

**STUDIES ON THE PERFORMANCE EVALUATION
OF A
LINEARLY EXTENSIBLE MULTIPROCESSOR NETWORK**

A THESIS

*submitted in fulfilment of the
requirements for the award of the degree*

of

DOCTOR OF PHILOSOPHY

in

ELECTRONICS AND COMPUTER ENGINEERING

CENTRAL LIBRARY
INDIAN INSTITUTE OF TECHNOLOGY
ROORKEE-247 667

510975'

By

11 - 7 - 82

MOHAMMAD QASIM RAFIQ




**DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING
UNIVERSITY OF ROORKEE
ROORKEE - 247 667 (INDIA)**

SEPTEMBER, 1995


CANDIDATE'S DECLARATION


I hereby certify that the work which is being presented in the thesis entitled "**STUDIES ON THE PERFORMANCE EVALUATION OF A LINEARLY EXTENSIBLE MULTIPROCESSOR NETWORK**" in fulfilment of the requirement for the award of the Degree of **Doctor of Philosophy** and submitted in the Department of Electronics & Computer Engineering of the University is an authentic record of my own work carried out during a period from May, 1993 to September, 1995 under the supervision of **Prof. J.P. Gupta**, and **Dr. Padam Kumar**.

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

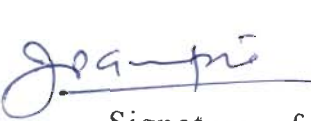


(MOHAMMAD QASIM RAFIQ)

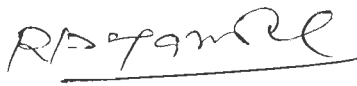
This is to certify that the above statement made by the candidate is correct to the best of our knowledge.

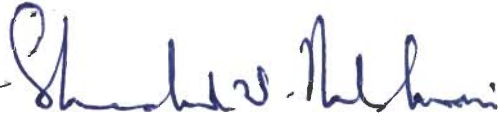
Date: 01.09.95 
Dr. Padam Kumar
Reader,
Department of Electronics &
Computer Engg.
University of Roorkee


Prof. J.P. Gupta
Member Secretary,
AICTE,
New Delhi

The Ph.D. viva-voce examination of MOHAMMAD QASIM RAFIQ Research Scholar, has been held on 27/7/96

 
Signature of
Supervisors


Signature of
H.O.D.


Signature of
External Examiner

DR. R. P. AGARWAL
PROFESSOR AND HEAD
DEPARTMENT OF ELECTRONICS
AND COMPUTER ENGINEERING
UNIVERSITY OF ROORKEE
ROORKEE-241 002

ABSTRACT

Exploiting parallelism is now a necessity to improve the throughput of the computer systems. In terms of hardware, this typically means providing multiple, simultaneously active processors. In terms of software, it means structuring a program as a set of largely independent subtasks. The structuring of a program is usually represented by a problem graph. The nodes of the graph denote the subtasks of a program and the links/arcs between them represent the precedence data relations among the subtasks. Research is active in the direction of developing new multiprocessor architectures and schedule the partitioned program onto it in order to achieve higher execution speeds and/or increased programming comfort.

The present work, reported in this thesis, is concerned with the development of a new multiprocessor network, called a Linearly Extensible Tree (LET) network and a dynamic scheduling scheme, named as Minimum Distance Scheduling (MDS) scheme, for parallel execution of tree-structured problems. In addition to this, simulation studies are carried out to compare the performance of LET multiprocessor network and the proposed scheduling scheme MDS with other similar multiprocessor networks and related scheduling schemes available in literature, on different types of problem graphs- general and tree-structured in particular.

The model proposed is a linearly extensible tree (LET) multiprocessor network, which exhibits the desirable properties of similar types of multiprocesor networks. The LET network combines linear extensibility with small number of processing elements per extension. The network has lower diameter, hence reduces the average path-length travelled by all messages and contains a constant degree per node.

A dynamic scheduling scheme MDS has been developed, which forces minimum distance constraint, based upon only the adjacency matrix information of the LET network, and with relatively small overhead, it oversees that the task arrives at the proper processor maintaining the

task relations even for grossly unbalanced problem graphs or in the presence of failing nodes or links. This scheduling scheme is compared with other available static and dynamic scheduling schemes in the literature for implementation on LET network and other similar multiprocessor networks for graph problems in general and tree-structured problems in particular. The superiority of the developed organisation i.e. LET network and MDS scheme on other existing organisation such as Binary deBruijn Multiprocessor (BDM) network and Round-Robin (R-R) scheme, BDM network and Minimum Load (ML) scheduling, hypercube and Dimension Exchange Method (DEM) scheme, hypercube network and Gradient Model (GM) scheme, and hypercube network and Hierarchical Balance Method (HBM) has been established.

Performance of LET network has also been studied for problems having an Acyclic Precedence Graph (APG) structures. In this connection a static Latest Precedence Scheduling (LPS) algorithm has been developed which runs faster compared to a similar other algorithm. The LPS algorithm preserves minimum distance. Performance of LET under this algorithm for APG'S has been tested and compared to other networks.

ACKNOWLEDGEMENTS

It is with deep sense of gratitude and reverence that I express my sincere thanks to my supervisors, **Prof. J.P. Gupta**, Prof. of Electronics and Computer Engineering and presently *Member Secretary*, All India Council for Technical Education, Govt. of India, and **Dr. Padam Kumar**, *Reader*, Electronics and Computer Engineering Department, University of Roorkee, for their erudite guidance and advice, constant keen interest at every stage of this work. Their untiring and painstaking efforts, constant encouragement and suggestions, methodical approach and invaluable help throughout, made it possible for me to complete this work in time. I consider myself very fortunate to be associated with such scholars of the subject, whose affection, guidance and scientific approach served a veritable incentive for completion of this work.

I was very fortunate to come in contact with **Prof. Derek R. Wilson**, professor at the University of Westminster, London, UK during his two visits to Roorkee. His valuable suggestions and the time spared for highly fruitful discussions is gratefully acknowledged.

I shall ever remain indebted to **Prof. R. P. Agarwal**, Head, Department of Electronics and Computer Engineering, for the facilities provided to me by the department and his kindness and generous help extended during the completion of the work.

I gratefully acknowledge the financial support received from the Government of India, Ministry of Human Resources under QIP scheme and also the administration of A.M.U. Aligarh for deputing me under this scheme. I am much obliged to **Prof. S.C. Handa**, Coordinator, QIP, for extending the facilities during my stay at the University Of Roorkee.

I am grateful to my elder brother **Prof. M.A. Siddiqui**, Prof. of Medicine, Principal & CMS, JN Medical College, AMU, Aligarh, for his kind support, concern, encouragement and advice throughout my life. His own achievements have greatly inspired me. His constant moral support and sharing of my household responsibilities enabled me to devote myself fully to my work at Roorkee.

I also take this opportunity to thank and acknowledge to bhabhis, **Mrs. Rafat Siddiqui**, **Mrs. Kamlesh Gupta** and **Mrs. Anita Kumar**, who were a constant source of encouragement and inspiration for me during my stay at Roorkee. I can not but remember the smile, devotion, understanding and trust that my wife, **Tazeen** and children, **Saman**, **Mansoor** and **Saboor**, who bore my neglect and separation from them which was inevitable to complete this research work.

Although it is not possible to name them individually, I can not forget my well wishers at A.M.U., Aligarh, especially **Prof. R.K. Gupta**, **Prof. A.K. Gupta**, **Prof. S. Basu**, **Prof. E. Hussain**, **Prof. F. Ghani**, **Prof. G.C. Upreti**, **Prof. K.A. Khan**, **Prof. M.T. Ahmad**, **Prof. S.A. Abbasi**, **Dr. S.K.A. Qasmi**, **Dr. Jamal Ahmad**, **Mr. S.H. Hasan**, **Mr. M.Naim Akhter**, **Dr. Sattar Husain** and **Mr. Z.H. Zaidi**, for their persistent support and cooperation.

This acknowledgement would be incomplete if I fail to express my deep sense of obligations to all my friends and colleagues at the University of Roorkee, for their warm friendship, encouragement, cooperation and valuable suggestions during my stay, especially **M/S Bahar Alam**, **A. Hasan**, **Zulfequar Ahmad**, **Khalid Moin**, **Rajiv Saxena**, **S.K. Sinha**, **M. Arif**, **S.A. Ansari**, **Ziauddin Ahmad**, **P. Mustajab** and **M. Shakeel**.

MOHAMMAD QASIM RAFIQ

CONTENTS

ABSTRACT	(i)
ACKNOWLEDGEMENTS	(iii)
CONTENTS	(iv)
1. INTRODUCTION	1
1.1 Parallelism as a concept	2
1.1.1 Pipelined processors	2
1.1.2 Vector Processors	3
1.1.3 Array Processors	3
1.1.4 Shared memory multiprocessors	3
1.1.5 Message passing multiprocessors	4
1.2 Need for performance evaluation	6
1.3 Motivation	7
1.4 Statement of the problem	8
1.5 Organization of the thesis	9
2. REVIEW OF MULTIPROCESSOR ARCHITECTURES AND SCHEDULING SCHEMES	11
2.1 Multiprocessor architecture	12
2.1.1 Shared-memory systems	13
2.1.2 Message passing systems	14
2.2 Multiprocessors interconnection networks	16
2.2.1 Interconnection organization	16
2.2.2 Network characteristics	16
2.2.3 Hypercube interconnection network	18
2.2.3.1 Hypercube topology	18
2.2.4 Basic concept of tree	21
2.2.4.1 Hypertree network	23
2.2.4.2 Binary de-Bruijn multiprocessor (BDM) network	24
2.2.4.3 Hyper de-Bruijn network	26
2.3 Properties of an interconnection network	26
2.4 Review of scheduling schemes	29
2.4.1 Introduction	29
2.4.2 Definitions	30
2.4.3 The classification scheme	33

2.4.3.1	Hierarchical classification	33
2.4.3.2	Flat classification characteristics	39
2.5	Problem structures	43
2.6	Multiprocessor architectures	43
2.7	Classification of scheduling algorithms	48
3.	LINEARLY EXTENSIBLE TREE NETWORK	59
3.1	Multiprocessor interconnection networks	59
3.2	Binary de-Bruijn Multiprocessor (BDM) network	60
3.3	Linearly extensible tree (LET) multiprocessor network	62
3.3.1.	Design and analysis	62
3.3.2	Properties of LET network	64
4.	MINIMUM DISTANCE DYNAMIC SCHEDULING SCHEME	71
4.1	The scheduling techniques	72
4.2	Minimum distance property	73
4.3	Minimum Distance Scheduling (MDS) scheme	74
4.4	The MDS algorithm	76
4.5	Simulation results	80
4.5.1	Dynamic load model	81
4.5.2	MDS Scheme on LET network	84
4.5.3	Comparison with other networks	89
5.	COMPARISON OF MDS SCHEME WITH OTHER SCHEDULING SCHEMES	101
5.1	Other scheduling schemes	101
5.1.1	Round-Robin (R-R) scheduling scheme	101
5.1.2	Minimum-Load (M-L) scheduling scheme	102
5.1.3	Dimension Exchange Method (DEM) scheduling scheme	104
5.1.4	Hierarchical Balancing Method (HBM) scheduling scheme	105
5.1.5	Gradient Model (GM) scheduling scheme	105
5.2	Performance Study of the MDS and other schemes on LET	105
5.3	Organizational model	111
5.4	Performance of various organizational models	112
5.4.1	Simulation results	113

6.	PERFORMANCE OF LET FOR GRAPHSTRUCTUED PROBLEMS	121
6.1	Introduction	121
6.2	Latest Precedence scheduling strategy	123
6.2.1	List Scheduling	124
6.2.2.	HDLF algorithm	124
6.2.3	Task Selection	126
6.3	Lates precedence schedulng algorithm	128
6.4	Graph generation	130
6.5	Performance of LET and other network	132
7	CONCLUSIONS AND FUTURE WORK	143
7.1	Conclusions	144
7.2	Future work	146
	REFERENCES	147

INTRODUCTION

High speed computing is essential to modern research and development as the demands for more and more computational power continue to grow. We have started relying more and more on computer simulation rather than on analysis or experimentation. A strong high-performance computer industry is essential for a successful modern economy. Critical security areas and a broad range of private sector activities depend on high-performance computers.

Most of the computer performance improvements made so far, have been based on technological developments. In fact the so called four generations of computers are defined by improved technologies (from tubes to VLSI). Semiconductor technology already has reached a point of maturity that significant switching-speed improvements are difficult to obtain. The fundamental limitation is the speed of light, because electrons can not move faster than this speed limit. Thus the source for increasing computing power must be looked for in an area other than switching technology. This has forced the designers to embrace parallel processing and to look for new architectural concepts that have the capability of providing orders-of magnitude performance increases. It is in this context that parallel processing plays a growing role in the computer industry [58,71].

The basis of most high-performance computer systems, according to Flynn's taxonomy[morgan], is the Multiple Instruction-stream and Multiple Data-stream (MIMD) organization. These systems employ multiple

processors which execute independent instruction streams accessing data autonomously. The design of such systems requires careful consideration of the number of processors and their interconnection topology, besides the proper choice of scheduling strategies for allocating the work.

Initial MIMD models often called *multiprocessors*, were based on the shared memory concept where the processors were connected to a number of memory modules to form a common global shared memory. However, such models can produce severe memory contention problems when processors try to access data residing on the same memory module. To overcome the memory contention problem the distributed memory MIMD model was introduced. In such a configuration the memory is distributed amongst the processors, with each processor and its associated local memory implemented on the same hardware device. If data needs to be accessed from local memory of another processor, then it is transferred using the interconnection network. Thus, the distributed MIMD model is scalable to higher orders of parallelism compared to shared memory model. However, the distributed model is associated with the overhead of interprocessor communication.

1.1 PARALLELISM AS A CONCEPT

It was natural that the first efforts in parallel processing were extensions of the sequential von Neumann Model [6]. The main techniques used to extend the sequential von Neumann model to parallel architectures are by employing the concept of 'Pipelining, Vector processors, Array processors and Multiprocessors.

1.1.1 Pipelined processors :

The process of pipelining divides a task T into subtasks

T_1, T_2, \dots, T_K and assigns them to a chain of processing stations. These sections are called pipeline segment processors. Parallelism is achieved when several segments operate simultaneously.

1.1.2 Vector processors :

Vector processors are specially designed to handle computations formulated in terms of vectors. A vector processor has a set of instructions that treat vectors as single operands. Since vectors are one-dimensional arrays and the same sequence of operation is repeated for every vector element, vector processing is ideally suited to pipeline arithmetic.

1.1.3 Array processors :

The term array processor refers to a synchronous parallel computer, in which the same instruction is performed on different data that each processor fetches from its own memory. This is why this model is sometimes called data-parallel model. The interconnection network facilitates data communication among the processing units, memory processing units and memories. This type of computer is also called Single-Instruction-Multiple-Data (SIMD) computer. The source of parallelism in SIMD computers is that one instruction operates on several operands simultaneously.

1.1.4 Shared-memory multiprocessors:

Another way of introducing parallelism is to use several processors, each including a control unit, an ALU, shared memory and I/O modules. Communication between processors is via a common memory. Each processor operates its own instruction stream, fetched either from the

local memory or from shared memory on the data fetched from the shared memory. The interconnection network facilitates data exchanges between processors, and between processor and shared memory.

1.1.5 Message passing multiprocessors:

In this model, the memory is distributed among the processors such that each processor has its own program and data memory. The communication of shared data is achieved via messages exchanged directly between processors through an interconnection network. It has been shown that this model is scaleable to several hundred, possibly thousands of processor/memory units and that this is the defacto standard of high performance parallel computers. However the design and optimisation of the interconnection network in these large parallel machines is an area that still requires considerable research and development.

Following terms are commonly used in multiprocessors:

a) *Processor complexity* - It refers to the CPU power and the internal structure of each processing element. Processor complexity varies from one architecture to another. *Homogeneous systems* - These systems have all processors with identical capabilities. *Heterogeneous systems* - These systems have processors which are not identical.

b) *Mode of operation* - It is a general term referring to both instruction control and data handling. The traditional mode of operation is command-flow, so called because the flow of events is triggered by commands derived from instruction sequences. Another method is to trigger operations as soon as their operands become available. This is known as data-flow operation. In this case, the control is determined by the availability of data. Yet another mode of control is demand-flow, in

which computations takes place only if their results are required by other computations. Combinations of these control modes are also possible.

c) *Memory structure* - It refers to the mode of operation and the organization of computer memory. In some new computer models, such as connectionist architecture and neural networks, memory consists of interconnection weights that indicate how easily connection can be made. In ordinary computers, memory organization and the size of the memory files are closely related to data structure.

d) *Interconnection network* - It refers to the hardware connections among processors and between processors and memories. The architecture of interconnection network should match the algorithm communication geometry as closely as possible. Computers with simple interconnection networks are efficient only for a small number of algorithms, whereas complex interconnection networks can be configured for a broad range of applications. Of course, the price paid in this case is increased cost and extra switching time.

e) *Number of processors and memory size* - It simply indicates how many processors the parallel system contains and how large the main memory is. In general, more processors provide more computing power which enables the system to approach more complex problems. When the size of the algorithm is greater than the size of the system, algorithm partitioning is required. Algorithm partitioning may have undesired side-effects, so ideally the number of processors should match the size of the algorithm.

1.2 NEED FOR PERFORMANCE EVALUATION

Currently, one of the most important issues in parallel processing is how to effectively utilize parallel computers that have become increasingly complex. It is estimated that many modern super computers and parallel processors deliver only 10 percent or less of their peak performance potential in a variety of applications. Yet high performance is the very reason why people build complex machines.

The causes of performance degradation are many. Performance losses occur because of mismatches among application, software, and hardware. In complex systems, mismatches may occur among software modules or hardware modules. The communication network bandwidth may not correspond to the speed of the processor or that of memory introducing unwanted latency.

Mapping applications to parallel computers and balancing parallel processors is indeed a very difficult task, and the state of understanding in this area is quite inadequate. Moreover, small changes in problem size while using different algorithms or different applications may have undesirable effects and can lead to performance degradation.

The various indices responsible for system performance include Load Imbalance Factor (LIF) and communication overhead, the complexity of the system and algorithm, efficiency of the system and speedup which are measures of different aspects of a computer system's performance. It is precisely in this area that the work presented in this thesis is based and it will be shown that a Linearly Extensible Tree (LET) network with high overall performance has been instantiated.

1.3 MOTIVATION

A declarative machine project, known as CTDNet, is being undertaken by the University Of Roorkee (India) in collaboration with University of Westminster (U.K) supported by UGC (India) and British Council. The main object of the project has been the development of an extensible parallel machine to support functional programming languages. Some background of the project, which has been the main motivation behind the work carried out in this thesis, is given below.

The first phase, which was named CTDNet [20], introduced the development of a reduction machine that was designed originally for a Cube Type Distributed (CTD) multiprocessor. A CTDNet program is a binary lambda graph in which every node represents an application expression. The evaluation strategy was basically eager, which encouraged concurrent computation, with lazy implementation of conditional and iteration structures. One major criticism of CTDNet relates to its efficiency: in simulation runs, the number of messages generated and the number of process instantiations were very high. Another criticism is that it had no provision for modeling input/output within the lambda calculus framework.

A later version CTDNet-2 was developed to overcome the shortcomings of the original version [86]. This handles program expressions by graph reduction rather than string reduction, which improved CTDnet-2's performance. However, the major drawback in such networks is due to the small grain size of the computation and the consequent heavy communication overhead.

In an effort to overcome the short comings of the original CTDNet, due to the development of supercombinators, which had a profound effect

on the design of the machine model and, not surprisingly, on CTDNet model as well, the latest version of CTDNet, known as CTDNet-3, was developed with the main objective of increasing the grain size in order to minimize the communication overheads.

For efficient implementation of functional language programmes (which are mostly tree structured task graphs), on parallel machines, apart from designing a parallel evaluation model, it is essential that a suitable topology for processor interconnection coupled with appropriate scheduling strategy is used and parallelism is exploited to the optimum extent possible. Suitable topology and appropriate scheduling strategy, referred to as an organization model, depends on various factors including the dynamic behavior of the programs. In CTDNet project [86], several types of networks were experimented for executing functional programs such as shared bus, ring network, tree structure, hypercube, de-Bruijn network etc., but the optimal network of any particular type of architecture was not established.

The present work is motivated by the requirement for the design and development of network model which can improve upon the results for other networks experimented on. The proposed architecture has been tested for binary, ternary and arbitrary task trees. Randomly generated structured task graphs, which are most commonly generated out of the functional language programs, have also been incorporated in the simulation runs for testing the performance of organisation model.

1.4 STATEMENT OF THE PROBLEM

The complete work, as presented in this thesis, can be divided into three parts. The first part is concerned with the design and development

of a tree structured network model. The main aim of this model is to reduce the run time overheads and therefore reduce latency in the interconnection network. The second part is concerned with the development of a scheduling scheme that can maintain a highly balanced load profile on the new network (LET) for tree structured problems and randomly generated task graph structures. Repeated simulation experiments have been performed to evaluate the performance of the interconnection system. The third and the last part of the thesis presents the implementation of different static/dynamic scheduling schemes on the developed systems and compares it with various other similar networks. A comparative simulation study has been carried out and the superiority of the system is established.

1.5 ORGANIZATION OF THE THESIS

This thesis is organized as follows:

* **Chapter 2:** *Review of the multiprocessors architectures and scheduling schemes.* In this chapter the basic concepts and properties of various tree-structured multiprocessor network has been discussed in the first part. In the second part, a review of the scheduling schemes, starting from the classification to the present scenario, based on tree structured problems and the tree type architectures is presented. Various factors influencing the run time overheads have been discussed.

* **Chapter 3:** *Linearly extensible tree network.* This chapter discusses the design and analysis of a linearly extensible tree network and its various properties followed by its comparison with the BDM and hypercube.

* **Chapter 4:** *Minimum distance dynamic scheduling scheme.* This chapter discusses different dynamic scheduling schemes suitable for tree type networks. A new dynamic scheduling scheme called Minimum Distance Scheduling (MDS) scheme has been described for LET network. The MDS scheme has been implemented, through simulation, on the LET network and other networks in order to confirm the performance of scheme.

* **Chapter 5:** *Comparison of MDS scheme with other scheduling schemes.* In this chapter other scheduling schemes have been discussed briefly. The MDS scheme along with these static/dynamic scheduling schemes have been implemented on LET network. The concept of organisation model i.e. the implementation of a scheduling scheme on a network has been discussed in short and a comparison of proposed organisation model with existing models has been done in the last.

* **Chapter 6:** *Performance of LET for graph - structured problems.* This chapter addressed the issue of static scheduling for graph structured problems. A static scheduling algorithm called latest precedence scheduling for acyclic precedence graphs has been described. The scheme has been implemented on the LET and other networks for graph structured problems. Various simulation experiments carried out have been reported here.

* **Chapter 7:** *Conclusion and future work.* This chapter presents conclusions and recommendations for future extensions of the work.

REVIEW OF MULTIPROCESSOR ARCHITECTURES AND SCHEDULING SCHEMES

Multiprocessing is the simultaneous execution of task on a parallel asynchronous computer system. A parallel asynchronous computer is a system whose active nodes are either processors or simple computers that cooperate closely but independently. These are multiple-instruction multiple-data stream (MIMD) computers. There are two basic multiprocessor models: shared-memory and message-passing systems. The shared-memory model provides a globally shared physical address space, which is highly desirable from the programmer's point of view. However, simultaneous access to shared-memory by many processors complicates the design of such systems. In contrast, message-passing model provides memory associated with each processor and data between processors is passed through messages.

MIMD computers are suitable for a much larger class of computations because they are inherently more flexible. This flexibility is achieved at the cost of a considerable more difficult mode of operation.

The organisation of this chapter is divided in two parts. In the first part of this chapter theory of shared memory and message passing multiprocessor architecture has been discussed. The properties of commonly used four multiprocessor architectures using Hypercube, de-Bruijn, hypertree and hyper de-Bruijn networks have been explained in detail. The second part of this chapter deals with basic definition and various classifications of scheduling schemes. Finally a review of different scheduling schemes on related multiprocessor networks has been given.

2.1 MULTIPROCESSOR ARCHITECTURE

A multiprocessor system is a single computer incorporating a number of independent processors that work together to solve a given problem. Figure 2.1 shows the relations among algorithm granularity, degree of hardware coupling and communication mode, and the difference between distributed and parallel processing. Distributed processing occurs when hardware resources cooperate loosely to process jobs. Examples of distributed processing are use of multiple-computers connected through networks to solve different jobs. Whereas in parallel processing hardware resources cooperate closely to process tasks simultaneously [22, 34, 58, 59, 71].

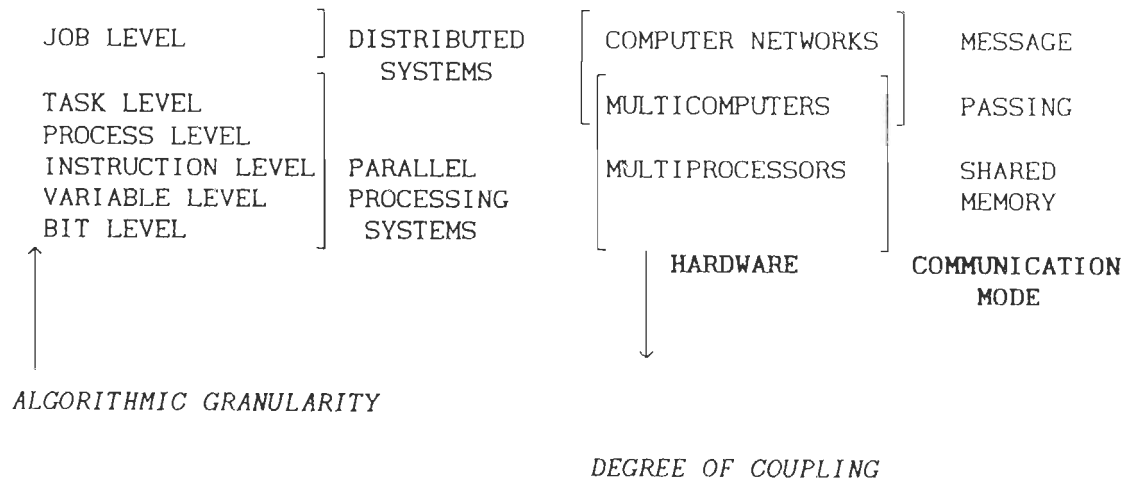


Figure 2.1 Relation between algorithmic granularity and multiprocessing systems.

There is an important difference between multiprocessors and multiple computers. A multiple computer consists of several computers - each with its own processor(s), memory, I/O and operating system, whereas a multiprocessor system has only one operating system and its processors share memory and I/O resources through an interconnection

network. As the number of processors increases, the interconnection network plays an important role in the overall performance of the system. Multiprocessors can be classified as shared-memory systems or message-passing systems.

2.1.1 Shared-memory systems

Multiprocessor architectures with shared-memory, also known as tightly coupled systems, have complete connectivity between processors and memory modules. A simplified block diagram of shared-memory systems is shown in Figure 2.2. It consists of a set of n processor elements, not necessarily identical, a set of m memory modules, and an interconnection network. The primary memory may be centralized ($m=1$) or partitioned into several modules. The common memory must be accessed by all processors in the system [71].

