choosing the task to be scheduled next, randomly.

5.4.4 Communication Cost

Communication between tasks that are allocated on the distant machines tend to increase the communication cost. Obviously, the tasks that are allocated on the same machine will incur zero communication cost. This advantage may necessitate that as much as possible, tasks that require communication be allocated to the same machine in order to reduce overall communication cost. However, this might result in load imbalance. The goal therefore is to optimize allocation of communicating tasks such that they are executed on same or neighbouring machines whilst balancing the load.

If tasks τ_i and τ_j are two tasks and τ_i is allocated to machine M_x then communication cost incurred due to computing τ_j on machine M_y is given by $c_{ij} \times d_{xy}$, where c_{ij} is the amount of communication between tasks τ_i and τ_j, and d_{xy} is the distance between machines M_x and M_y.

Distance between two machines can easily be computed using the given topology graph. Fuzzification can be applied to c_{ij} considering how large c_{ij} is. The initial objective is to find the largest possible value of communication between any two tasks. Let it be denoted as $maxitc$. Table 5.3 details the criteria, linguistic

variables and the Triangular Fuzzy Numbers that are used for the fuzzification of communications between tasks. If a given task τ_i has h communicating tasks i.e.

Linguistic Variable	ITC Range	Membership Function
NO	$-\infty$ to 0	(0.0,0.0,0.0)
VL	0 to $0.2maxitc$	(0.0,0.1,0.3)
LO	$0.2maxitc$ to $0.4maxitc$	(0.1,0.3,0.5)
ME	$0.4maxitc$ to $0.6maxitc$	(0.3,0.5,0.7)
HI	$0.6maxitc$ to $0.8maxitc$	(0.5,.07,0.9)
VH	$0.8maxitc$ to $maxitc$	(0.7, 0.9, 1.0)

Table 5.3: Linguistic Variables for Fuzzification of ITC Cost

$\tau_1, \tau_2, \tau_3, \ldots, \tau_h$, which are allocated to machines $M_1, M_2, M_3, \ldots, M_h$ respectively then the total fuzzy communication cost incurred due to computing τ_i on machine M_j is given by the following equation:

$$\widetilde{\nabla}_{CC}(M_j) = \sum_{r=1}^{h} d(j,r) \otimes \tilde{c}(i,r)$$

Where, $\tilde{c}(i,r)$ denotes the amount of communication between tasks τ_i and τ_r, and $d(j,r)$ stands for the distance between machines M_j and M_r. For all cases when $r = j$ (i.e. when cost of computing τ_i on any of the machines $M_1, M_2, M_3, \ldots, M_h$ is determined) the corresponding term in the summation part of above equation is 0. In other words when $r = j$, $d(j,r) \otimes \tilde{c}(i,r) = 0$.

Machines with higher communication costs should have fewer chances of executing τ_i so above equation is revised and normalized as follows:

$$\widetilde{\nabla}_{CC}(M_j) = \frac{D \otimes \sum_{r=1}^{h} \tilde{c}(i,r) - \sum_{r=1}^{h} d(j,r) \otimes \tilde{c}(i,r)}{D \otimes \sum_{r=1}^{h} \tilde{c}(i,r)} \quad (2)$$

In equation (2), D stands for the diameter of the underlying topology and the term $D \otimes \sum_{r=1}^{h} \tilde{c}(i,r)$ corresponds to the maximum value of communication cost which may ensue between task τ_i and its communicating tasks.

5.4.5 Load Balancing

Considering only the execution and communication costs during task allocation may result in the assignment of heavier computational load to capable machines. Therefore, computational load should also be taken into account. By doing so fair allocation of tasks can be ensured. This work employs cumulative fuzzy expected time to compute of all previously assigned tasks to measure the computational load of a machine.

Let tasks $\tau_1, \tau_2, \tau_3, \ldots, \tau_k$,$(k < m)$, have been allocated to machine M_j, then the total load on machine M_j (which has been assigned k tasks), is given by ,

$$\widetilde{\bigtriangledown}_{LD}(M_j) = \sum_{i=1}^{k} \tilde{e}_{ij}$$

where $\tilde{e}_{ij}$ denotes the fuzzy expected time to compute of task τ_i on M_j. Once again machines with higher load should have less probability of executing a new task, so the computational load calculation is revised and normalized as follows:

$$\widetilde{\bigtriangledown}_{LD}(M_j) = \frac{(\sum_{j=1}^{n} \widetilde{\bigtriangledown}_{LD}(M_j) - \widetilde{\bigtriangledown}_{LD}(M_j))}{\sum_{j=1}^{n}(\sum_{j=1}^{n} \widetilde{\bigtriangledown}_{LD}(M_j) - \widetilde{\bigtriangledown}_{LD}(M_j))} \tag{3}$$

5.4.6 System Reliability

The reliability of an electronic component is the probability that the component is operational for execution of tasks that are assigned to it. The reliability of a heterogeneous multicomputer system can therefore be defined as the product of the components reliabilities. That is, the product of the probability that each machine be operational during task execution and the probability that each link be operational during inter-task communication. The importance of link reliability in a distributed system has been discussed by Awasthi et al. [32] in which they proposed that it is essential to consider link reliability as a component of cost function related to a distributed system. In a fuzzy environment the HMS reliability $\widetilde{R}$ can

be formulated as:

$$\widetilde{R} = \widetilde{R}_M \otimes \widetilde{R}_L$$

Where $\widetilde{R}_M$ stands for all machines fuzzy reliability and $\widetilde{R}_L$ stands for all links fuzzy reliability.

If a given task τ_i has communicating tasks i.e. $\tau_1, \tau_2, \tau_3, \ldots, \tau_h$, which are allocated to machines $M_1, M_2, M_3, \ldots, M_h$ respectively then the fuzzy system reliability for computing τ_i on machine M_j can be computed using following equation:

$$\widetilde{\bigtriangledown}_{REL}(M_j) = \widetilde{R}_j \otimes \prod_{i=1}^{h} \widetilde{R}_i \otimes \prod_{l \neq i}^{h} \widetilde{R}_{jl} \tag{4}$$

In equation (4) the first product term represents the total fuzzy reliability of all machines involved and the second product term represents the total fuzzy reliability of all links involved. Equation (4) further ensures that machines with higher system reliability are preferred for prospective assignment.

This study assumes that reliabilities of all machines and links constituting the HMS are represented as triangular fuzzy numbers. For experimental purposes they have been generated randomly using a computer program.

5.4.7 The Task Allocation Algorithm

Algorithm 4, which is developed based on the concepts presented above, handles all allocations. The algorithm is straight forward. For each task ready to be allocated, the suitability score of each machine is calculated considering various parameters described in equations (1)-(4). Machines are ranked according to their suitability scores and the task is awarded to the highest ranked machine.

Lemma 5.4.4 For m tasks and n machines the complexity of algorithm 4 is $O(m(n+1))$.

Proof:

1. The priority assignment algorithm takes m steps to assign priorities to m tasks.

Algorithm 4 $FLBTA-I$ Allocation Algorithm

begin
Data: Directed Acyclic Graph (T), Interconnection Network Topology Graph (M), ITC Matrix, Fuzzy Machine Reliabilities, Fuzzy Link Reliabilities
Result: $FA : T \rightarrow M$ (A Task Mapping)
Generate $FETC$ using Algorithm 2.
Generate Task Priorities List $priority(\tau)$ using Algorithm 3
Apply criteria shown in Table 5.3 to fuzzify ITC and obtain $FITC$
for $i = 0$ **to** $|T| - 1$ **do**
  Select Task τ_p, which has the highest priority in $priority(\tau)$
  for $j = 0$ **to** $|M| - 1$ **do**
    Compute $\widetilde{\nabla}_{EC}(M_j), \widetilde{\nabla}_{CC}(M_j), \widetilde{\nabla}_{LD}(M_j), \widetilde{\nabla}_{REL}(M_j)$ (equations (1) to (4))
    $\widetilde{\nabla}(M_j) \leftarrow \widetilde{\nabla}_{Ec}(M_j) + \widetilde{\nabla}_{CC}(M_j) + \widetilde{\nabla}_{LD}(M_j) + \widetilde{\nabla}_{REL}(M_j)$
    $score[j] \leftarrow rank(\widetilde{\nabla}(M_j))$
  end for
  if $score[l] > score[z] \quad \forall \quad z = 1, 2, \ldots, n \quad and \quad z \neq l$ **then**
    $k \leftarrow l$
  end if
  Award task τ_p to machine M_k
  $\{T\} \leftarrow \{T\} - \tau_p$
  $\{priority(\tau)\} \leftarrow \{priority(\tau)\} - priority(\tau_P)$
end for
end

2. To allocate a task $\tau \in T$, the algorithm ranks exactly n machines. So, mn steps are required to allocate m tasks.

Clearly, $(m + mn)$ active operations (steps) are required to allocate m tasks to n machines. Hence, the complexity of algorithm 4 is $O(m(n + 1))$.

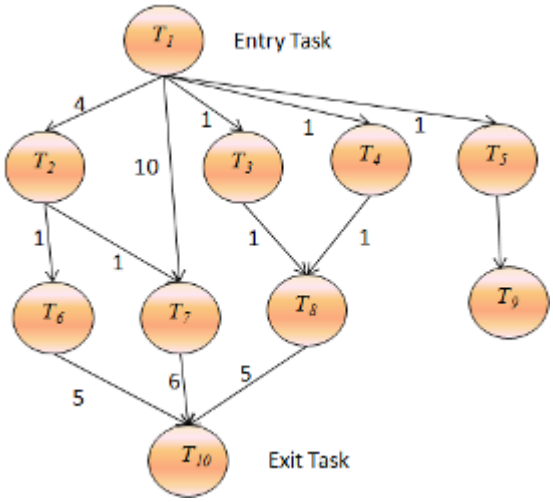

Figure 5.2: Task Graph

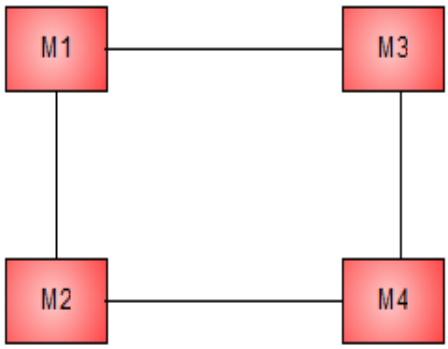

Figure 5.3: 4-Machine Graph

5.4.8 Experimental Setup and Implementation of FLBTA - I

To evaluate the proposed fuzzy load balancing task allocation model, randomly generated scenarios are used. Randomly generated scenarios are employed for following reasons:

1. It is desirable to obtain the results that validate the effectiveness of the proposed model over a broad range of conditions.

2. A generally accepted set of HMS benchmark tasks does not exist, and

3. It is not clear what characteristics a "typical" HMS task would exhibit.

The proposed fuzzy approach is implemented on a number of task and machine graphs. The model is simulated on an Ubuntu 12.04 LTS operating system by coding the algorithms presented within this thesis in C using CodeBlocks 12.11 compiler on an Intel Core i7 machine. The file FLBTA.sh runs the simulation. Following subsections present the results of various experiments:

4-machine 10-task Allocations:

Consider the task graph and machine graph shown in Figure 5.2 and 5.3 respectively. Task Graph analogous to one shown in Figure 5.2 can be generated using the random graph generators described by Topcuoglu et al. [210] or by Arabnejad and Barbosa [21]. The ITC matrix can be easily obtained by the task graph. The allocation proceeds as follows:

1. Obtain $FETC$ matrix using Algorithm 2.

2. High task and high machine heterogeneity is assumed so according to Ali et al. [17], $R_{task} = 10^5$ and $R_{mach} = 10^2$. Table 5.4 shows the computer generated $FETC$ matrix.

3. Obtain $FITC$ matrix using criteria shown in Table 5.3. Table 5.5 shows sample 10×4 excerpt from computer generated 10×10 ITC matrix.

4. Obtain Task Priorities using Algorithm 3. Table 5.8 shows the normalized average execution times of tasks and their priorities. It may also be noticed from Table 5.8 that T_1 has the highest (top) priority value and T_{10} has the lowest priority value. It indicates that T_1 is scheduled first and T_{10} is scheduled in the last.

	M_1	M_2	M_3	M_4
T_1	(5.174835e+006, 8.884815e+005, 2.795788e+005)	(4.809022e+006, 9.090524e+005, 6.356008e+005)	(3.102382e+006, 5.121537e+005, 2.987722e+005)	(4.343956e+006, 6.401172e+005, 1.098332e+005)
T_2	(7.271830e+006, 1.050841e+006, 5.954816e+005)	(2.556015e+006, 9.082768e+004, 2.443019e+004)	(2.905656e+006, 2.142637e+005, 9.082958e+004)	(4.778006e+006, 1.594033e+005, 5.197385e+004)
T_3	(2.750266e+006, 1.499191e+006, 1.679337e+005)	(3.066452e+006, 1.560214e+006, 1.358900e+005)	(1.191007e+006, 7.790295e+005, 1.735577e+005)	(4.266388e+006, 3.631704e+006, 1.337320e+005)
T_4	(2.636090e+006, 1.909979e+006, 1.137709e+006)	(2.329733e+006, 8.998402e+005, 3.988450e+005)	(4.168872e+005, 2.614417e+005, 2.806509e+004)	(5.917446e+005, 3.720859e+005, 6.786513e+004)
T_5	(1.215392e+005, 4.744388e+004, 1.361199e+004)	(1.383768e+005, 7.430940e+004, 1.334456e+004)	(7.920654e+004, 2.957462e+004, 3.717079e+003)	(4.459773e+004, 2.105972e+004, 2.041603e+003)
T_6	(7.862227e+005, 5.018878e+005, 1.256203e+005)	(1.022397e+006, 4.602895e+005, 3.322727e+004)	(1.039007e+006, 5.100162e+005, 1.076456e+005)	(1.255537e+006, 8.865920e+005, 1.508172e+005)
T_7	(1.105282e+006, 6.522251e+005, 3.076089e+005)	(5.460207e+005, 4.251507e+004, 1.212778e+004)	(8.641674e+005, 3.731596e+005, 2.673224e+005)	(7.035541e+005, 2.807508e+005, 6.343622e+004)
T_8	(2.753674e+006, 6.726262e+004, 2.219791e+004)	(2.081680e+006, 1.214727e+005, 4.749267e+004)	(3.581122e+006, 8.339630e+004, 3.473049e+004)	(5.660913e+006, 2.458267e+005, 8.428964e+004)
T_9	(1.438785e+006, 3.053171e+005, 2.995934e+004)	(1.010269e+006, 1.133963e+005, 8.089765e+003)	(1.596561e+006, 7.218953e+005, 9.984331e+004)	(2.608167e+006, 6.657863e+005, 8.124595e+004)
T_{10}	(3.133960e+006, 2.036002e+006, 1.128517e+006)	(1.796165e+006, 8.294919e+005, 1.428774e+005)	(3.789968e+006, 6.839037e+005, 5.590871e+005)	(2.462176e+006, 1.239195e+006, 7.399158e+005)

Table 5.4: Computer Generated $FETC$ Matrix

	T_1	T_2	T_3	T_4
T_1	(0.000000e+000,0.000000e+000,0.000000e+000)	(1.000000e-001,3.000000e-001,5.000000e-001)	(0.000000e+000,1.000000e-001,3.000000e-001)	(0.000000e+000,1.000000e-001,3.000000e-001)
T_2	(0.000000e+000,1.000000e-001,3.000000e-001)	(0.000000e+000,0.000000e+000,0.000000e+000)	(7.000000e-001,9.000000e-001,1.000000e+000)	(0.000000e+000,0.000000e+000,0.000000e+000)
T_3	(0.000000e+000,0.000000e+000,0.000000e+000)	(0.000000e+000,0.000000e+000,0.000000e+000)	(1.000000e-001,3.000000e-001,5.000000e-001)	(0.000000e+000,0.000000e+000,0.000000e+000)
T_4	(0.000000e+000,0.000000e+000,0.000000e+000)	(0.000000e+000,0.000000e+000,0.000000e+000)	(0.000000e+000,0.000000e+000,0.000000e+000)	(0.000000e+000,1.000000e-001,3.000000e-001)
T_5	(0.000000e+000,1.000000e-001,3.000000e-001)	(0.000000e+000,0.000000e+000,0.000000e+000)	(0.000000e+000,0.000000e+000,0.000000e+000)	(0.000000e+000,0.000000e+000,0.000000e+000)
T_6	(0.000000e+000,1.000000e-001,3.000000e-001)	(0.000000e+000,0.000000e+000,0.000000e+000)	(0.000000e+000,0.000000e+000,0.000000e+000)	(0.000000e+000,0.000000e+000,0.000000e+000)
T_7	(0.000000e+000,0.000000e+000,0.000000e+000)	(0.000000e+000,0.000000e+000,0.000000e+000)	(0.000000e+000,0.000000e+000,0.000000e+000)	(0.000000e+000,1.000000e-001,3.000000e-001)
T_8	(0.000000e+000,0.000000e+000,0.000000e+000)	(0.000000e+000,0.000000e+000,0.000000e+000)	(0.000000e+000,1.000000e-001,3.000000e-001)	(0.000000e+000,0.000000e+000,0.000000e+000)
T_9	(0.000000e+000,0.000000e+000,0.000000e+000)	(0.000000e+000,0.000000e+000,0.000000e+000)	(0.000000e+000,0.000000e+000,0.000000e+000)	(0.000000e+000,0.000000e+000,0.000000e+000)
T_{10}	(0.000000e+000,0.000000e+000,0.000000e+000)	(0.000000e+000,1.000000e-001,3.000000e-001)	(0.000000e+000,0.000000e+000,0.000000e+000)	(0.000000e+000,0.000000e+000,0.000000e+000)

Table 5.5: Sample 10×4 excerpt from computer generated 10×10 $FITC$ matrix

	M_1	M_2	M_3	M_4
M_1	0	1	1	2
M_2	1	0	2	1
M_3	1	2	0	1
M_4	2	1	1	0

Table 5.6: 4-Machine Distance Matrix

Machine	**Reliability**
M_1	(.9570, .9393, .9331)
M_2	(.9714, .9408, .9333)
M_3	(.9274, .9009, .9002)
M_4	(.9504, .9297, .9156)

Table 5.7: Randomly Generated 4-Machine Fuzzy Reliability

Task	**Normalized Average Execution Time**	**Priority**
T_1	33.0	95
T_2	28.0	76
T_3	36.0	80
T_4	20.0	90
T_5	1.0	94
T_6	13.0	26
T_7	9.0	43
T_8	21.0	61
T_9	14.0	46
T_{10}	31.0	16

Table 5.8: 10-Task Normalized Average Execution Times and Priorities

5. Table 5.6 and 5.7 list the machine connectivity matrix and randomly generated machine reliability matrix. The interconnection network topology is assumed to be a hypercube.