Data exchange between processors and memories is frequent and intense. The interconnection network is a potential bottleneck for these systems. While memory contention (memory access conflict) has always been a performance factor in uniprocessor systems, it becomes more important in parallel shared-memory systems simply because of the need of many processors to simultaneously access the same memory locations. To decrease the communication traffic in the network and the chance of memory contention, several alternative solutions exist. A local memory (LM), directly accessed by the processor, may be placed near the processor, thus reducing the number of memory requests through the interconnection network. Also, a cache memory may be provided in order to increase the memory bandwidth. A memory mapping (MM) unit is required to decide which memory requests are local and which are global.

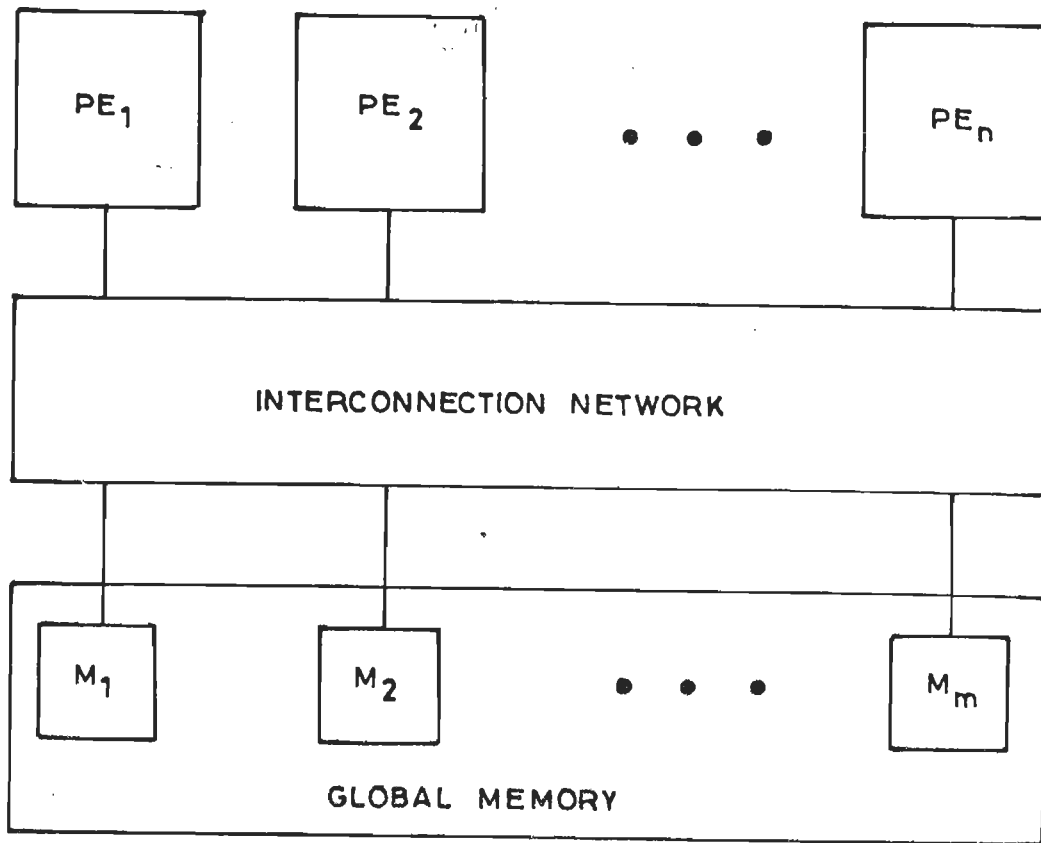


Figure 2.2 Shared-memory systems

The major limitation of a shared-memory system is the possibility of primary memory access conflicts. This restriction tends to put an upper bound on the number of processors that can be effectively incorporated in the system and supported by a single operating system. Shared-memory systems are efficient for small to medium sized multiprocessors.

2.1.2 Message-passing systems

Message-passing systems, also known as loosely coupled systems, consist of several computer modules and an interconnection network. Each

computer module has a processor, a memory and an I/O interface. Data communication is carried out through messages, not through shared variables as in the previous case. The length of message varies, but usually each message consists of a number of fixed-size packets. Inter computer communication follows a predetermined communication protocol. Thus the active node in a message-passing system is a computer and the degree is not as great as that of shared-variable systems [58,71] as shown in Figure 2.3.

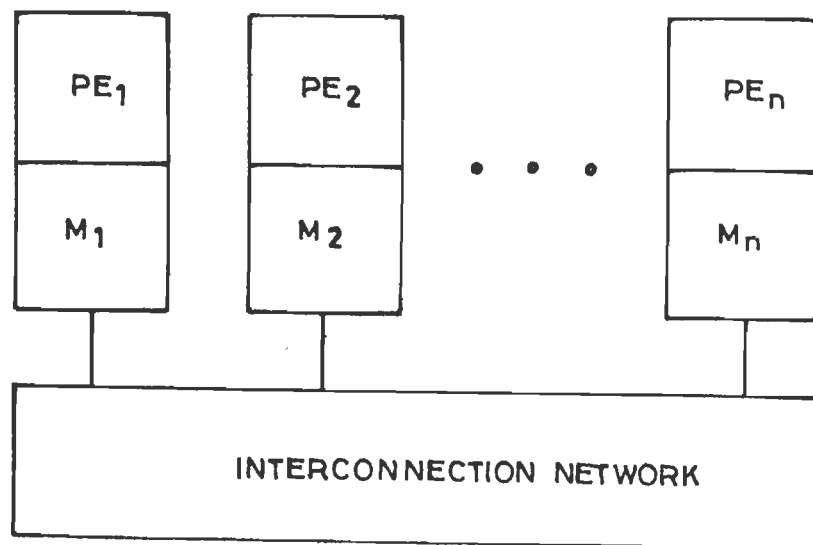


Figure 2.3 Message-passing systems

Shared-memory models can perform message-passing primitives easily, but the reverse is not true. This is because data structures are shared among processors in the former model. Properties such as simplicity and scalability make message-passing multiprocessors prime candidates for very large systems. Message-passing systems are more efficient for problems that can be partitioned into larger tasks that do not interact very frequently.

2.2 MULTIPROCESSOR INTERCONNECTION NETWORKS

2.2.1 *Interconnection organisation*

As a result of increasing the number of functional modules in a multiprocessor, the interconnection network becomes increasingly complex. Examples of multiprocessor interconnection networks are - time-shared or common buses, crossbar switches, multiport memories, Hypercubes, meshes and multistage interconnection networks. The multistage interconnection networks are feasible interconnections for large multiprocessor systems. Multistage interconnections allow processor-to-processor and processor-to-memory communications in a more general way than the other organisations [46-48,58,59,71].

2.2.2 *Network characteristics*

Some important characteristics of a multistage interconnection network are its mode of operation, switching technique, routing technique and interconnection network topology [34,58,59,71].

(i) Operation modes

There are two basic modes of network operation: synchronous and asynchronous. In the synchronous mode, the network is centrally supervised. The connection paths are established simultaneously and remain set until the control disconnects them. In the asynchronous mode, connection paths are setup or disconnected on an individual basis. The asynchronous mode of operation is more appropriate for multiprocessor systems.

(ii) Switching techniques

There are three basic switching techniques: circuit switching, packet switching and wormhole switching. Circuit switching sets up the switches and ports and establishes a dedicated path between an input-output pair. This technique is efficient for larger transmissions. Packet switching refers to a technique in which messages between any two terminals are broken into several shorter, fixed-length packets, which are routed independently to their destination using store-and-forward procedures. In wormhole switching a message is also broken into smaller parts (called flits), as in packet switching; however, the difference is that here all flits follow the same route. Compared with circuit switching, packet switching is efficient for shorter and more frequent transmissions.

(iii) Routing techniques

The routing technique is the method of establishing communication paths and resolving conflicts. Three basic routing techniques have been considered: centralized, distributed and adaptive. In the centralized routing scheme, a central control makes all the logic decisions needed to setup communication paths. This scheme is more flexible for small to medium scale systems. In the distributed scheme, logical decisions are made locally, based on current conditions. In adaptive scheme, information about the network is collected globally, but routing decisions are made locally.

(iv) Interconnection network topology

The network topology is the way in which the switches are

interconnected. The topology is perhaps the most important factor determining network performance. The binary Hypercube having a robust topology, is highlighted in the next section.

2.2.3 Hypercube interconnection network

Many interconnection structures for loosely-coupled multiprocessors have been proposed in the literature. Examples includes: Binary tree, Lattice structure, Flip net, Omega net, Indirect n-cube, de-Bruijn net, Hypertree [9,37,40] and Hyper de-Bruijn network [37]. Some of the most successful networks are based on the Hypercube.

The Hypercube represents a class of message-passing architectures using cube (or exchange) interconnection topology. Hypercube networks are some of the first and most successful commercial multiprocessors. Each node in this network is connected through bidirectional, asynchronous point-to-point communication channels to n other nodes. The first Hypercube system was built at Caltech in the early 1980's as an experimental parallel computer for scientific numeric computation [22,58,61,71].

2.2.3.1 Hypercube topology

A Hypercube multiprocessor consists of 2^n processors. Consecutively numbered with binary integers using a string of n bits. Each processor is connected to every other processor whose binary number differs from its own by exactly one bit. The connection scheme places the processors at the vertices of an n -dimensional cube. Hypercube interconnection networks for n nodes varying from 1 to 4 are shown in Figure 2.4. The Hypercube has the property that it can be defined inductively. A

Hypercube of order 0 is a single node, and the Hypercube of the order $n+1$ is constructed by taking two Hypercubes of order n and connecting their respective nodes. Some important properties of this interconnection used in parallel processing, are given below [10,20,37,89] :

- 1) As the number of processor increases, the number of connection wires and related hardware (such as ports) increases only logarithmically, so that the systems with a very large number of processors become feasible.
- 2) A Hypercube is a super set of other interconnection networks such as rings, multistage cube network, trees etc. because these can be embedded into a Hypercube by ignoring some Hypercube connections [10,20,89].
- 3) Hypercubes are scalable - a property that results directly from the fact that Hypercube interconnections can be defined recursively.
- 4) Hypercubes have simple routing schemes. A message-routing policy may send a message to the neighbour whose binary tag agrees with the tag of the final destination in the next bit position, with the bits scanned in some order. The path length for sending a message between any two nodes is exactly the number of bits in which their tag bits differ. Numerous possible paths connecting any two nodes exist; this redundancy can be used to enhance the communication bandwidth and the fault tolerance of the Hypercube.

Hypercube nodes are usually identical. However, they do not have to be identical as long as their message-routing protocols are the same. In a heterogeneous system, some nodes may have special I/O or processing capabilities [10,46-48,51,58,59,85].

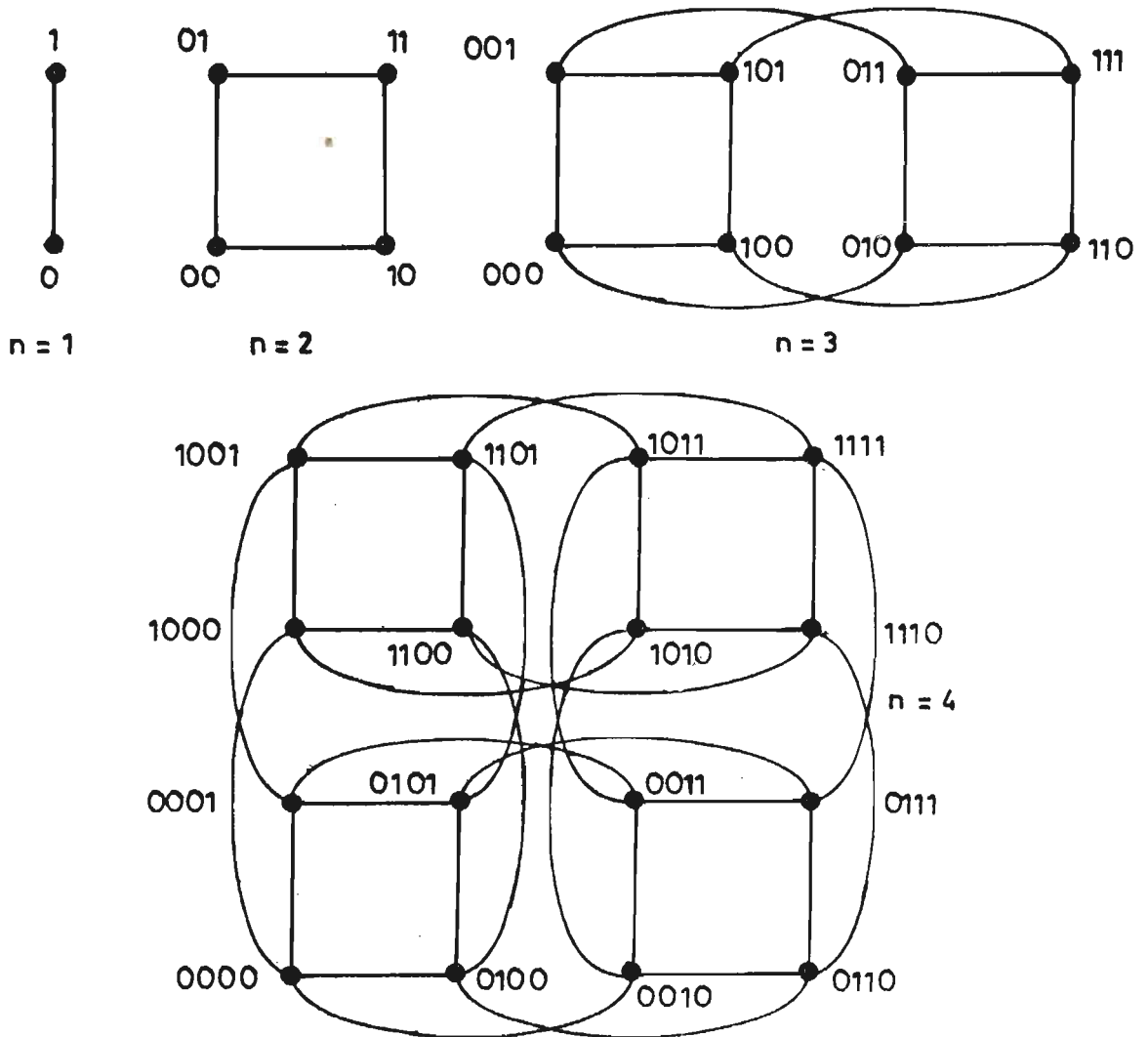


Figure 2.4 Hypercube interconnections

Hypercube computers have been contemplated since the appearance of microprocessors in 1970's. The first Hypercube multiprocessor was built and used at Caltech in 1983. Since then, interest in this type of parallel computer has grown so fast that today there are several companies offering a wide range of Hypercubes largely classified under two generation of Hypercubes. The machine in the first generation

include the Caltech Cosmic Cube, the Intel iPSC 1, the AMETEK System 14, the NCUBE/ten and the Floating Point Systems T Series. Whereas the second generation of Hypercubes began with the advent of some more powerful message-passing schemes in 1988. Representative machines of this generation are the Intel iPSC 2 and AMETEK 2010 series. The main characteristics of this generation are: 1) 32-bit processors with integrated floating-point accelerators, 2) large node memory facilitated by the availability of megabyte-chip RAM technology, and 3) message routing that is performed in hardware and become invisible to programmer [58,61,71]. The elimination of software overhead for message-passing, coupled with the technological improvements in CPU and memory, has improved the node performance by one or two orders of magnitude.

The disadvantage of this network from requirements point of view, is that it is not truly expansible. Whenever the number of nodes grows beyond a power of two, all nodes have to be changed since they have to be provided with an additional port. Thus, the module of this network is not constant predefinable building block. Moreover, a useful expansion of this structure has to occur by doubling the number of nodes.

2.2.4 Basic concept of tree

A tree is an acyclic connected graph [32,70]. The following theorem summarizes the basic properties of trees:

Let $G(V,E)$ be a graph. Then G is a tree if and only if one of the following properties holds:

- 1) G is connected and $|E| = |V| - 1$,
- 2) G is acyclic and $|E| = |V| - 1$,
- 3) There exists a unique path between every pair of vertices in G .

The proof of this theorem is straight forward. An end point of a tree is a tree vertex of degree one. A nontrivial tree has from 2 to $|V| - 1$ end points. A center of a tree is a tree vertex of minimum eccentricity. A tree has exactly 1 or 2 centers. A star is a tree of diameter 1 or 2. An arbitrary acyclic graph is called forest.

A rooted tree is a tree in which a distinguish vertex V , call^{no} has the root. The level of a vertex is defined as: vertices at a distance i from the root lie at level $(i+1)$; V itself lies at level 1. The height of the rooted tree is defined as its maximum level.

We call onward a digraph a tree if its underlying undirected graph is a tree in the sense; a vertex V is a root of a digraph G if there are directed paths from V to every other vertex in G . A digraph is a directed tree and contains a root; a vertex of outdegree zero is an end point in a directed tree.

An ordered tree is a directed tree in which the set of children of each vertex is ordered. A binary tree is an ordered tree in which no vertex has more than two children. One of the children is called the left child, while the other is called a right child. The sub tree rooted at the left child of V is called the left sub tree of V , and the one at the right is the right sub tree of V . In a complete binary tree, every vertex has either two children or none. In a balanced complete binary tree, every end point has the same level.

An N -ary tree is a generalization of binary trees where we allow each vertex to have as many as N ordered children. In a complete balanced N -ary tree, every end point has the same level.

2.2.4.1 Hypertree network

A hypertree network combines the best features of the binary tree and the n-dimensional Hypercube [9,40,42]. The basic skeleton of Hypertree is a binary tree structure, and as in the half-ring or full-ring structures, additional links in Hypertree are horizontal and connect nodes which lie in the same level of the tree. In particular, they are chosen to be a set of n-cube connections, connecting nodes which differ by only one bit in their address, shown in Figure 2.5.

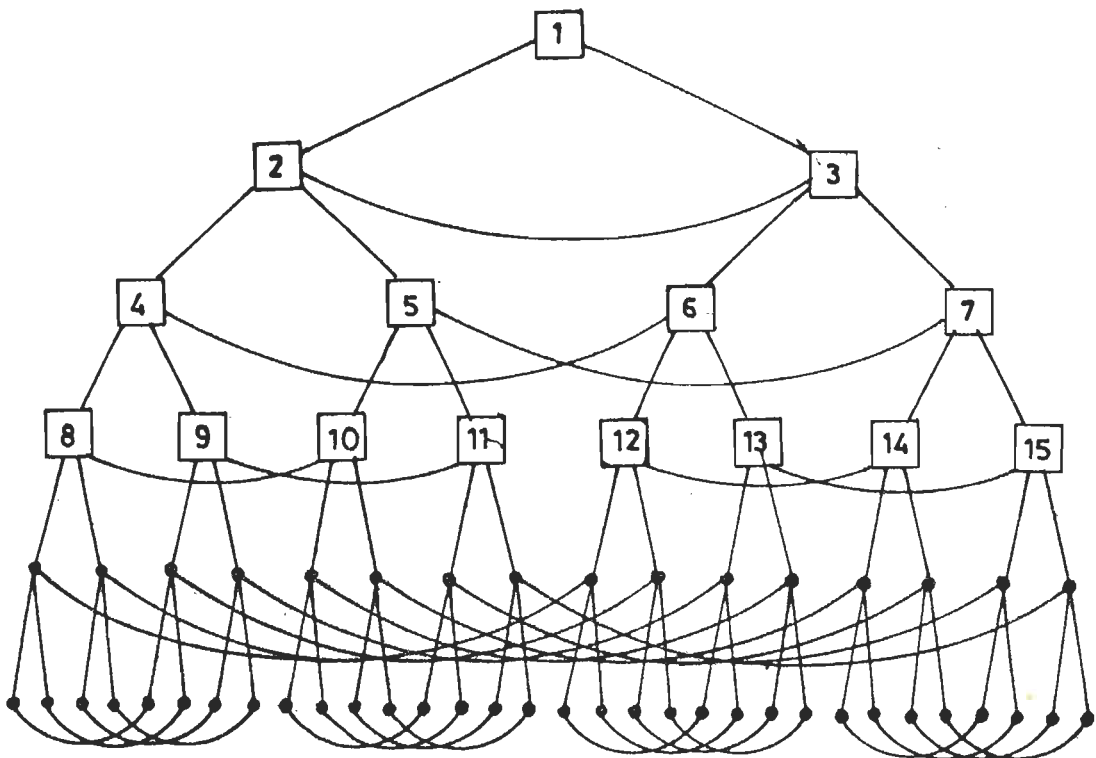


Figure 2.5 Hypertree interconnections

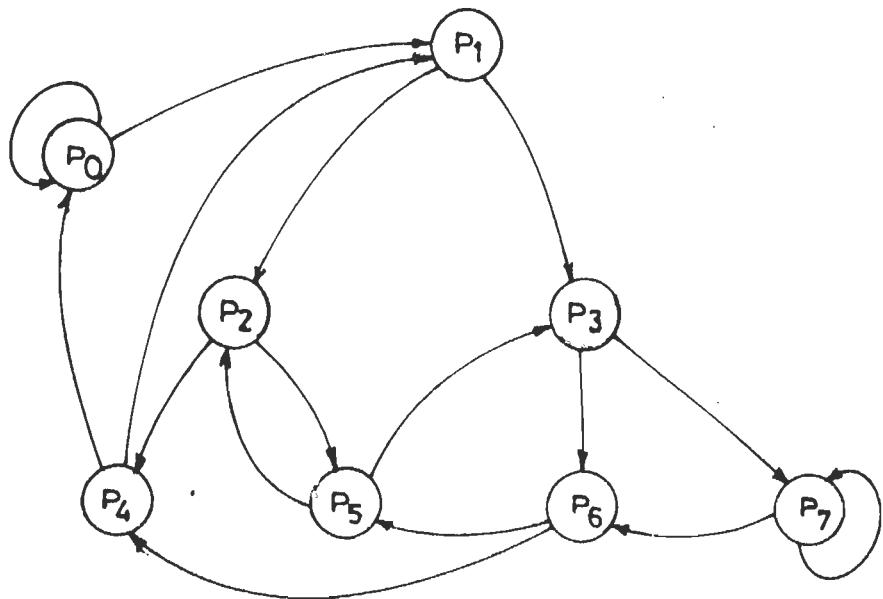
The two underlying structures permit two distinct logical views of the system. Problems which map particularly nicely onto a tree structure can take advantage of the binary tree, while those that can use the symmetry of the n-cube can be assigned to the processors in a way that

efficiently uses the n -cube links. The regular structure allows the implementation of simple routing algorithms, which require no detailed knowledge of network interconnections. With relatively small additional overhead, a routing algorithm can be constructed that is robust enough, so that messages will arrive at the proper node even for grossly unbalanced trees or in the presence of failing nodes or links. This is a requirement for easy expansibility of the system and for graceful degradation in the presence of communication hardware failures. The network is readily expansible in an incremental way. All nodes have a fix number of ports regardless of the size of the network.

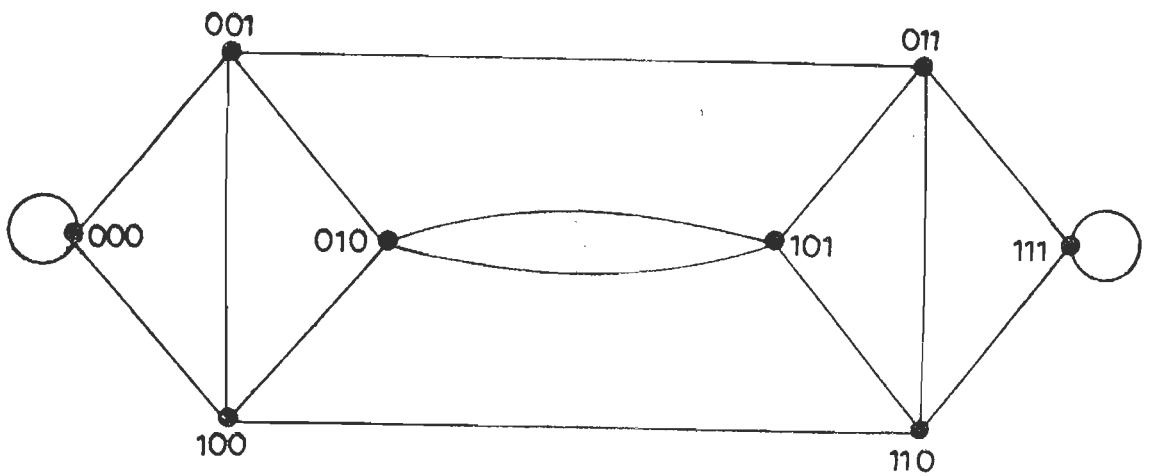
2.2.4.2 *Binary de-Bruijn Multiprocessor (BDM) network*

The de-Bruijn interconnection network has the property of a binary tree folding onto itself giving the appearance of infinite depth [82,83,90]. Let Q be a set of N identical processors, the topology then can be defined in terms of two functions L and R , which map Q into Q . The function L and R establish links from the processor P to the left and right child processors $L(P)$ and $R(P)$, respectively, for each processor P in Q . It turns out, as a result of the above definition, that each processor has a connectivity of four in the network depicted in Figure 2.6.

The network possesses a versatile topology. It admits an N -node linear array, an N -node ring, $(N-1)$ node complete binary trees etc. The de-Bruijn networks are proven to be fault tolerant as well as extensible.



Interconnection scheme for eight processors



Undirected binary deBruijn graph

Figure 2.6 Binary de-Bruijn interconnections

2.2.4.3 Hyper de-Bruijn network

The new class of graph called the Hyper de-Bruijn graph [37] have the following properties:

- 1) These networks allows the construction of large networks with any desired degree. In particular, 2^n node network (for $n \geq 4$) can be designed to have any degree in the range of 4 to n .
- 2) These networks admit simple routing algorithms. They are flexible and fault tolerant, scalable and partitionable.
- 3) These networks have many computationally important topologies as sub networks, shown in Figure 2.7.

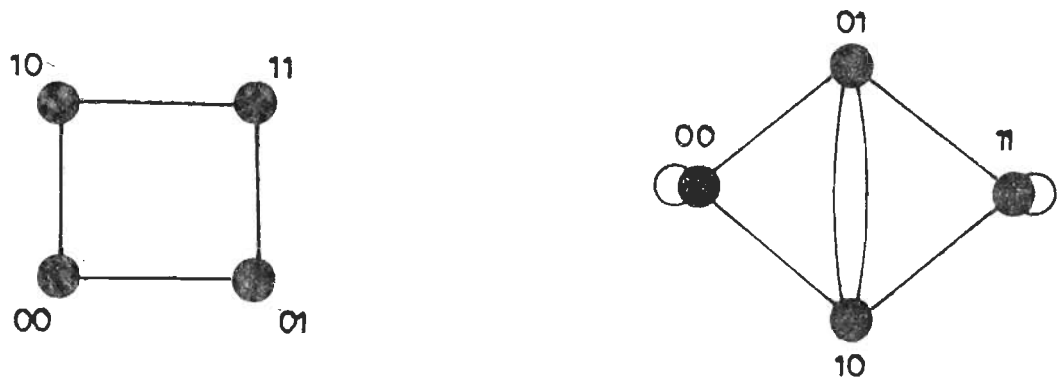


Figure 2.7 Hyper de-Bruijn interconnections

2.3 PROPERTIES OF AN INTERCONNECTION NETWORK

The selection of right topological layout for processing elements requires the incorporation of some important topological properties. These properties are listed below [37,40,47] :

1) *Worst case distances* : One important measure of the power of an interconnection network is the distance that message must travel in the network. It is advantageous to make this parameter as low as possible, since it will not only reduce traveling time for messages but also minimize message density in the links.

2) *Average distance* : Of even more importance may be the average path length travelled by all messages. In order to obtain a meaningful comparison between different networks, some normalization has to be made, in particular if networks between processors with different numbers of ports per node are considered. With no limits on the number of ports, a fully interconnected network could be designed, which would lead to an average distance of one. For a constant communication bandwidth per node, the bandwidth available through each port is then B/p , where p is the number of ports and B is the total bandwidth available from that processor [40].

Another factor influencing the average message path length is the distribution of pairs of communicating nodes. In the absence of any specific information about the communication patterns required by a particular task, one might assume a uniform distribution in which all nodes send messages with equal probability to all other nodes.

3) *Routing algorithms* : One of the desirable requirements for a large network of processors is that messages can be routed by each intermediate processor without total knowledge of all the details of the network, since the storage of that information within each node can use up an exorbitant amount of memory space. The requirement that this

information be resident at each node also enormously complicates the process of modifying the network.

4) *Message density* : Another major goal in the design of an efficient network topology is to distribute traffic as evenly as possible over all existing links.

5) *Expansibility* : Among the important parameters of the network are its modularity and expansibility, and specifically the smallest increment by which the system can be expanded in a useful way. It is generally unreasonable to demand that a system must remain balanced in all stages of expansion. Since this may imply that its size must be increased in large steps, i.e. powers of two.

6) *Fault tolerance* : The requirements for fault tolerance have greatly increased in recent years as systems have become increasingly complex. Certainly, an important feature of structures is that it must continue to work correctly, although perhaps with reduced performance, when one or more components have failed. Specifically, we expect such a system to continue to operate properly in the presence of failure of a single link or even a single node with all its attached links, as long as that particular node is not involved in the computation, i.e. the node is neither the source nor the destination for any messages [40,42].

When problem structure is such that it recursively decomposes itself into identical subproblems, it is natural to select an interconnection network which can be viewed as a virtual tree structure of arbitrary depth.

In addition to designing an appropriate network, the efficient management of parallelism on an interconnection network involves optimizing conflicting performance indices, like the minimization of communication and scheduling overheads and uniform distribution of load among the processors. Such issues are addressed at the organisational level by appropriate scheduling mechanisms. An introduction and a review of the scheduling schemes follows in the next section.

2.4 REVIEW OF SCHEDULING SCHEMES

2.4.1 Introduction

In response to ever-growing need for speeding up computationally intensive tasks, a number of parallel computer architectures, where several processing elements are connected by an interconnection network, have been proposed. Most of these architectures may be classified into : (a) *Dedicated architectures* which aim at maximizing the achievable performance for a particular task or a class of similar tasks. There usually exists relatively little room for optimizing the assignment of decomposed sub tasks on these architectures. and (b) *General purpose architectures*, which provide a good average performance for a broad range of tasks. Therefore, scheduling becomes an important problem for such type of architectures, since it has a substantial effect on the system performance and utilization [14-17,25,57]. To determine which task module of a parallel program has to be executed on which processor of a multiprocessor system, so as to minimize the total execution time of the program is a classical multiprocessor scheduling problem [23,29,30,36,77].

The scheduling problem assumes a set of processors and a set of tasks (jobs) which are to be serviced by these processors according to a specific policy. Depending on the nature and constraints of the tasks as well as of the processors, the problem is to find an efficient policy for managing the access to and alternatively the use of processors by various tasks to optimize some desired performance measure. To effectively exploit a more powerful computing facility of a multiprocessor, the architecture of the processor network and the problem structure both should be studied separately for the consideration of scheduling policy.

The basic scheduling theory, classification and existing scheduling schemes are now described.

2.4.2 Definitions

The core of all the efforts to exploit the potential power of distributed computation are the issues related to the management and allocation of system resources relative to the computational load of the system. The general scheduling problem has been described a number of times and in a number of different ways [28,39,59,100] and is usually a restatement of the classical notions of job sequencing in the study of production management. For the purposes of distributed process scheduling, the broader view of the scheduling function is taken as a *resource management* which is basically a mechanism or policy used to efficiently and effectively manage the access to and use of a resource by its various consumers. Hence, the scheduling problem may be viewed as consisting of the following three main components :

- 1) Consumer(s).
- 2) Resource(s).
- 3) Scheduler and Policy.

One can observe the behavior of the scheduler in terms of how the *policy* affects the *resources* and *consumers*. The scheduler may be viewed in terms of how it affects either or both resources and consumers. This relationship between the scheduler, policies, consumers, and resources is show in Figure 2.8.

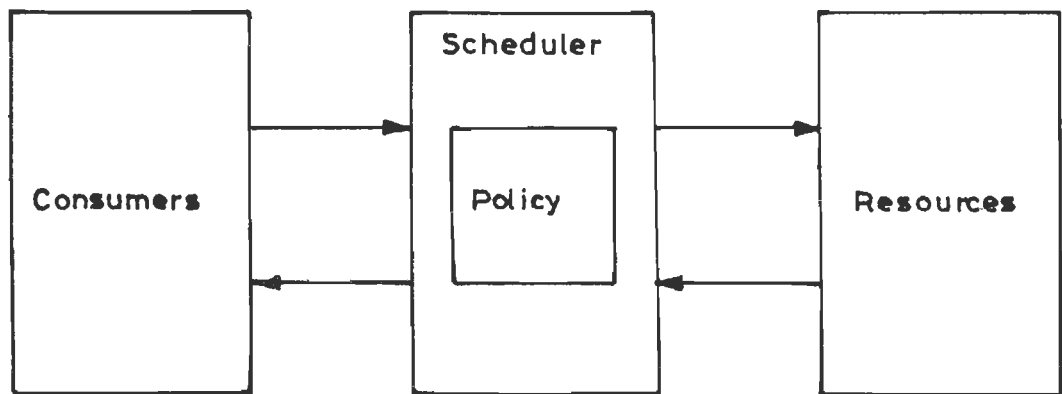


Figure 2.8 Scheduling system

One by-product of the general scheduling problem is the unification of the two terms in common use in the literature [10,23,53,59,84]. There is often an implicit distinction between the terms *scheduling* and *allocation*. However, it can be argued that these are merely alternative formulations of the same problem, with allocation posed in terms of resource allocation (from the resources point of view), and scheduling viewed from the consumers point of view. Thus, allocation and

scheduling are merely two terms describing the same general mechanism, but described from different view points.

When considering the decision-making policy of a scheduling system, there are two fundamental components: responsibility and authority. When responsibility for making and carrying out policy decisions is shared among the entities in a distributed system, we say that the scheduler is *distributed*. When authority is distributed to the entities of a resource management system, we call this *decentralized*. This differentiation exists in many other organization structures. Any system which possesses decentralized authority must have distributed responsibility, but it is possible to allocate responsibility for gathering information and carrying out policy decisions, without giving the authority to change past or make future decisions.

The terms *dynamic scheduling* and *adaptive scheduling* are quite often attached to various proposed algorithms in the literature [25,59,84] but there appears to be some confusion as to the actual difference between these two concepts. The more common property to find in a scheduler (or resource management subsystem) is the dynamic property. In dynamic situation, the scheduler takes into account the current state of affairs as it perceives them in the system. This is done during the normal operation of the system under a dynamic and unpredictable load. In an adaptive system, the scheduling policy itself reflects changes in its environment. The difference here is one of level in the hierarchical solution to the scheduling problem. Whereas a dynamic solution takes environmental inputs into account when making its decisions, an adaptive solution takes environmental stimuli into account to modify the scheduling policy itself.

2.4.3 *The classification scheme*

The usefulness of the four-category taxonomy of computer architecture presented by Flynn [23] has been well demonstrated by the ability to compare the systems through their relation to that taxonomy. The goal of the taxonomy given here is to provide a commonly accepted set of terms and to provide a mechanism to allow comparison of past work in the area of distributed scheduling in a qualitative way. In addition, it is hoped that the categories and their relationships to each other have been chosen carefully enough to indicate areas in need of future work as well as to help classify future work.

The taxonomy will be kept as small as possible by proceeding in a hierarchical fashion for as long as possible, but some choices of characteristics may be made independent of previous design choices, and thus will be specified as a set of descriptors from which a subset may be chosen. The taxonomy, while discussed and presented in terms of distributed process scheduling, is applicable to a larger set of resources. In fact, the taxonomy could usefully be employed to classify any set of resource management systems. However, the attention is focused on the area of process management since it is in this area which we hope to drive relationships useful in determining potential areas for future work.

2.4.3.1 *Hierarchical classification*

The structure of the hierarchical portion of the taxonomy is shown in Figure 2.9. A discussion of the hierarchical portion then follows.

1) *Local versus Global*: At the highest level, we may distinguish between local and global scheduling. Local scheduling is involved with the

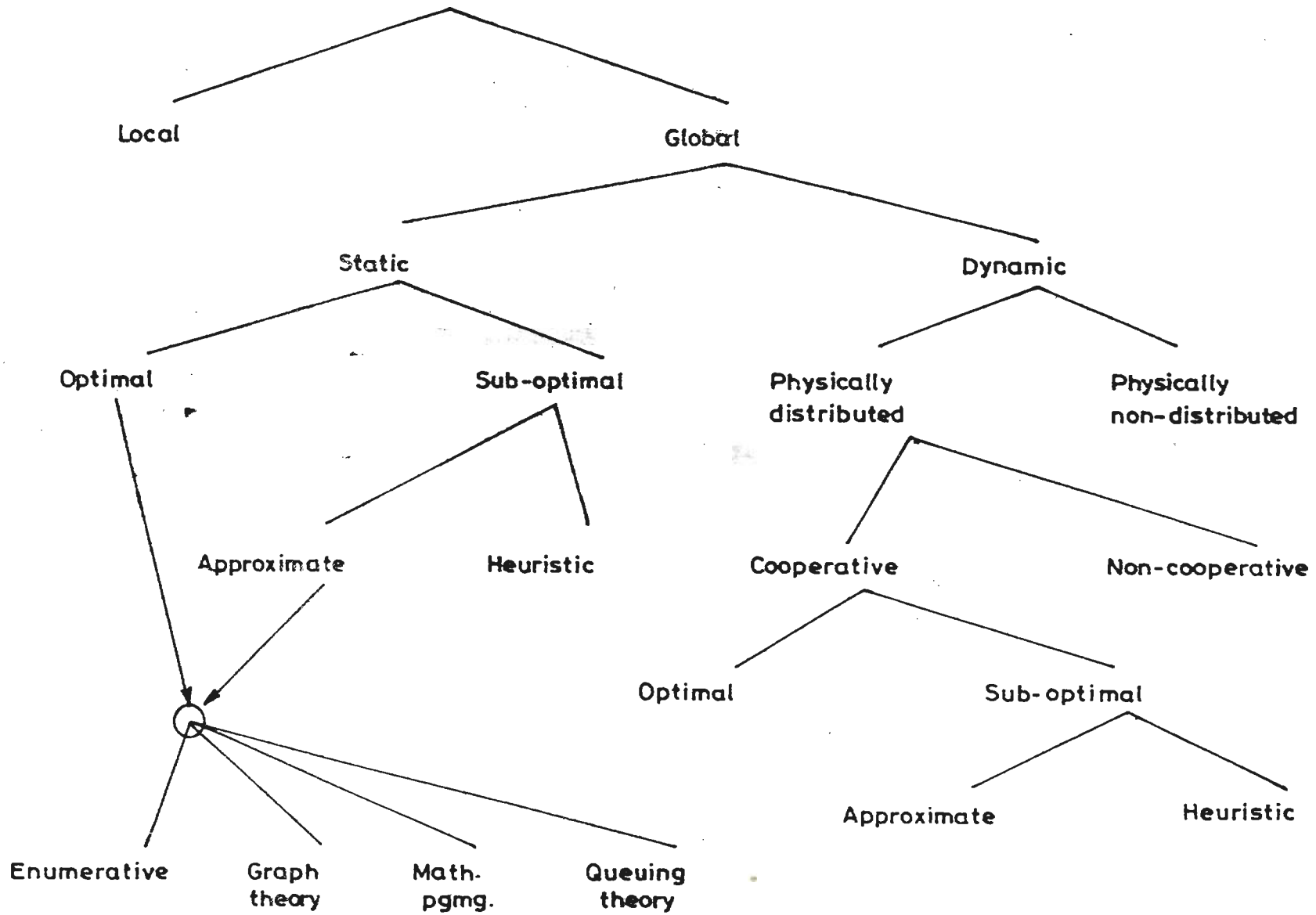


Figure 2.9 Task scheduling characteristics

assignment of processes to the time-slices of a single processor. Since the area of scheduling on single processor systems [25,28,84] as well as the area of sequencing or job-shop scheduling [20,29,53,84] has been actively studied for a number of years, this taxonomy will focus on global scheduling. Global scheduling is the problem of deciding where to execute a process, and the job of local scheduling is left to the operating system of the processor to which the process is ultimately allocated. This allows the processors in a multiprocessor environment to work with increased autonomy while reducing the responsibility (and consequently overhead) of the global scheduling mechanism. It may be noted that this does not imply that global scheduling must be done by a single central authority. Rather, the problems of local and global scheduling view as separate issues, and (at least logically) separate mechanisms are at work solving each of them.

2) *Static versus Dynamic*: The next level of hierarchy (beneath global scheduling) is a choice between static and dynamic scheduling. This choice indicates the time at which the scheduling or assignment decisions are made.

a) *Static scheduling* : In the case of static scheduling, information regarding the total mix of processes in the system as well as all the independent subtasks involved in a job or task force, is assumed to be available by the time the program object modules are linked into load modules. Hence, each executable image in a system has a static assignment to a particular processor, and each time that process image is submitted for execution, it is assigned to that processor. A more relaxed definition of static scheduling may include algorithms that

schedule task forces for a particular hardware configuration. Over a period of time, the topology of the system may change, but characteristics describing the task force remain the same. Hence, the scheduler may generate a new assignment of processes to processors to serve as the schedule until the topology changes again. This can be further subdivided into the following:

Optimal versus Sub-optimal: In the case that all information regarding the state of the system as well as the resource needs of a process are known, an optimal assignment can be made based on some criterion function [10,15,84]. Examples of optimization measure are minimizing total process completion time, maximizing utilization of resources in the system, or maximizing system throughput. In the event that these problems are computationally not feasible, sub-optimal solutions may be tried [66,84]. Within the realm of sub-optimal solutions to the scheduling problem, we may think of two general categories.

Approximate versus Heuristic: In approximate scheduling we use the same formal computational model for the algorithm, but instead of searching the entire solution space for an optimal solution, one is satisfied when we find a *good* one. These solutions are categorized as suboptimal-approximate. The assumption that a good solution can be recognized may not be so significant, but in the cases where the metric is available for evaluating a solution, this technique can be used to decrease the time taken to find an acceptable solution (schedule).

The second branch beneath the sub-optimal category is labeled heuristic [23,84]. This branch represents the category of static

algorithms which make the most realistic assumptions about *a priori* knowledge of the concerned process and system loading characteristics. It also represents the solution to the static scheduling problem which require the most reasonable amount of time and other system resources to perform their function. The most distinguishing feature of heuristic schedulers is that they make use of special parameters which affect the system in indirect ways. Often, the parameter being monitored is correlated to system performance in an indirect way rather than a direct way, and this alternate parameter is much simpler to monitor or calculate.

Optimal versus Sub-optimal Approximate techniques : Regardless of whether a static solution is optimal or sub-optimal approximate, there are four basic categories of task allocation algorithms which can be used to arrive at an assignment of processes to processors :

- 1) Solution space enumeration and search [93]
- 2) Graph theoretic [42,69,97]
- 3) Mathematical programming [14,17,67,83]
- 4) Queuing theoretic [17,18].

The graph theoretic technique uses a graph to represent a task, and applies the minimal-cut algorithm to the graph to get the task assignment with minimum interprocessor communication. The mathematical programming approach formulates task assignment as an optimization problem, and solves it with mathematical programming techniques.

b) *Dynamic scheduling*: In the dynamic scheduling problem, the more realistic assumption is made that very little *a priori* knowledge is

available about the resource needs of a process. It is also unknown that in what environment the process will be executed during its lifetime. In the static case, a decision is made for a process image before it is ever executed, while in the dynamic case no decision is made until a process begins its life in the dynamic environment of the system. Since it is the responsibility of the running system to decide where a process is to execute, it is only natural to next ask where the decision itself is to be made.

Distributed versus Non-distributed: Dynamic scheduling is further subdivided into distributed and non-distributed scheduling. In distributed scheduling the work involved in making decisions is physically distributed among the processors [22,23,54,84], whereas in non-distributed scheduling the responsibility for the task of global dynamic scheduling physically resides on a single processor [2,71,77,87]. Here the concern is with the logical authority of the decision-making process.

Cooperative versus Non-cooperative: Within the realm of distributed dynamic global scheduling, we may also distinguish between those mechanisms which involve cooperation between the distributed components (cooperative) and those in which the individual processors make decisions independent of the actions of the other processors (non-cooperative). Here the degree of autonomy is the parameter which each processor has in determining how its own resources should be used. In the non-cooperative case individual processors act alone as autonomous entities and arrive at decisions regarding the use of their

resources independent of the effect of their decision on the rest of the system. In the cooperative case each processor has the responsibility to carry out its own portion of the scheduling task, but all processors are working towards a common system-wide goal.

As in the static case, the taxonomy tree has reached a point where we may consider optimal, sub-optimal approximate, and sub-optimal heuristic solutions. The same discussion as was presented above for static case applies here as well.

In addition to the hierarchical portion of taxonomy already discussed, there are a number of other distinguishing characteristics which scheduling systems may have. The following sections which do not fit uniquely under any particular branch of the tree-structured taxonomy given so far, are presented here as a flat extension to the scheme, but are still important in the way that they describe the behavior of the scheduler. The placement of these characteristics near the bottom of the tree is not intended to be an indication of their relative importance or any other relation to other categories of the hierarchical portion. Their position are primarily to reduce the size of the description of the taxonomy.

2.4.3.2 Flat classification characteristics

1) *Adaptive versus Non-adaptive*: An adaptive solution to the scheduling problem is one in which the algorithms and parameters used to implement the scheduling policy change dynamically according to the previous and current behavior of the system in response to previous decisions made by the scheduling system. In contrast to an adaptive scheduler, a non-adaptive scheduler would be one which does not necessarily modify

its basic control mechanism on the basis of the history of the system activity. An example of the nonadaptive scheduler would be the one which always weighs its inputs in the same way regardless of the history of the systems behavior.

2) *Load Balancing*: This category of policies, which has received a great deal of attention recently [10,11,16,26,36,44,60,65], approaches the problem with the philosophy that being fair to the hardware resources of the system is good for the users of that system. The basic idea is to attempt to balance the load on all processors in such a way as to allow progress by all processes on all nodes to proceed at approximately the same rate. This solution is more effective when the nodes of a system are homogeneous since this allows all nodes to know a great deal about the structure of the other nodes. Normally, information would be passed about the network periodically or on demand [6,15,41,56,89], in order to allow all nodes to obtain a local estimate concerning the global state of the system. Then the nodes act together in order to remove work from heavily loaded nodes and place it at lightly loaded nodes. This is a class of solution which relies heavily on the assumption that the information at each node is quite accurate in order to prevent processes from endlessly being circulated about the system without making much progress. Another concern here is deciding on the basic unit used to measure the load on individual nodes.

3) *Bidding*: In this class of policy mechanisms, a basic protocol framework exists which describes the way in which the processes are assigned to processors. The resulting scheduler is one which is usually

cooperative in the sense that enough information is exchanged (between nodes with tasks to execute and nodes which may be able to execute tasks) so that an assignment of tasks to processors can be made which is beneficial to all nodes in the system as a whole. A wide variety of possibilities exist concerning the type and amount of information exchanged in order to make decisions [1,18,23,84]. Each node in the network is responsible for two roles with respect to the bidding process: *manager* and *contractor*. The manager represents the task in need of a location to execute, and the contractor represents a node which is able to do work for other nodes. A single node may take on both of these roles, and that there are no nodes which are strictly managers or contractors alone. The manager announces the existence of a task in need of execution by a task announcement, then receives bids from the other nodes (contractors). The amount and type of information exchanged are the major factors in determining the effectiveness and performance of a scheduler employing the notion of bidding. A very important feature of this class of schedulers is that all nodes generally have full autonomy.

4) *Probabilistic*: This classification has existed in scheduling systems for some time [6,29,41,50,56]. The basic idea for this scheme is motivated by the fact that in many assignment problems the number of permutations of the available work and the number of mappings to processors so large, that in order to analytically examine the entire solution space would require extremely large amount of time. One of the solutions is to randomly choose some process as the next to assign. Repeatedly using this method, a number of different schedules may be generated, and then this set is analyzed to choose the best from among

those randomly generated. An alternative view of probabilistic schedulers are those which employ the principles of decision theory in the form of team theory [23,84]. These would be classified as probabilistic.

5) *One-Time Assignment versus Dynamic Reassignment*: In this classification, we consider the entities to be scheduled. If the entities are *jobs* in the traditional batch processing sense of the term [17,20,23], then we consider the single point in time in which a decision is made as to where and when the job is to execute. While this technique technically corresponds to a dynamic approach, it is also static in the sense that once a decision is made to place and execute a job, no further decisions are made concerning the job. We would characterize this class as one-time assignment. In this mechanism, the only information usable by the scheduler to make its decision is the information given to it by the user or submitter of the job.

In contrast, solutions in the dynamic reassignment class try to improve on earlier decisions by using information on smaller computation units the executing subtasks of jobs or task forces. This adaptation takes the form of migrating processes (including current process state information).

The general scheduling problem consist of efficiently scheduling or assigning program (tasks) to the processors in a multiprocessor system. The performance of the system depends basically on mapping of the tasks to processors and then distribution of the tasks to each processor (load balancing). Therefore, these scheduling problems can be classified according to the structure of the program (problem structure) and the

architecture of the processors (topology of the network) to which they are scheduled. There are different problem structures and various architectures reported so far. Some important structures and architectures are discussed here.

2.5 PROBLEM STRUCTURES

The various configurations involved in the problem structure are [42,69,72], shown in Figure 2.10, is listed below.

- (a) chain structure
- (b) ring type
- (c) up-rooted and down rooted tree
- (d) mesh type and
- (e) precedence type.

These problem structures may be again different in nature which may be the following :

- (1) tasks having different execution time.
- (2) number of tasks are arbitrary.
- (3) inter-task data transfer are difficult and non-negligible and
- (4) task precedence relations are not constrained.

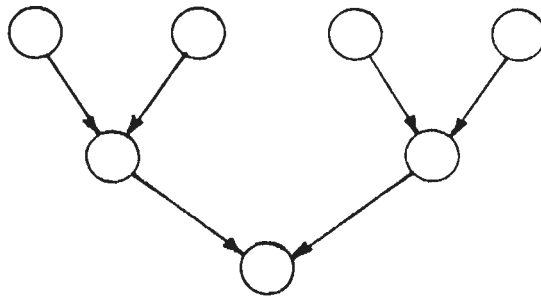
2.6 MULTIPROCESSOR ARCHITECTURES

The various topologies of the processors (system architecture) upon which scheduling of various problem structures have been studied are shown in Figure 2.11, and are discussed below.

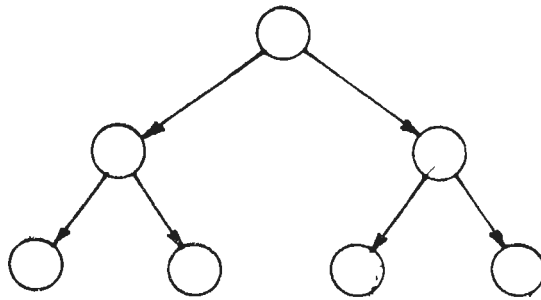
Regular networks are divided into static and dynamic networks. In static network topologies, the example networks [34] are shown in Figure 2.12.



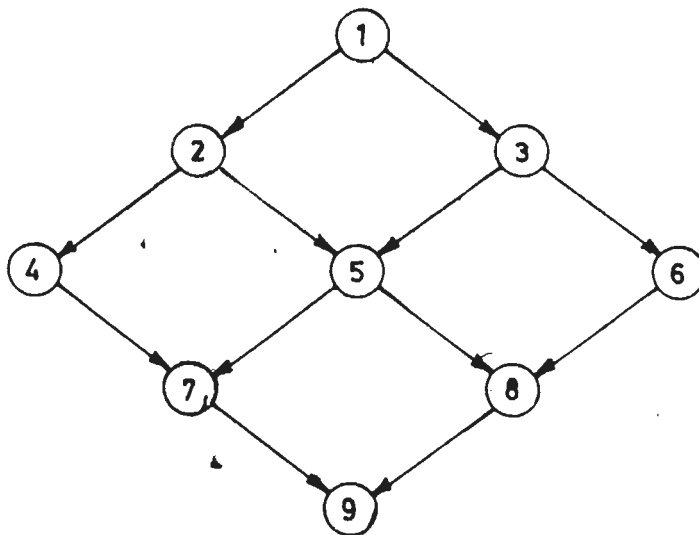
(a) LINEAR CHAIN



(b) DOWN-ROOTED TREE



(c) UP-ROOTED TREE



(d) MESH

Figure 2.10 Various problem structures

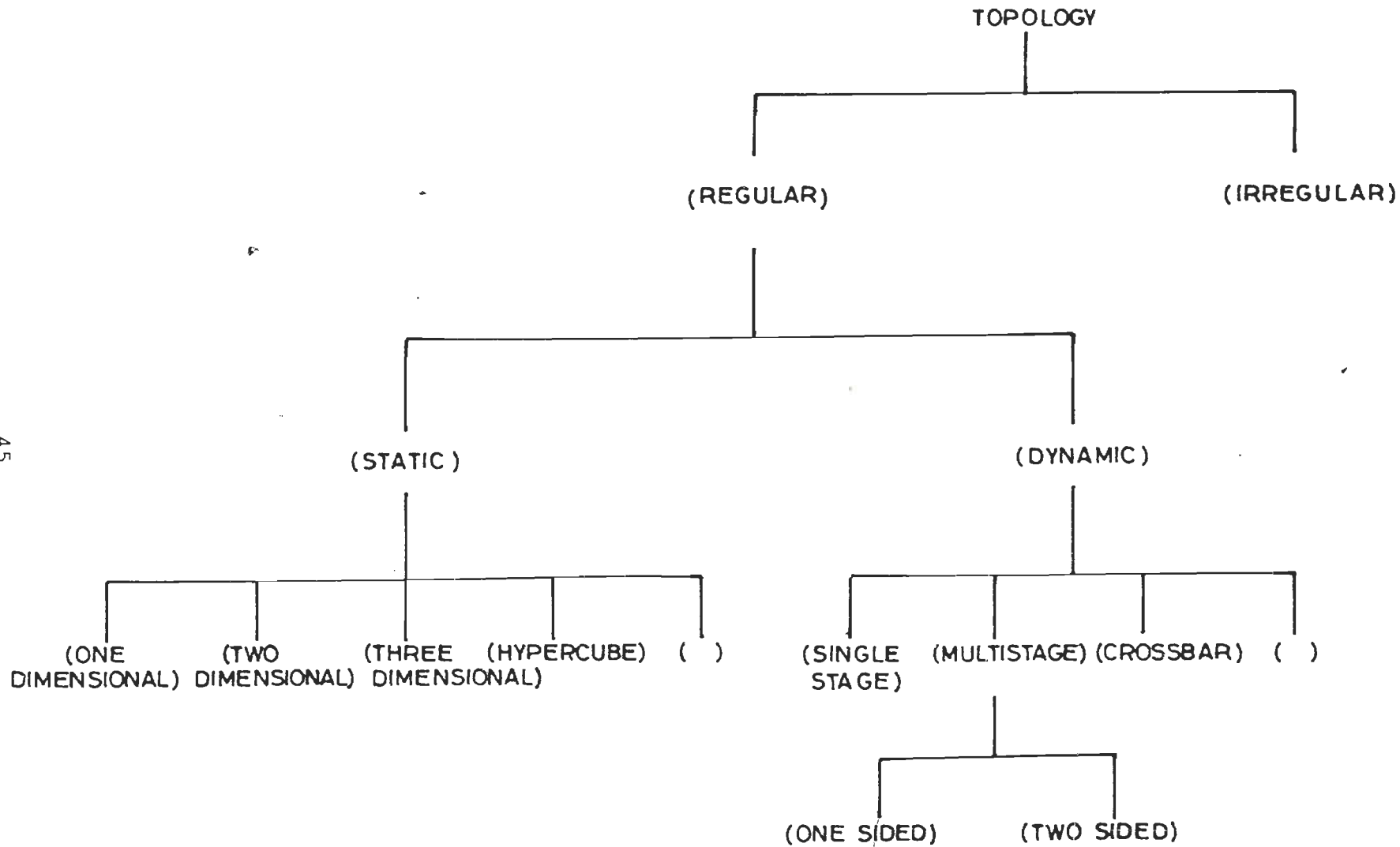


Figure 2.11 Various processor topologies

- * Linear array
- * Ring
- * Star
- * Tree
- * Near-neighbor mesh
- * Systolic array
- * Completely connected
- * Chordal ring
- * 3-cube
- * 3 cube connected cycle etc.