6. For the sake of simplicity, it is assumed that all links in the machine graph are equally reliable. The reliability of each link is the fuzzy number $(\tilde{0.9796})$ which may be considered as $(0.9796, 0.9796, 0.9796)$ (i.e. a triangular fuzzy number).

7. Priority wise Final allocation is presented in Table 5.9. Table 5.10 shows the summary of allocations.

8. Computer generated allocation results can be accessed using the url: `https://github.com/jahangir2016/results-new`. The file flbta1-4-10.txt contains the results.

Task	$score[1]$	$score[2]$	$score[3]$	$score[4]$	Allocated To
T_1	$-3.627326e-001$	$2.955747e-001$	$\mathbf{3.257845e+000}$	$1.277145e+000$	M_3
T_5	$9.168058e+000$	$9.131133e+000$	$9.244981e+000$	$\mathbf{1.243547e+001}$	M_4
T_4	$5.447038e+000$	$7.538294e+000$	$8.580367e+000$	$\mathbf{1.130240e+001}$	M_4
T_3	$\mathbf{7.103060e+000}$	$6.561715e+000$	$6.981317e+000$	$3.116477e+000$	M_1
T_2	$-2.571012e+000$	$\mathbf{1.011819e+001}$	$4.623762e+000$	$4.117594e+000$	M_2
T_8	$4.208941e+000$	$\mathbf{6.226380e+000}$	$2.447019e+000$	$2.359387e+000$	M_2
T_9	$8.473719e+000$	$7.063032e+000$	$7.905082e+000$	$\mathbf{9.759001e+000}$	M_4
T_7	$\mathbf{7.892464e+000}$	$6.722391e+000$	$7.695515e+000$	$7.517310e+000$	M_1
T_6	$7.938504e+000$	$7.758004e+000$	$\mathbf{8.248931e+000}$	$7.571169e+000$	M_3
T_{10}	$2.458924e+000$	$\mathbf{7.218460e+000}$	$2.626766e+000$	$5.146332e+000$	M_2

Table 5.9: 10-Task Allocation Results

8-machine 17-task Allocations:

Consider the task graph and machine graph shown in Figure 5.4 and 5.5 respectively. The ITC matrix can be easily obtained by the task graph. The allocation

Machine	No. of Tasks Allocated
M_1	2
M_2	3
M_3	2
M_4	3

Table 5.10: 10-Task Allocation Summary

proceeds as follows:

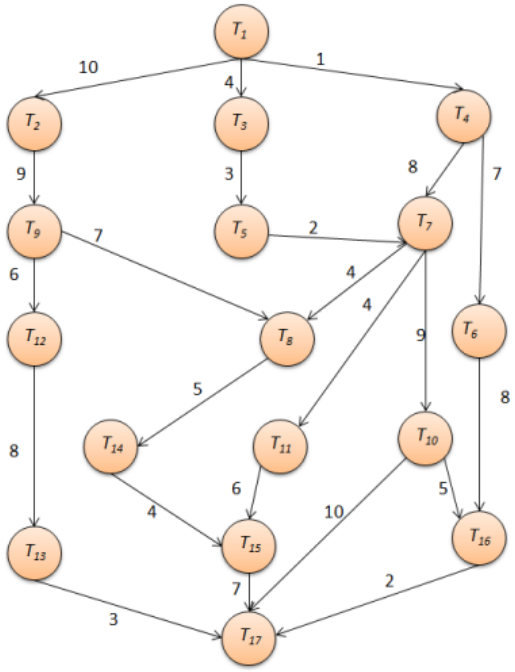

Figure 5.4: 17-Task DAG

1. Obtain $FETC$ matrix using Algorithm 2.

2. High task and high machine heterogeneity is assumed so according to Ali et al. [17], $R_{task} = 10^5$ and $R_{mach} = 10^2$. Table 5.11 shows sample 17×4 excerpt from computer generated 17×8 $FETC$ matrix.

3. Obtain $FITC$ matrix using criteria shown in Table 5.3.

	M_1	M_2	M_3	M_4
T_1	4.046272e+005, 8.385184e+004, 1.050163e+004	8.228888e+006, 3.265090e+006, 2.501323e+005	1.533784e+006, 6.368622e+005, 9.648965e+004	3.329971e+006, 9.268799e+005, 8.806199e+004
T_2	3.119617e+006, 2.615997e+005, 3.881855e+004	6.235697e+006, 8.566147e+005, 2.294863e+004	1.395092e+006, 5.925411e+005, 1.057943e+004	3.329699e+006, 2.307138e+005, 2.333760e+004
T_3	6.019862e+006, 5.208601e+005, 1.794866e+004	4.548469e+006, 2.211507e+005, 2.417894e+004	6.126231e+006, 1.091125e+006, 1.080361e+004	6.633451e+006, 1.836069e+006, 3.409098e+005
T_4	1.875031e+006, 4.538920e+005, 3.940853e+004	9.111113e+005, 2.072447e+005, 1.888814e+004	1.323132e+006, 4.809720e+005, 6.917544e+004	5.326815e+006, 3.554527e+005, 6.610975e+004
T_5	6.435656e+006, 1.694586e+005, 3.991689e+004	2.141583e+006, 7.512814e+004, 3.685346e+003	2.443951e+006, 4.359060e+004, 3.390224e+003	3.700970e+006, 1.337420e+005, 1.432051e+004
T_6	6.798696e+006, 9.394047e+004, 3.336239e+003	7.732050e+006, 5.642330e+004, 1.110297e+004	5.237530e+005, 2.375702e+004, 4.934213e+003	6.101342e+006, 8.300244e+004, 1.359351e+004
T_7	1.756977e+006, 1.102737e+005, 6.568523e+004	1.577710e+006, 8.610288e+005, 2.719630e+005	1.714902e+006, 8.507785e+005, 1.932347e+005	1.623222e+006, 5.589245e+005, 1.805344e+005
T_8	2.472571e+006, 1.731276e+005, 2.737942e+004	1.663823e+006, 2.047798e+005, 3.127146e+004	4.097270e+006, 7.255212e+005, 3.113613e+005	3.817810e+006, 2.205496e+005, 1.092487e+005
T_9	5.574646e+005, 5.621383e+004, 1.229050e+004	2.261992e+006, 6.037371e+005, 1.841023e+005	1.711418e+006, 4.611822e+005, 1.079268e+005	1.542664e+006, 7.568705e+004, 3.970702e+004
T_{10}	3.560327e+006, 6.929272e+005, 2.572484e+004	9.712875e+005, 2.612933e+005, 3.532419e+004	9.892584e+005, 1.111392e+005, 4.916346e+004	1.385603e+006, 2.353064e+005, 7.710401e+004
T_{11}	1.044319e+006, 4.540770e+005, 2.433214e+005	7.216124e+005, 3.502822e+004, 2.666898e+004	2.322572e+006, 3.672868e+004, 2.054649e+004	3.460146e+006, 8.762432e+005, 1.108995e+005
T_{12}	2.959033e+006, 2.580712e+005, 3.158552e+004	3.223764e+006, 1.662307e+006, 1.079133e+006	1.574740e+006, 5.142164e+005, 2.061691e+005	1.553832e+005, 6.115855e+004, 3.664121e+004
T_{13}	2.388651e+005, 2.069235e+005, 7.923197e+004	4.619197e+005, 2.063043e+005, 1.033539e+005	6.949313e+005, 1.069045e+005, 3.302491e+004	3.611466e+005, 1.624624e+005, 7.292926e+004
T_{14}	8.526506e+005, 4.181575e+005, 1.715285e+004	7.138524e+005, 2.287454e+005, 1.009968e+005	8.639144e+005, 2.112830e+005, 7.878361e+004	3.834913e+005, 1.170104e+005, 2.276380e+004
T_{15}	1.141715e+006, 2.247492e+005, 4.881331e+004	1.470256e+006, 3.639248e+005, 2.308235e+005	1.663336e+006, 4.030932e+005, 2.858938e+005	9.258317e+005, 4.152695e+005, 2.876316e+005
T_{16}	1.424514e+006, 8.415666e+005, 1.521500e+004	1.785970e+006, 2.045264e+005, 1.442016e+005	3.532431e+005, 1.040509e+005, 2.668477e+004	2.355628e+005, 1.779522e+004, 1.540810e+004
T_{17}	6.413720e+005, 4.391041e+005, 3.648887e+004	2.767449e+006, 3.870127e+005, 1.235928e+004	2.178678e+006, 1.365975e+006, 5.712821e+004	2.561642e+006, 7.318182e+005, 7.817550e+004

Table 5.11: 17×4 Excerpt From Computer Generated $FETC$ Matrix

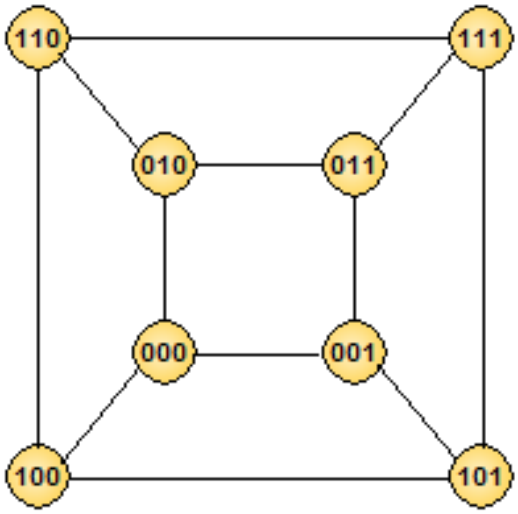

Figure 5.5: 8-Machine Graph

	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8
M_1	0	2	1	1	1	3	2	2
M_2	2	0	1	1	3	1	2	2
M_3	1	1	0	2	2	2	1	3
M_4	1	1	2	0	2	2	3	1
M_5	1	3	2	2	0	2	1	1
M_6	3	1	2	2	2	0	1	1
M_7	2	2	1	3	1	1	0	2
M_8	2	2	3	1	1	1	2	0

Table 5.12: 8-Machine Distance Matrix

Machine	Reliability
M_1	(0.9741,0.9543,0.9502)
M_2	(0.9747,0.9622,0.9586)
M_3	(0.9882,0.9684,0.9581)
M_4	(0.9829,0.9555,0.9534)
M_5	(0.9737,0.9595,0.9561)
M_6	(0.9846,0.9707,0.9580)
M_7	(0.9773,0.9723,0.9681)
M_8	(0.9545,0.9512,0.9507)

Table 5.13: Randomly Generated 8-Machine Fuzzy Reliability

4. Obtain Task Priorities using Algorithm 3. Table 5.14 shows the normalized average execution times of tasks and their priorities.

Task	Normalized Average Execution Time	Priority
T_1	5.0	60
T_2	6.0	53
T_3	5.0	59
T_4	4.0	47
T_5	3.0	47
T_6	3.0	21
T_7	1.0	42
T_8	2.0	34
T_9	4.0	43
T_{10}	4.0	17
T_{11}	3.0	18
T_{12}	8.0	20
T_{13}	10.0	17
T_{14}	2.0	28
T_{15}	15.0	5
T_{16}	25.0	9
T_{17}	9.0	2

Table 5.14: 17-Task Normalized Average Execution Times and Priorities

5. Table 5.12 and 5.13 list the machine connectivity matrix and randomly generated machine reliability matrix. The interconnection network topology is assumed to be a hypercube.

6. For the sake of simplicity it is assumed that all links in the machine graph are equally reliable. The reliability of each link is the fuzzy number $(0.\tilde{9}796)$

which may be considered as $(0.9796, 0.9796, 0.9796)$ (i.e. a triangular fuzzy number).

7. Priority wise Final allocation is presented in Table 5.16. Table 5.15 shows the summary of allocations.

8. Computer generated allocation results can be accessed using the url: `https://github.com/jahangir2016/results-new`. The file flbta1-8-17.txt contains the results.

8-machine 41-task Allocations:

Consider the task graph and machine graph shown in Figures 5.8 and 5.5 respectively. The ITC matrix can be easily obtained by the task graph. The allocation proceeds as follows:

1. Obtain $FETC$ matrix using Algorithm 2.

2. High task and high machine heterogeneity is assumed so according to Ali et al. [17], $R_{task} = 10^5$ and $R_{mach} = 10^2$.

3. Obtain $FITC$ matrix using criteria shown in Table 5.3.

4. Obtain Task Priorities using Algorithm 3. Table 5.23 shows the normalized average execution times of tasks and their priorities.

5. Table 5.12 and 5.13 list the machine connectivity matrix and randomly generated machine reliability matrix. The interconnection network topology is assumed to be a hypercube.

6. For the sake of simplicity it is assumed that all links in the machine graph are equally reliable. The reliability of each link is the fuzzy number $(0.\tilde{9}796)$ which may be considered as $(0.9796, 0.9796, 0.9796)$ (i.e. a triangular fuzzy number).

Task	$score[1]$	$score[2]$	$score[3]$	$score[4]$	$score[5]$	$score[6]$	$score[7]$	$score[8]$	Allocated To
T_1	4.250892e+000	1.167216e+000	3.766972e+000	3.073753e+000	3.240938e+000	2.030062e+000	3.840723e+000	3.179646e+000	M_1
T_3	3.813045e+000	1.430946e+001	1.481046e+001	1.433562e+001	1.322765e+001	1.181215e+001	1.492550e+001	1.289262e+001	M_7
T_2	3.156856e+000	1.316158e+001	1.284067e+001	1.255360e+001	1.446272e+001	1.414299e+001	1.840887e+000	1.288538e+001	M_5
T_4	4.063211e+000	1.416034e+001	1.412138e+001	1.418098e+001	3.794985e+000	1.647452e+001	1.219302e+001	1.570889e+001	M_6
T_5	1.510978e+001	1.706827e+001	1.789429e+001	1.679428e+001	3.603892e+000	6.198246e+000	1.426431e+001	1.625562e+001	M_3
T_9	1.416921e+001	1.673733e+001	4.696467e+000	1.642651e+001	3.910579e+000	5.926561e+000	1.278339e+001	1.578841e+001	M_2
T_7	1.474768e+001	4.770876e+000	2.377902e+001	6.169498e+001	2.191928e+001	2.659566e+001	4.998139e+001	6.145597e+001	M_4
T_8	5.299338e+001	4.652345e+000	2.404030e+001	5.245592e+001	2.217639e+001	2.686243e+001	5.122069e+001	6.164925e+001	M_8
T_{14}	5.220800e+001	4.244929e+000	2.353735e+001	5.272039e+001	2.220355e+001	2.726188e+001	5.036700e+001	5.693309e+001	M_8
T_{16}	5.117873e+001	3.317460e+000	2.165318e+001	5.287898e+001	2.065466e+001	2.501344e+001	4.923271e+001	3.841968e+001	M_4
T_6	5.296611e+001	4.937823e+000	2.351350e+001	2.724896e+001	2.134753e+001	2.567711e+001	5.008022e+001	4.075107e+001	M_1
T_{12}	2.360278e+001	2.227525e+000	2.409979e+001	2.831897e+001	2.130899e+001	2.743609e+001	4.881655e+001	3.936071e+001	M_7
T_{11}	2.607404e+001	4.716624e+000	2.398287e+001	2.708356e+001	2.236344e+001	2.693256e+001	5.108062e+000	4.958470e+001	M_8
T_{10}	4.324728e+001	3.632918e+001	4.326946e+001	4.371981e+001	4.253964e+001	4.424710e+001	5.184096e+000	2.647512e+001	M_6
T_{13}	4.498722e+001	3.511567e+001	4.296991e+001	4.317918e+001	4.103250e+001	1.167795e+001	2.413145e+000	2.591909e+001	M_1
T_{15}	2.506920e+001	3.709886e+001	4.424581e+001	4.560594e+001	4.295603e+001	1.161111e+001	4.001254e+000	2.671171e+001	M_4
T_{17}	2.559136e+001	3.784804e+001	4.475150e+001	2.280019e+001	4.395857e+001	1.171760e+001	5.143972e+000	2.798026e+001	M_3

Table 5.15: 17-Task Allocation Results

Machine	No. of Tasks Allocated
M_1	3
M_2	1
M_3	2
M_4	3
M_5	1
M_6	2
M_7	2
M_8	3

Table 5.16: 17-Task Allocation Summary

7. Table 5.17 shows the summary of allocations. The speedup and efficiency values are approximately same as they are in 8-machine 17-task allocations.

8. Computer generated allocation results can be accessed using the url: `https://github.com/jahangir2016/results-new`. The file flbta1-8-41.txt contains the results.