In dynamic network topologies, the example networks are:

- * Single stage
- * Multistage
- * Cross bar etc.

Among the better known networks, on which much work has been done in particular to allow multiple, simultaneous connections between processor banks and memory banks to permit sharing of data or concurrent cooperation on the same tasks, are lattice structure, the flip net, the omega net, the indirect n-cube, the perfect shuffle, the augmented data manipulator, the deBruijn network, the generalized connection network, the Banyan partitioner, and the n-cube network [38,40,60]. It is understood that one of the chief properties of many of these networks is the efficient interconnection of nodes in the n-dimensional Hypercube, or n-cube configuration. Connecting all the processors in a Hypercube topology, the hardware complexity increases as the network size grows, whereas in deBruijn network merits for such situations, because number

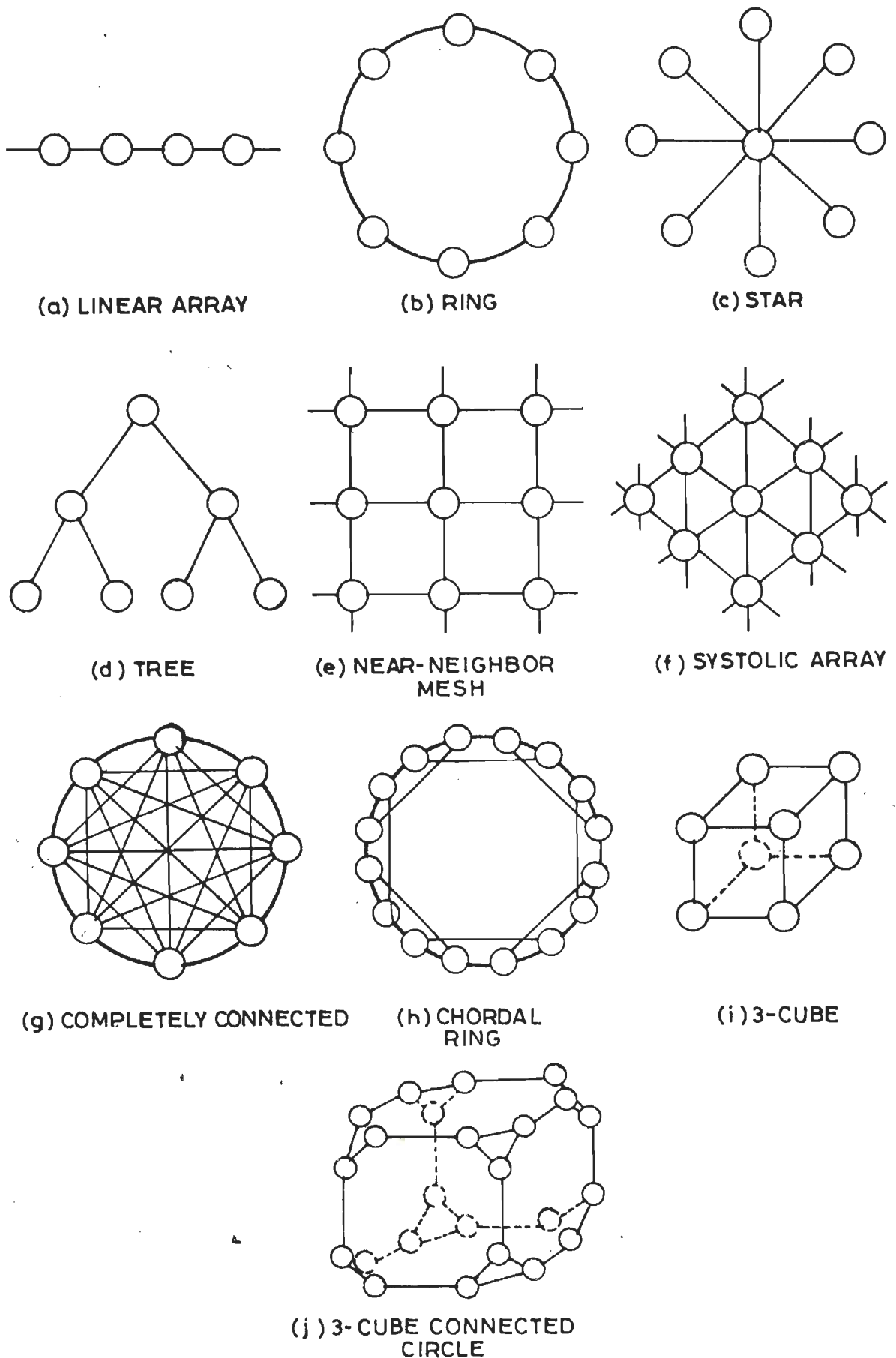


Figure 2.12 Examples of some network topologies

of connections per processor is limited to *Four*, and does not increase with the size of the network [40,72,75,79]. The n-cube is particularly compact. Whenever the number of nodes grows beyond a power of two, all nodes have to be changed since they have to be provided with an additional port. An expansion of this structure has to occur by doubling the number of nodes. An incompletely populated n-cube lacks the above mentioned properties which make it attractive in the first place and also due to the fact that it is not truly expansible. Tree structures, on the other hand, are expansible in a natural way, and even unbalanced trees still retain most of the properties that make trees attractive [38,40,42]. If the number of ports per node has to be limited, a binary tree structure, requiring only *Three* ports per node, looks particularly attractive [33,38]. For effective cooperation of several processors on the same task, or for fast access to distributed data, high communications bandwidth is typically required. If all processors are simply connected onto one and the same bus, this shared resource becomes a bottleneck preventing simultaneous communication between different pairs of processors, and the effective throughput of the system may actually go down as the number of processors is increased. A suitable interconnection network is thus needed which provides as much bandwidth as possible between any pair of processors. One possible approach is to combine one processor and its memory with one node of the switching network, thus creating a regular network.

2.7 CLASSIFICATION OF SCHEDULING ALGORITHMS

Rawlins [84] has reported the following six natural kind of algorithms.

(1) *Classical algorithm:*

- May be very slow
- Never lies
- Always stops
- Always predictable
- Doesn't use random numbers

(2) *Heuristic algorithm:*

- Always fast, if it stops
- May not solve the problem (usually this is dependent on the input and the condition is usually difficult to check)
- May not stop
- May not be predictable
- Sometimes uses random numbers.

(3) *Approximation algorithm:*

- Always fast
- Gives a near answer to the proposed problem, or the answer to a near problem
- Always stops
- Always predictable
- May use random numbers

(4) *Randomized algorithm:*

- Usually fast
- Never lies
- Always stops- Usually unpredictable (often uses sampling)
- Uses random numbers

(5) *Probabilistic algorithm:*

- Always fast
- Usually tells the truth
- Always stops
- May be unpredictable
- May use random numbers

(6) *Ergodic algorithm:*

- Always fast, if stops
- Usually tells the truth
- May not stop
- Unpredictable (each repetition is independent)
- Uses random numbers

There are so many variants involve, which are responsible for optimizing the mapping and load balancing and hence finally improve the performance of the multiprocessor system. The difficulty of solution varies with inclusion or exclusion of pre-emption, network topology, the number of parallel processors, cardinality, communication-overhead, precedence constraints etc. Surveys of the rapidly expanding area of deterministic scheduling theory and task allocation are given in [25,51,52,84].

C. Shen and W Tsai [93], used an optimal, enumerative approach to the task assignment problem. The criterion function is defined in terms of optimizing the amount of time a task will require for all interprocess communication and execution, where the task submitted by users are assumed to be broken into suitable modules before execution.

610975

The cost function is called a minimax criterion, since it is intended to minimize the maximum execution and communication time required by any single processor involved in the assignment. Graphs are then used to represent the module to processor assignments. The algorithm is used to find the optimal search of this solution space and to achieve a certain degree of processor load balancing also.

The model presented by Ma et. al. [67], considers an optimum mathematical programming formulation employing a branch and bound technique to search the solution space. The goals of the solution are to minimize interprocessor communications and balance the utilization of all processors. The model given defines a cost function which includes interprocessor communication cost and processor execution costs. The assignment is then represented by a summation of all costs incurred in the assignment. The algorithm then used to search the solution spaces (consisting of all potential assignments) is derived from the basic branch and bound technique.

In case of dynamic solutions, a more realistic assumption is made that very little a priori knowledge is available about the resource needs of a process. Unlike the static case, no decision is made for a process before it is executed [25]. A dynamic programming is applied to schedule program tasks, with no consideration to inter-task communication, on to multiprocessor system [72]. Even for two processors, this approach failed when number of tasks involved exceeds 50. Using the max-flow min-cut strategy, gives an algorithm for optimal scheduling of program modules or tasks, without precedence constraints, on to a two processor system. Extension of this approach to three or more processors does not appear to be feasible.

An efficient $O(n)$ algorithm was developed by Hu [49], where the task processing time is equal and the task graph is tree shaped. For arbitrary precedence among the tasks, then Coffman and Graham [27] presented an $O(n^2)$ algorithm for two processors. If any of these restrictions are relaxed then the problem becomes NP-hard.

A mapping strategy is proposed by Lee and Aggarwal [57], using communication overhead as an objective function to evaluate the optimality of mapping a problem graph on to a system graph assuming all program tasks to be identical and the number of tasks to be less than or equal to the number of processors in the system. The strategy in which all problem nodes are activated simultaneously i.e. in the same phase, tested using Hypercube as system graph. They tried to achieve good initial assignment and then employ a pair wise exchange method to optimize the assignment. In extension of their work, Aggarwal and Chaudhary [25], developed a generalized mapping scheme which shows the concept of pseudo processors for achieving a deadlock free mapping and reducing communication overheads for generalized system. Recently a heuristic algorithm for scheduling parallel program tasks onto arbitrary multiprocessor topology assuming non-negligible inter processor communication has been proposed. However, their algorithm does not guarantee contention-free communication.

Bokhari [14-17], describes a mapping scheme assuming no cardinality variation. The objective function is the number of edges of the problem graph that fall on the edges of the system graph. Therefore, the objective function takes into account only the matched edges (cardinality). However, the unmatched edges may, in some cases, determine the systems performance. The problem graph edges are also

assumed to be identical, although in general they could have different traffic intensities, represented as weights.

Another simplifying assumption is made in the quadratic assignment problem [43]. The objective function is the sum of products of the weights of problem edges and the distance of the corresponding system edges for all problem edges i.e. the sum of communication overheads of all problem edges, which seems to be a reasonable measure. However, this measure does not specify exactly what is to be minimized (maximized) in parallel processing application. Moreover, the actual distance of the system edges is not really independent of the problem graph unless the problem edges share none of the system edges [51]. McDowell and Appelbe [69] discuss the problem of assigning processes to the processors interconnected as a ring. The problem graphs are restricted to binary trees, and a heuristic algorithm is suggested to minimize the communication delays. A tight, necessary condition for finding assignments of program fragments to linearly connected processors that require no communication delays, is presented.

Ravikanth et al [82,83], proposed the properties of ideal performance in a multiprocessor system, to reduce communication overhead, scheduling overhead and distribution of load among processors. Selecting an interconnection network, for applicative programs, which turns out to be deBruijn graph [82]. They considered the following properties for performance enhancement :

- * Minimum distance property.
- * Static scheduling.
- * Minimum load imbalance.

Restricting to the above properties, their architecture minimizes

communication overhead for a class of applicative programs that unfolds as complete binary tree and the network achieves an ideal performance using simple static scheduling mechanism. They concluded that the performance can not be improved by dynamic policy, which is more complex and incurs higher overheads. They showed that for arbitrary tree structures, the network is able to achieve a fairly good distribution of tasks using static scheduling. The performance of the network improves as the problem size grows larger.

In modification to this scheme, Reddy [86] proposed a scheduling scheme and named it as minimum-load scheduling. In the Ravikanth's scheme, when an arbitrary tree structure is mapped onto the network, the difference in the number of tasks scheduled on left and the right processors may be more than one. This anomaly has cumulative effect of increasing the load among the processors as the size of the tree structure increases. To improve the performance, while scheduling arbitrary tree structures, a better approach would be to exchange the information with the immediate neighbours before scheduling a task. Implemented on de-Bruijn and QDM networks, this scheme shows better results for tree structured task graphs.

There have been other efforts to configure interconnection networks to execute binary tree task structures. Martin [68] proposed the twisted torus, which is modification of hyper torus. Sequin [91] further modified it to obtain the doubly twisted torus. These networks, through a static scheduling strategy, distribute binary trees assigning tasks at level 'k' of the tree unevenly among 'k+1' processors. Since the number of tasks at a level are related exponentially to the level number, these networks don't achieve optimal performance.

In another effort, Reeves et.al [103] reported five dynamic load balancing strategies which illustrate the trade off between 1) knowledge-the accuracy of each balancing decision, and 2) overhead - the amount of added processing and communication incurred by the balancing process. The Sender (Receiver) Initiated Diffusion strategies are asynchronous schemes which only uses near-neighbour information. The Hierarchical Balancing Method organizes the system into a hierarchy of subsystems within which balancing is performed independently. The Gradient Model employs a gradient map of the proximities of underloaded processors in the system to guide the migration of tasks between overloaded and underloaded processors. Finally, the Dimension Exchange Method requires a synchronization phase prior to load balancing and then balances iteratively.

Rommel [87] by using another approach, reported a general formula for the probability of unequal load distribution in the system. This probability can be used to define the likelihood of load balancing success in a distributed system. The work is divided into one regarding information passing, and a second regarding actual distributed scheduling algorithm. The approach for information passing include: sender directed, receiver directed bidding and focused addressing. The algorithm is based on the modification of Livny & Melmans probability approach which is extended to consider a generalized server [65]. For obtaining better load balance, another powerful mapping technique known as Scatter Decomposition exists, which uses probabilistic model of work load in one dimension [75]. It has been shown that scatter decomposition minimizes the average processor work load variance and thus better load balance.

Recently, Barmon et al [7] reported a dynamic load balance algorithm in a distributed computer system. Considering N identical processors connected through a reliable and bi-directional communication links. The processors maintained two Q 's for tasks, as ready Q and local Q . The algorithm distributes the tasks through a local processor or transferred for processing to a neighboring processor via a reliable communication network to the entire system. The performance of the system is measured in terms of its overall response time and the simulation results suggest that these algorithms perform well by utilizing the design parameters properly. The problem of load redistribution in a distributed system is recognized as load balancing. In their attempt, Hwa-Chun Lin and C. Raghavendra [62] proposed a dynamic load balancing policy with a central job dispatcher. The design of the policy is motivated by the operation of a single Q multiserver Queuing system and gave best result for small job transfer delays. A modified M/M/N model of queuing system gave the accurate estimate of average response time for mean job transfer delays.

Research in last decade has given a number of task scheduling algorithms for multiprocessing system including, First Come First Serve (FCFS), Shortest processor Time First (STF), Smallest Memory requirement First (SMF), Non-Scheduling (NS), Random Scheduling (RS), Arrival Balanced Scheduling (ABS), End Balanced Scheduling (EBS) and Continual Balanced Scheduling (CBS), for both, static and dynamic scheduling [12]. The above algorithms have different performance for different applications but if the number of application tasks is greater than twice the number of processors, CBS and EBS out perform the other schedulers.

From the above review, it is apparent that myriad of multiprocessor scheduling strategies exist which can be applied to specific structure of programs and specific system architectures. An optimal scheduling can be made based on some objective functions to enhance the performance of overall system [7]. The objective functions mostly reported include communication overhead and load balancing among the processors [25,57,82,83,86]. Several researchers have categorically worked on these two aspects to achieve optimal solution [82,83,86].

The scheduling schemes reported by Ravikanth et. al. [82], Reeves et. al. [103], and by Reddy [86] consider the same performance indices are implemented on the similar type of networks. In the next chapter a Linearly Extensible Tree (LET) multiprocessor network has been proposed and analysed and the properties of this network has been compared with similar networks.

LINEARLY EXTENSIBLE TREE NETWORK

The demand for higher and higher computation speed and the signs of saturation in integrated circuit technology has given a filip to the development in multiprocessor systems. The multiprocessing approach to parallelism is the most generalized and flexible one, but to a great extent its success depends on interconnection topology. To this day, the problem of interconnecting processors to achieve high computation bandwidth and scalable parallelism has not been fully solved. The choice of the topology of the interconnection network is critical in the design of massively parallel computer systems. For this reason, a plethora of interconnection network proposals have appeared in the literature, and an enormous amount of research has centered on the design and analysis of these networks [42]. Motivated from the above discussion, a new multiprocessor network named as Linearly Extensible Tree (LET) network has been proposed, which is suitable for tree-structured problem graphs. In this chapter, an analysis of the network, its various properties and a brief comparison with a similar type of network has been given.

3.1 MULTIPROCESSOR INTERCONNECTION NETWORKS

Interconnection networks are often modelled as undirected or directed graphs. The nodes of such a graph represent the processors, and the edges indicate the communication links between various processors. The length of a path between two nodes is the number of edges encountered in the path. The diameter of a network is the largest

distance between any two nodes. The degree of a network is the largest degree of all nodes in that network. Extensibility is the property which facilitates constructing large-sized systems out of small-sized systems with minimum changes in the configuration of a node of the system

A tree uses the minimum number of links in a connected network, but it has unacceptably poor communication properties. On the other hand, the fully connected network is prohibitively expensive, since the number of links grows as $O(N^2)$ for a N -node network. Between these two extremes, different families of interconnection networks exist [42]. Some of the desirable properties of interconnection networks are: high fault tolerance, small diameter, small degree, simple routing algorithms, efficient layout, high bandwidth and extensibility. Many of these properties make contradictory demands and hence a compromise is necessary in the designing of the network.

3.2 BINARY DEBRUIJN MULTIPROCESSOR (BDM) NETWORK

As the proposed network is a modification of BDM hence a description of BDM is given first. Binary deBruijn Multiprocessor (BDM) network has the property of a binary tree folding onto itself giving the appearance of infinite depth. It is also called a virtual tree network. The network may be defined as follows [82] :

Let Q be a set of N identical processors, where N is a power of 2. The interconnection between processors are governed by two functions L and R , which map Q into Q as follows :

If $Q = \{P_1, P_2, \dots, P_{N-1}\}$ then

$$L(P_i) = P_{2i \bmod N} \quad \text{and}$$

$$R(P_i) = P_{(2i+1) \bmod N} \quad \text{for all } P_i \text{ in } Q$$

The functions L and R establish links from the processor P to the processor L(P) and R(P), for each processor in Q. It turns out, as a result of the above definitions, that each processor has a connectivity of four in the network. Figure 3.1 (repeated here for convenience) shows, for instance, the interconnection for eight processors.

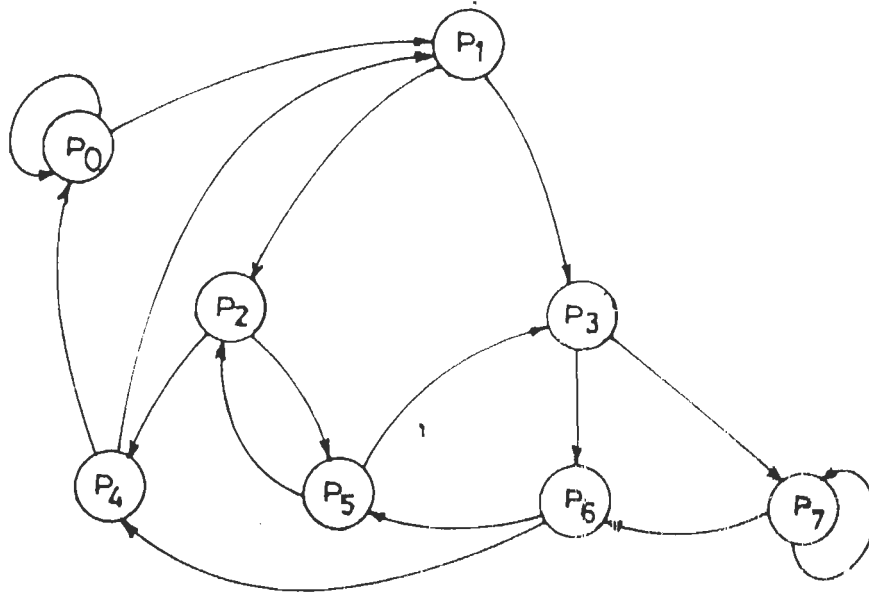


Figure 3.1 Binary deBruijn Multiprocessor (BDM) network with eight processors

The BDM possesses a versatile topology having a fixed degree per node. It is capable of high fault tolerance and can admit a number of same family networks. The performance studies on this network show that the network is highly suitable for complete binary tree type problem structures because perfect mapping of problem graph onto the network is possible.

Apart from the above mentioned properties of the BDM, it has some draw backs. The number of nodes in the network grows exponentially. An

extension in the network, like in a hypercube, requires doubling of the processor count. Thus extension complexity for n th extension is $O(2^n)$. In contrast to this, a network with linear growth complexity would be more practical, because in such a network, extensions would be less costly. A modified BDM network is proposed which is a recursively connected Linearly Extensible Tree (LET) network.

3.3 LINEARLY EXTENSIBLE TREE (LET) MULTIPROCESSOR NETWORK

3.3.1 Design and analysis

As said earlier, the LET network grows linearly in a binary tree like shape. In a binary tree, the number of nodes at a level j is 2^j , whereas in LET network the number is $(j+1)$. The network itself may be defined through connection functions in a manner similar to that for BDM.

Let Q be a set of N identical processors, represented as

$$Q = \{P_0, P_1, \dots, P_{N-1}\}$$

The number of processors N in the network is given by

$$N = \sum_{k=1}^{d+1} k \quad \dots (3.1)$$

where d is the depth of the network. For different depths, networks having 1, 3, 6, 10, 15, 21, ... processors are possible. Equation (3.1) is quadratic in d , solving which we get,

$$d = -\frac{3}{2} \pm \frac{\sqrt{(1+8N)}}{2} \quad \dots (3.2)$$

As negative depth has no meaning, the negative sign may be dropped.

Thus depth d is $O(N^{1/2})$. In a binary tree the depth is $O(\log_2 N)$. The depth of LET network grows faster than that of binary tree mainly because of lesser 'accommodation' at each level.

In order to define the link functions, we denote each processor in the set Q as P_{ij} , j being the level in LET network where the processor P_i resides. As per the LET policy, only $(j+1)$ processors exist at level j . Thus at level 0, P_0 exists and hence it may be redesignated as P_{00} . Similarly, the processors at level 1 are P_{11} and P_{21} . Figure 3.2 shows this arrangement.

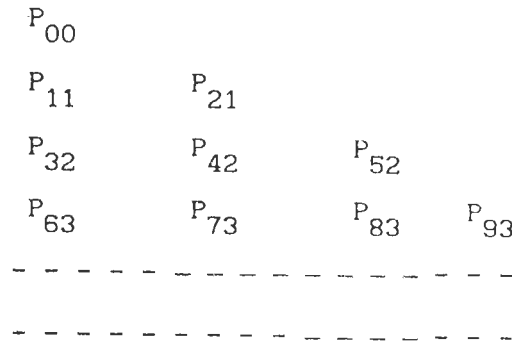


Figure 3.2 Arrangement of processors in LET

Let Q' be the set of redesignated processors of Q . Thus,

$$Q' = \{P_{ij}\}, \quad 0 \leq i \leq N-1, \quad 0 \leq j \leq d$$

The link functions L and R define a mapping from Q' to Q given as

$$\begin{aligned}
 L(P_{ij}) &= P_{(i+j+1) \bmod N}, \text{ and} \\
 R(P_{ij}) &= P_{(i+j+2) \bmod N} \text{ for all } P_{ij} \text{ in } Q' \quad \dots (3.3)
 \end{aligned}$$

The two functions in Eq. (3.3) indicate the links between various processors in the network. Figure 3.3 shows a LET network for six

processors along with its adjacency matrix. Figure 3.4 shows a bigger network for $N = 36$. To avoid cluttering of figure, the fold-back connections have been shown by dotted lines to an extra dummy level of processors. If the fold-back connections are ignored, it can be seen from Eq. (3.2), that the number of leaf nodes L in a network of N processors is given by

$$L = \frac{\sqrt{(1+8N)}}{2} - \frac{1}{2}$$

i.e. $L = d+1$. The width of the network (maximum number of nodes at a level) grows much slowly compared to that in a binary tree.

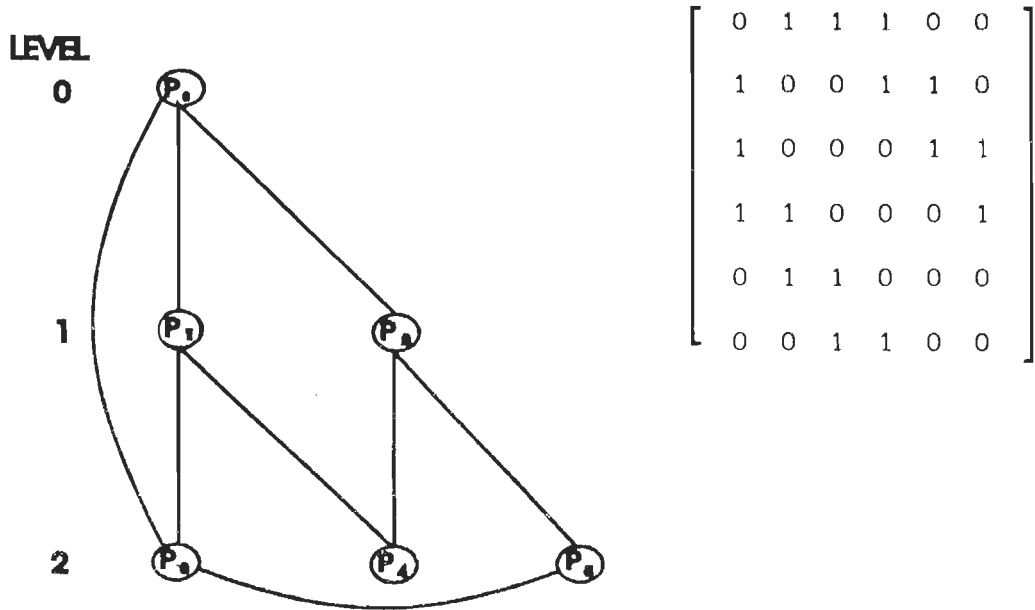


Figure 3.3 LET network with six processors

Adjacency matrix of LET

3.3.2 Properties of the LET network :

Here some properties of the LET network have been compared to de-Bruijn and Hypercube networks [37,40].

A. Number of Nodes

The number of nodes in a multiprocessor network plays a vital role by virtue of which the complexity of the system is affected. Lesser the number of node, lesser is the system complexity and it is more economical. The number of nodes in the LET network is $N = \sum_{k=1}^n k$, whereas the number of nodes in Hypercube and de-Bruijn network 2^n .

B. Degree of Node

Degree or connectivity of a node in a multiprocessor system is the number of connections required at each node. Connectivity of a node determines the hardware complexity of the network. The higher the connectivity, the higher is the hardware complexity and hence the cost of the network. A constant connectivity implies easy extensibility with minimum change in the hardware structure of each node. The degree of node in the proposed structure is always 4 or less . The connectivity of Hypercube increases with the size, though in case of de-Bruijn, it is constant at 4.

C. Extensibility

Extensibility is a property which facilitates constructing large sized systems out of small ones with minimum changes in the configuration of the nodes. In the proposed network, the extension complexity increases linearly because each extension requires adding a single layer of (n+1) nodes. Hypercube and de-Bruijn networks though are extensible but the complexity increases exponentially by the power of 2.

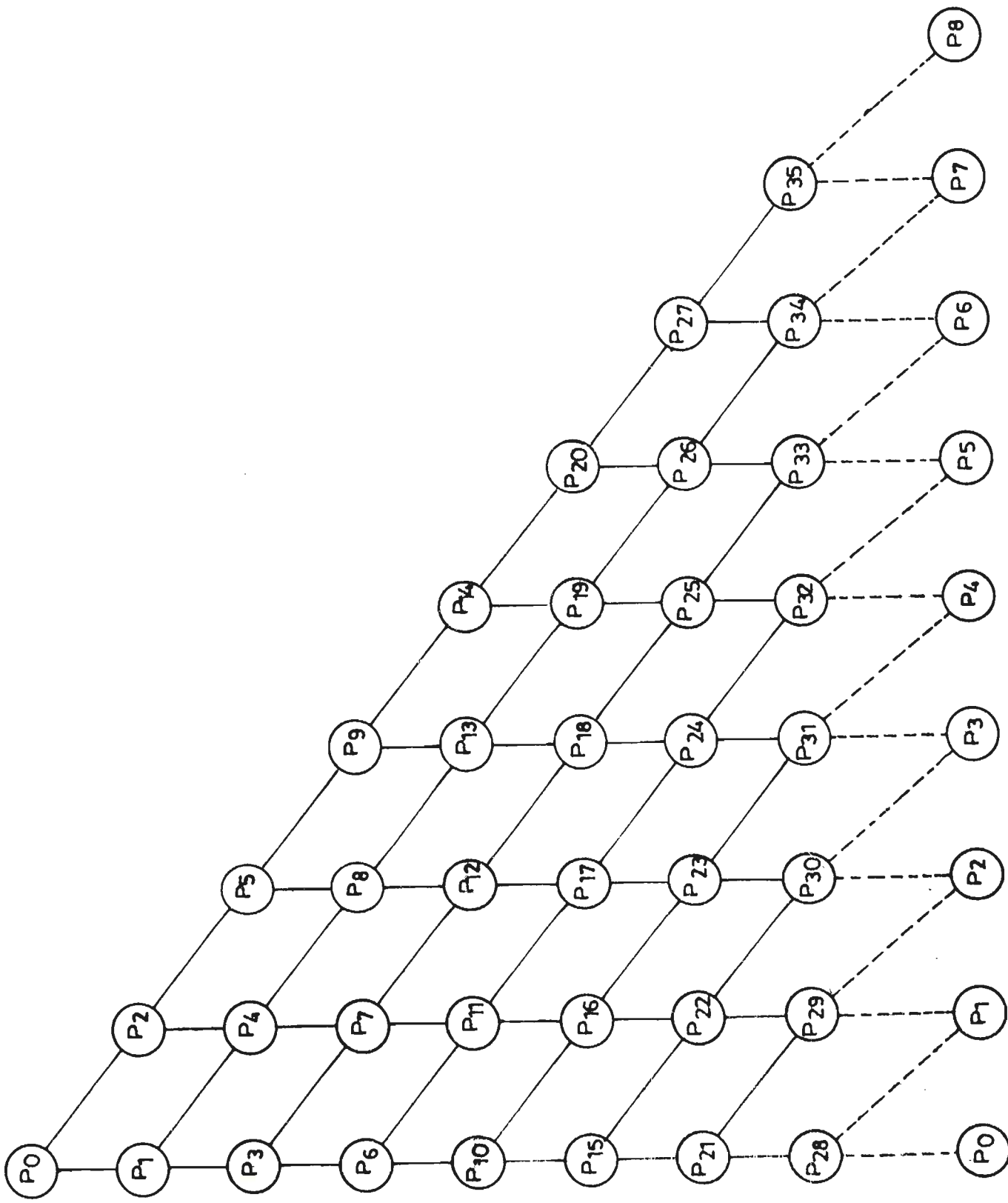


Figure 2.4.1 FT network for M=26

D. Diameter

The diameter of a multiprocessor network is a measure of the maximum internode distance in the network. This property is important in determining the distance involved in communications and hence the performance of the multiprocessor system. The diameter of a network is bound to increase as the size grows unless there is no limit on the number of links. In the case of de-Bruijn and Hypercube, the diameter increases by one as the number of processors is doubled. Ignoring the fold-back connections, the diameter of LET network also increases by one on each extension, although the fold-back connections will tend to provide short cuts between top portion processors and the bottom layer processors which are otherwise highly distant. The ratio of the number of processors served by back connections to total processors is $2/(d+1)$. As this ratio keeps on decreasing as the network grows, its effect on reduction in diameter is negligible. Table 3.1 shows the diameter of networks of different depths. The results have been obtained using shortest path algorithm. It may be seen that the diameter does not always increase with the addition of a layer of processors

Table 3.1
Diameter of various sized LET network

Depth	0	1	2	3	4	5	6	7	8	9	10	11	15	20
Number of Processors	1	3	6	10	15	21	28	36	45	55	66	78	136	231
Diameter	0	1	2	3	4	4	5	6	7	8	8	9	12	16

Table 3.2 summarizes comparison of the above properties of Hypercube, de-Bruijn and the LET network topology.

Table 3.2
Summary of parameters

Parameter	Hypercube	de-Bruijn	LET network
No. of processors	$N = 2^n$	$N = 2^n$	$N = \sum_{k=1}^n k$
Degree	n	4	4
Extensibility	$\frac{n}{2}$	$\frac{n}{2}$	$n+1$
Diameter	$O(\log_2 N)$	$O(\log_2 N)$	$O(\sqrt{N})$

E. Leaf to leaf distance

Message path lengths in a network depend on the communication patterns. The statistical distribution of the pairs of communicating nodes can be used in calculating average message path lengths. Even in a network with large diameter, the message path lengths can be kept low if the communication between distant nodes can be avoided. If the communication patterns are not known, then a uniform distribution i.e. all nodes sending messages to all other nodes with equal probability, has to be assumed. In such a case a small diameter will help.

In many situations, leaf to leaf distances are important in determining message path lengths. In a binary tree, there is a lack of direct links between leaves. Messages between remote leaves have to

travel up towards root and then come down to reach the target. In LET network, the back connections provide a direct long distance link between a leaf and some node in the vicinity of the root. As a result of this, it is expected that the longest inter-leaf distance in LET network will be $1/2$ or less of that in a simple binary tree.

In conclusion, it may be said that a new network topology (LET) for multiprocessor systems has been proposed as an attempt to combine some desirable features of linearly extensible tree structures and compact Hypercube or de-Bruijn structures. The proposed architecture exhibits better connectivity, lesser number of nodes and linear extensibility over Hypercube and BDM networks. In the next chapter, a dynamic scheduling scheme has been discussed which takes into account the adjacency matrix of network interconnections and with relatively small additional overhead, it oversees that the tasks will arrive at the minimum distance processors even for grossly unbalanced problem trees or even in the presence of failing nodes or links.

MINIMUM DISTANCE DYNAMIC SCHEDULING SCHEME

To make efficient use of a multiprocessor network it is necessary to distribute work in a manner which keeps all the processors equally busy. If a problem could be divided into totally independent modules then the problem of distributing is trivial: for m modules and n processors assign $\text{round}(m/n)$ modules to each processor and the rest to any processor. Even if different modules have different weights, the problem of dividing work uniformly among processors is not very difficult.

Scheduling problem acquires all its complexities due to data dependencies between different modules of a problem. All computation problems get decomposed into interconnected tasks. The nature of interconnection amongst problem tasks govern the kind of interconnections among processors which can lead to optimal scheduling. A naive solution is to have a network matching exactly in size and interconnections with the problem graph. The solution is obviously unacceptable.

The problem of load balancing among the processors may be tackled through static or dynamic allocation. Static scheduling requires partitioning the problems into a set of parallel tasks and then statically allocating them to processors so as to have maximum balance. A constraint on achievable degree of balance is the requirement of keeping the necessary evil of communication overhead to the minimum. If two communicating tasks are scheduled onto the same processor, the

communication delay is zero; otherwise, the delay equals the message start up time, plus the data size divided by the transfer rate. A process start time is dictated by the communication delay, which is the function of the number of hops from one processor to another along the interconnection network of the machine.

Static partitioning and allocation is unsuited to applications where the shape of the computational graph alters dynamically as the program executes. This situation arises quite often in the implementation of functional languages through reduction machines [53]. The aim of this chapter is to study the performance of the proposed LET network for tree-structured problems which occur so commonly in functional programs.

A dynamic scheduling scheme to suit the LET network has been described here. The basic property of this scheme is to minimise the distance between communicating tasks hence named as Minimum Distance Scheduling (MDS). At the end performance studies have been carried out using MDS scheme on the LET network and the comparative analysis has been done on other networks considering identical problem parameters.

4.1 THE SCHEDULING TECHNIQUES

Generally, scheduling techniques can be classified into two categories. In the first category, an application comprising a task or set of tasks with a priori knowledge about their characteristic is scheduled to the system nodes before run time. This type of scheduling problem is better described as the assignment/mapping problem. This assignment can be done in a number of ways using various optimization techniques, depending upon the nature of the application and the target

system. These types of techniques have also been termed as static scheduling techniques. The second class of scheduling which takes into account the notion of time, is used to assign tasks to processors by considering the current state of the system. The state information, concerning current load on individual nodes and the availability of resources is time dependent. These type of strategies don't assume a priori knowledge about the tasks and are known as dynamic scheduling techniques.

The scheduling is guided by some constraints which may be different from application to application. Since the performance of a parallel architecture can be characterized mainly by communication delays, distribution of load among the processors and scheduling overheads, a close correspondence between the structure of the problem and the architecture of processors is desired in order to minimize these overheads [82,86].

Load balancing attempts to improve system performance by redistributing the work load submitted to the system. It necessitates some means of maintaining a global view of the system activity, and a mechanism for redistributing tasks to processors on a network where they will most benefit in terms of execution time. A strong load imbalance produces a poor increase in performance when performed on a multiprocessor. By contrast, a near perfect load balance will produce linear improvements in the performance.

4.2 MINIMUM DISTANCE PROPERTY

In multiprocessor networks, interprocessor communication costs are significant relative to intraprocessor costs and have a substantial

effect on system performance. In order to reduce these overheads, a scheduling strategy must be designed for an assignment of task to processor which minimizes execution and communication costs.

Minimum distance is the property which assures the minimization of the communication in distributing subtasks and collecting partial results. A scheduling scheme with this property minimizes overheads thus guaranteeing maximum possible speedup. This property may be formally stated as [82] :

If T and T_1 are two tasks from a task tree of a given problem such that T is the parent of T_1 and if P and P_1 are the processors on which T and T_1 are scheduled, then P should be directly connected to P_1 in the network.

In the algorithm developed, adjacency matrix of the network is used to satisfy the minimum distance property. All the above mentioned techniques ultimately conclude that the target machine's interconnection network affect the schedule and, in turn, the running program. Thus the scheduling heuristic needs to include the target machine's characteristics. After having a deep and concise following of the above features, a Minimum Distance Scheduling (MDS) strategy for the multiprocessor network (LET) has been developed.

4.3 THE MINIMUM DISTANCE SCHEDULING (MDS) SCHEME

The scheme has been developed for a tree type problem structure. It is dynamic in the sense that no apriori knowledge of problem tree is assumed except that the problem can be represented as a tree. The simulation of dynamic load is discussed in section 4.5.

The approach tries to maximise the load balancing among processors under the constraint of the need to keep message path lengths to one hop (minimum distance property). Mostly any load balancing algorithm will consider the overall load at a processor. However, in this algorithm we take into account the 'active load' only for this purpose. In a tree type problem structure, it is expected that load at a particular level only has to compete for processor time and hence the load at other levels should not be considered for balancing. This load at a level in the problem tree, we define as active load.

In view of the above, the algorithm calculates ideal load value for each level, which is used by load balancer to detect load imbalances and make load migration decisions. The load imbalance factor for kth level of task tree, denoted as LIF_k , is defined as:

$$LIF_k = [\max \{ load_k(P_i) \} - (ideal-load)_k] / (ideal-load)_k$$

where $(ideal-load)_k = [load_k(P_0) + load_k(P_1) + \dots + load_k(P_{N-1})] / N$,

and $\max (load_k(P_i))$ denotes the maximum load pertaining to level k of the task tree on a processor P_i , $0 \leq i \leq N-1$, and $load_k(P_i)$ stands for the load on processor P_i due to kth level of the task tree.

Based on ideal-load value, the donor (overloaded) processors and acceptor (underloaded) processors are identified. Migration of task, if any, can take place between donor and acceptor processors only. Migration from a donor processor is done under the constraints of minimum distance. As this constraint is always applied, a task existing at the donor processor must have satisfied this constraint and hence

must be transferred to an acceptor, which is directly connected to the donor. Thus for every donor, there is a set of Minimum Distance Acceptors (MDA). Tasks are not allowed to migrate to acceptors which are outside this set. With reference to Figure 3.3 of LET network,

$$MDA(P_0) = \{ P_1, P_2, P_3 \}$$

if P_0 is a donor.

The scheme may be seen to have three phases, namely:

- 1) zero distance scheduling, which is done dynamically at each level.
- 2) processor selection, to select the donor and acceptor processors based on ideal-load, and
- 3) task migration under minimum distance constraint.

4.4 THE MDS ALGORITHM

The algorithm starts mapping the root task on the processor P_0 . At any subsequent level of the problem tree, each newly spawned task is initially scheduled on the same processor as its parent task. This provides a Zero Distance Scheduling (ZDS). However, this leads to lot of imbalance in load. In fact, if ZDS policy is followed then tasks will continue to be scheduled on P_0 only.

After zero distance schedule at a level k , the ideal load is calculated and the donor and acceptor processors are identified. These processors are arranged into priority queues according to the amount of ZDS k th level load (denoted as ZDS_k) at each processor, for donors it is a descending priority queue whereas acceptors have an ascending priority queue. The lower the ZDS_k load of an acceptor, the higher is its acceptance capacity. These queues are used in task migration phase.

A Load Table (LT) is maintained for task migration and for final calculation of LIF value at each level. LT has a row for each level of task tree and has columns corresponding to each processor. An element of this table may be referred as $LT [l,p]$ where l (i.e. ≥ 0) is the level number and $p(0 \leq p \leq N-1)$ is the processor number. Thus in the beginning, $LT[0,0] = T_1$, where T_1 is the root task of the problem tree.

The whole algorithm, in a "C" like notation, is given below:

```

mds()
{
    map root_task on P0;
    store (root_task); /*store(task) will store the subtasks in a list,
                        let n be the length of this list */
    k = 1;
    do
    {
        for (count = 0; count  $\leq$  n; count ++)
        {
            Tc = select(list);
                /* select(list) retrieves a task from the list */
            store(Tc);
            Tf = father(Tc);
                /* father(task) returns the father of the task */
            Pf = processor(Tf);
            /* processor(task) returns the processor on which task is
                scheduled */
            map Tc on Pf /* this is zero distance scheduling */
        }
        update (k); /* update (k) modifies the kth row of LT */
        schedule (k);
        k = k+1;
    }
}

```

```

        } while ( k < k_max);
    }

schedule ( int k )
{
    IL = ideal_load(k);
    for (itno = 0; itno < 2; itno ++ ) { /* number of iterations */
        for (i = 0; i < N; i ++ ) {
            If (load(Pi) > IL)
                add_dQ(Pi);
            /* add_dQ(Pi) puts the processor Pi in donor priority queue dQ */
            /* load Pi gives the load on Pi from LT[k,i] */
            else add_aQ(Pi);
        } /* add_aQ(Pi) puts the processor Pi in accepter priority queue aQ */
        while(dQ not empty) {
            Pi = delete(dQ);
            si = MDA(Pi);
            /* si is the set of minimum distance accepters for Pi */
            assign(Pi);
        } /* assign(Pi) tries to transfers a load equal to excess of Pi from Pi
        to the highest priority accepter from si. If not successful, Pi continues
        to be a donor with reduced overload.*/
        update(k);
    }
}

```

Tables 4.1 and 4.2 show the computer generated progress of load table for complete ternary and arbitrary ternary trees (upto level 5) respectively on a LET of 6 processors. In each row the entries are: donor processors, ZD schedule, MD schedule, ideal-load and LIF obtained. The next section deals with the details of simulation.

Table 4.1
Load Table for complete ternary task tree upto depth 5.

P_0								
ZDS[1]	3	0	0	0	0	0	0	3
MDS[1]	0	1	1	1	0	0	0	3
Ideal-load[1]	= 0.5		rounded IL = 1.0					
LIF[1]	= 0.00							
	P_1	P_2	P_3					
ZDS[2]	0	3	3	3	0	0	0	9
MDS[2]	1	1	2	1	2	2	2	9
Ideal-load[2]	= 1.5		rounded IL = 2.0					
LIF[2]	= 33.33							
	P_2	P_4	P_5					
ZDS[3]	3	3	6	3	6	6	6	27
MDS[3]	3	5	5	5	5	4	4	27
Ideal-load[3]	= 4.50		rounded IL = 5.0					
LIF[3]	= 11.11							
	P_2	P_4	P_5					
ZDS[4]	9	9	18	9	18	18	18	81
MDS[4]	13	13	14	14	13	14	14	81
Ideal-load[4]	= 13.50		rounded IL = 14.0					
LIF[4]	= 3.70							
	P_2	P_3	P_5					
ZDS[5]	39	39	42	42	39	42	42	243
MDS[5]	39	41	41	41	41	40	40	243
Ideal-load[5]	= 40.50		rounded IL = 41.0					
LIF[5]	= 1.23							

Table 4.2

Load Table for arbitrary ternary task tree upto depth 5.

P_0							
ZDS[1]	3	0	0	0	0	0	3
MDS[1]	1	1	0	1	0	0	3
Ideal-load[1]	= 0.5		rounded IL = 1.0				
LIF[1]	= 0.00						
P_0							
ZDS[2]	2	1	0	1	1	1	6
MDS[2]	2	1	0	1	1	1	6
Ideal-load[2]	= 1.00		rounded IL = 1				
LIF[2]	= 0.00						
P_1		P_2					
ZDS[3]	2	3	3	2	1	0	11
MDS[3]	2	2	1	2	2	2	11
Ideal-load[3]	= 1.83		rounded IL = 2				
LIF[3]	= 9.09						
P_1							
ZDS[4]	3	6	3	3	4	2	21
MDS[4]	4	4	3	4	4	2	21
Ideal-load[4]	= 3.50		rounded IL = 4.0				
LIF[4]	= 14.29						
P_1		P_4					
ZDS[5]	7	10	8	7	9	4	45
MDS[5]	8	8	8	8	9	4	45
Ideal-load[5]	= 7.50		rounded IL = 8.00				
LIF[5]	= 20.00						

4.5 SIMULATION RESULTS

The above mentioned MDS scheduling scheme has been implemented on Tata ELXI mini-super computer in the "C" language in the same environment. The simulation run consists of generating task trees and 'executing' them on the network of processors i.e. on a six processor LET network under the above discussed scheduling scheme, for a fixed set

of parameter values. The parameter values which are fixed for a particular run are the size of the network (by level) and the tree to be generated (i.e. maximum depth). The execution grain size is assumed to be uniform for all the tasks in the task tree. The respective average LIF's and, seed number for generating arbitrary trees, number of trees, the probability and fan out for the generation of the task trees etc. are used as parameters for a given class of trees. In order to study the behavior of scheduling mechanisms, each parameter is varied independently over a wide range of values. The entire process is then repeated over several classes of tree by varying the probability distribution associated with the random variables 'Spawn' and 'Fan out'.

4.5.1 *Dynamic load model*

The scheduling performance of the strategy has been tested on the LET network by simulating artificial dynamic loads. In order to characterize a non-deterministic load, the total problem is conceived to be an arbitrary tree which unwinds itself level by level. A task scheduled on a processor spawns an arbitrary number of subtasks, which are part of the whole problem tree. Thus the load on each processor is varying at run time creating unbalance, and balancer has to be invoked after each unwinding step.

For a meaningful simulation, tree structures that form a representative sample of programs are needed which are to be executed on the networks. To meet this requirement, a set of randomly created tree structures are used, whose generation is governed by two random variables 'Spawn' and 'Fanout'. The random variable 'Spawn' decides whether a node should be a leaf node or an internal node and the random

variable 'Fanout' decides the number of children a node should have. A tree is built in a breadth-first manner, starting from the root node.

By repeated application of the following operations on the nodes at the lowest level of the partially constructed tree, a tree structure is generated up to a pre-specified level (depth):

If the depth of the node is equal to the pre - specified depth

then the node is a leaf node

else if the value assumed by the random variable Spawn is zero then

the node is a leaf node

else the node is an internal one and the number of children

it has is equal to the value assumed by the random

variable Fanout for that node.

In a tree thus generated, each node represents a task. The tree is considered as a test problem on which the schemes are to be applied. Any particular class of tree is characterized by probability distributions associated with the random variables 'Spawn' and 'Fanout'. Thus, different classes of trees are generated by associating different probability values with 'Spawn' and 'Fanout'.

The experiments have been based upon various types of randomly generated tree structures, which fall into one of the following categories.

- 1) random binary tree structures (tree structures having a maximum fanout of two), and
- 2) random ternary tree structures (tree structures having a maximum fanout of three)

In the former case 'Fanout' is uniformly distributed over the range {1,2}, and for the later, it is uniformly distributed over the range {1,2,3}.

In order to obtain tree structures having different amount of parallelism, different probability values are assigned to the random variable 'Spawn'. The following probability distributions have been employed in order to generate various tree structures.

$$\begin{array}{ll} P(\text{Spawn} = 1) = 1.0, & P(\text{Spawn} = 0) = 0.0, \\ P(\text{Spawn} = 1) = 0.9, & P(\text{Spawn} = 0) = 0.1, \\ P(\text{Spawn} = 1) = 0.8, & P(\text{Spawn} = 0) = 0.2 \end{array}$$

where Spawn = 0 implies leaf node and Spawn = 1 implies internal node. It has been observed that for distributions having lower values for $P(\text{Spawn} = 1)$, the tree structures generated are too sparse to be meaningful.

A task tree is generated and executed in a parallel and breadth-first manner starting from the root task which is assigned to some given processor in the network. The task tree grows as tasks randomly spawn new tasks and scheduler schedule them onto neighbouring processors (or itself) as per the scheduling rule. A task after spawning sub-tasks enters into a wait state. A waiting task becomes executable at a later point of time, when all of its sub-tasks have completed execution. An executable task on being selected by the processor, executes to produce a result packet. The result packet is then forwarded to its destination.

Using the above simulated dynamic load, the performance of the network has been tested for MDS scheme. The performance is measured in terms of load imbalance left after a balancing action at each level of task tree. A constraint that has been forced in the scheduling is to maintain minimum distance i.e. task do not migrate to under loaded processors in a way so as to make the distance from parent task more than one hop in the processor network.

4.5.2 MDS scheme on LET network

To study the behavior of the dynamic Minimum Distance Scheduling (MDS) Scheme on the LET network, the LIFs are computed for different classes of task structures. The estimation of LIF is obtained and the curves are plotted as the LIF against the problem size (in terms of task tree depth) shown in Figures 4.1 - 4.4.

The trend of the curves obtained, indicates the average behaviour of the load imbalance factor with respect to the level in the task tree for different randomly generated tree structures when MDS scheme is implemented on the LET network. It has been observed that LIF shows a similar behaviour in both the cases of binary and ternary tree task structures, rising initially from zero to a peak and then reducing asymptotically. The number of tasks, in those levels of the tree where the LIF shows rising trend, is less than the number of processors in the network. The LIF starts falling in these graphs once sufficient number of tasks are available in the network. In case of arbitrary binary task trees, the peak value of LIF remains in between 35-45%, whereas in case of arbitrary ternary task trees, the peak value of the average LIF never exceeds more than 25%.