Machine	No. of Tasks Allocated
M_1	5
M_2	7
M_3	4
M_4	3
M_5	3
M_6	6
M_7	6
M_8	7

Table 5.17: 41-Task 8-Machine Allocation Summary

12-machine 30-task Allocations:

Consider the task graph and machine graph shown in Figure 5.6 and 5.7 respectively. The ITC matrix can be easily obtained by the task graph. The allocation proceeds as follows:

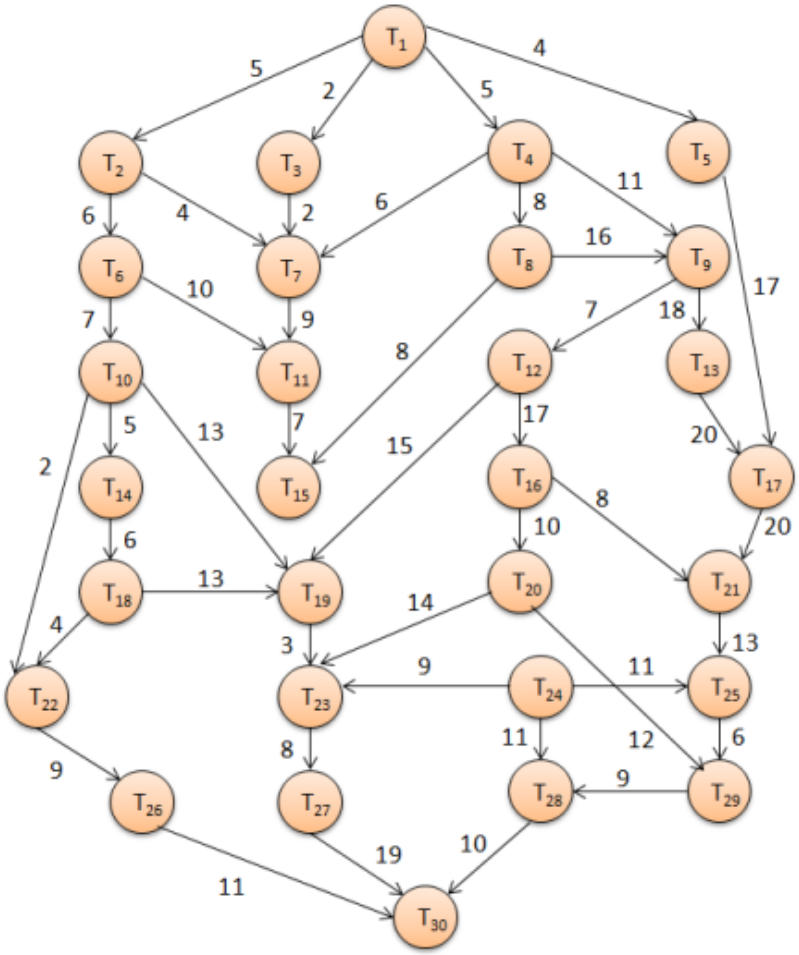

Figure 5.6: 30-Task DAG

1. Obtain $FETC$ matrix using Algorithm 2.
2. High task and high machine heterogeneity is assumed so according to Ali et al. [17], $R_{task} = 10^5$ and $R_{mach} = 10^2$.
3. Obtain $FITC$ matrix using criteria shown in Table 5.3.

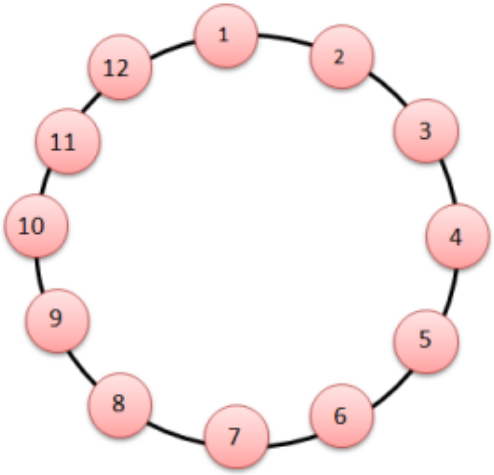

Figure 5.7: 12-Machine Graph

	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}	M_{11}	M_{12}
M_1	0	1	2	3	4	5	6	5	4	3	2	1
M_2	1	0	1	2	3	4	5	6	5	4	3	2
M_3	2	1	0	1	2	3	4	5	6	5	4	3
M_4	3	2	1	0	1	2	3	4	5	6	5	4
M_5	4	3	2	1	0	1	2	3	4	5	6	5
M_6	5	4	3	2	1	0	1	2	3	4	5	6
M_7	6	5	4	3	2	1	0	1	2	3	4	5
M_8	5	6	5	4	3	2	1	0	1	2	3	4
M_9	4	5	6	5	4	3	2	1	0	1	2	3
M_{10}	3	4	5	6	5	4	3	2	1	0	1	2
M_{11}	2	3	4	5	6	5	4	3	2	1	0	1
M_{12}	1	2	3	4	5	6	5	4	3	2	1	0

Table 5.18: 12-Machine Distance Matrix

Machine	Reliability
M_1	(0.9842,0.9561,0.9503)
M_2	(0.9512,0.9501,0.9501)
M_3	(0.9824,0.9752,0.9570)
M_4	(0.9669,0.9523,0.9504)
M_5	(0.9723,0.9585,0.9513)
M_6	(0.9654,0.9550,0.9523)
M_7	(0.9517,0.9505,0.9505)
M_8	(0.9601,0.9561,0.9501)
M_9	(0.9730,0.9527,0.9517)
M_{10}	(0.9564,0.9558,0.9556)
M_{11}	(0.9605,0.9584,0.9580)
M_{12}	(0.9804,0.9639,0.9616)

Table 5.19: Randomly Generated 12-Machine Fuzzy Reliability

4. Obtain Task Priorities using Algorithm 3. Table 5.20 shows the normalized average execution times of tasks and their priorities.

5. Table 5.18 and 5.19 list the machine connectivity matrix and randomly generated machine reliability matrix. The interconnection network topology is assumed to be a ring (as size 12 is not supported by the hypercube).

6. For the sake of simplicity it is assumed that all links in the machine graph are equally reliable. The reliability of each link is the fuzzy number $(\tilde{0.9796})$ which may be considered as $(0.9796, 0.9796, 0.9796)$ (i.e. a triangular fuzzy number).

7. Table 5.21 shows the summary of allocations.

8. Computer generated allocation results can be accessed using the url: `https://github.com/jahangir2016/results-new`. The file flbta1-12-30.txt contains the results.

Task	Normalized Average Execution Time	Priority
T_1	1.0	429
T_2	2.0	404
T_3	4.0	176
T_4	28.0	372
T_5	13.0	283
T_6	5.0	368
T_7	33.0	144
T_8	5.0	361
T_9	20.0	144
T_{10}	18.0	297
T_{11}	6.0	95
T_{12}	9.0	223
T_{13}	9.0	303
T_{14}	8.0	221
T_{15}	14.0	87
T_{16}	2.0	205
T_{17}	17.0	235
T_{18}	10.0	206
T_{19}	16.0	102
T_{20}	16.0	117
T_{21}	12.0	173
T_{22}	23.0	154
T_{23}	5.0	70
T_{24}	13.0	186
T_{25}	11.0	100
T_{26}	2.0	83
T_{27}	2.0	45
T_{28}	4.0	75
T_{29}	6.0	76
T_{30}	6.0	28

Table 5.20: 30-Task Normalized Average Execution Times and Priorities

Machine	No. of Tasks Allocated
M_1	2
M_2	3
M_3	3
M_4	3
M_5	2
M_6	3
M_7	3
M_8	1
M_9	2
M_{10}	3
M_{11}	3
M_{12}	2

Table 5.21: 30-Task Allocation Summary

16-machine 41-task Allocations:

Consider the task graph and machine graph shown in Figures 5.8 and 5.9 respectively. The ITC matrix can be easily obtained by the task graph. The allocation proceeds as follows:

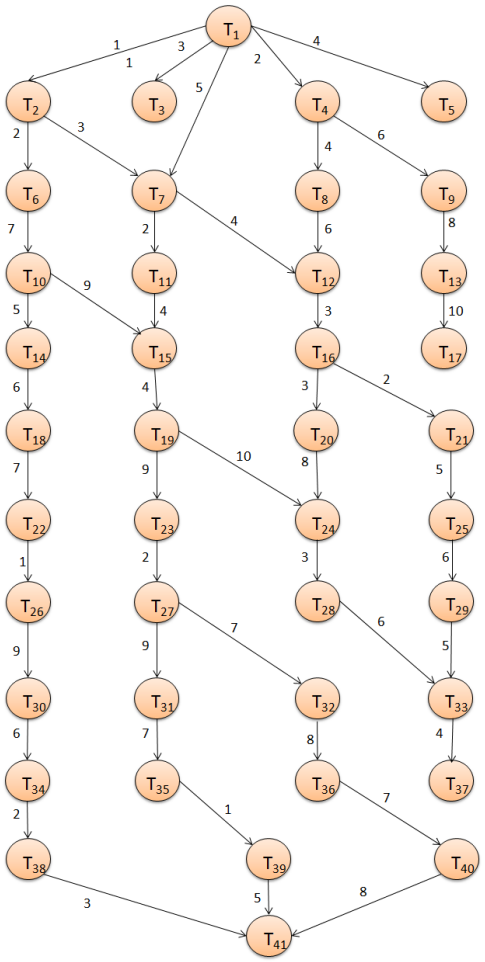

Figure 5.8: 41-Task DAG

1. Obtain $FETC$ matrix using Algorithm 2.
2. High task and high machine heterogeneity is assumed so according to Ali et al. [17], $R_{task} = 10^5$ and $R_{mach} = 10^2$.
3. Obtain $FITC$ matrix using criteria shown in Table 5.3.

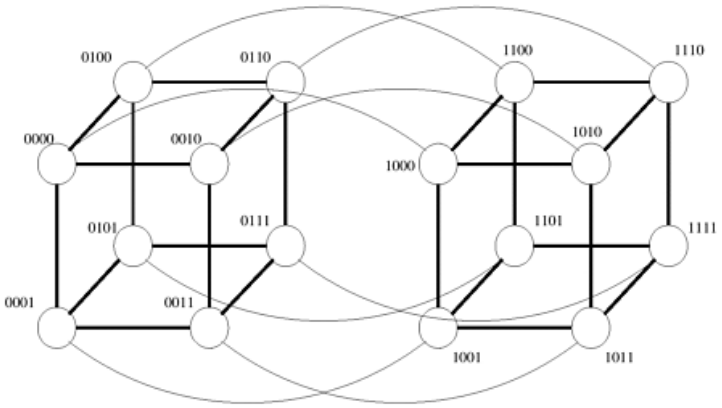

Figure 5.9: 16-Machine Graph

Machine	Reliability
M_1	(0.9712,0.9601,0.9531)
M_2	(0.9896,0.9683,0.9649)
M_3	(0.9662,0.9609,0.9543)
M_4	(0.9727,0.9713,0.9646)
M_5	(0.9642,0.9552,0.9509)
M_6	(0.9554,0.9509,0.9502)
M_7	(0.9651,0.9535,0.9523)
M_8	(0.9884,0.9503,0.9501)
M_9	(0.9797,0.9795,0.9530)
M_{10}	(0.9776,0.9760,0.9738)
M_{11}	(0.9543,0.9518,0.9514)
M_{12}	(0.9648,0.9644,0.9592)
M_{13}	(0.9632,0.9528,0.9503)
M_{14}	(0.9808,0.9632,0.9619)
M_{15}	(0.9741,0.9584,0.9544)
M_{16}	(0.9849,0.9679,0.9575)

Table 5.22: Randomly Generated 16-Machine Fuzzy Reliability

4. Obtain Task Priorities using Algorithm 3. Table 5.23 shows the normalized average execution times of tasks and their priorities.

5. Table 5.22 lists the randomly generated machine reliability matrix. Machine distance matrix can be easily deduced form the machine graph shown in Figure 5.9. The interconnection network topology is assumed to be a hypercube.

6. For the sake of simplicity it is assumed that all links in the machine graph are equally reliable. The reliability of each link is the fuzzy number $(0.\tilde{9}796)$ which may be considered as $(0.9796, 0.9796, 0.9796)$ (i.e. a triangular fuzzy number).

7. Table 5.24 shows the summary of allocations.

8. Computer generated allocation results can be accessed using the url: `https://github.com/jahangir2016/results-new`. The file flbta1-16-41.txt contains the results.

16-machine 50-task Allocations:

Consider the task graph and machine graph shown in Figures 5.10 and 5.9 respectively. The ITC matrix can be easily obtained by the task graph. The allocation proceeds as follows:

1. Obtain $FETC$ matrix using Algorithm 2.

2. High task and high machine heterogeneity is assumed so according to Ali et al. [17], $R_{task} = 10^5$ and $R_{mach} = 10^2$.

3. Obtain $FITC$ matrix using criteria shown in Table 5.3.

4. Obtain Task Priorities using Algorithm 3. Table 5.27 shows the normalized average execution times of tasks and their priorities.

Task	Normalized Average Execution Time	Priority
T_1	24.000000	163
T_2	60.000000	149
T_3	10.000000	26
T_4	17.000000	107
T_5	5.000000	2
T_6	111.000000	137
T_7	32.000000	141
T_8	45.000000	89
T_9	44.000000	36
T_10	5.000000	126
T_{11}	65.000000	109
T_{12}	21.000000	74
T_{13}	65.000000	19
T_{14}	67.000000	22
T_{15}	54.000000	95
T_{16}	24.000000	68
T_{17}	87.000000	7
T_{18}	85.000000	119
T_{19}	36.000000	68
T_{20}	99.000000	59
T_{21}	31.000000	54
T_{22}	21.000000	101
T_{23}	115.000000	50
T_{24}	19.000000	47
T_{25}	67.000000	52
T_{26}	11.000000	88
T_{27}	62.000000	39
T_{28}	68.000000	45
T_{29}	1.000000	40
T_{30}	68.000000	71
T_{31}	39.000000	35
T_{32}	50.000000	30
T_{33}	84.000000	33
T_{34}	2.000000	53
T_{35}	63.000000	33
T_{36}	4.000000	27
T_{37}	11.000000	11
T_{38}	12.000000	28
T_{39}	41.000000	25
T_{40}	19.000000	25
T_{41}	37.000000	20

Table 5.23: 41-Task Normalized Average Execution Times and Priorities

Machine	No. of Tasks Allocated
M_1	1
M_2	3
M_3	2
M_4	1
M_5	3
M_6	4
M_7	4
M_8	2
M_9	2
M_{10}	2
M_{11}	3
M_{12}	3
M_{13}	2
M_{14}	2
M_{15}	4
M_{16}	3

Table 5.24: 41-Task Allocation Summary

5. Table 5.22 lists the randomly generated machine reliability matrix. Machine distance matrix can be easily deduced form the machine graph shown in 5.9. The interconnection network topology is assumed to be a hypercube.

6. For the sake of simplicity it is assumed that all links in the machine graph are equally reliable. The reliability of each link is the fuzzy number $(0.\tilde{9}796)$ which may be considered as $(0.9796, 0.9796, 0.9796)$ (i.e. a triangular fuzzy number).

7. Table 5.25 shows the summary of allocations. The speedup and efficiency values are approximately same as they are in 16-machine 41-task allocations.

8. Computer generated allocation results can be accessed using the url: `https://github.com/jahangir2016/results-new`. The file flbta1-16-50.txt contains the results.

32-machine 50-task Allocations:

Consider the task graph and machine graph shown in Figures 5.10 and 5.11 respectively. The ITC matrix can be easily obtained by the task graph. The allocation proceeds as follows:

1. Obtain $FETC$ matrix using Algorithm 2.

Machine	No. of Tasks Allocated
M_1	5
M_2	4
M_3	2
M_4	2
M_5	3
M_6	3
M_7	4
M_8	3
M_9	4
M_{10}	2
M_{11}	1
M_{12}	4
M_{13}	3
M_{14}	4
M_{15}	1
M_{16}	5

Table 5.25: 50-Task 16-Machine Allocation Summary

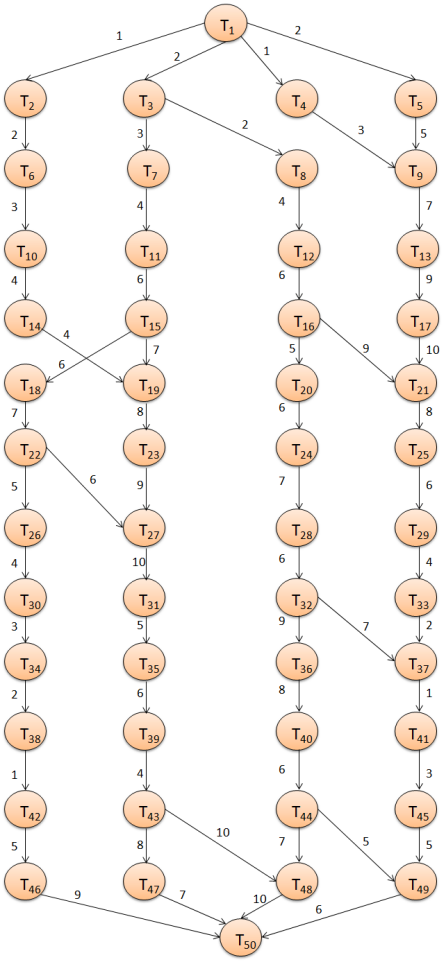

Figure 5.10: 50-Task DAG

Machine	Reliability
M_1	(0.9858,0.9578,0.9555)
M_2	(0.9650,0.9633,0.9507)
M_3	(0.9527,0.9505,0.9505)
M_4	(0.9811,0.9664,0.9602)
M_5	(0.9549,0.9540,0.9515)
M_6	(0.9633,0.9538,0.9535)
M_7	(0.9887,0.9750,0.9652)
M_8	(0.9833,0.9776,0.9503)
M_9	(0.9856,0.9614,0.9517)
M_{10}	(0.9541,0.9508,0.9507)
M_{11}	(0.9667,0.9606,0.9557)
M_{12}	(0.9536,0.9533,0.9527)
M_{13}	(0.9787,0.9715,0.9674)
M_{14}	(0.9802,0.9626,0.9517)
M_{15}	(0.9833,0.9747,0.9542)
M_{16}	(0.9791,0.9604,0.9530)
M_{17}	(0.9641,0.9612,0.9530)
M_{18}	(0.9502,0.9502,0.9502)
M_{19}	(0.9656,0.9509,0.9502)
M_{20}	(0.9666,0.9615,0.9581)
M_{21}	(0.9641,0.9546,0.9513)
M_{22}	(0.9642,0.9519,0.9519)
M_{23}	(0.9567,0.9559,0.9540)
M_{24}	(0.9523,0.9518,0.9504)
M_{25}	(0.9564,0.9518,0.9514)
M_{26}	(0.9565,0.9505,0.9504)
M_{27}	(0.9548,0.9511,0.9506)
M_{28}	(0.9702,0.9662,0.9661)
M_{29}	(0.9721,0.9672,0.9619)
M_{30}	(0.9551,0.9529,0.9522)
M_{31}	(0.9640,0.9524,0.9512)
M_{32}	(0.9810,0.9563,0.9539)

Table 5.26: Randomly Generated 32-Machine Fuzzy Reliability

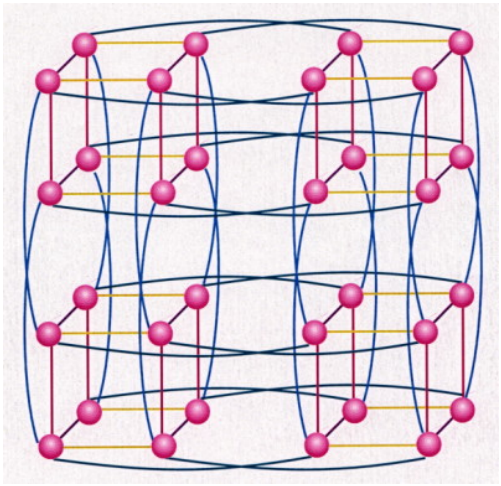

Figure 5.11: 32-Machine Graph

2. High task and high machine heterogeneity is assumed so according to Ali et al. [17], $R_{task} = 10^5$ and $R_{mach} = 10^2$.