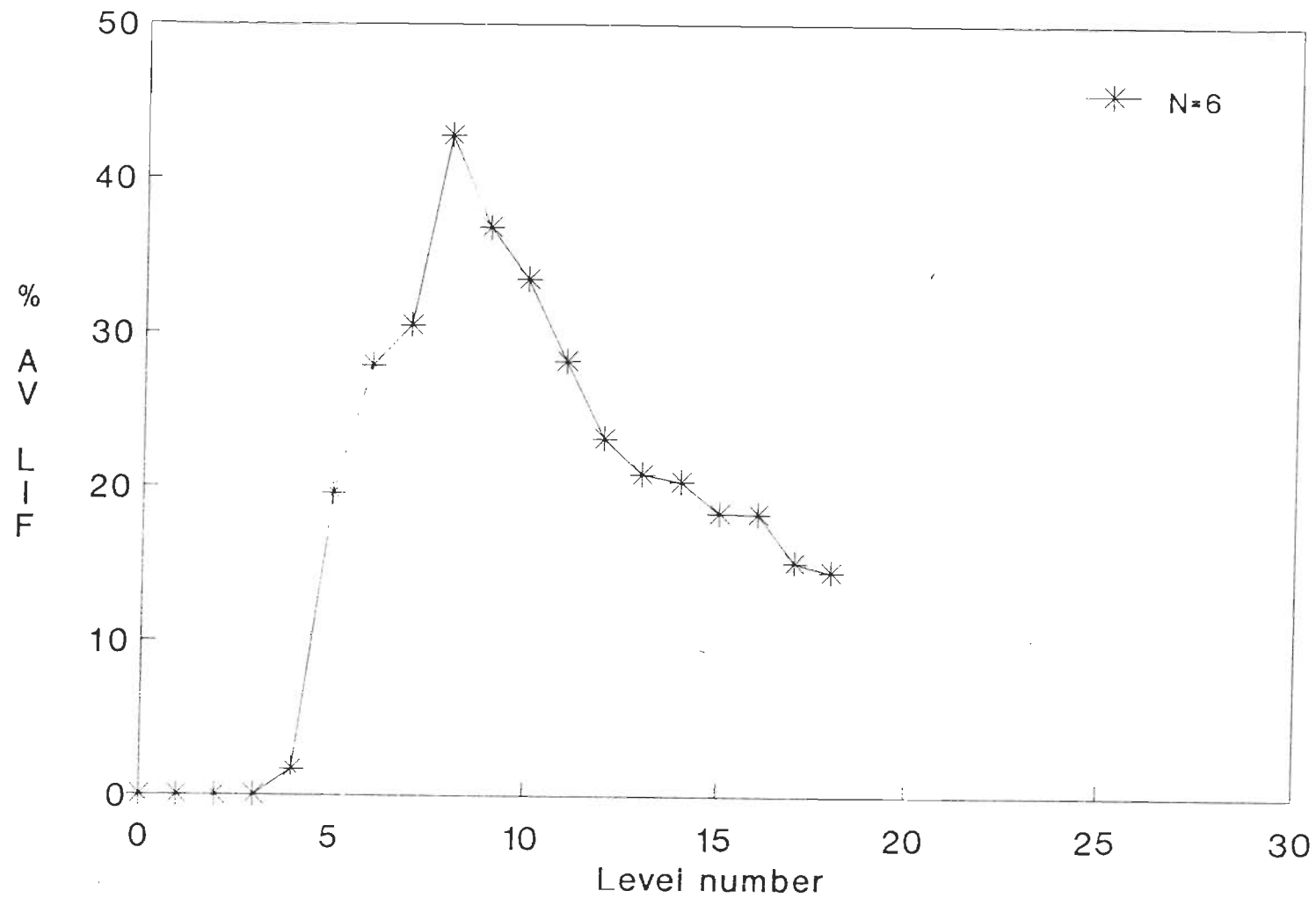


Fig. 4.1 LIF for arbitrary binary tree
(Probability = 0.9)

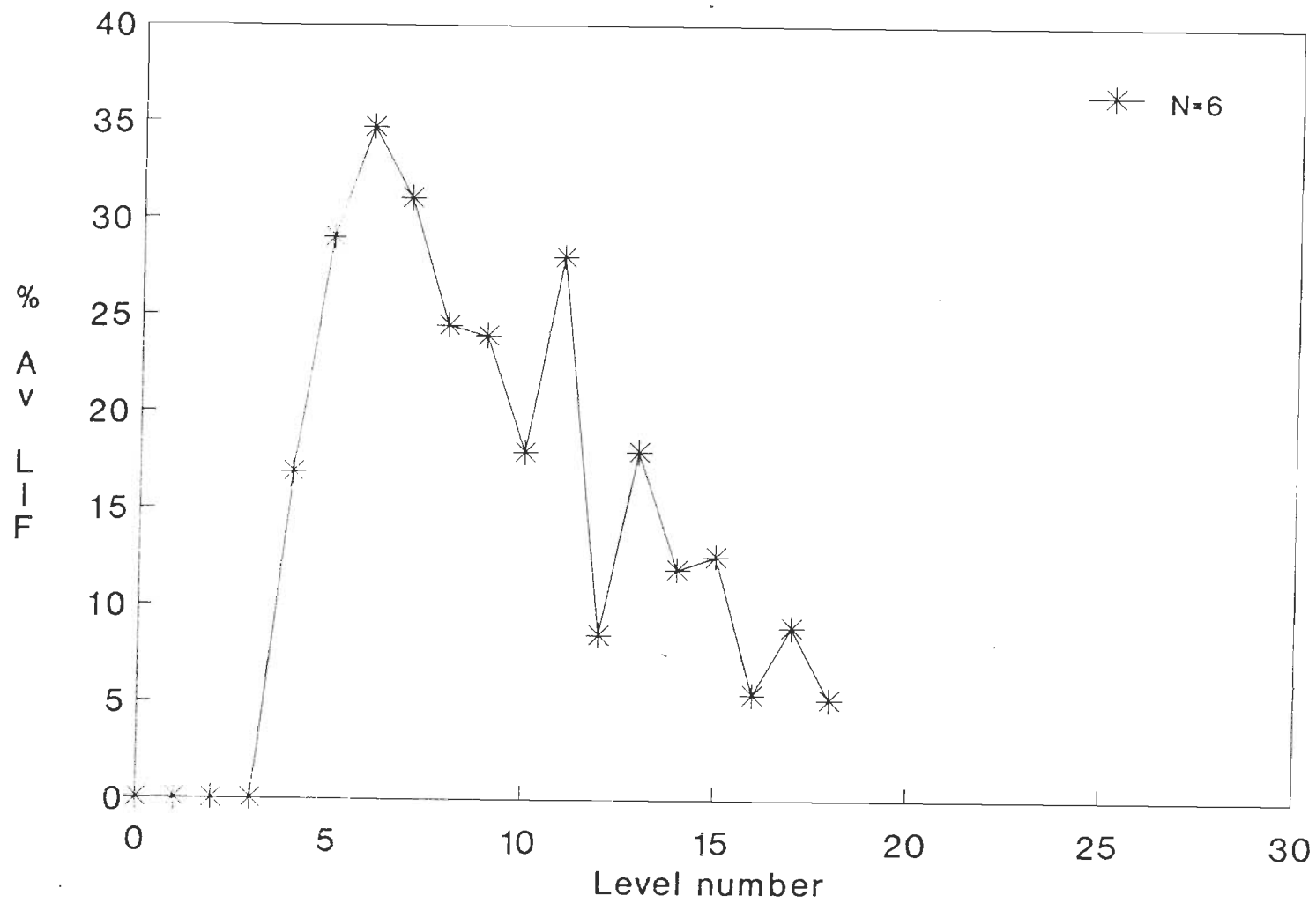


Fig. 4.2 LIF for arbitrary binary tree
(Probability = 1.0)

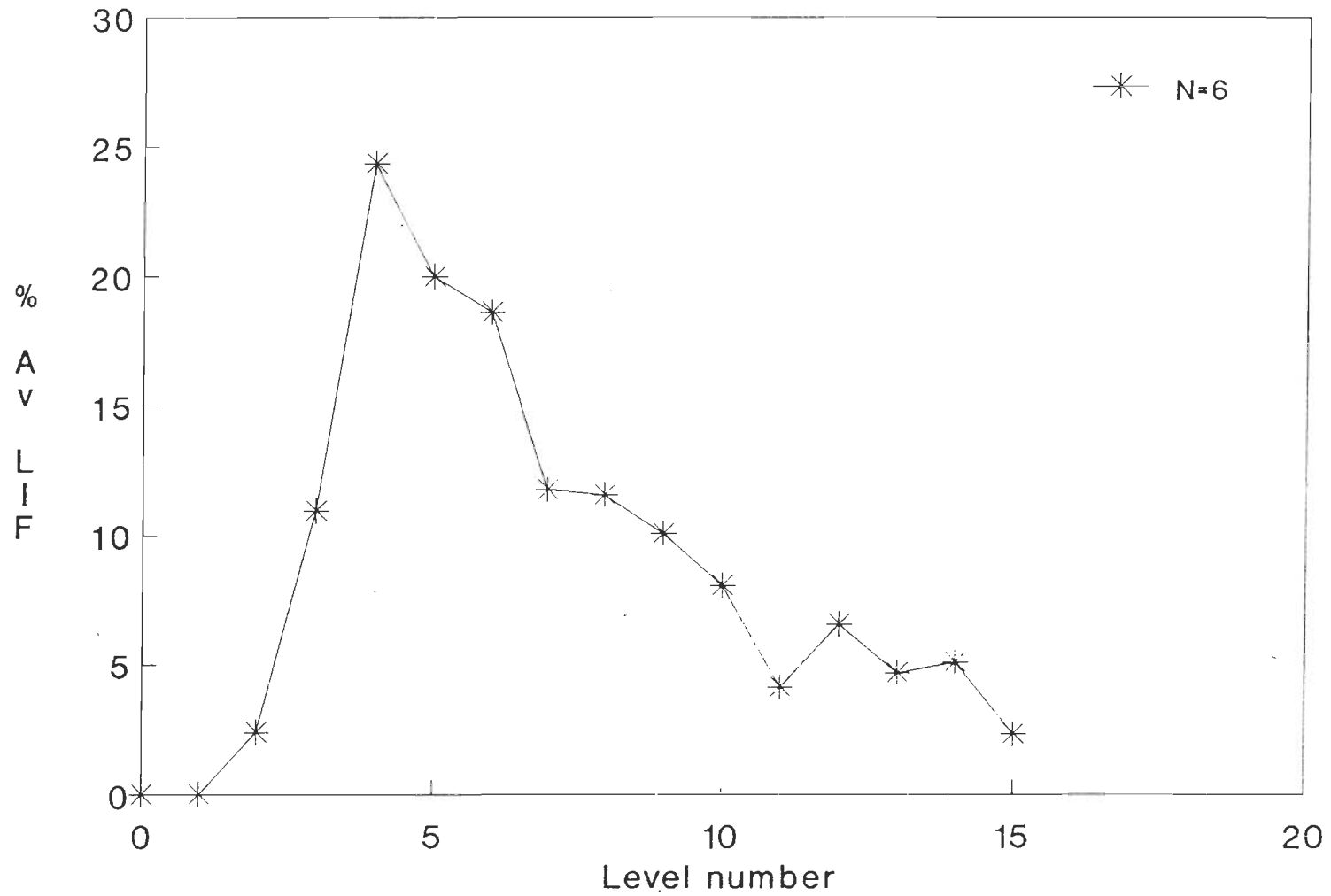


Fig. 4.3 LIF for arbitrary ternary tree
(Probability = 0.9)

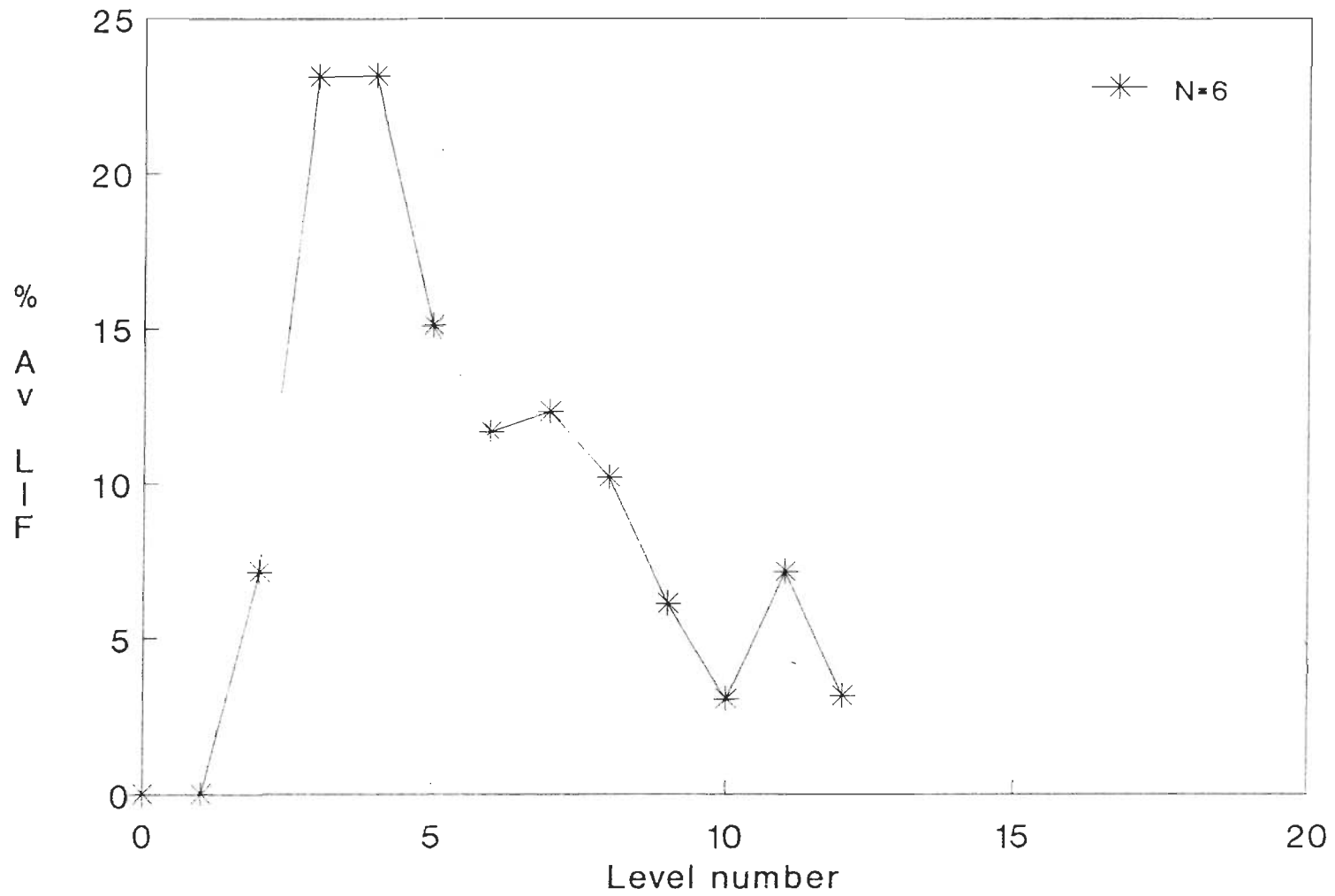


Fig. 4.4 LIF for arbitrary ternary tree
(probability = 1.0)

From the above observation, we can say that the MDS scheme is performing well within the limitation for binary task trees but performing better for ternary task graphs on LET network.

4.5.3 Comparison with other networks

The same MDS scheme is implemented on other networks also and the same performance indices are evaluated. First, the simulation study was carried out on BDM network. Figures 4.5 - 4.9 show the average LIF verses level of the task tree for binary and ternary task trees.

The comparative simulation study indicates that the MDS scheme is performing good for binary task graphs on BDM network in comparison to the LET network. On the other hand, the scheme is performing poorly for ternary task graphs on BDM in comparison to LET network. The LET network is outperforming BDM for ternary task graphs for any probability of spawning the tasks.

This dynamic scheduling scheme when implemented on other networks, like Hypercube, BDM and Binary along with LET, the performance results are shown in Figures 4.10-4.13.

The performance results indicate that LIF is always higher for LET network in comparison to other networks for binary task graphs. This indicates that LET is not performing nicely and not giving good performance for binary task trees with respect to other networks but even then the performance of LET is comparable to other networks for binary task graphs as depicted in Figure 4.10. On the contrary, the LET network is outperforming for ternary task graphs in comparison to Hypercube, BDM and Binary networks as shown in Figures 4.11-4.13.

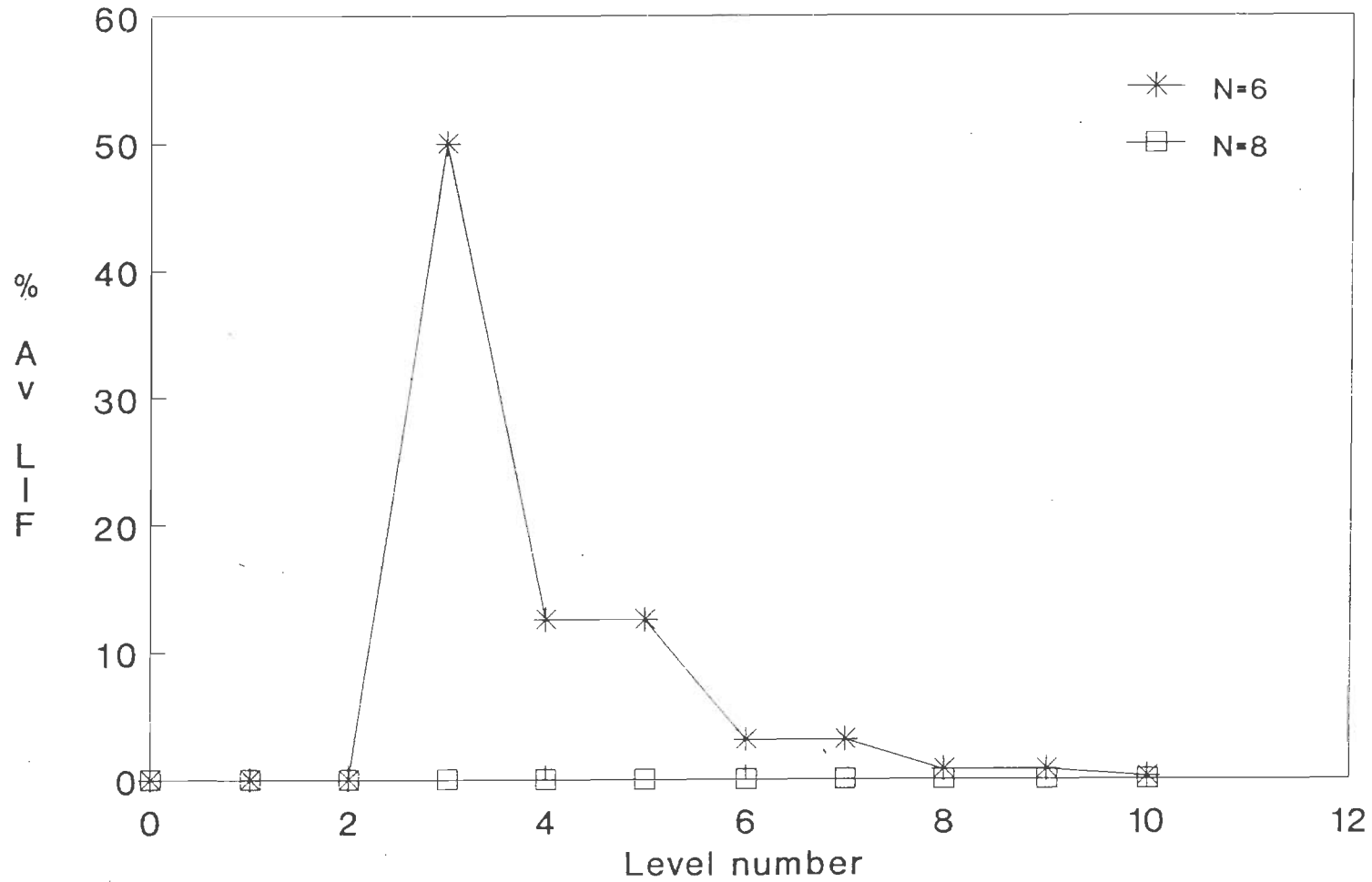


Fig 4.5 LIF for complete binary tree

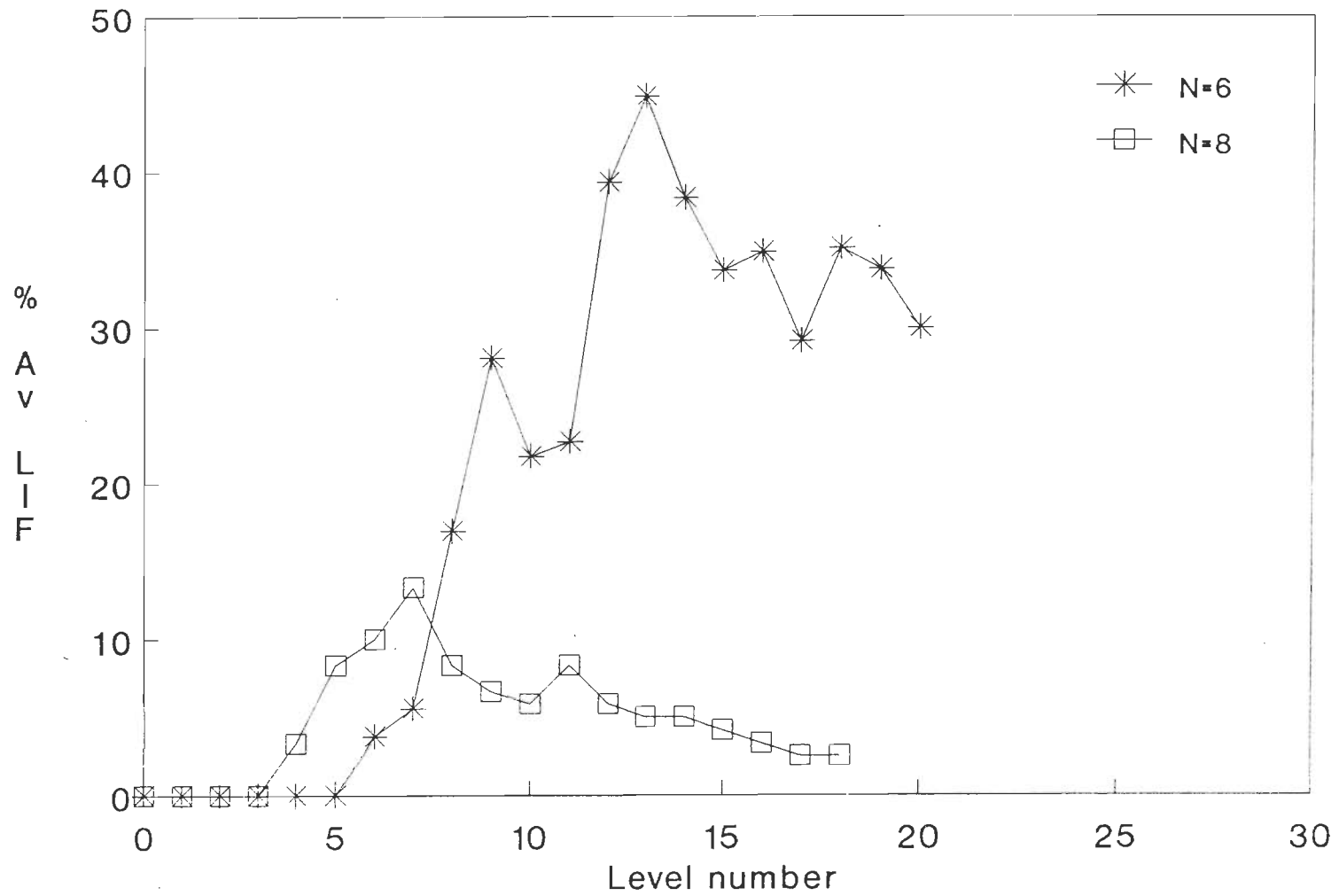


Fig. 4.6 LIF for arbitrary binary tree
(Probability = 0.75)

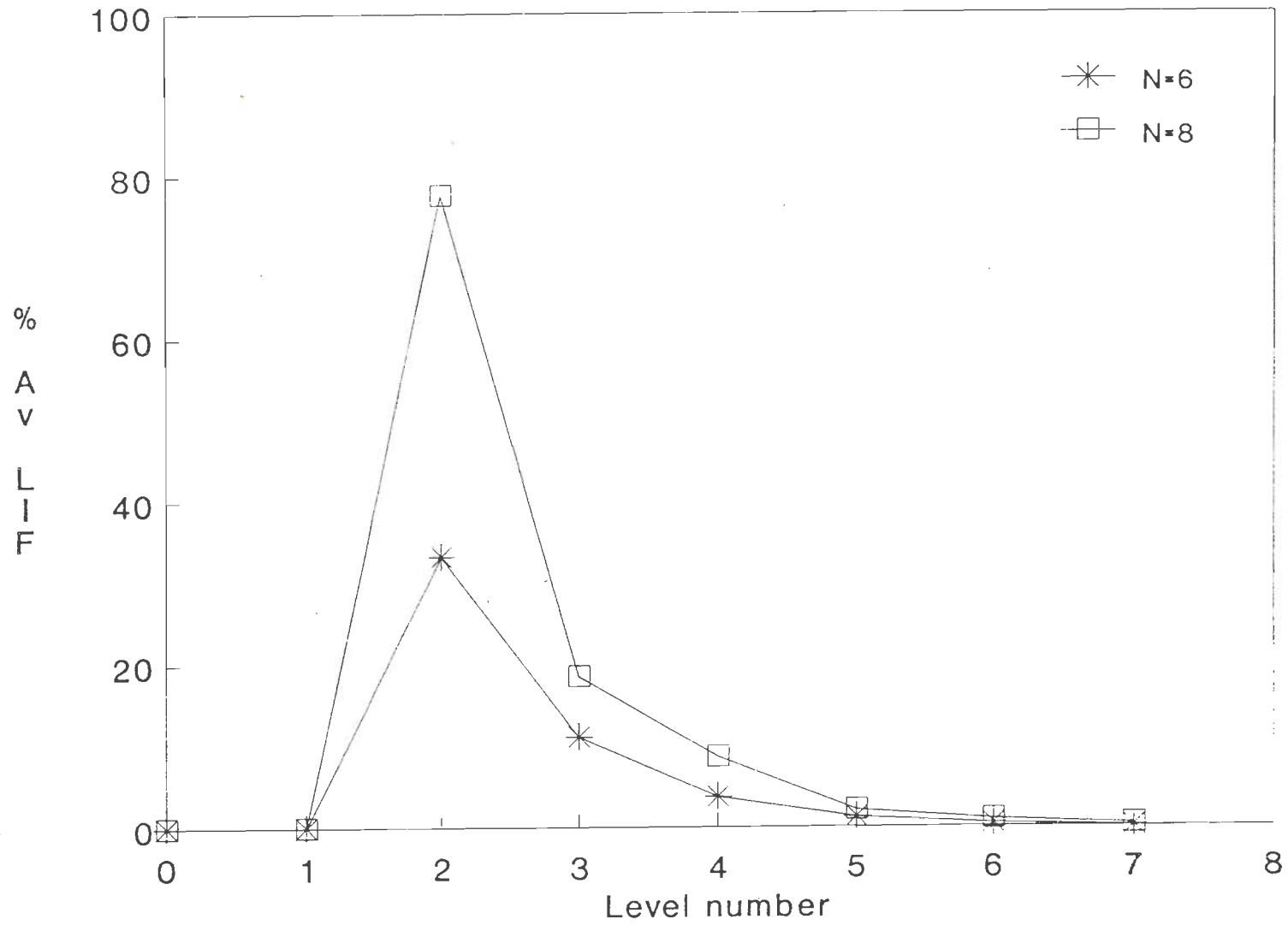


Fig. 4.7 LIF for complete ternary tree

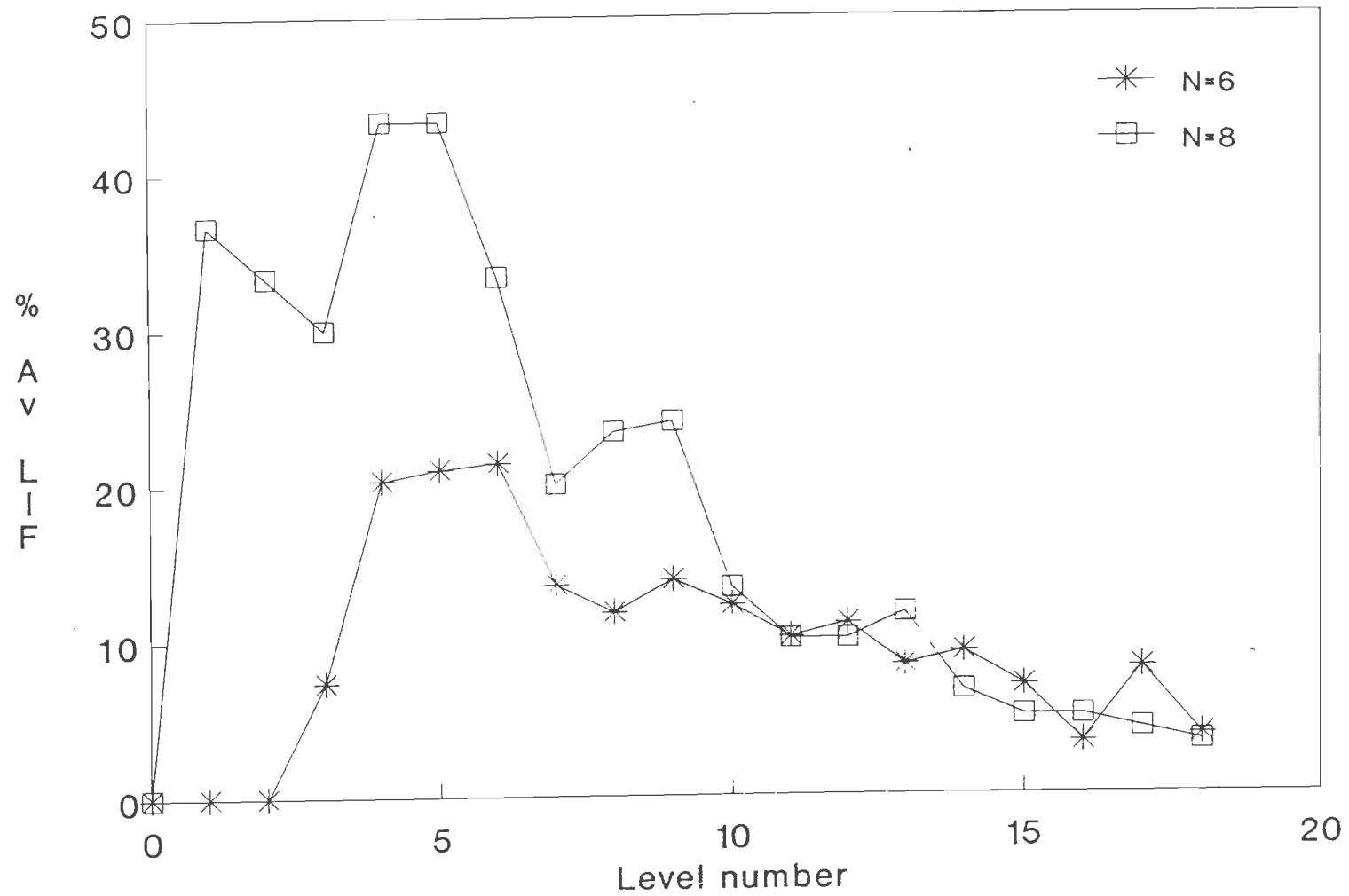


Fig. 4.8 LIF for arbitrary ternary tree
(probability = 0.75)

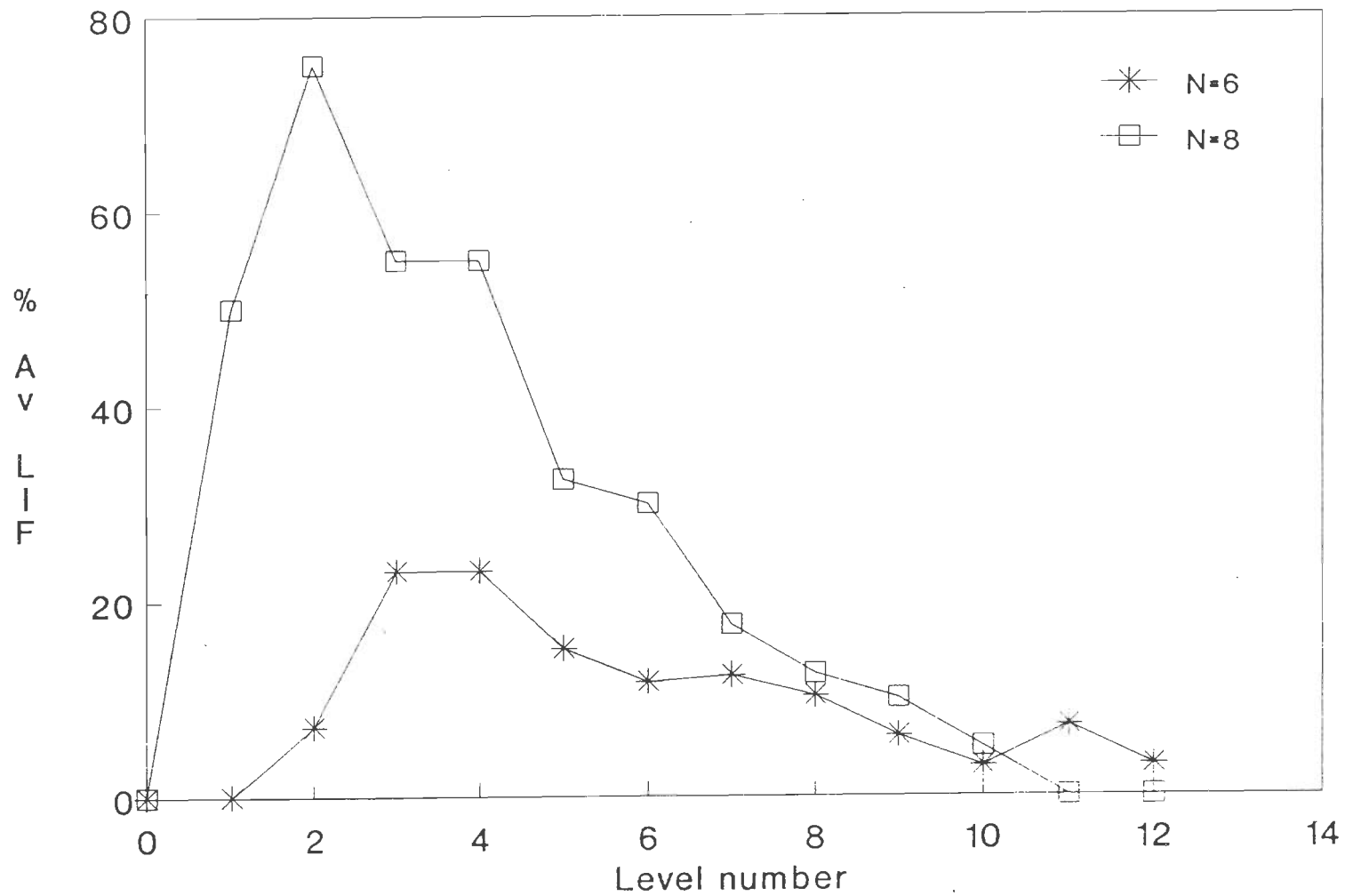


Fig. 4.9 LIF for arbitrary ternary tree
(Probability = 1.0)

The reason for better load balance in BDM for complete binary trees is that the network itself is a virtual binary tree. In other words for binary tree problems BDM has perfectly matched topology. However, in case of ternary trees and to some extent in arbitrary binary trees BDM enjoys no such advantage or it may be said that LET network is not handicapped.

Better performance of LET network for ternary tree structures is due to lesser number of processors. The comparison in various curves is for 6 processor LET and 8 processor BDM or Hypercube. It is always possible to obtain a better load balance in a smaller compact network. It may be argued that comparison should have been made with 10 processor LET network. However, third level of extension produces $8(=2^3)$ processor networks for BDM or Hypercube whereas a 6 processor network for LET. The performance of 10 processor LET will have to be compared to a 16 processor BDM.

It may be concluded that LET connection topology is able to provide a complete network with lesser number of processors. for a comparison on equal footing, if we were to conceive a 6 processor BDM it will be a 'broken' network not exhibiting full properties of BDM. This means LET is providing the 'completeness' of 8 processor BDM with 6 processors only and hence better load balancing.

As LET is a linearly growing structure, an equivalent LET will always have lesser number of processors compared to BDM or Hypercube (Hypercube loses on another front that growth in size, it requires processors with larger number of interconnection ports, which is constant in BDM or LET network.). Thus another advantage flowing out of

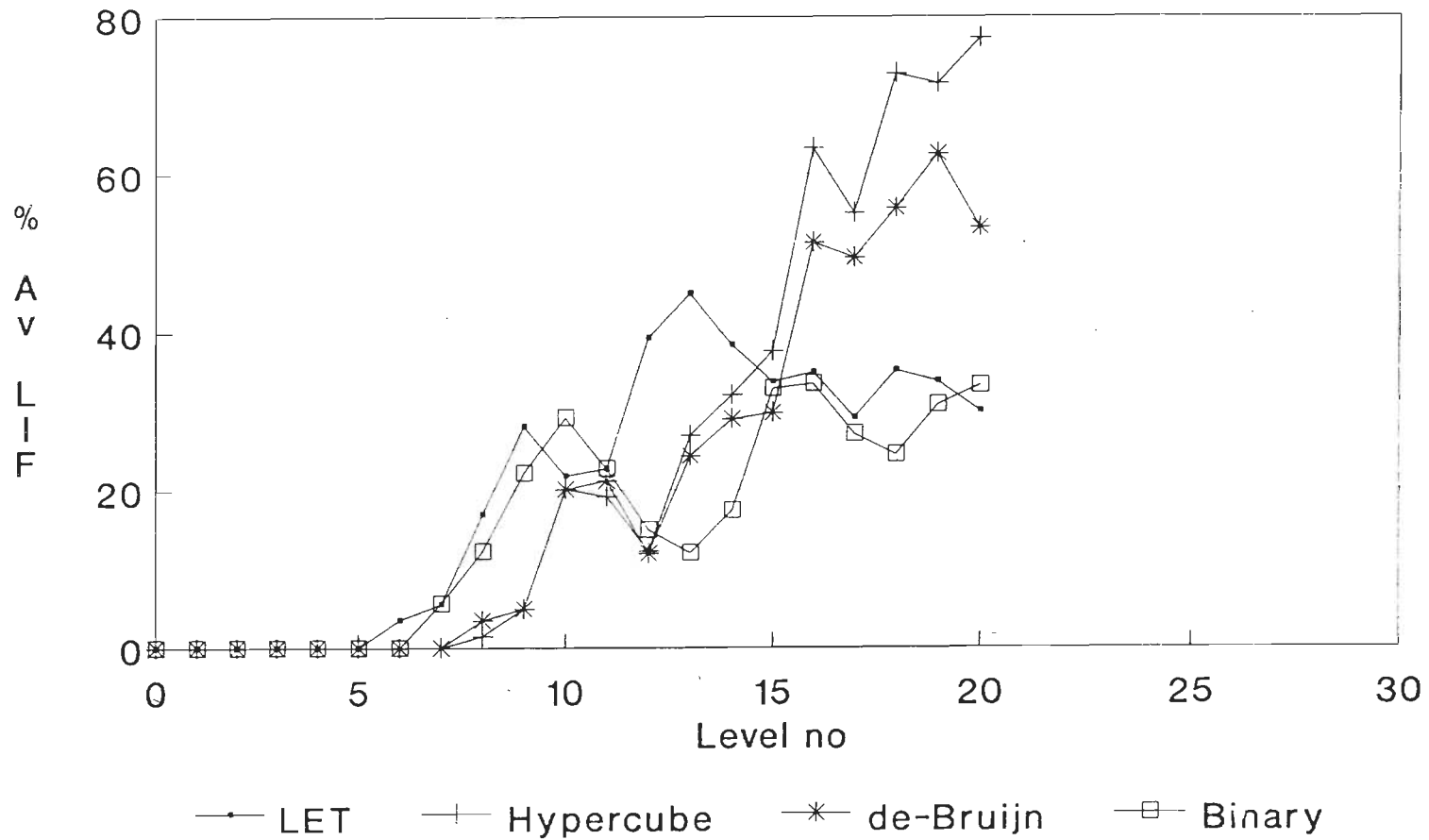


Fig. 4.10 MDS scheme on various networks
for arbitrary binary tree
(Probability = 0.75)

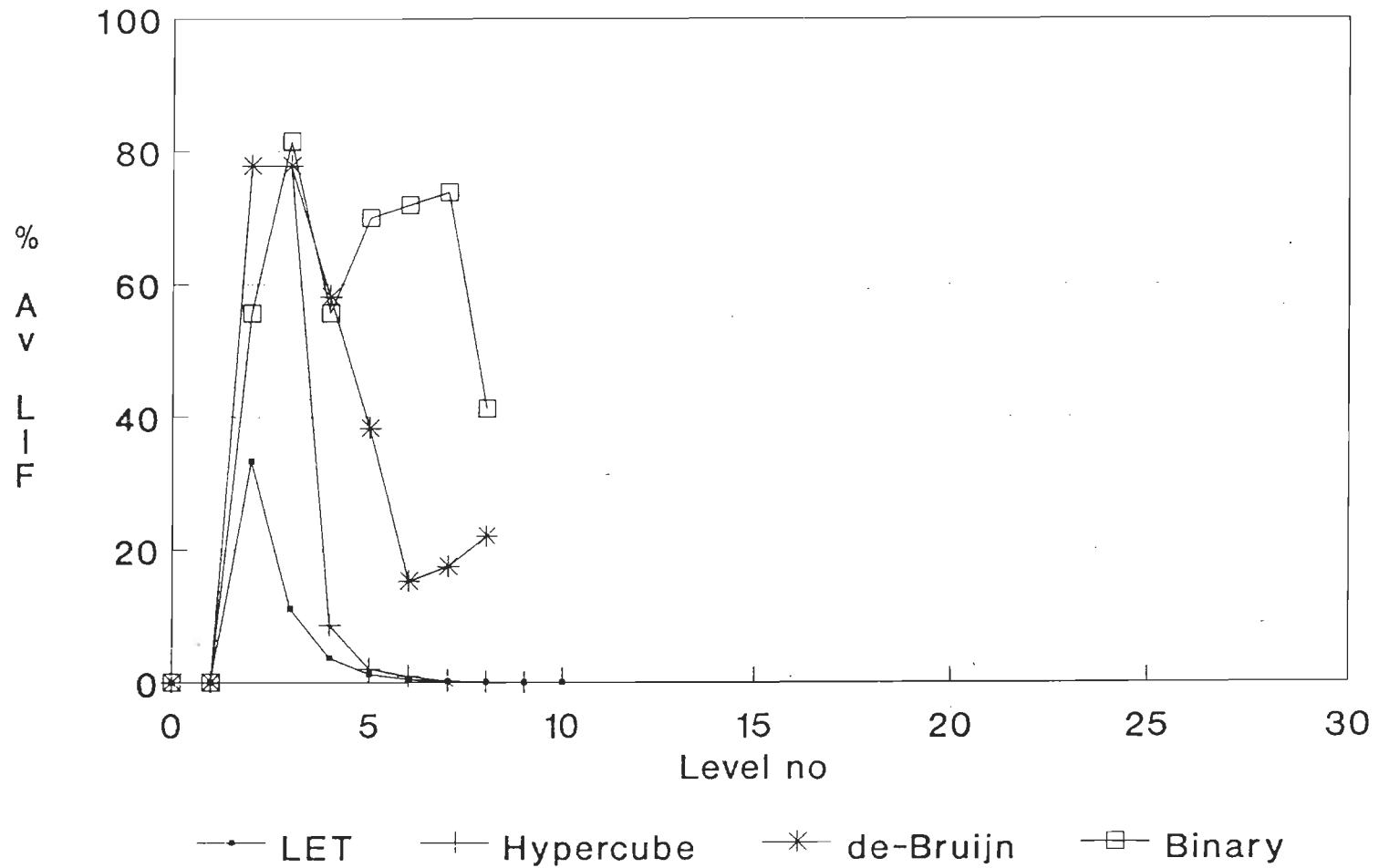


Fig. 4.11 MDS scheme on various networks for complete ternary tree

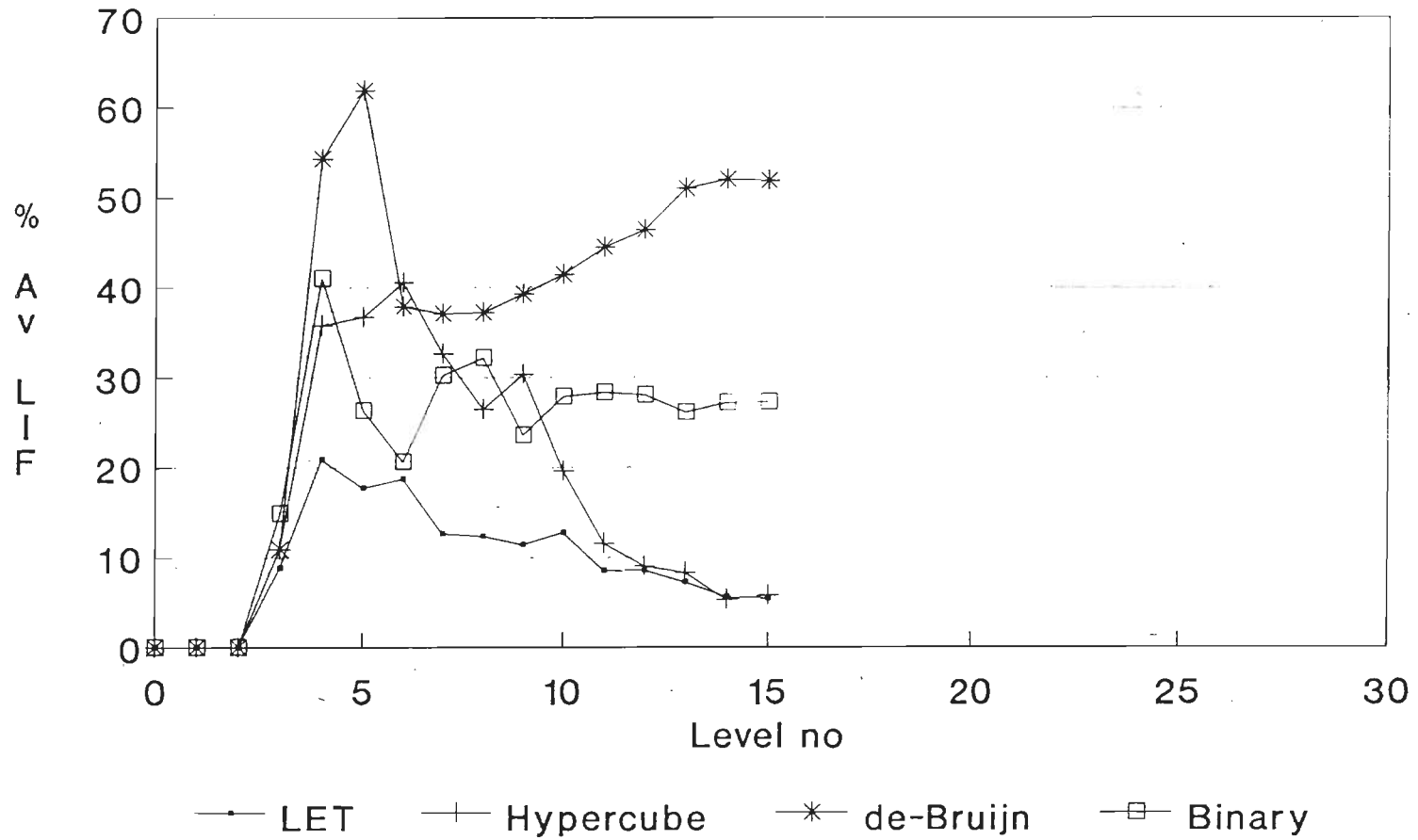


Fig. 4.12 MDS scheme on various networks
for arbitrary ternary tree
(Probability = 0.85)

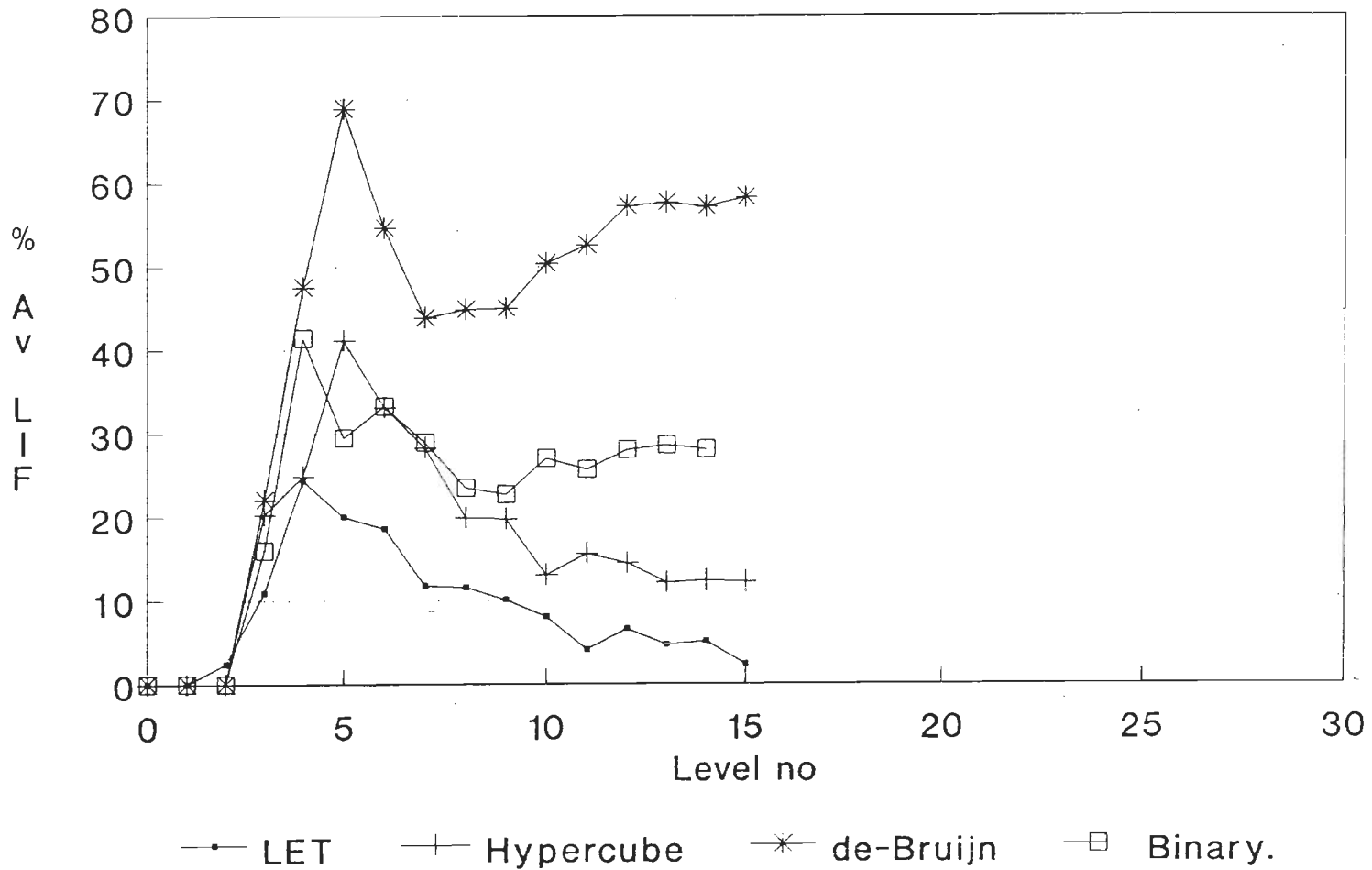


Fig. 4.13 Proposed scheme on various networks for arb. ter.

(Probability = 0.9)

linear extension in LET, is better load balancing achievable besides lower cost.

The next chapter deals with the comparison of MDS scheme with other existing scheduling schemes on LET network. Also, the proposed organisation model (pair of proposed network and the scheduling scheme) has been compared with some other models.

COMPARISON OF MDS SCHEME WITH OTHER SCHEDULING SCHEMES

Efficient management of parallelism involves optimizing conflicting performance indices like minimization of communication and scheduling overheads and of load imbalance among processors etc. A solution to this problem can be obtained at the organizational level by selecting a suitable layout and appropriate scheduling mechanisms.

In this chapter different scheduling schemes have been described and various organizational models (pair of a network and a scheduling scheme designed for it) have been compared with the proposed organizational model. The comparison is based on the simulation studies under the same environmental conditions.

5.1 OTHER SCHEDULING SCHEMES

The following are few scheduling schemes which have given optimal performance on the particular multiprocessor networks for which they have been designed.

5.1.1. *Round-Robin (R-R) scheduling scheme*

Ravikanth et.al. [82], implemented a Round-Robin scheduling scheme onto a binary de-Bruijn multiprocessor network. The scheduling strategy is described as:

Let a task T on a processor P from Q spawn two subtasks T1 and T2 then, a simple scheduling amounts to assigning T1 to left of the processor and T2 to the right of the processor P's childs.

The scheduling strategy assigns the tasks spawned by a processor in a round robin fashion between its left and right child processor. It maps a binary tree of arbitrary depth in an optimal manner onto the network. This is because the structure of the network is like that of a (virtual) binary tree of infinite depth. The effectiveness of the strategy for arbitrary tree structures is estimated by studying the load imbalance factor (i.e. how the maximum load on a processor deviates from the ideal load) for every level of the task tree.

To this end the term Load-Imbalance-Factor (LIF) is used which is defined for every level of the task tree. It represents the deviation of the maximum load on a processor from the ideal load.

The load-imbalance-factor for k th level of the task tree is defined as (the expression exists in an earlier chapter 4) :

$$LIF_k = [\max \{ \text{load}_k(P_i) \} - \text{ideal load}_k] / \text{ideal load}_k$$

where $\text{ideal load}_k = [\text{load}_k(P_0) + \text{load}_k(P_1) + \dots + \text{load}_k(P_{N-1})] / N$,

and $\max(\text{load}_k(P_i))$ denotes the maximum load pertaining to level k of the task tree on a processor P_i in Q ., and $\text{load}_k(P)$ stands for the load on processor P due to the k th level of the task tree.

5.1.2. *Minimum-Load (M-L) scheduling scheme*

To improve the performance for scheduling arbitrary tree structured tasks, Reddy [86], improves the round-robin strategy for Binary de-Bruijn Multiprocessor(BDM) and Quaternary de-Bruijn Multiprocessor (QDM). The approach is to exchange the information with the immediate

neighbours before scheduling a task. The load information can be extracted from the locality of a processor and is used while scheduling a task onto a processor.

A subtask generated by processor P is scheduled onto left processor, or right processor child or P itself wherever the active-load is minimum.

The term active-load refers to the load on a processor corresponding to the level that is active(executable) in the task tree structure. All the other tasks pertaining to lower levels of task tree are known as passive-load. Since the tasks that undergo reduction, are taken from active-load only, it is logical that the scheduling decision should be based only on the active-load information, and not on total load of the processor.

Reeves et.al. [103] proposed several dynamic scheduling strategies for highly parallel computers. These schemes have been tested after appropriate modification for applying them to the LET network. The general model of the dynamic load balancing is mainly based on the load balancing profitability determination at various sites in a multiprocessor network. Whenever profitable, a balancer is invoked which migrates tasks to achieve a more uniform distribution of the load on the processors. Each donor processor, during balancing, selects most suitable tasks (based on task dependencies) for migration, thus maintaining minimum distance.