3. Obtain $FITC$ matrix using criteria shown in Table 5.3.

4. Obtain Task Priorities using Algorithm 3. Table 5.27 shows the normalized average execution times of tasks and their priorities.

5. Table 5.26 lists the randomly generated machine reliability matrix. Machine distance matrix can be easily deduced form the machine graph shown in 5.9. The interconnection network topology is assumed to be a hypercube.

6. For the sake of simplicity it is assumed that all links in the machine graph are equally reliable. The reliability of each link is the fuzzy number $(0.\tilde{9796})$ which may be considered as $(0.9796, 0.9796, 0.9796)$ (i.e. a triangular fuzzy number).

7. Table 5.28 shows the summary of allocations.

8. Computer generated allocation results can be accessed using the url: `https://github.com/jahangir2016/results-new`. The file flbta1-32-50.txt contains the results.

Task	Normalized Average Execution Time	Priority
T_1	138.000000	608
T_2	115.000000	363
T_3	92.000000	460
T_4	144.000000	599
T_5	250.000000	550
T_6	137.000000	357
T_7	113.000000	367
T_8	35.000000	440
T_9	26.000000	535
T_{10}	11.000000	347
T_{11}	140.000000	320
T_{12}	1.000000	417
T_{13}	158.000000	528
T_{14}	213.000000	291
T_{15}	165.000000	309
T_{16}	105.000000	414
T_{17}	23.000000	451
T_{18}	183.000000	259
T_{19}	253.000000	254
T_{20}	19.000000	348
T_{21}	165.000000	401
T_{22}	217.000000	253
T_{23}	48.000000	221
T_{24}	210.000000	327
T_{25}	17.000000	385
T_{26}	114.000000	218
T_{27}	122.000000	220
T_{28}	150.000000	297
T_{29}	151.000000	323
T_{30}	189.000000	192
T_{31}	46.000000	171
T_{32}	91.000000	246
T_{33}	26.000000	268
T_{34}	76.000000	154
T_{35}	86.000000	112
T_{36}	95.000000	209
T_{37}	248.000000	215
T_{38}	84.000000	147
T_{39}	44.000000	100
T_{40}	72.000000	169
T_{41}	20.000000	180
T_{42}	94.000000	145
T_{43}	138.000000	96
T_{44}	152.000000	161
T_{45}	72.000000	124
T_{46}	153.000000	95
T_{47}	160.000000	60
T_{48}	40.000000	87
T_{49}	123.000000	99
T_{50}	247.000000	56

Table 5.27: 50-Task Normalized Average Execution Times and Priorities

Machine	No. of Tasks Allocated
M_1	1
M_2	2
M_3	1
M_4	2
M_5	3
M_6	2
M_7	1
M_8	3
M_9	1
M_{10}	1
M_{11}	2
M_{12}	2
M_{13}	2
M_{14}	2
M_{15}	1
M_{16}	2
M_{17}	2
M_{18}	1
M_{19}	1
M_{20}	2
M_{21}	1
M_{22}	1
M_{23}	1
M_{24}	1
M_{25}	1
M_{26}	1
M_{27}	2
M_{28}	1
M_{29}	2
M_{30}	2
M_{31}	2
M_{32}	1

Table 5.28: 50-Task Allocation Summary

5.4.9 Performance Evaluation of $FLBTA-I$

Performance Metrics

The following metrics have been used to gauge the performance of the underlying fuzzy task allocation model.

- **Speedup:** For an underlying task graph, the speedup value is determined by dividing the sequential execution time by the parallel execution time (1.e. makespan of the allocation). The sequential execution time is obtained by allocating all tasks to a single machine that minimizes the total execution time of all tasks. Hence,

$$Speedup = \frac{\min\limits_{i}\{se_i\}}{\max\limits_{i}\{te_i\}}$$

  where, se_i is the sequential execution time of the parallel application on machine M_i.

- **Efficiency:** The efficiency of a parallel application is a measure of how well are the processors utilized in a parallel system. It is defined as follows:

$$Efficiency = \left(\frac{Speedup}{No.of Processors}\right) \times 100$$

$FLBTA-I$ has been evaluated on number of different task and machine graphs using the above metrics. For different scales of hypercube interconnection network, results are presented in Table 5.29:

Figure 5.12 shows the speedup and efficiency variations as the system is scaled up from 4 to 32 machines. From Table 5.29 and Figure 5.12, it is obvious that:

- The speedup increases as the number of machines in the system are increased.

- The efficiency decreases as the system is scaled up. This is because of the increase in the communication overhead. This tradeoff between speedup and efficiency is inherent to a parallel system [95].

No. of Machines	Average Speedup (Over 10 Runs)	Efficiency(% approx.)
4	3.2	84
8	6.6	82
12	8.7	72
16	11.0	69
32	20.1	63

Table 5.29: $FLBTA - I$ Performance

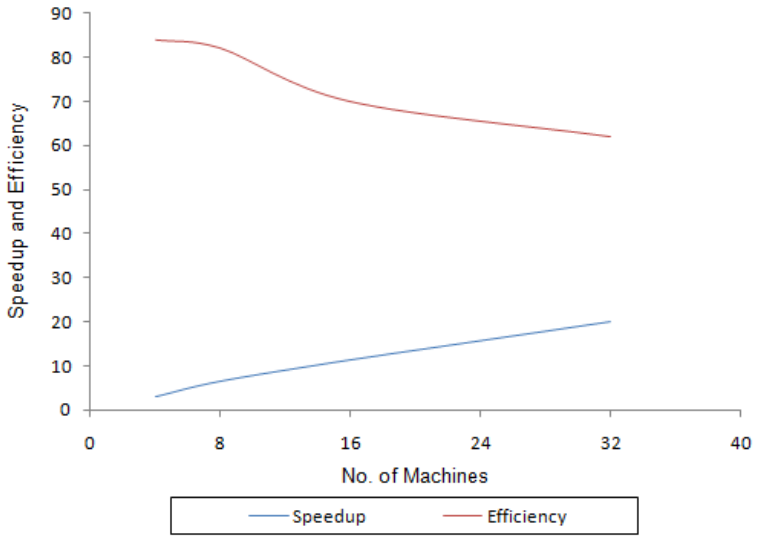

Figure 5.12: Speedup and Efficiency Variation

5.4.10 Load Balancing

In each case the $FLBTA-I$ performs load balancing by appropriately allocating tasks as shown in Table 5.10. Figures 5.13-5.16 show the load distribution (cumulative execution time) on various machines when tasks are allocated to different machine configurations using $FLBTA-I$:

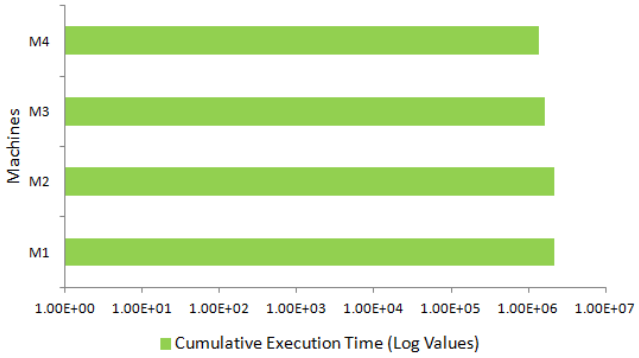

Figure 5.13: Load Balancing in 4-Machine Allocations

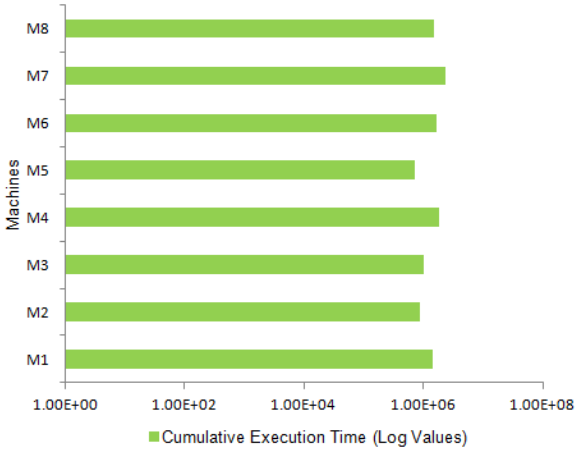

Figure 5.14: Load Balancing in 8-Machine Allocations

5.4.11 Comparative Study

Topcuoglu et al. [210] have proposed Heterogeneous-Earliest-Finish-Time ($HEFT$) and Critical-Path-on-a-Processor ($CPOP$) for mapping randomly generated task

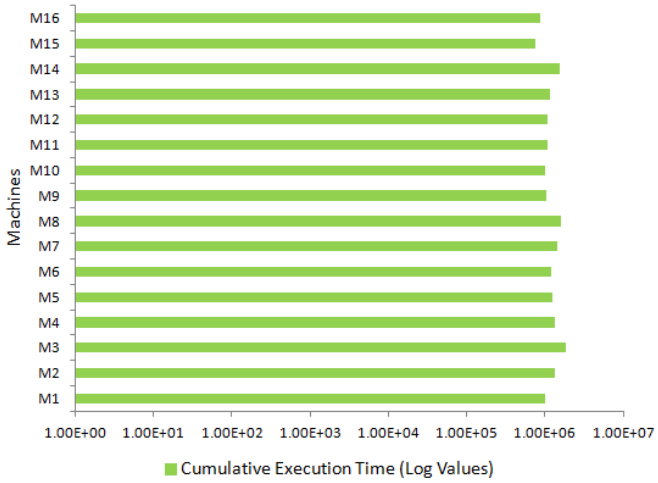

Figure 5.15: Load Balancing in 16-Machine Allocations

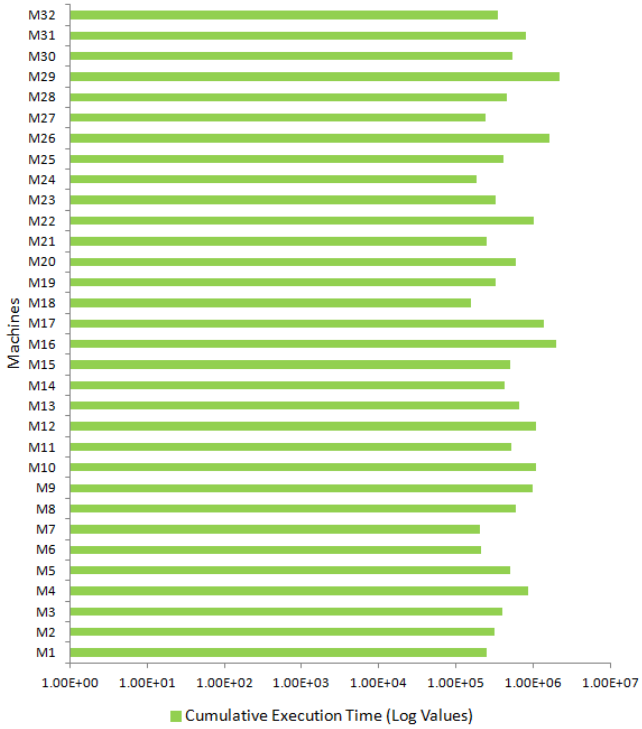

Figure 5.16: Load Balancing in 32-Machine Allocations

graphs on a bounded number of heterogeneous processors. Based on experimental studies, the authors reported that $HEFT$ and $CPOP$, outperformed task- scheduling heuristics like DLS (Dynamic Level Scheduling), MH (Mapping Heuristic) and LMT (Levelized-Min Time) algorithms when evaluated under the very metrics that are under considered within this thesis. Like $HEFT$ and $CPOP$ algorithms, $FLBTA-I$ also uses randomly generated scenarios so $HEFT$ and $CPOP$ were chosen as the algorithms for comparative study. The following subsection discusses each of them briefly:

Heterogeneous-Earliest-Finish-Time ($HEFT$)

It is an application-mapping algorithm for a limited number of heterogeneous processors. It has two major phases namely $TaskPrioritizing$ and $ProcessorSelection$. The $TaskPrioritizing$ phase assigns priorities to the tasks and the $ProcessorSelection$ phase selects the best processor for allocating a task at hand whilst minimizing the finishing time of tasks. The algorithm requires the tasks to be ordered by their scheduling priorities based on upward and downward ranking.

The worst case complexity (when the task graph is dense) of $HEFT$ algorithm for m tasks and n machines is $O(m^2n)$ and best case complexity is $O(e \times n)$, where e is number of edges in task graph.

Critical-Path-on-a-Processor ($CPOP$)

The $CPOP$ algorithm is analogous to $HEFT$ in the sense that it also uses the $TaskPrioritizing$ and $ProcessorSelection$ phases. However, it uses a different attribute for determining the task priorities and employs a different strategy for selecting the best processor for allocating the task at hand.

The timing complexity of $CPOP$ is $O(m^2n)$, for m machines and n tasks.

- Table 5.30 compares $FLBTA - I$ allocation algorithm results for the allocation of Gauss Elimination Algorithm graph for matrix dimension of 5 [210], with results of $HEFT$ and $CPOP$. From Table 5.30 it is clear that

$FLBTA-I$ provides better results for the performance metrics under consideration.

- As the system is scaled up, efficiency decreases rapidly in case of $HEFT$ and $CPOP$, while with $FLBTA-I$ it decreases gradually.
- Both algorithms i.e. $HEFT$ and $CPOP$ assume a fully connected interconnection network. No such restriction is imposed on $FLBTA-I$ and hence it is meant for all types of networks.
- The timing complexities of $HEFT$ and $CPOP$ are $O(m^2n)$, whereas timing complexity of $FLBTA-I$ in all cases is $O(m(n+1))$. Clearly $FLBTA-I$ is the better performing algorithm. Figure 5.17 and 5.18 compare the speedup and efficiency of all three algorithms graphically.

No. of Machines	$FLBTA-I$		$HEFT$		$CPOP$	
	Speedup	Efficiency(%)	Speedup	Efficiency (%)	Speedup	Efficiency (%)
4	3.2	84	3.1	79	3.08	77
8	6.6	82	5.9	74	5.7	72
16	11.0	69	9.7	61	9.4	59
32	20.1	63	17.9	56	16.6	52

Table 5.30: $FLBTA-I$ Vs. $HEFT$ and $CPOP$

5.5 Fuzzy Load Balancing Task Allocation Model - II

Most studies in task allocation assume that resource requirements of the task are not known a *priori*. However, better allocation and load balancing should be possible if the allocator uses information on task resource requirements to make allocation decisions. The results obtained by Devarakonda and Iyer [90] show that it is possible to predict the CPU, memory, and I/O requirements of a task using a statistical pattern-recognition technique. The criteria proposed by Qureshi and Majeed [180] for classifying a task as CPU Intensive or Memory Intensive or IO Intensive

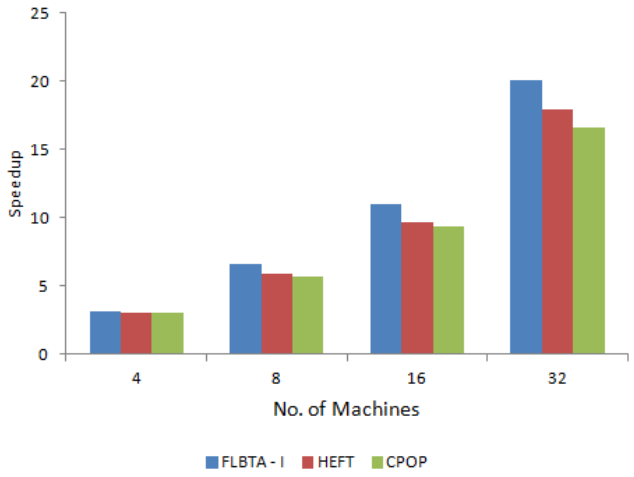

Figure 5.17: Speedup Comparison

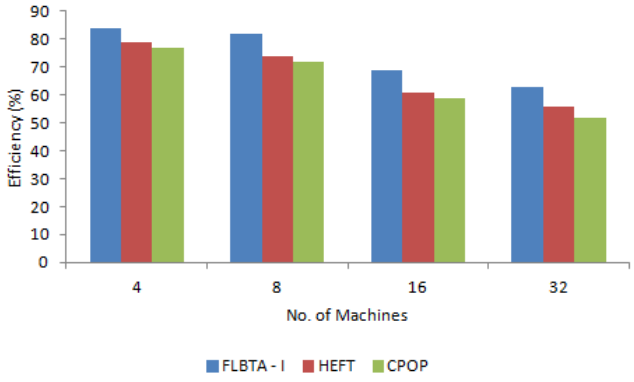

Figure 5.18: Efficiency Comparison

or a Mixed Nature uses a percentage of task work related to these components. This approach appears more appealing as with the advent of Open Computing Language (OpenCL), determining the percentage of task work related to a particular machine component has become possible [116]. OpenCL is a framework for writing programs that execute across heterogeneous platforms consisting of central processing units (CPUs), graphics processing units (GPUs), digital signal processors (DSPs), Field-programmable gate arrays (FPGA) and other components. It is an open standard for parallel programming supported by many hardware vendors including AMD, Intel, NVIDIA and IBM. This thesis extends the task classification criteria proposed by Qureshi and Majeed [180], assuming that it is possible to determine the requirements of a task for a machine component by determining the percentage of task work related to that particular component. Considering these results the Fuzzy Load Balancing Task Allocation Model - II (FLBTA - II) adds following improvement to FLBTA - I:

5.5.1 Selecting Best Machine to Execute the Task

The following subsection describes how to select the best machine to execute a task.

Task Requirements Fuzzification

Let a given task be τ and $\tau^{CPU}, \tau^{GPU}, \tau^{MEM}, \tau^{NCP}$ and τ^{IO} be the task requirements for CPU, GPU, Numeric-Coprocessor, Memory and I/O Bandwidth, where τ^x means percentage of task work related to machine component x. Only five vital machine components are used within this study, however, the idea may easily be extended to incorporate more components. Irrespective of the methods we use to determine (described in [90] [180]) crisp value of τ^x is obtained. As a first step, task allocation using fuzzy logic requires fuzzification of inputs. During this process the crisp input values are obtained and transformed into their corresponding linguistic variables. These fuzzy inputs called antecedents form their correspond-

ing membership graphs depending on the membership functions chosen for them to represent. This study uses the triangular membership functions. This crisp value of τ^x (Universe of disclosure) is first normalized in the range $[0.0, 1.0]$ and then transformed on the linguistic scale using any of the five linguistic variables Very Low (VL), Low (LO), Medium (ME), High (HI) and Very High (VH). Table 5.31 shows the fuzzy partition of input using triangular membership functions.

Input Range(%)	Normalized Segments	Linguistic Variable	Corresponding TFN
$0 \leq \tau^x < 2.0$	$[0.00, 0.05]$	Very Low (VL)	$(0.0, 0.00, 0.01)$
$2.0 \leq \tau^x < 10.0$	$[0.05, 0.25]$	Low (LO)	$(0.0, 0.03, 0.05)$
$10.0 \leq \tau^x < 20.0$	$[0.25, 0.50]$	Medium (ME)	$(0.10, 0.50, 0.7)$
$20.0 \leq \tau^x < 30.0$	$[0.50, 0.75]$	High (HI)	$(0.50, 0.70, 0.9)$
$30.0 \leq \tau^x < 40.0$	$[0.75, 1.00]$	Very High (VH)	$(0.70, 0.90, 1.0)$

Table 5.31: Linguistic Variables For Fuzzification of τ^x

For a given task τ, if the CPU related work τ^{CPU}, is rated high, then it is denoted as $\tilde{r}(\tau, \tau^{CPU})$, and is assigned the membership value $(0.50, 0.70, 0.9)$.

Machine Capabilities Fuzzification

All machines (computers) forming the HMS under consideration are diversely capable i.e. differ in the capacity and capability of components. This machine heterogeneity can be represented on a linguistic scale. Let $\tilde{m}(M, x)$ be the relative capability measure of machine M with regard to component x then it can be represented on the linguistic scale using five linguistic variables (Table 5.32). Relative capability implies that a machine M which has the maximum measure of a given component receives the highest rating with regard to that component. Other machines for that component are rated with respect to M. The triangular membership functions for required fuzzification are also given in Table 5.32. A HMS is more complex than any other form of parallel systems. The added complexity could increase the possibilities of system failures. Hence, ensuring reliability of each machine belonging to the system is of critical importance while allocating

Linguistic Variable	Triangular Membership Function
Least Capable (LEC)	$(0.0, 0.0, 0.2)$
Low Capable (LOC)	$(0.0, 0.2, 0.4)$
Moderately Capable (MOC)	$(0.2, 0.4, 0.6)$
Highly Capable (HIC)	$(0.4, 0.6, 0.8)$
Extremely Capable (EXC)	$(0.6, 0.8, 1.0)$

Table 5.32: Linguistic variables for Machine Capability Fuzzification

the tasks. It is assumed that reliabilities of all machine components are known and have been expressed as Triangular Fuzzy Numbers. Let $\widetilde{R}_k$ denotes the fuzzy reliability of a component k of machine M_j. Then, for a given task τ, which requires p machine components to execute, the Yaakob and Kawata [226] equation is modified to determine the relative capability of machine M_j to execute τ, as follows:

$$\widetilde{\bigtriangledown}_{MC}(M_j) = \frac{1}{p} \otimes \sum_{i=1}^{p} [\tilde{m}(M_j, x_i) \otimes \tilde{r}(\tau, \tau^{x_i})] + \prod_{k=1}^{p} \widetilde{R}_k$$

if $\widetilde{R}_k = (z_k - a_k, z_k, z_k + b_k)$ (i.e. a triangular fuzzy number) then we have:

$$\prod_{k=1}^{p} \widetilde{R}_k = \left(\prod_{k=1}^{p} (z_k - a_k), \prod_{k=1}^{p} (z_k), \prod_{k=1}^{p} (z_k + b_k) \right)$$

So, the machine capability equation can be rewritten as follows:

$$\widetilde{\bigtriangledown}_{MC}(M_j) = \frac{1}{p} \otimes \sum_{i=1}^{p} [\tilde{m}(M_j, x_i) \otimes \tilde{r}(\tau, \tau^{x_i})] + \left(\prod_{k=1}^{p} (z_k - a_k), \prod_{k=1}^{p} (z_k), \prod_{k=1}^{p} (z_k + b_k) \right) \quad (5)$$

5.5.2 Link Reliability

As each machine's reliability has been included in the machine capability equation 5, the system reliability equation is modified as follows. If a given task τ_i has communicating tasks i.e. $\tau_1, \tau_2, \tau_3, \ldots, \tau_h$, which are allocated to machines $M_1, M_2, M_3, \ldots, M_h$ respectively then the fuzzy system reliability for computing τ_i on machine M_j can be computed using following equation:

$$\widetilde{\bigtriangledown}_{REL}(M_j) = \prod_{l \neq i}^{h} \widetilde{R}_{jl} \quad (6)$$

5.5.3 Modified Allocation Algorithm

The modified algorithm according to above improvements to $FLBTA-I$ is listed below:

5.5.4 Experimental Setup and Implementation of FLBTA - II

The proposed fuzzy approach is implemented on a number of task and machine graphs. The model is simulated on an Ubuntu 12.04 LTS operating system by coding the algorithms presented within this thesis in C using CodeBlocks 12.11 compiler on an Intel Core i7 machine. The file FLBTAM.sh runs the simulation. Following subsections present the results of various experiments:

4-machine 10-task Allocations:

We once again consider the problem of allocating 10 tasks to 4 machines, shown in Figure 5.2 and 5.3 respectively. The allocation proceeds as follows:

1. Obtain $FETC$ matrix using Algorithm 2.
2. High task and high machine heterogeneity is assumed so according to Ali et al. [17], $R_{task} = 10^5$ and $R_{mach} = 10^2$. Table 5.4 shows the computer generated $FETC$ matrix.
3. Obtain Task Priorities using Algorithm 3. Table 5.8 shows the normalized average execution times of tasks and their priorities.
4. Obtain Fuzzified Task Requirements (FTR) and Fuzzified Machine Capabilities (FMC) matrices using criteria shown in table 5.31 and 5.32. Tables 5.33 and 5.34 list these matrices.
5. Obtain $FITC$ matrix using criteria shown in Table 5.3. Table 5.5 shows sample 10×4 excerpt from computer generated 10×10 ITC matrix.

Algorithm 5 $FLBTA-II$ Allocation Algorithm

begin
Data: Directed Acyclic Graph (T), Interconnection Network Topology Graph (M), ITC Matrix, Fuzzy Machine Component Reliabilities, Fuzzy Link Reliabilities,Task Requirements, Machine Capabilities
Result: $FA : T \rightarrow M$ (A Task Mapping)
Generate $FETC$ using Algorithm 2.
Generate Task Priorities List $priority(\tau)$ using Algorithm 3
Apply criteria shown in Table 5.3 to fuzzify ITC and obtain $FITC$
Apply criteria shown in table 5.31 and 5.32 to fuzzify task requirements and machine capabilities.
for $i = 0$ **to** $|T| - 1$ **do**
  Select Task τ_p, which has the highest priority in $priority(\tau)$
  for $j = 0$ **to** $|M| - 1$ **do**
    Compute $\widetilde{\nabla}_{MC}(M_j), \widetilde{\nabla}_{EC}(M_j), \widetilde{\nabla}_{CC}(M_j), \widetilde{\nabla}_{LD}(M_j), \widetilde{\nabla}_{REL}(M_j)$ (equations (5), (1) to (3),(6))
    $\widetilde{\nabla}(M_j) \leftarrow \widetilde{\nabla}_{MC}(M_j) + \widetilde{\nabla}_{EC}(M_j) + \widetilde{\nabla}_{CC}(M_j) + \widetilde{\nabla}_{LD}(M_j) + \widetilde{\nabla}_{REL}(M_j)$
    $score[j] \leftarrow rank(\widetilde{\nabla}(M_j))$
  end for
  if $score[l] > score[z] \quad \forall \quad z = 1, 2, \ldots, n \quad and \quad z \neq l$ **then**
    $k \leftarrow l$
  end if
  Award task τ_p to machine M_k
  $\{T\} \leftarrow \{T\} - \tau_p$
  $\{priority(\tau)\} \leftarrow \{priority(\tau)\} - priority(\tau_P)$
end for
end

Task	Requirements				
	CPU	GPU	NCP	MEM	IO
T_1	HI	VH	VH	HI	ME
T_2	VL	VH	HI	VH	HI
T_3	LO	VH	VH	VH	VH
T_4	VH	VH	VH	LO	VL
T_5	LO	HI	VL	VH	HI
T_6	VH	VL	HI	VH	VH
T_7	ME	HI	VH	VL	LO
T_8	VH	VH	VH	VH	HI
T_9	HI	VH	HI	VH	VH
T_{10}	VL	VH	VH	LO	VH

Table 5.33: Fuzzified Task Requirements

Machine	Capability Rating				
	CPU	GPU	NCP	MEM	IO
M_1	MOC	LEC	LOC	HIC	LOC
M_2	HIC	LOC	MOC	LOC	LEC
M_3	MOC	EXC	MOC	LOC	EXC
M_4	EXC	MOC	LOC	LEC	HIC

Table 5.34: Fuzzified Machine Capabilities

Machine	Capability Rating				
	CPU	GPU	NCP	MEM	IO
M_1	(0.9769,0.9731,0.9656)	(0.9762,0.9594,0.9543)	(0.9707,0.9584,0.9503)	(0.9732,0.9592,0.9565)	(0.9655,0.9578,0.9555)
M_2	(0.9526,0.9509,0.9507)	(0.9641,0.9598,0.9559)	(0.9524,0.9514,0.9512)	(0.9682,0.9681,0.9516)	(0.9509,0.9508,0.9507)
M_3	(0.9605,0.9593,0.9545)	(0.9666,0.9577,0.9552)	(0.9568,0.9546,0.9519)	(0.9806,0.9705,0.9703)	(0.9855,0.9791,0.9508)
M_4	(0.9892,0.9616,0.9514)	(0.9658,0.9509,0.9502)	(0.9695,0.9584,0.9506)	(0.9851,0.9827,0.9506	(0.9879,0.9593,0.9564)

Table 5.35: Fuzzified Machine Components Reliabilities

6. Table 5.6 and 5.35 list the machine connectivity matrix and randomly generated machine component reliabilities matrix respectively. The interconnection network topology is assumed to be a hypercube.

7. For the sake of simplicity it is assumed that all links in the machine graph are equally reliable. The reliability of each link is the fuzzy number $(\tilde{0.9796})$ which may be considered as $(0.9796, 0.9796, 0.9796)$ (i.e. a triangular fuzzy number).

8. Table 5.39 shows speedup and efficiency for the allocation under consideration. It is evident from Table 5.39 that $FLBTA - II$ provides better results in comparison of $FLBTA - I$.

9. Computer generated allocation results can be accessed using the url: `https://github.com/jahangir2016/results-new`. The file flbta2-4-10.txt contains the results.

8-machine 17-task Allocations:

Consider the task graph and machine graph shown in Figure 5.4 and 5.5 respectively. The allocation proceeds as follows:

1. Obtain $FETC$ matrix using Algorithm 2.

2. High task and high machine heterogeneity is assumed so according to Ali et al. [17], $R_{task} = 10^5$ and $R_{mach} = 10^2$.Table 5.11 shows sample 17×4 excerpt from computer generated 17×8 $FETC$ matrix.

3. Obtain $FITC$ matrix using criteria shown in Table 5.3.

4. Obtain Task Priorities using Algorithm 3. Table 5.14 shows the normalized average execution times of tasks and their priorities.

5. Obtain Fuzzified Task Requirements (FTR) and Fuzzified Machine Capabilities (FMC) matrices using criteria shown in table 5.31 and 5.32. Table 5.36 and 5.37 list these matrices.

Task	Requirements				
	CPU	GPU	NCP	MEM	IO
T_1	VH	VH	HI	ME	VH
T_2	VH	VH	VH	ME	VL
T_3	ME	VH	ME	VH	VH
T_4	VH	ME	VH	VH	VH
T_5	VH	VH	VH	VH	VH
T_6	VH	VH	VH	VH	VH
T_7	VH	ME	VH	ME	VH
T_8	VH	VH	VH	HI	VH
T_9	LO	VH	VH	VH	VH
T_{10}	VH	VH	VH	VH	VH
T_{11}	VH	LO	HI	VH	VH
T_{12}	ME	ME	VH	VH	VH
T_{13}	ME	VH	HI	ME	VH
T_{14}	VH	VL	VH	ME	VH
T_{15}	VH	ME	VH	VH	VH
T_{16}	VH	VH	VH	VH	VH
T_{17}	VH	VH	VH	VH	VH

Table 5.36: Fuzzified Task Requirements

Machine	Capability Rating				
	CPU	GPU	NCP	MEM	IO
M_1	MOC	EXC	HIC	MOC	EXC
M_2	EXC	LEC	HIC	MOC	EXC
M_3	HIC	EXC	HIC	HIC	EXC
M_4	LEC	EXC	HIC	EXC	HIC
M_5	HIC	MOC	LEC	MOC	HIC
M_6	EXC	MOC	MOC	LEC	LOC
M_7	MOC	HIC	LOC	MOC	MOC
M_8	HIC	HIC	MOC	EXC	LEC

Table 5.37: Fuzzified Machine Capabilities

Machine	Capability Rating				
	CPU	GPU	NCP	MEM	IO
M_1	(0.9743,0.9534,0.9526)	(0.9765,0.9740,0.9641)	(0.9713,0.9641,0.9566)	(0.9589,0.9571,0.9540)	(0.9696,0.9660,0.9600)
M_2	(0.9896,0.9635,0.9615)	(0.9523,0.9502,0.9500)	(0.9709,0.9685,0.9645)	(0.9638,0.9609,0.9596)	(0.9895,0.9882,0.9534)
M_3	(0.9525,0.9513,0.9503)	(0.9797,0.9668,0.9614)	(0.9833,0.9802,0.9522)	(0.9867,0.9737,0.9595)	(0.9651,0.9602,0.9544)
M_4	(0.9610,0.9595,0.9573)	(0.9576,0.9555,0.9514)	(0.9787,0.9603,0.9566)	(0.9714,0.9549,0.9504)	(0.9654,0.9613,0.9586)
M_5	(0.9545,0.9538,0.9532)	(0.9549,0.9530,0.9502)	(0.9594,0.9579,0.9522)	(0.9894,0.9632,0.9578)	(0.9699,0.9635,0.9523)
M_6	(0.9671,0.9645,0.9604)	(0.9758,0.9598,0.9530)	(0.9875,0.9610,0.9547)	(0.9630,0.9588,0.9522)	(0.9570,0.9505,0.9505)
M_7	(0.9773,0.9544,0.9533)	(0.9721,0.9711,0.9651)	(0.9595,0.9551,0.9514)	(0.9741,0.9595,0.9585)	(0.9590,0.9510,0.9503)
M_8	(0.9768,0.9636,0.9562)	(0.9779,0.9637,0.9553)	(0.9648,0.9636,0.9595)	(0.9821,0.9569,0.9511)	(0.9863,0.9707,0.9692)

Table 5.38: Fuzzified Machine Components Reliabilities

6. Obtain $FITC$ matrix using criteria shown in Table 5.3. Table 5.5 shows the sample 10×4 excerpt from computer generated 10×10 ITC matrix.

7. Table 5.6 and 5.38 list the machine connectivity matrix and randomly generated machine component reliabilities matrix respectively. The interconnection network topology is assumed to be a hypercube.

8. For the sake of simplicity it is assumed that all links in the machine graph are equally reliable. The reliability of each link is the fuzzy number $(\tilde{0.9796})$ which may be considered as $(0.9796, 0.9796, 0.9796)$ (i.e. a triangular fuzzy number).

9. Table 5.39 shows speedup and efficiency for the allocation under consideration. It is evident from Table 5.39 that $FLBTA - II$ provides better results in comparison of $FLBTA - I$.

10. Computer generated allocation results can be accessed using the url: `https://github.com/jahangir2016/results-new`. The file flbta2-8-17.txt contains the results.

All experiments which are carried out to validate $FLBTA - I$, are also repeated with $FLBTA - II$. Results of all experiments are available on Internet and can be accessed using the url: `https://github.com/jahangir2016/results-new`.

Table 5.39 lists speedup and efficiency values obtained in various cases. From 5.39 it is obvious that $FLBTA - II$ produces better results than $FLBTA - I$. Figures 5.19 - 5.20 compare the speedup and efficiency values obtained in two approaches graphically.

Chapter Summary

In this chapter two load balancing task allocation models namely $FLBTA - I$ and $FLBTA - II$ based on fuzzy logic have been developed and tested. Three novel algorithms have also been presented. The performance of the algorithms has

No. of Machines	Average Speedup (Over 10 Runs)	Efficiency(% approx.)
4	3.4	85
8	6.7	84
12	9.6	80
16	11.4	73
32	20.19	63

Table 5.39: $FLBTA-II$ Performance

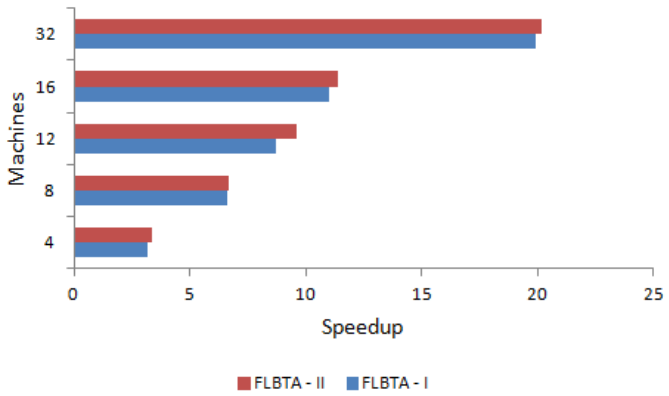

Figure 5.19: $FLBTA-I$ and $FLBTA-II$ Speedup Comparison

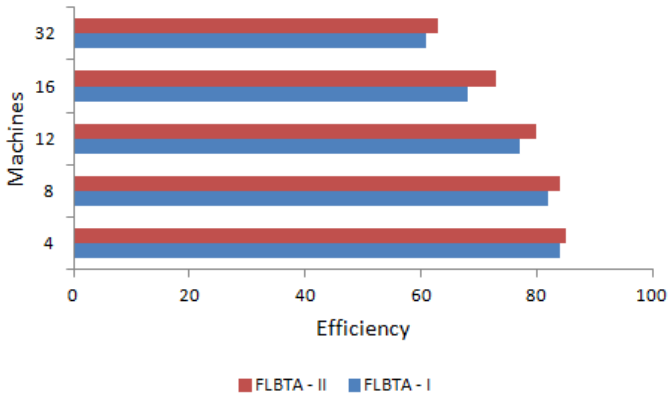

Figure 5.20: $FLBTA-I$ and $FLBTA-II$ Efficiency Comparison

been recorded through experiments on large sets of randomly generated data. A comparative study with existing approaches (Heterogeneous-Earliest-Finish-Time ($HEFT$) and Critical-Path-on-a-Processor ($CPOP$)) has also been conducted. The performance of algorithms based on $FLBTA-I$ and $FLBTA-II$ has been found to be better for the representative examples used within this thesis. Simulation results indicate that both $FLBTA-I$ and $FLBTA-II$, fairly and successfully divide the computational load across heterogeneous machines. Their complexities compared to existing approaches of interest are low. Simulation results further indicate that fuzzy logic is an appealing approach to solve the task allocation problem.

Chapter 6

Conclusions and Future Directions

This thesis has addressed two challenging issues related to parallel and distributed systems: The design of an economical interconnection network topology and task allocation strategy while load balancing. In the first part of this work, a new interconnection network topology termed as STH has been designed and evaluated. Its suitability for use as a multiprocessor interconnection network has been explored. Various properties of the proposed topology have been derived and compared with some other topologies on a number of interconnection network evaluation parameters. Procedures for routing and broadcasting on the proposed topology have also been discussed and a simple routing algorithm has been presented. Following are the distinctive features of the proposed topology:

1. It is highly scalable i.e. it is capable of supporting any given network size.

2. Comparative study presented in Chapter 3 indicates that with a reduced diameter, better average distance, low traffic density, low cost, maximum number of links, high bisection width and tremendous scalability, STH is highly suitable for massively parallel systems.

3. It provides a great architectural support for parallel computing.

The second part of this thesis is related to the development of load balancing task allocation models for parallel and distributed systems. Two mathematical models,

based on fuzzy logic, have been developed in this work. The salient features of the proposed models are as follows:

1. The distributed system has been assumed to be a heterogeneous multicompter system i.e. each machine within the system is diversely capable. In other words, a task may have different execution times on two different machines.

2. Communication links between machines differ in bandwidth. Two machines may use any of the links available between them to communicate. The quality of the link is taken into account by the proposed allocation models as these models tend to maximize the reliability of the system which includes link reliability as well.

3. Instead of maximizing or minimizing a particular parameter, these models tend to optimize a set of parameters consisting of execution cost, communication cost, load balancing and system reliability, thereby leading to better solutions.

4. The proposed algorithms provide high quality solutions and these algorithms perform better than some other known algorithms as is evident from the comparative study presented in Chapter 5.

5. The algorithms are capable of tackling the task allocation problem under a variety of input parameters and are therefore robust in this sense. They are general in nature and are meant to map any given DAG of arbitrary structure on any given machine graph.

6.1 Summary of Contributions

In addition to the literature reviews presented in this work, a scalable topology has been designed along with its routing and broadcasting algorithms. Further, two mathematical models for task allocation in a parallel and distributed system have

also been presented. Based on our mathematical models we have further developed three more algorithms: Two for task allocation and one is a fuzzy algorithm used for calculating expected time to compute matrix. Our main contributions can be categorized as follows:

6.1.1 Literature Reviews

In the first part of this thesis, an overview of the proposed interconnection network topologies has been presented. An addition to the general classification of interconnection network topologies has been made based on the lessons learnt from the overview. Some well known interconnection networks have been reviewed and five of them namely the Hypercube (HC), Double Loop Hypercube (DLH), Hex-Cell (HX), 2D-Mesh (ME) and Hyper Mesh (HM) have been selected for comparative study considering their capability of representing the given network size (scalability).

In the second part of this thesis an overview of various techniques used to solve the task allocation problem has been presented. Taxonomy of task allocation models based on the solution techniques used has also been presented. Accordingly, the allocation models have been classified as mathematical programming based, graph theoretic based, state space search based and heuristics based.

The literature reviews presented in the thesis may be used as a starting point for prospective researchers in the field of interconnection networks design and task allocation.

6.1.2 Development of Mathematical Models

In this thesis, two load balancing task allocation models namely $FLBTA-I$ and $FLBTA-II$ based on fuzzy logic have been developed and verified. In these models, to exploit the available computational power of a HMS, the workload distribution is considered for a heterogeneous multicomputer system. A set of parameters which need to be optimized during the allocation process is identi-

fied. The approach used in the models attempt to keep all the machines busy all the time for tasks execution. Three novel algorithms have also been presented. The performance of the algorithms has been recorded through experiments on large sets of randomly generated data. A comparative study with the existing approaches (Heterogeneous-Earliest-Finish-Time ($HEFT$) and Critical-Path-on-a-Processor ($CPOP$)) has also been conducted. Simulation results indicate that the proposed models fairly and successfully divide the computational load across heterogeneous machines. Their complexities compared to existing approaches of interest are significantly low and they are effective in finding optimal assignments in all cases.

Simulation results further indicate that fuzzy logic is an appealing approach to solve the task allocation problem and may allow researchers further opportunities to develop innovative solutions to this complex problem.

6.1.3 Algorithms

As pointed out earlier, four novel algorithms have been presented in this thesis:

1. An algorithm for routing and broadcasting on $STH(m, n)$ interconnection network.

2. Algorithm for estimating task execution times under given machine and task heterogeneities.

3. Algorithm based on Fuzzy Load Balancing Task Allocation Model - I ($FLBTA-I$).

4. Algorithm based on Fuzzy Load Balancing Task Allocation Model - II ($FLBTA-II$).

6.2 Concluding Remarks

- As a conclusion from the literature review pertaining to the first part of the thesis, it is very difficult to design a scalable and economical interconnection network topology. Even if a topology is somehow derived, devising a routing and broadcasting procedure for it still remains a challenge. A topology is considered as "acceptable", if besides being economical and scalable, it is supported by efficient routing and broadcasting procedures.

- From the literature review pertaining to the second part of the thesis, it can be inferred that it is very difficult to compare and evaluate various task allocation approaches. In most of the approaches, several assumptions are made; both for application graph and for the machine graph and therefore these approaches greatly depend on them.

- As is the general practice, numerous papers reported in literature compare their results against previously proposed solutions. Each of them attempts to prove that their results are better than others. Therefore, it is almost impossible to select a single best solution from a wide range of tasks to machine allocation.

- Mathematical programming techniques are more flexible compared to other allocation techniques. They always lead to an optimal solution. System constraints like inter-task communication cost, processor speed limit, memory limit can be easily formulated using mathematical programming approaches. However, they are time and space hungry and can't be used as general solution to task allocation problems.

- Graph theoretic techniques are simple but are not meant for generalized allocation of m tasks on n machines due to their high complexity. Moreover, formulating system constraints using these techniques is a challenge.

- State space search techniques are flexible compared to graph theoretic techniques and system constraints can easily be formulated using them. Like

mathematical programming techniques, they also guarantee optimal solution but once again are constrained by their high complexities.

- Heuristics provide most efficient and effective methods of obtaining near optimal solutions for task allocation problems. But, it is a tedious job to choose best values for the parameters required by them and it is very much possible that the solution obtained from them is far away from the optimal solution. When used for obtaining a solution near to an optimal solution, their timing complexity can grow exponentially. Despite their many shortcomings, they are still favoured by researchers.

- Finally, fuzzy logic is an appealing approach to solve the task allocation problem and may be observed as a wonderful tool for researchers to develop further innovative solutions to this complex problem.

6.3 Future Research

Some of the important issues pertaining to distributed systems that are to be investigated are listed below:

6.3.1 Task Partitioning

An important issue related to distributed systems is partitioning an application into a set of tasks. A number of techniques for efficiently partitioning an application to enhance the system performance have been reported in literature. Some partitioning techniques for solution of partial differential equations have been proposed by Berger and Bokhari [40] [39], but they are not suitable for other problems.

6.3.2 Estimation on Communication Cost

The work presented in this thesis assumes that the amount of communication among tasks is known *a priori*. However, just as task execution times were es-

timated, it is worth exploring techniques for estimation of communication costs. This problem involves choosing an appropriate metric for this cost and then deriving values for this cost.

6.3.3 Designing Suitable Multicomputer Architecture

Designing a suitable multicomputer system architecture on which task allocation can take place easily is an important issue which needs to be investigated further.

6.3.4 Improving the Allocation Models for Network Contention

The task allocation models presented in this thesis assume that there is no contention on the network and communication among machines take place smoothly. Realistically however, contention takes place on the network. Therefore, there is a scope to improve the task allocation models presented in this thesis to handle the network contention.

6.3.5 Designing a Dynamic Task Allocator

The task allocators addressed in this work are meant for static task allocation wherein certain information related to the tasks and machines is available before the execution begins. The future goal could be to improve these allocators to handle dynamic task allocations which is a more active research domain.

List of Publications

Published Papers

1. Alam J., Kumar R., ”STH: A Highly Scalable and Economical Topology for Massively Parallel Systems”, Indian Journal of Science and Technology, Vol. 4, No. 12, December 2011-12 (ISSN 0974- 6846).

2. Alam J., Kumar R., Khan Z., ”Linearly Extendible Arm (LEA) A Constant Degree Topology for Designing Cost Effective Product Networks”, Ubiquitous Computing and Communication Journal, Vol. 2 No. 5, June 2010 (ISSN 1992-8424).

3. Alam J., Kumar R., ”Eight Row Matrix (ERM): A New Topology for Designing Cost Effective Product Networks”, International Journal of Engineering Science and Technology, Vol. 2 No. 4, pp. 330 - 340, 2010 (ISSN: 0975-5462).

4. Alam J., Khan Z., Kumar R., ”Performance analysis of Dynamic load Balancing Techniques for Parallel and Distributed Systems”, International Journal of Computer and Network Security, Vol. 2, No.2, pp. 123 - 127 , 2010 (ISSN 20762739).

Communicated Papers

1. Alam J., Kumar R., ”A Fuzzy Load Balancing Task Allocator for Parallel and Distributed System”, Sadhna Journal (A Journal of Indian Academy of Science) (March 2015)

2. Alam J., Kumar R., A Task Allocation Model for Effective Load Balancing on Heterogeneous Multicomputer System (HMS), International Journal of Parallel Programming, (April 2015)

References

[1] http://www.phy.ornl.gov/csep/ca/node22.html.

[2] Abd-El-Barr, M. and Al-Somani, T. F. (2011). Topological properties of hierarchical interconnection networks: A review and comparison. *Journal of Electrical and Computer Engineering*, 2011(1).

[3] Abdelzaher, T. and Shin, K. (2000). Period-based load partitioning and assignment for large real-time applications. *IEEE Computer*, 49(1):81–87.

[4] Abraham, S. and Padmanabhan, K. (1989). Performance of the direct binary n-cube network for multiprocessors. *IEEE Computers*, 38(7):1000–1011.

[5] Abuelrub (2002). On hyper-mesh multicomputers. *J. Institute of Maths and Comput. Sci.*, 12(2):81–83.

[6] Abuelrub, E. (2007). Data communication and parallel computing on twisted hypercubes. In *World Congress on Engg.*, UK.

[7] Abuelrub, E. (2008). A comparative study on topological properties of the hyper-mesh interconnection network. In *World Congress on Engg.*, pages 616–621, UK.

[8] Adam, T., Chandy, K., and Dickson, J. (1974). A comparison of list scheduling for parallel processing systems. *Communications of the ACM*, 17:685–690.

[9] Agrawal, N. and Ravikumar, C. (1996). Fault-tolerant routing in multiply twisted cube topology. *J. Sys. Architec.*, 42(4):279–288.

[10] Aguilar, J. (2003). Heuristic algorithm based on a genetic algorithm for mapping parallel programs on hypercube multiprocessors. *International Journal of Computer Systems Science and Engineering*, 4:217–222.

[11] Ahmad, I. and Kwok, Y. (1999). On paralleling the multiprocessor scheduling problem. *IEEE Computers*, 10:414–432.

[12] Ahmad, I., Kwok, Y., and Dhodi, M. (2001). *Scheduling parallel programs using genetic algorithms*. John Wiley and Sons, New York, USA.

[13] Ahmad, I., Kwok, Y., and Wu, M. (1996). Analysis, evaluation, and comparison of algorithms for scheduling task graphs on parallel processors. In *International Symposium on Parallel Architectures, Algorithms, and Networks*, pages 207–213.

[14] Al-Sadi, J., Day, K., and Ould-Khaoua, M. (2002). Unsafety vectors: A new fault-tolerant routing for binary n-cubes. *J. Sys. Architec.*, 47(9):783–793.

[15] Al-Tawil, K., Abd-El-Barr, M., and Ashraf, F. (1997). A survey and comparison of warmhole routing techniques in mesh networks. *IEEE Network*, 11(2).

[16] Alam, J. and Kumar, R. (2010). Linearly extendible arm (lea) a constant degree topology for designing scalable and cost effective interconnection networks. *J. Ubiquitous Computing and Communication*, 5(2):40–48.

[17] Ali, S., Siegel, H., Maheswaran, M., and Hensgen, D. (2000). Representing task and machine heterogeneities for heterogeneous computing system. *Tamkang Journal of Science and Engineering*, 3(3):195–207.

[18] Altenbernd, P. and Hansson, H. (1998). The slack method: A new method for static allocation of hard real-time tasks. *Real Time Systems*, 15(2):103–130.

[19] Amawy, A. and Latifi, S. (1991). Properties and performance of folded hypercubes. *IEEE Parallel and Distributed Systems*, 2(1):31–42.

[20] Appadoo, S., Bector, C., and Bhatt, S. (2013). Possibilistic characterization of (m,n)-trapezoidal fuzzy numbers with applications. *Journal of Interdisciplinary Mathematics*, 14(4):347–372.

[21] Arabnejad, H. and Barbosa, J. (2014). List scheduling algorithm for heterogeneous systems by an optimistic cost table. *IEEE Parallel and Distributed Systems*, 25(3):682–694.

[22] Arora, R. and Rana, S. (1979). On module assignment in two processor distributed systems. *Information Processing Letters*, 9(3):113–117.

[23] Arora, R. and Rana, S. (1980). Heuristic algorithms for process assignment on distributed computing systems. *Information Processing Letters*, 11(4,5):199–203.

[24] Arora, R., Rana, S., and Sharma, N. (1981). On the design of process assigner for distributed computing systems. *The Australian Computer Journal*, 13(3):77–82.

[25] Arunkumar, S. and Chockalingam, T. (1992). Randomized heuristics for the mapping problem. *Int. J. High Speed Computing*, 4(4):289–300.

[26] Asanovic, K., Bodik, R., Demmel, J., Keaveny, T., Keutzer, K., Kubiatowicz, J., Morgan, N., Patterson, D., Sen, K., Wawrzynek, J., Wessel, D., and Yelick, K. (2009). A view of the parallel computing landscape. *Commun. ACM*, 52(10):56–67.

[27] Attiya, G. and Hamam, Y. (2004a). Reliability oriented task allocation in heterogeneous distributed systems. In *Ninth International Symposium on Computers and Communications (ISCC)*, pages 68–73. IEEE.

[28] Attiya, G. and Hamam, Y. (2004b). Task allocation for minimizing programs completion time in multicomputer systems. In *ICCSA - Lecture Notes in Computer Science*, volume 3044, pages 97–106. Springer-Verlag.

[29] Attiya, G. and Hamam, Y. (2004c). Two phase algorithm for load balancing in heterogeneous distributed systems. In *12th Euromicro Conference on Parallel, Distributed and Network-Based Processing*, pages 434–439. IEEE.

[30] Attiya, G. and Hamam, Y. (2006a). Allocation for maximizing reliability of distributed systems: A simulated annealing approach. *J. Parallel and Distributed Processing*, 66(10):1259–1266.

[31] Attiya, G. and Hamam, Y. (2006b). Task allocation for maximizing reliability of distributed systems:asimulated annealing approach. *Journal of Parallel and Distributed Computing*, 66(10):1259–1266.

[32] Awasthi, L., Mishra, M., and Joshi, R. (2012). An efficient coordinated checkpointing approach for distributed computing systems with reliable channels. *International Journal of Computers and Applications*, 34.

[33] Awwad, A., , Ayyoub, A., and Ould-Khaoua, M. (2003). On topological properties of arrangement star network. *J. Sys. Architec.*, 48(11-12):325–336.

[34] Aykanat, C. and Haritaoglu, I. (1995). An efficient mean field annealing formulation for mapping unstructured domains to hypercubes. *Lecture Notes in Computer Science*, 980:115–120.

[35] Ayyoub, A. and Day, K. (1998). The hyperstar interconnection network. *J. Parallel and Distributed Processing*, 48(2):175–199.

[36] Batcher, K. E. (1968). Sorting networks and their applications. In *Proc. Spring Joint Computer Conf.*, pages 307–314.

[37] Bellman, R. and Zadeh, L. (1970). Decision-making in a fuzzy environment. *Management Science*, 17(4):141–164.

[38] Benes, V. E. (1962). On rearrangeable three-stage connecting networks. *Bell Systems Tech. J.*, 41(5):1481–1492.

[39] Berger, M. and Bokhari, S. (1985). A partitioning strategy for pdes across multiprocessors. In *International Conference on Parallel Processing*, pages 166–170.

[40] Berger, M. and Bokhari, S. (1987). A partitioning strategy for non-uniform problems on multiprocessors. *IEEE Computers*, C-36(5):570–580.

[41] Berman, O. and Ashrafi, N. (1993). Optimization models for reliability of modular software systems. *IEEE Software Engineering*, 19(11):1119 – 1123.

[42] Bhuyan, L. N. (1987). Interconnection networks for parallel and distributed processing. *IEEE Computers (Special Issue)*, 20(6):9–75.

[43] Bhuyan, L. N., Yang, Q., and Agrawal, D. P. (1989). Performance of multiprocessor interconnection networks. *IEEE Computers*, 22(2):25–37.

[44] Billionnet, A. (1994). Allocating tree-structured programs in a distributed system with uniform communication costs. *IEEE Parallel and Distributed Systems*, 5(4):445–448.

[45] Billionnet, A., Costa, M., and Sutter, A. (1992). An efficient algorithm for a task allocation problem. *Journal of ACM*, 39(3):205–218.

[46] Bokhari, S. (1981a). On the mapping problem. *IEEE Computers*, 30(3):207–214.

[47] Bokhari, S. (1981b). A shortest tree algorithm for optimal assignment across space and time in distributed processor system. *IEEE Software Engineering*, SE-7(6):583–589.

[48] Bokhari, S. (1987). *Assignment Problems in Parallel and Distributed Computing*. Kluwer Academic Publisher.

[49] Bokhari, S. (1988). Partitioning problems in parallel, pipeline and distributed computing. *IEEE Computers*, 37(1):45–57.

[50] Boku, T., Umemura, M., Makino, J., Fukushige, T., Susa, H., and Ukawa, A. (2002). Heterogeneous multi-computer system: a new platform for multi-paradigm scientific simulation. In *ICS '02: Proceedings of the 16th international conference on Supercomputing*, pages 26–34, New York, NY, USA 2002. ACM Press.

[51] Bollinger, S. and Midkiff, S. (1988). Processor and link assignment in multicomputers using simulated annealing. In *ICPP*, pages 1–7.

[52] Bollinger, S. and Midkiff, S. (1991). Heuristic technique for processor and link assignment in multicomputers. *IEEE Computer*, 40(3):325–333.

[53] Bouvry, P., Chassin, J., Dobruck, M., Hluch, L., and Margalef, T. (1994). Mapping and load balancing on distributed memory systems. In *Symposium on microcomputer and microprocessor applications.*, volume 1, pages 315–324.

[54] Bouvry, P., Chassin, J., and Trystram, D. (1995). Efficient solution for mapping parallel programs. In *Lecture Notes in Computer Science*, volume 966, pages 379–390. Springer-Verlag.

[55] Bowen, N., Nikolaou, C., and Ghafoor, A. (1992). On the assignment problem of arbitrary process systems to heterogeneous distributed computer systems. *IEEE Computers*, 41(3):257–273.

[56] Braun, T. D., Siegel, H. J., Beck, N., Maheswaran, M., Reuther, A. I., Robertson, J. P., Theys, M. D., Yao, B., Hensgen, D., and Freund, R. F. (2001). A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *J. Parallel and Distributed Computing*, 61(6):810–837.

[57] Buckley, J. (2006). *Fuzzy Probability and Statistics.* Springer-Verlag Berlin Heidelberg.

[58] Buckley, J. and Jowers, L. (2008). *Monte Carlo Methods in Fuzzy Optimization*. Springer-Verlag Berlin Heidelberg.

[59] Bultan, T. and Aykanat, C. (1992). A new mapping heuristic based on mean field annealing. *J. Parallel and Distributed Processing*, 16(4):292–305.

[60] Buyya, R. (2000). The design of paras microkernel. `http://www.buyya.com/microkernel`.

[61] Cai, W. and Lee, B. (1998). File allocation with balanced response time in a distributed multi-server information system. *Information and Software Technology*, 40(1):27–35.

[62] Casavant, T. and Kuhl, J. (1988). A taxonomy of scheduling in general-purpose distributed computing systems. *IEEE Software Engineering*, 14(2):141–154.

[63] Chang, P., Chang, D., and Kavi, K. M. (2001). File allocation algorithms to minimize data transmission time in distributed computing system. *J. of Information Science and Engineering*, 17:633–646.

[64] Chang, P., Chen, D., and Kavi, K. (2000). Multimedia file allocation on vc networks using multipath routing. *IEEE Computers*, 49(9):977–979.

[65] Chartrand, G. and Lesniak, L. (1986). *Graph and diagraph*. Wadsworth and Books, California.

[66] Chaudhary, V. and Aggarwal, J. (1993). A generalized scheme for mapping parallel algorithms. *IEEE Parallel and Distributed Systems*, 4(3):328–346.

[67] Chen, C., Agrawal, D. P., and Burke, J. R. (1993). dbcube: a new class of hierarchical multiprocessor interconnection networkswith area efficient layout. *IEEE Parallel and Distributed Systems*, 4(12):1332–1344.

[68] Chen, P. Y., Yew, P. C., and Lawrie, D. (1982). Performance of packet switching in buffered single-stage shuffle-exchange networks. In *3rd Int. Conf. on Distributed.*, pages 622–627.

[69] Cheng, T. and Sin, C. (1990). A state-of-the-art review of parallel-machine scheduling research. *Euro. J. of Operations Research*, 47:271–292.

[70] Chern, M. and Chen, G. (1989). An lc branch and bound algorithm for the module assignment problem. *Information Processing Letters*, 32(2):61–71.

[71] Chiu, G. and Chon, K. (1998). Efficient fault-tolerant multicast scheme for hypercube multicomputers. *IEEE Parallel and Distributed Systems*, 9(10):952–962.

[72] Choi, H., Daknis, S., Narahari, B., and Simba, R. (1997). Algorithms for mapping task graphs to a network of heterogeneous workstations. In *International Conference on High Performance Computing, Chennai, India*.

[73] Chretienne, P. (1989). A polynomial algorithm to optimally schedule tasks on a virtual distributed system under tree-like precedence constraints. *European Journal of Operational Research*, 43:225–230.

[74] Chu, T. and Abraham, J. (1981). Load balancing in distributed systems. *IEEE Software Engineering*, SE-8(4):401–412.

[75] Chu, W., Holloway, L., Lan, M., and Efe, K. (1980a). Distributed enumeration on network computers. *IEEE Computers*, C-29(9):818–825.

[76] Chu, W., Holloway, L., Lan, M., and Efe, K. (1980b). Task allocation in distributed data processing. *IEEE Computers*, 13(11):57–69.

[77] Chu, W. W. (1969). Optimal file allocation in a multiple computer system. *IEEE Computers*, C-18(10):885–889.

[78] Clos, C. (1953). A study of non-blocking switching networks. *Bell Systems Tech. J.*, 32:406–424.

[79] Coffman, E. (1976). *Computer and Job-Shop Scheduling Theory*. Wiley, New York.

[80] Coffman, E. and Graham, R. (1972). Optimal scheduling for two-processor systems. *Acta Informatica*, 1:200–213.

[81] Colib, J. and Chretienne, P. (1991). C.p.m scheduling with small communication delays and task duplication. *Operations Research*, 39(3):680–684.

[82] Crandall, P. and Quinn, M. (1993). Block data decomposition for data-parallel programming on a heterogeneous workstation network. In *2nd International symposium on high-performance distributed computing*, pages 42–49. IEEE Computer Society Press, Spokane.

[83] Cull, P. and Larson, S. (1998). Smaller diameters in hypercube-variant networks. *Telecommunication Systems*, 10(1-2):175–185.

[84] Dally, W. and Towles, B. (2004). *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Press, San Francisco.

[85] Daoud, M. and Kharma, N. (2006). An efficient genetic algorithm for task scheduling in heterogeneous distributed computing systems. In *IEEE Congress on Evolutionary Computation Sheraton Vancouver Wall Centre Hotel*, BC Canada. IEEE.

[86] Das, S. and Banerjee, A. (1992). Hyper petersen network: Yet another hypercube-like topology. In *Frontiers 92.*, pages 270–277, USA.

[87] Day, K. and Ayyoub, A. (1997). The cross product of interconnection networks. *IEEE Parallel and Distributed Systems*, 8(2):109–118.

[88] Day, K. and Tripathi, A. (1994). A comparative study of topological properties of hypercubes and star graphs. *IEEE Parallel and Distributed Systems*, 5(1):31–38.

[89] Deeter, D. and Smith, A. (1997). Heuristic optimization of network design considering all-terminal reliability. In *Reliability and Maintainability Symposium*, page 194 199, Philadelphia, USA.

[90] Devarakonda, M. and Iyer, R. (1989). Predictability of process resource usage: A measurement-based study of unix. *IEEE Software Engineering*, 15(12):1579–1586.

[91] D'Hollander, E. and Opsommer, J. (1987). Implementation of an automatic program partitioner on a homogeneous multiprocessor. In *International Conference on Parallel Processing*, pages 517–520.

[92] Dongra, J. J., Meuer, H., and Strohmaier, E. (1997). Top500 supercomputer sites. *Supercomputer*, 67:89–120.

[93] Dubis, D. and Prade, H. (1978). Operations on fuzzy numbers. *International Journal of Systems Science*, 9:613626.

[94] Duncan, R. (1990). A survey on parallel computer architectures. *Computer*, 23(2):5–16.

[95] Eager, D., Zahorjan, J., and Lazowska, E. (1989). Speedup versus efficiency in parallel systems. *IEEE Computers*, 38(3):408–423.

[96] Efe, K. (1982). Heuristic models of task assignment scheduling in distributed systems. *IEEE Computers*, 15(6):50–56.

[97] Efe, K. (1992). The crossed cube architecture for parallel computation. *IEEE Parallel and Distributed Systems*, 3(5):513 – 524.

[98] Efe, K. and Fernandez, A. (1994). Computational properties of mesh connected trees: A versatile architecture for parallel computation. In *1994 International Conf. on Parallel Processing.*, volume 1, pages 72–76.

[99] El-Dessouki, O. I. and Huen, W. H. (1980). Task allocation in distributed data processing. *IEEE Computers*, 13(11):67–69.

[100] El-Dissouki, O. and Huen, W. (1980). Distributed enumeration on between computers. *IEEE Computers*, 29(9):818–825.

[101] El-Rewini, H. and Abd-El-Barr, M. (2005). *Advanced computer architecture and parallel processing*. Wiley-Interscience, John Wiley and Sons, Inc., NY.

[102] El-Rewini, H., Ali, H., and Lewis, T. (1995). Task scheduling in multiprocessing systems. *IEEE Computers*, pages 27–37.

[103] El-Rewini, H. and Lewis, T. (1998). *Distributed and Parallel Computing*. Manning and Prentice Hall.

[104] El-Rewini, H., Lewis, T., and Ali, H. (1994). *Task Scheduling in Parallel and Distributed Systems*. Prentice Hall, Englewood Cliffs.

[105] Fattah, M., Daneshtalab, M., Liljeberg, P., and Plosila, J. (2013). Smart hill climbing for agile dynamic mapping in manycore systems. In *50th ACM / EDAC / IEEE Design Automation Conference (DAC)*, pages 1–6, Austin, TX,USA. IEEE.

[106] Feng, T. (1981). A survey of interconnection networks. *IEEE Computer Mag.*, 14(12):12–27.

[107] Fernadez, E. B. and Bussell, B. (1973). Bounds on the number of processors and time for multiprocessor optimal schedules. *IEEE Computers*, C-22(8):745–751.

[108] Fernandez-Baca, D. and Medepalli, A. (1989). Allocating modules to processors in a distributed system. *IEEE Software Engineering*, 15(11):1427 – 1436.

[109] Flynn, M. J. (1995). *Computer Architecture: Pipelined and Parallel Processor Design*. Jones and Bartlett.

[110] Friedman, R., Manor, S., and Guo, K. (2002). Scalable stability detection using logical hypercube. *IEEE Parallel and Distributed Systems*, 13(9):972–984.

[111] Gabow, H. (1982). An almost linear algorithm for two-processor scheduling. *Journal of the ACM*, 29(3):766–780.

[112] Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, CA.

[113] Gerasoulis, A. and Yang, T. (1992). A comparison of clustering heuristics for scheduling dags on multiprocessors. *J. Parallel and Distributed Processing*, 16(4):276–291.

[114] Goyal, A. and Agerwala, T. (1984). Performance analysis of future shared atorage systems. *IBM Journal of Reseaech and Development*, 28(1):95–107.

[115] Graham, S. and Seidel, S. (1993). The cost of broadcasting on star graphs and k-ary hypercubes. *IEEE Computers*, 42(6):756–759.

[116] Grewe, D. and O'Boyle, M. (2011). A static task partitioning approach for heterogeneous systems using opencl. In *20th international conference on Compiler construction: part of the joint European conferences on theory and practice of software*, pages 286–305.

[117] Hagin, A. Problem formulations, models and algorithms for mapping distributed multimedia applications to distributed computer systems. Technical Report 13/1994, Universitat Stuttgart, Facluta Informatik.

[118] Hamam, Y. and Hindi, K. (2000). Assignment of program modules to processors: a simulated annealing approach. *Euro. J. Operation Research*, 122(2):509–513.

[119] Hellmann, M. Introduction to fuzzy logic. `http://epsilon.nought.de/tutorials/fuzzy/fuzzy.pdf`.

[120] Hluchy, L., Dobrucky, M., and Dobrovodsky, D. (1996). A task allocation tool for multicomputers. In *Scientific Conference with International Participation-Electronics, Computers and Informatics*, pages 129–134, Kosice, Herlany.

[121] Hord, R. (1993). *Parallel Supercomputing in MIMD Architectures*. CRC Press.

[122] Hsieh, C. (2003). Optimal task allocation and hardware redundancy policies in distributed computing system. *European Journal of Operational Research*, 147:430–447.

[123] Hsieh, C. and Hsieh, Y. (2003). Reliability and cost optimization in distributed computing system. *Computer and Operation Research*, 30:1103–1119.

[124] Hui, C. and Chanson, S. (1997). Allocating task interaction graphs in heterogeneous networks. *IEEE Parallel and Distributed Systems*, 8(9):908–925.

[125] Hwang (2004). *Advanced computer architecture: parallelism, scalability, programmability.* 6th Reprint, TMH.

[126] Hwang, G. and Tseng, S. (1993). A heuristic task assignment algorithm to maximize reliability of a distributed system. *IEEE Reliability*, 42(3):408–415.

[127] Iqbal, M. (1986). Icase report no. 86-40. Technical report.

[128] Iqbal, M., Salt, J., and Bokhari, S. (1986). A comparative analysis of static and dynamic load balancing strategies. In *International Conference on Parallel Processing*, pages 1040–1047.

[129] Ismail, M. and Zin, M. (2010). Development of simulation model in heterogeneous network environment: Comparing the accuracy of simulation model for data transfers measurementover wide area network (wan). *International Journal of Multimedia and Ubiquitous Engineering*, 5(4):347–372.

[130] Johnson, D. S., Papadimitriou, C., and Yannakakis, M. (1988). How easy is local search? *J. Computer and System Sciences*, 37:79–100.

[131] Kaddoura, M., Ranka, S., and Wang, A. (1996). Array decomposition for non-uniform computational environments. *Journal of Parallel and Distributed Processing*, 36:91–105.

[132] Kafil, M. and Ahmed, I. (1998). Optimal task assignment in heterogeneous distributed computing systems. *IEEE Concurrency*, 6(3):42–51.

[133] Kartik, S. and Murthy, C. (1997). Task allocation algorithms for maximizing reliability of distributed computing systems. *IEEE Computers*, 46(6):719–724.

[134] Khan, A., McCreary, C., and Jones, M. (1994). A comparison of multiprocessor scheduling heuristics. In *International Conference on Parallel Processing*, volume II, pages 243–250.

[135] Kim, S. and Browne (1998). A general approach to mapping of parallel computations upon multiprocessor architectures. In *International Conference on Parallel Processing*, volume 3, pages 1–8.

[136] Kim, S. and Veidenbaum, A. V. (1999). Interconnection network organization and its impact on performance and cost in shared memory multiprocessors. *Parallel Computing*, 25(3):283–309.

[137] Klasing, R. (1998). Improved compressions of cube-connected cycles networks. *IEEE Parallel and Distributed Systems*, 19(8):803–812.

[138] Kohler, W. (1975). A prelimnary evaluation of the critical path method for scheduling tasks on multiple processor systems. *IEEE Computers*, 24(12):1235–1238.

[139] Koziris, N., Romesis, M., Tsanakas, P., and Papakonstantinou, G. (1998). An efficient algorithm for the physical mapping of clustered tasks graphs onto multiprocessor architectures. In *8th Euromicro Workshop on Parallel and Distributed Processing*.

[140] Kruskal, C. and Weiss, A. (1984). Allocating independent subtasks on parallel processors extended abstract. In *International Conference on Parallel Processing*, pages 236–240.

[141] Kuck, D. J. (1978). *The Structure of Computers and Computations*, volume I. John Wiley and Sons, Inc., NY.

[142] Kumar, J. M. and Patnaik, L. M. (1992). Extended hypercube: a hierarchical interconnection network of hypercubes. *IEEE Parallel and Distributed Systems*, 3(1):45–57.

[143] Lawrie, D. H. (1975). Access and alignment of data in an array processor.. *IEEE Computer*, C-24(12):1145–1155.

[144] Lee, C. (1991). Parallel machines scheduling with nonsimultaneous machine available time. *Discrete Applied Mathematics*, 30(1):53–61.

[145] Lee, C. and Shin, K. (1997). Optimal task assignment in homogeneous networks. *IEEE Parallel and Distributed Systems*, 8(2):119–129.

[146] Lee, C. H., Lee, D., and M.Kim (1992). Optimal task assignment in linear array network. *IEEE Computers*, 41(7):877–880.

[147] Lee, S. Y. and Aggarwal, J. K. (1987). A mapping strategy for parallel processing. *IEEE Computers*, 36(4):433–442.

[148] Leiserson, C. E. (1985). Fat-trees: Universal networks for hardware-efficient supercomputing. *IEEE Computer*, C-34(10):892–901.

[149] Levitin, G., Lisnianski, A., Ben-Haim, H., and Elmakis, D. (1998). Redundancy optimization for series-parallel multistate systems. *IEEE Reliability*, 47(2):165–172.

[150] Liang, L., Zhou, X.-G., Wang, Y., and Peng, C.-L. (2007). Online hybrid task scheduling in reconfigurable systems. In *International Conference on Computer Supported Cooperative Work in Design (CSCWD) 2007.*, pages 1072 – 1077, Melbourne, Victoria.

[151] Lin, J. and Yeh, P. (2001). Automatic test data generation for path testing using gas. *Information Sciences*, 131(1-4):47–64.

[152] Lisper, B. and Mellgren, P. (2001). Response-time calculation and priority assignment with integer programming methods. In *Work-in-progress and Industrial Sessions, 13th Euromicro Conference on Real-Time Systems*.

[153] Lo, V. (1988). Heuristic algorithms for task assignment in distributed systems. *IEEE Computers*, 37(11):1384–1397.

[154] Lo, V. (1993). *Heuristic Algorithms for Task Assignment in Distributed Systems*. PhD thesis, University of Illusion at Urbana.

[155] Loh, P. K. K., Hsu, W. J., Wentong, C., and Sriskanthan, N. (1996). How network topology affects dynamic loading balancing. *IEEE Parallel and Distributed Technology: Systems and Applications*, 4(3):25–35.

[156] Louri, A., , Weech, B., and Neocleous, C. (1998). A spanning multichannel linked hypercube: A gradually scalable optical interconnection network for massively parallel computing. *IEEE Parallel and Distributed Systems*, 9(5):497–512.

[157] Louri, A. and Neocleous, C. (1997). A spanning bus connected hypercube: A new scalable optical interconnection network for multiprocessors and massively parallel systems. *J. Lightwave Technology.*, 15(7):1241–1252.

[158] Louri, A. and Sung, H. (1994). Scalable optical hypercube-based interconnection network for massively parallel computing. *Appl. Optics*, 33(32):7588–7598.

[159] Ma, P., Lee, E., and Tsuchiya, M. (1982). A task allocation for distributed computing systems. *IEEE Computers*, 31(1):41–47.

[160] Magirou, V. and Milis, J. (1989). An algorithm for the multiprocessor assignment problem. *Operations Research Letters*, 8(6):351–356.

[161] Malluhi, Q. M. and Bayoum, M. A. (1994). Hierarchical hypercube: a new interconnection topology for massively parallel systems. *IEEE Parallel and Distributed Systems*, 5(1):17–30.

[162] Manoharan, S. (1997). Augmenting work-greedy assignment schemes with task duplication. In *International Conference on Parallel and Distributed Systems*, pages 772–779.

[163] Marcogliese, R. and Novarese, R. (1981). Module and data allocation methods in distributed systems. In *International Conference on Distributed Computing Systems*, pages 50–59. Springer-Verlag.

[164] McMillen, R. J. (1984). A survey of interconnection networks. In *IEEE Globecom.*, page 105.

[165] Moore, G. (1965). Cramming more components onto integrated circuits. *Electronics*, 38(8).

[166] Norman, M. and Thanisch, P. (1993). Models of machines and computation for mapping in multicomputers. *ACM Computing Surveys*, 25(3):263–302.

[167] Omara, F. A. and Arafa, M. M. (2010). Genetic algorithms for task scheduling problem. *J. Parallel and Distributed Computing*, 70(1):13–22.

[168] Opricovic, S. and Tzeng, G. (2004). Compromise solution by mcdm methods: a comparative analysis of vikor and topsis. *Euro. J. of Operational Research*, 156(2):445455.

[169] Orduna, J., Silla, F., and Duato, J. (2001). A new task mapping technique for communication-aware scheduling strategies. In *30th International Workshops on Parallel Processing (ICPP) Workshop*, page 349354, Valencia, Spain.

[170] Page, A. J., Keane, T. M., and Naughton, T. J. (2010). Multi-heuristic dynamic task allocation using genetic algorithms in a heterogeneous distributed system. *J. Parallel and Distributed Computing*, 70(7):758–766.

[171] Palis, M. (1996). Task clustering and scheduling for distributed memory parallel architectures. *IEEE Parallel and Distributed Systems*, 7(1):46–55.

[172] Parhami, B. (2000). Challenges in interconnection network design in the era of multiprocessor and massively parallel microchips. In *proceeding of International Conf. Communications in Computing (CIC-2000)*, pages 241–246, Las Vegas, Nevada, USA.

[173] Patel, J. H. (1981). Performance of processor-memory interconnections for multiprocessors. *IEEE Computer*, C-30(10):771–780.

[174] Patelm, A., Kasalik, A., and McCrosky, C. (2006). Area efficient vlsi layout for binary hypercube. *IEEE Computers*, 49(2):160–169.

[175] Peng, D. T., Shin, K., and Abdelzaher, T. (1997). Assignment and scheduling communicating periodic tasks in distributed real-time systems. *IEEE Software Engineering*, 23(12):745–758.

[176] Preparata, F. P. (1981). The cube-connected cycles: a versatile network for parallel computation. *Communications ACM*, 24(5):300–309.

[177] Price, C. and Krishnaprasad, S. (1984). Software allocation models for distributed computing systems. In *4th International Conference on Distributed Computing systems*, pages 40–48.

[178] Price, C. and Pooch, U. (1982). Search techniques for non-linear multiprocessor scheduling problems. *Naval Research Logistics Quarterly*, 29(2):213–233.

[179] Quinn, M. J. (2012). *Parallel Processing and Parallel Algorithms: Theory and Computation*. Mc Graw Hill.

[180] Qureshi, K. and Majeed, B. (2011). Task partitioning, scheduling and load balancing strategy for mixed nature of tasks. *J. Supercomputing*, 59(3):1348–1359.

[181] Rao, G., Stone, H., and Hu, T. (1979). Assignment of tasks in a distributed processor system with limited memory. *IEEE Computers*, C-28(4):291–299.

[182] Reid, D. (1995). Executing join queries in an uncertain distributed environment. *Mathematical and Computer Modelling*, 22(3):9–23.

[183] Roosta, S. H. (1993). *Parallel Computing: Theory and Practice*. Springer-Verlag.

[184] Rosenthal, A. (1982). Dynamic programming is optimal for non-serial optimization problems. *SIAM J. Comput.*, C-11(1):47–59.

[185] Saad, Y. and Schultz, M. (1998). Topological properties of the hypercube. *IEEE Computers*, 33(7):867–872.

[186] Sadayappan, P. (1987). Nearest-neighbour mapping of finite element graphs onto processor meshes. *IEEE Computers*, 36(12):1408–1424.

[187] Sadayappan, P., Ercal, F., and Ramanujam, J. (1990). Cluster partitioning approaches to mapping parallel programs onto a hypercube. *Parallel Computing*, 13(1):1–16.

[188] Sagar, G. and Sarje, A. (1991). Task allocation model for distributed systems. *Int. J. Systems Sciences*, 22(9):1671–1678.

[189] Sarje, A. and Sagar, G. (1991). Heuristic model for task allocation in distributed computer systems. In *Computers and Digital Techniques IEE Proceedings-E*, volume 138, pages 313–318. IEE.

[190] Scherson, I. D. Ortogonal graphs and the analysis and construction of a class of mins. In *ICPP 89.*, pages 380–388.

[191] Selvakumar, S. and Murthy, C. (1994). Static task allocation of concurrent programs for distributed computing systems with processor and resource heterogeneity. *Parallel Computing*, 20(6):835–851.

[192] Sharieh, A., Qatawneh, M., Almobaideen, W., and Sleit, A. (2008). Hexcell: modeling, topological properties and routing algorithm. *Eur.J. Sci. Res.*, 22(2):457–468.

[193] Shatz, S. M. and Wang, J.-P. (1992). Task allocation for maximizing reliability of distributed computer systems. *IEEE Computers*, 41(9):1156–1168.

[194] Shen, C. and Tasi, W. (1985). A graph matching approach to optimal task assignment in distributed computing system using a minmax criterion. *IEEE Computers*, C-34(3):197–203.

[195] Shi, Y., Hou, Z., and Song, J. (2000). Hierarchical interconnection networks with folded hypercubes as basic cluster. In *4th International Conference/Exhibition on High Performance Computing in the Asia-Pacific Region.*, volume 1, pages 134–137.

[196] Siegel, H. J. (1985). *Interconnection networks for large-scale parallel processing*. Lexington Books.

[197] Siegel, H. J., McMillen, R. J., and Mueller, P. T. (1979). A survey of interconnection methods for reconfigurable parallel processing systems. In *Nat. Computer Conf., AFIPS.*, pages 529–542.

[198] Sih, G. and Lee, A. (1993). De-clustering: A new multiprocessor scheduling technique. *IEEE Parallel and Distributed Systems*, 4(6):625–637.

[199] Sinclair, J. (1987). Efficient computation of optimal assignments for distributed tasks. *J. Parallel and Distributed Computing*, 4(4):342–362.

[200] Sinnen, O., To, A., and Kaur, M. (2011). Contention-aware scheduling with task duplication. *Journal of Parallel and Distributed Computing*, 71(1):77–86.

[201] Srinivasan, S. and Jha, N. (1999). Safety and reliability driven task allocation in distributed systems. *IEEE Parallel and Distributed Systems*, 10(3):238–251.

[202] Stone, H. (1977). Multiprocessor scheduling with the aid of network flow algorithms. *IEEE Software Engineering*, SE-3(1):85–93.

[203] Stone, H. (1978). Critical load factors in two-processor distributed systems. *IEEE Software Engineering*, 4(3):254–258.

[204] Stone, H. (1993). *High Performance Computer Architecture.* Addison Wesley.

[205] Stone, H. S. (1971). Parallel processing with the perfect shuffle. *IEEE Computer*, 20(2):153–161.

[206] Thurber, K. J. (1974). Interconnection networks—a survey and assessment. In *Nat. Computer Conf., AFIPS.*, pages 909–919.

[207] Tindell, A., Burns, A., and Wellings, A. (1992). Allocating hard real time tasks: An np-hard problem made easy. *Springer J. Real Time Systems*, 4(2):145–165.

[208] Tom, A. and Murthy, C. (1999). Optimal task allocation in distributed systems by graph matching approach and state space search. *J. Systems and Software*, 46(1):59–75.

[209] Tompkins, M. F. (2003). Optimization techniques for task allocation and scheduling in distributed multi-agent operations. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA.

[210] Topcuoglu, H., Hariri, S., and Wu, M. (2002). Performance-effective and low complexity task scheduling for heterogeneous computing. *IEEE Parallel and Distributed Systems*, 13(3):260–274.

[211] Towsley, D. (1986). Allocating programs containing branches and loops within a multiple processing system. *IEEE Software Engineering*, SE-12(10):1018–1024.

[212] Tripathi, A., Sarker, B., and Vidyarthi, D. (2000). A ga based multiple task allocation considering load. *J. High Speed Computing*, 11(4):203–214.

[213] Verma, A. and Raghavendra, C. (1994). *Interconnection Networks for Multiprocessors and Multicomputers: Theory and Practice*. IEEE Computer Society Press.

[214] Verma, A. K. (1997). Reliability-based optimal task-allocation in distributed-database management systems. *IEEE Reliability*, 46(4):452–459.

[215] Vidyarthi, D., Tripathi, A., and Sarker, B. (2001). Allocation aspects in distributed computing systems. *IETE Technical Review*, 18:449–454.

[216] Vidyarthi, D., Tripathi, A., and Sarker, B. (2006). Cluster based load partitioning and allocation in distributed computing systems. *International Journal of Computers and Applications*, 28(4).

[217] V.K.Balakrishnan (2007). *Schaums Outlines Graph Theory*. Fifth Reprint, TMH.

[218] Wang, L. and Tasi, W. (1988). Optimal assignment of task modules with precedence for distributed processing by graph matching approach and state space search. *BIT Numerical Mathematics*, 28(1):54–68.

[219] Ward, M. and Pomero, D. (1984). Assigning parallel executable intercommunicating subtasks to processors. In *4th International Conference on Parallel Processing*, pages 392–394, Bellaire, MI.

[220] Wilkinson, B. and Allen, M. *Parallel Programming Techniques and Applications Using Networked Workstations and Parallel Computers*. 2nd Edition, Pearson Prentice Hall,USA.

[221] Williams, E. (1983). Assigning processes to processors in distributed systems. In *International Conference on Parallel Processing*, pages 404–406.

[222] Woo, S., Yang, S., Kim, S., and Han, T. (1997). Task scheduling in distributed computing systems with a genetic algorithm. In *High Performance Computing on the Information Superhighway HPC Asia*, pages 301–305.

[223] Wu, C. and Liu, I. (1980a). Assignment of tasks and resources for distributed processing. In *IEEE COMPCON Fall 1980*, pages 655–662.

[224] Wu, C. and Liu, I. (1980b). A partitioning algorithm for parallel and distributed processing. In *International Conference on Parallel Processing*, pages 254–255.

[225] Xu, M., Xu, J., and Hou, X. (2005). Fault diameter of cartesian product graphs. *Info. Processing Letters.*, 93(5):245–248.

[226] Yaakob, S. and Kawata, S. (1999). Workers placement in an industrial environment. *Fuzzy Sets and Systems*, 106:289–297.

[227] Yang, B., Hu, H., and Guo, S. (2009). Cost-oriented task allocation and hardware redundancy policies in heterogeneous distributed computing systems considering software reliability. *Computers and Industrial Engineering*, 56(4):1687–1696.

[228] Yang, C. and Miller, B. (1988). Critical path analysis for the execution of parallel and distributed programs. In *International Conference on Distributed Computing Systems*, pages 366–373.

[229] Yanping, L. and Haijiang, L. (2009). Welding multi-robot task allocation for biw based on hill climbing genetic algorithm. In *Technology and Innovation Conference 2009 (ITIC 2009)*, pages 1–8, Xian, China. IET.

[230] Yao, J. and Chaing, J. (2003). Inventory without backorder with fuzzy total cost and fuzzy storing cost defuzzified by centroid and signed distance. *Operations Research*, 148:401–409.

[231] Yin, P., Shao, B., Cheng, Y. P., and Yeh, C. (2009). Metaheuristic algorithms for task assignment in distributed computing systems: A comparative and integrative approach. *The Open Artificial Intelligence Journal*, 3(1):16–26.

[232] Yin, P.-Y., Yu, S.-S., Wang, P.-P., and Wang, Y.-T. (2007a). Multi-objective task allocation in distributed computing systems by hybrid particle swarm optimization. *Applied Mathematics and Computation*, 184(2):407–420.

[233] Yin, P.-Y., Yu, S.-S., Wang, P.-P., and Wang, Y.-T. (2007b). Task allocation for maximizing reliability of a distributed system using hybrid particle swarm optimization. *Journal of Systems and Software*, 80(5):724–735.

[234] Youssef, A. (1995). Design and analysis of product networks. In *Fifth Sym. Frontiers of Massively Parallel Sys.*, pages 521–528, USA.

[235] Youssef, A. and Narahari, B. (1990). Banyan-hypercube networks. *IEEE Parallel and Distributed Systems*, 1(2):160–169.

[236] Youyao, L., , Jungang, H., and Huimin, D. (2008). A hypercube based scalable interconnection network for massively parallel system. *J. Computer*, 3(10):58–65.

[237] Zadeh, L. (1965). Fuzzy sets. *Information and Control*, 8(3):338353.

[238] Zadeh, L. (1975). The concept of a linguistic variable and its application to approximate reasoning. *Information Sciences*, 8(3):199–249.

[239] Zhang, X. (1991). System effects of interprocessor communication latency in multicomputers. *IEEE Micro*, 11(2):12–15.

[240] Zheng, S., Cong, B., and Bttayeb, S. (1993). The star-hypercube hybrid interconnection networks. In *ISCA Intl. Conf. Comput. Appl. Design, Simulation and Analysis.*, pages 98–101, USA.