The balancer uses the concept of balancing domains which reduces the overhead of the balancing process, but does not ensure a balanced load for the entire system. This trade-off is illustrated in the scheduling strategies.

5.1.3. Dimension Exchange Method (DEM) scheduling scheme

The DEM strategy was conceptually designed for a Hypercube system. In this scheme small domains are balanced first and these are then combined to form larger domains until ultimately the entire system is balanced. The balancing is performed iteratively in each of $\log N$ dimensions for an N processor Hypercube configuration. Balancing is initiated by any overloaded processor whose load level rises above the average level by some threshold. The DEM scheme can be described as:

DEM is a global, fully synchronous approach. Load balancing is performed in an iterative fashion by folding an N processor system into $\log_2 N$ dimensions and balancing one dimension at a time.

In other words, in case of an N processor system, balancing is performed iteratively in each of the $\log N$ dimensions. All processor pairs in the first dimension, those processors whose addresses differ in only the least significant bit, balance the load between themselves. Next, all processor pairs in the second dimension balance the load between themselves and so forth, until each processor has balanced its load with each of its neighbours.

To implement this scheduling scheme onto LET network as given in Fig. 3.3 in chapter 3, the pairs (or the dimension) and the order in which tasks are transferred to balance the six processor network (refer to Figure 3.3) are:

Processors	(P_0-P_1) ,	(P_0-P_2) ,	(P_0-P_3)
Processors	(P_1-P_3) ,	(P_2-P_4)	
Processors	(P_1-P_4) ,	(P_2-P_5) ,	(P_3-P_5)

5.1.4. Hierarchical Balancing Method (HBM) scheduling scheme

The HBM is a dynamic, synchronous, global approach which organizes the system into a hierarchy of subsystems. Load balancing is initiated at the lowest levels in the hierarchy with small subsets of processors and ascends to highest level which encompasses the entire system. This scheme centralizes the balancing process at different levels of the problem tree with increasing degree of knowledge at higher levels. The scheme is most suited to tree-structured networks.

In the LET network, whole of the network can not be considered at one level, like Hypercube network. In LET network, the hierarchy level starts from the bottom. The information of the leaf nodes are transferred to their respective fathers at the next higher level. In the separate domains, father and the two childs, migrate tasks from overloaded processors to under loaded processors, the same process takes place in other domain also till the whole network is balanced.

5.1.5. Gradient Model (GM) scheduling scheme

This scheme employs a gradient map of the proximities of the under loaded processors in the system to guide the migration of tasks between overloaded and under loaded processors. The proximity of a processor is the shortest distance to lightly loaded processor. The scheme has a big overhead of updating the proximity map from time to time. The number of messages generated for this updating depend on network topology besides some factors which are problem structure dependent.

5.2 PERFORMANCE STUDY OF THE MDS AND OTHER SCHEMES ON LET

The above mentioned scheduling schemes including the proposed MDS scheme are implemented on Tata ELXI mini-super computer in the C

language in the same environment. The simulation run consists of generating arbitrary task trees and 'executing' them on the network of processors i.e. on the six processor LET network under the above discussed scheduling schemes, for a fixed set of parameter values. The parameter values which are fixed for a particular run are the size of the network and the tree to be generated (i.e. maximum depth of its leaves). The execution grain size is assumed to be uniform for all the tasks in the task tree. The respective average LIFs and, seed number for generating arbitrary trees, number of trees, the probability and fan out for the generation of the task trees etc. are used as performance indices, for a given class of trees under a fixed set of parameter values. In order to understand the behavior of scheduling mechanisms, each parameter is varied independently over a wide range of values. The entire process is then repeated over several classes of tree by varying the probability distribution associated with the random variables Spawn and Fan out.

To study the behavior of different scheduling schemes on the LET network, the LIFs are computed for different classes of task structures. The estimation of LIF is obtained and the curves are plotted as the LIF against the problem size (in terms of task tree depth) shown in Figures 5.1 - 5.4.

The trend of the curves obtained, indicate that the proposed scheduling scheme out performs other schemes in case of complete binary tree. The LIF obtained through MDS scheme decreases smoothly to a minimum value of LIF, whereas for RR scheme it decreases from the very high peak and have the very high value of LIF as compared to the MDS scheme which has the lowest value, and similar is the case with ML

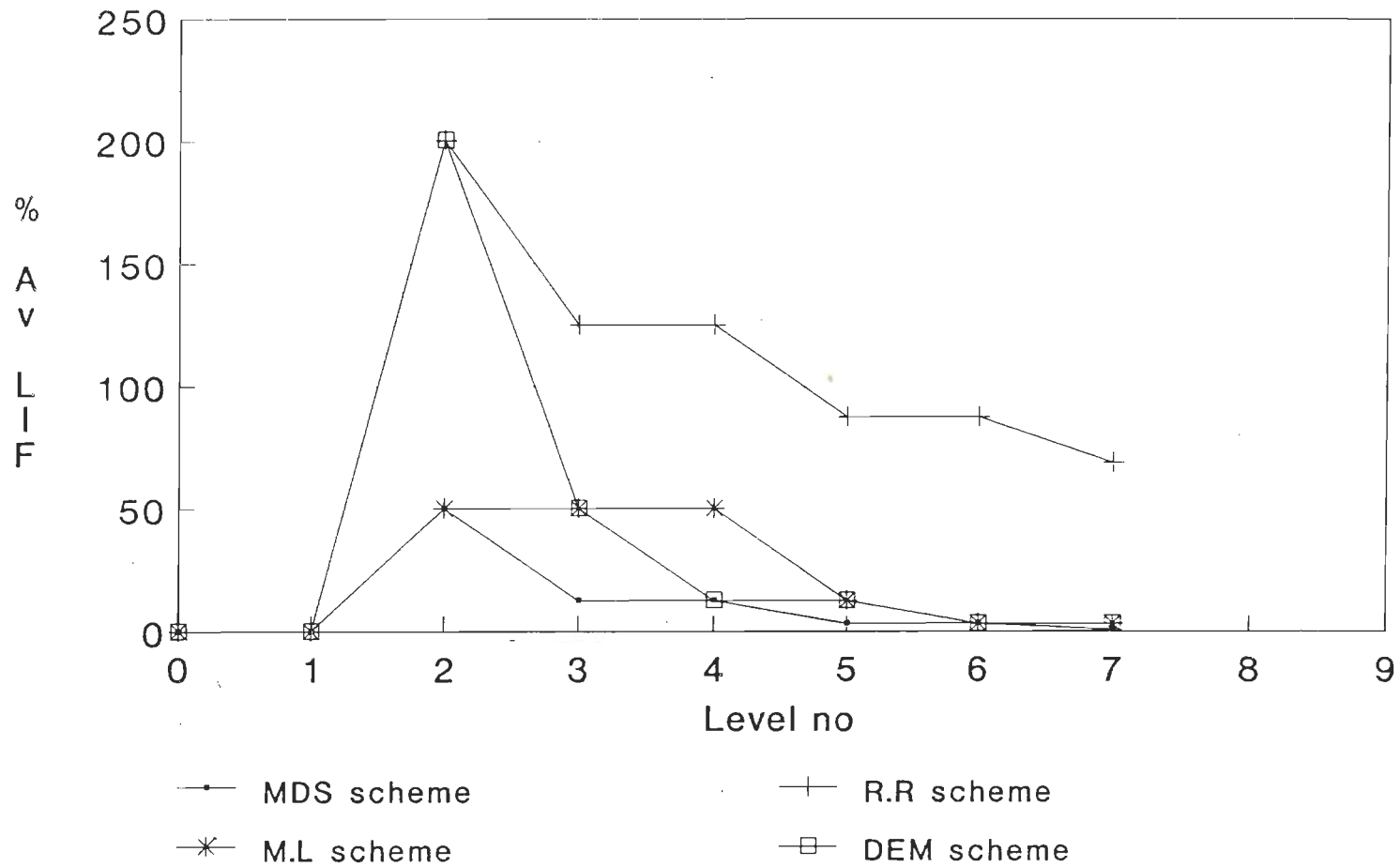


Fig. 5.1 Various scheduling schemes on LET for binary tree

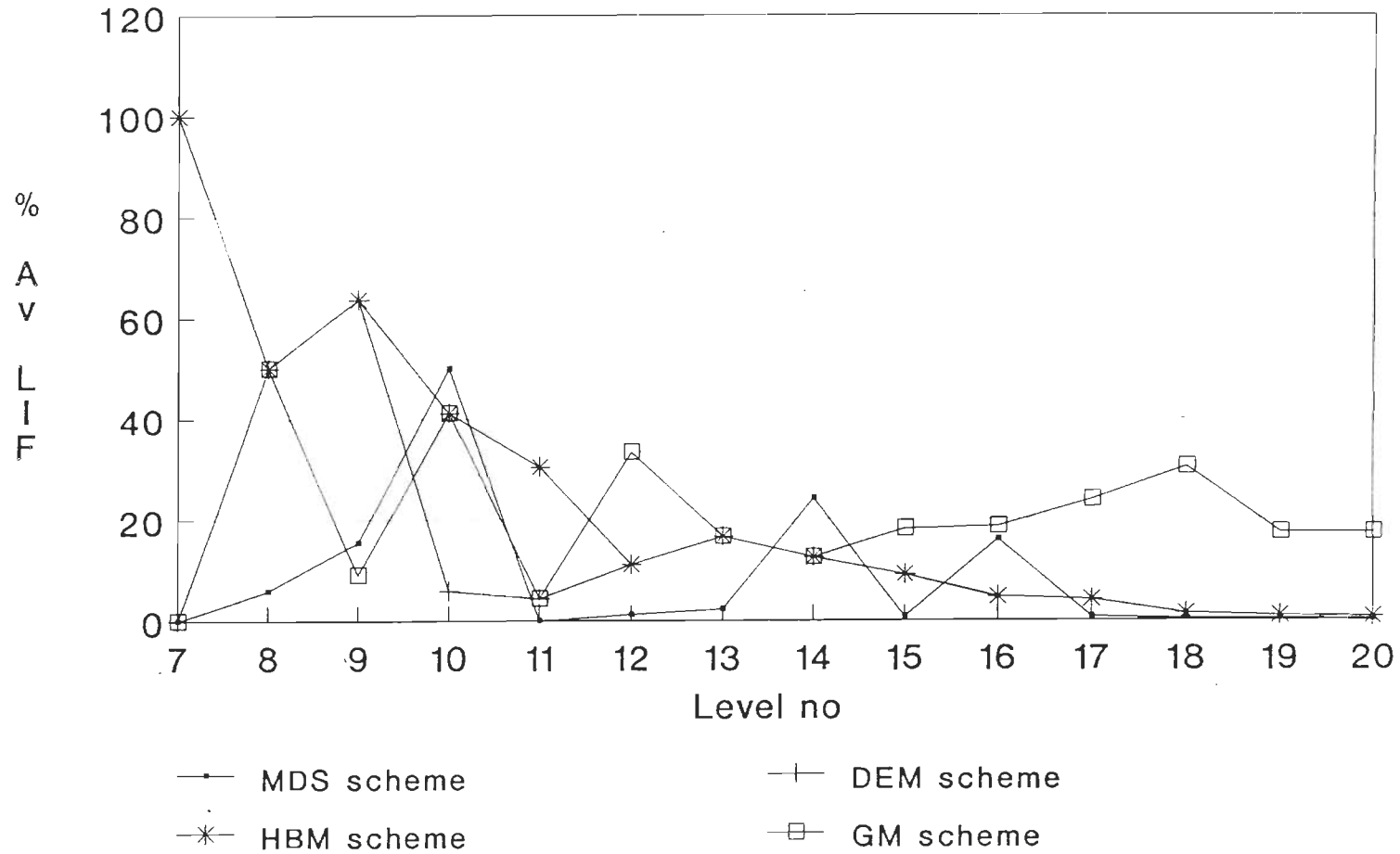


Fig. 5.2 Various dynamic scheduling sch.
on LET for arbitrary binary tree

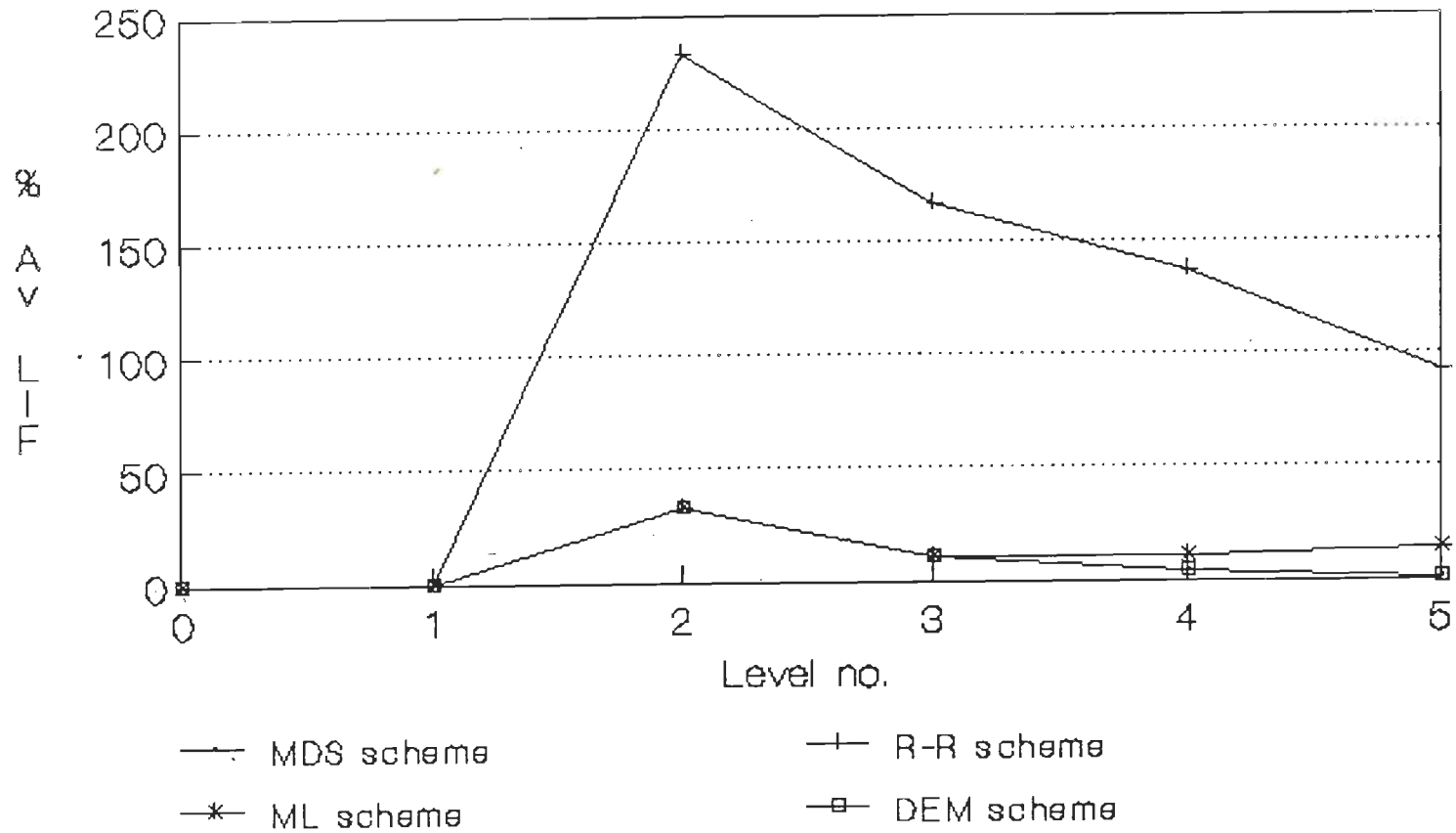


Fig.5.3 Various scheduling sch.
on LET for complete tree.

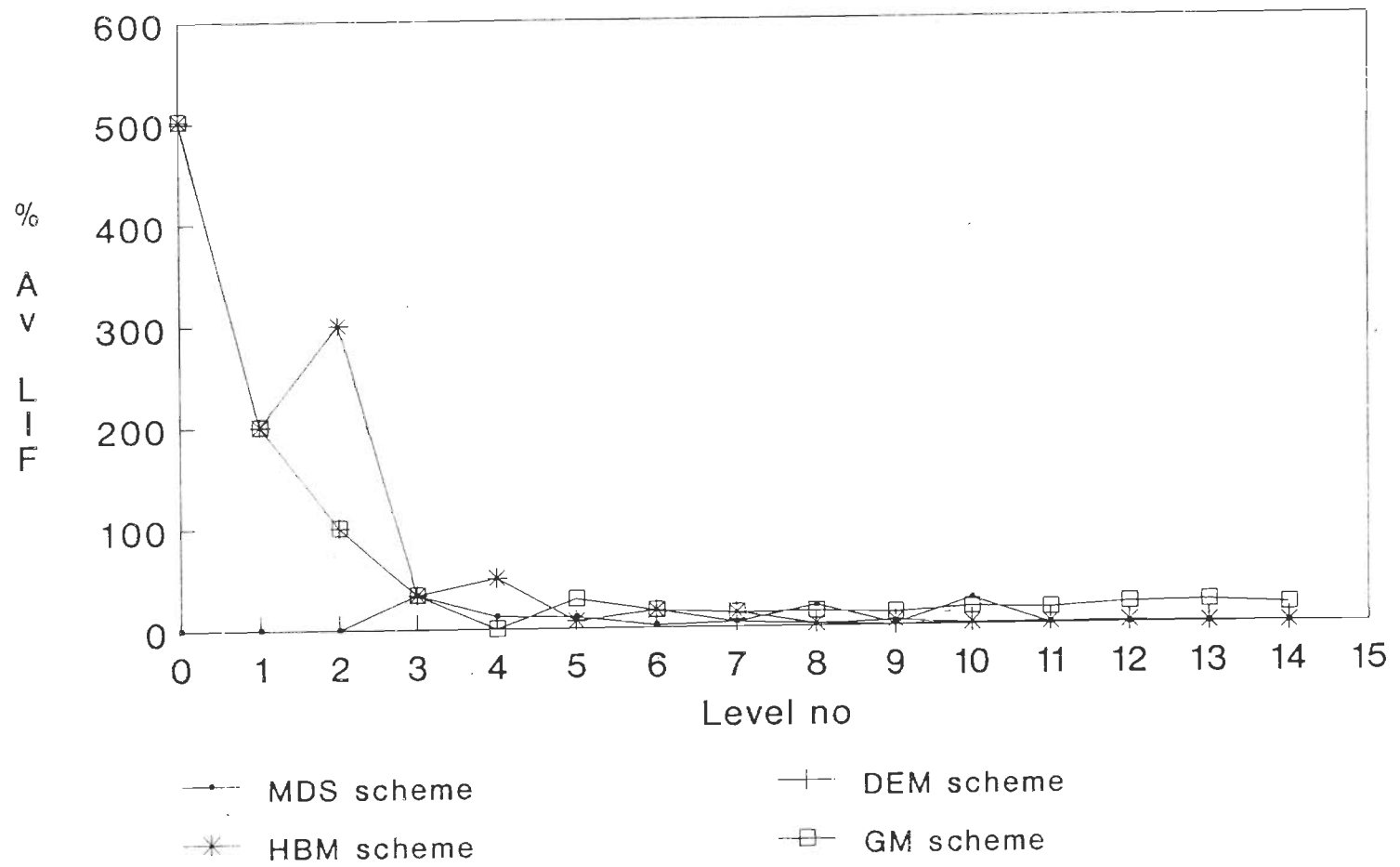


Fig.5.4 Various dynamic scheduling sch. on LET

scheme, which decreases very slowly in comparison to the proposed MDS scheme. On the other hand, the DEM scheduling scheme shows a close behavior with respect to the proposed scheme for complete binary tree as indicated in the Figure 5.1.

For complete ternary task trees, the MDS and DEM schemes show approximately similar trend of curves for all level of the tree as shown in Figure 5.3. The RR and ML schemes once again indicate very high peaks for the same levels of the tree in comparison to the MDS scheme. Figures 5.2. and 5.4., show the behavior of the various schemes for arbitrary binary and arbitrary ternary trees. The average behavior of Load Imbalance Factor for a wide range of randomly generated tree structures shows that the MDS scheme is performing better than rest of discussed static as well as dynamic scheduling schemes at all levels of the trees.

5.3. THE ORGANIZATIONAL MODEL

Architecture design addresses itself to the task of configuring a physical structure that best meets the problem requirements. Since performance of a parallel architecture can be characterised mainly by communication delays, distribution of load among processors and scheduling overheads, a close correspondence between the structure of the problem and the architecture is desired in order to minimise these overheads. Thus, in the context of parallel architectures, this essentially implies the need for the right choice of the interconnection network and a scheduling strategy which together can achieve the mapping of the problem onto the machine so as to minimise overheads [82,83].

The basic objective of the research problem is to develop a good organizational model (suitable topological layout and appropriate

scheduling strategy) which should have a close correspondence with the structure of the problem. If the parallelism is to be worthwhile, the communication and scheduling overheads should be significantly lower. Since communication overheads are directly dependent on the distance through which sub-tasks as well as results have to travel in order to reach their respective destinations, it becomes imperative to minimize the distance between the processor scheduling the sub-task and the processor executing it. However, this can only be achieved by an uneven distribution of tasks among processors. Adopting a centralized scheduling mechanism for achieving as uniform a load distribution as possible would certainly be unacceptable due to the higher overheads associated with such a scheme.

5.4 PERFORMANCE OF VARIOUS ORGANISATIONAL MODELS

To draw general conclusions about the utility of the scheduling scheme and versatility of the interconnection scheme on which it is implemented, it is desirable to experiment on tree-structured tasks that abstract the behavior of parallel programs. A simulation run consists of generating a task tree and executing it on various networks of processors (de-Bruijn, hypercube and proposed networks) under different scheduling schemes, for a fixed set of parameter values. Since a task tree that is randomly generated is but a single instance from a class of trees, the simulation run is repeated over several task trees (30 in case of dense trees and 40 in case of sparse trees), chosen arbitrarily from that class. The respective average LIFs are used as performance indices. In order to understand the behavior of scheduling mechanisms, each parameter is varied independently over a wide range of values. The

entire process is then repeated over several classes of trees by varying the probability distribution associated with the random variables Spawn and Fan out as done in the previous chapter.

To evaluate the relative merits of various scheduling schemes on different architectures, and suitability of one's study, extensive simulation study have been conducted. The static scheduling scheme of Ravikanth et. al. [82,83], onto the eight-processor de-Bruijn multiprocessor for different task trees, to evaluate the Load Imbalance Factor (LIF) and scheduling overheads, results have been obtained. Reeves et.al. [103], applied the dynamic Dimension Exchange Method (DEM) scheduling strategy onto eight-processor hypercube architecture. The DEM scheme is applied on the hypercube using arbitrary tree-structured task trees onto it and different results are obtained. Reddy's [86] static scheduling scheme is implemented on eight-processor de-Bruijn multiprocessor for various tree-type programs, LIF and scheduling overhead are calculated for different level of the task trees. The proposed dynamic scheduling scheme is also implemented on six-processor LET network in the same environment for similar problems and the results for the LIF are obtained.

5.4.1 *Simulation Results*

A comparative pair-wise (scheduling scheme v/s architectures) study has been done by simulating the above schemes onto the corresponding networks. The simulation studies conducted on tree-structured problems in the same environment are presented in Figures 5.5 to 5.8.

The plotted curves are self explanatory, as the study indicate that the load imbalance factor (LIF) approaches zero asymptotically as one

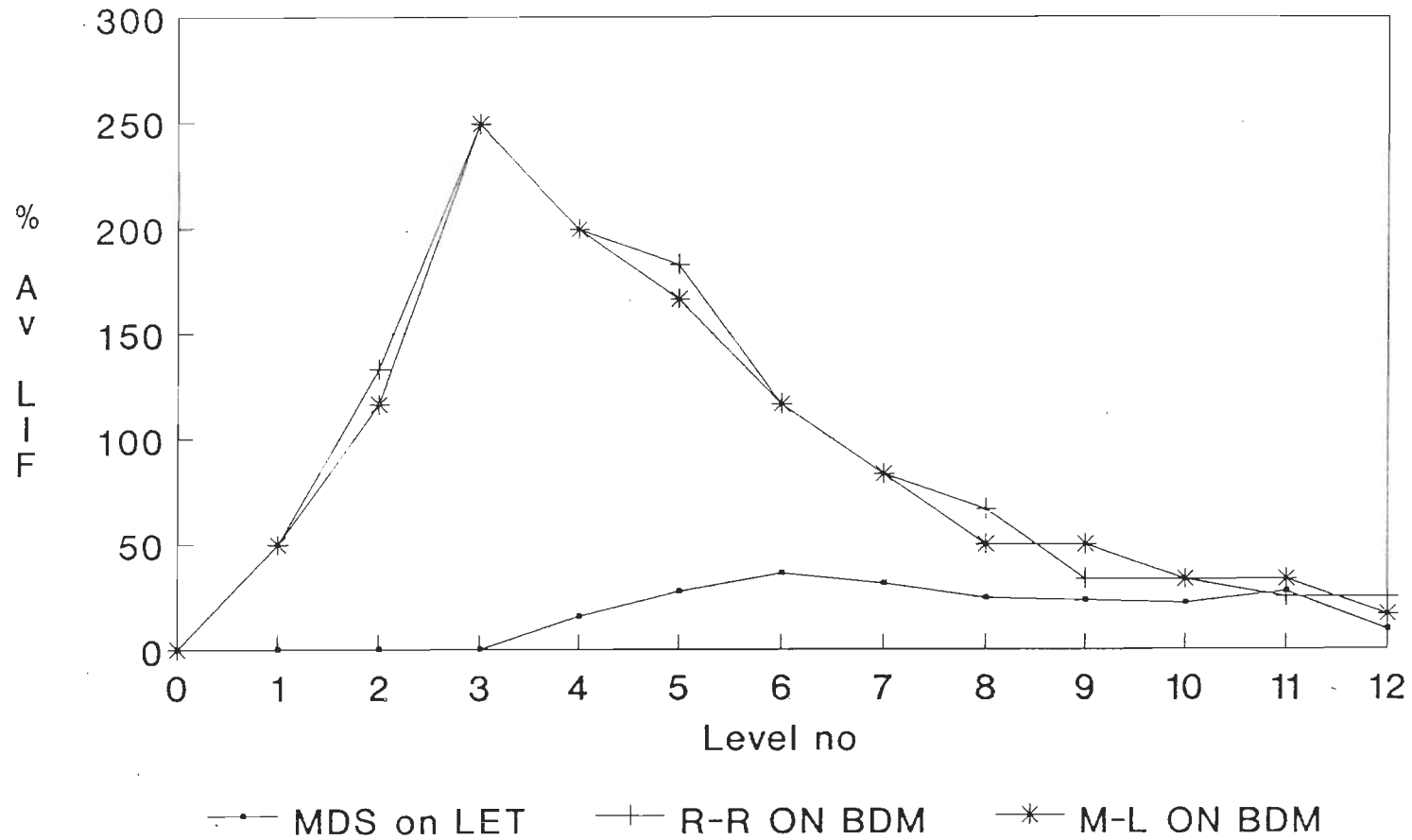


Fig. 5.5. Various static scheduling sch.
 on deBruijn and MDS on LET for
 arbitrary binary tree

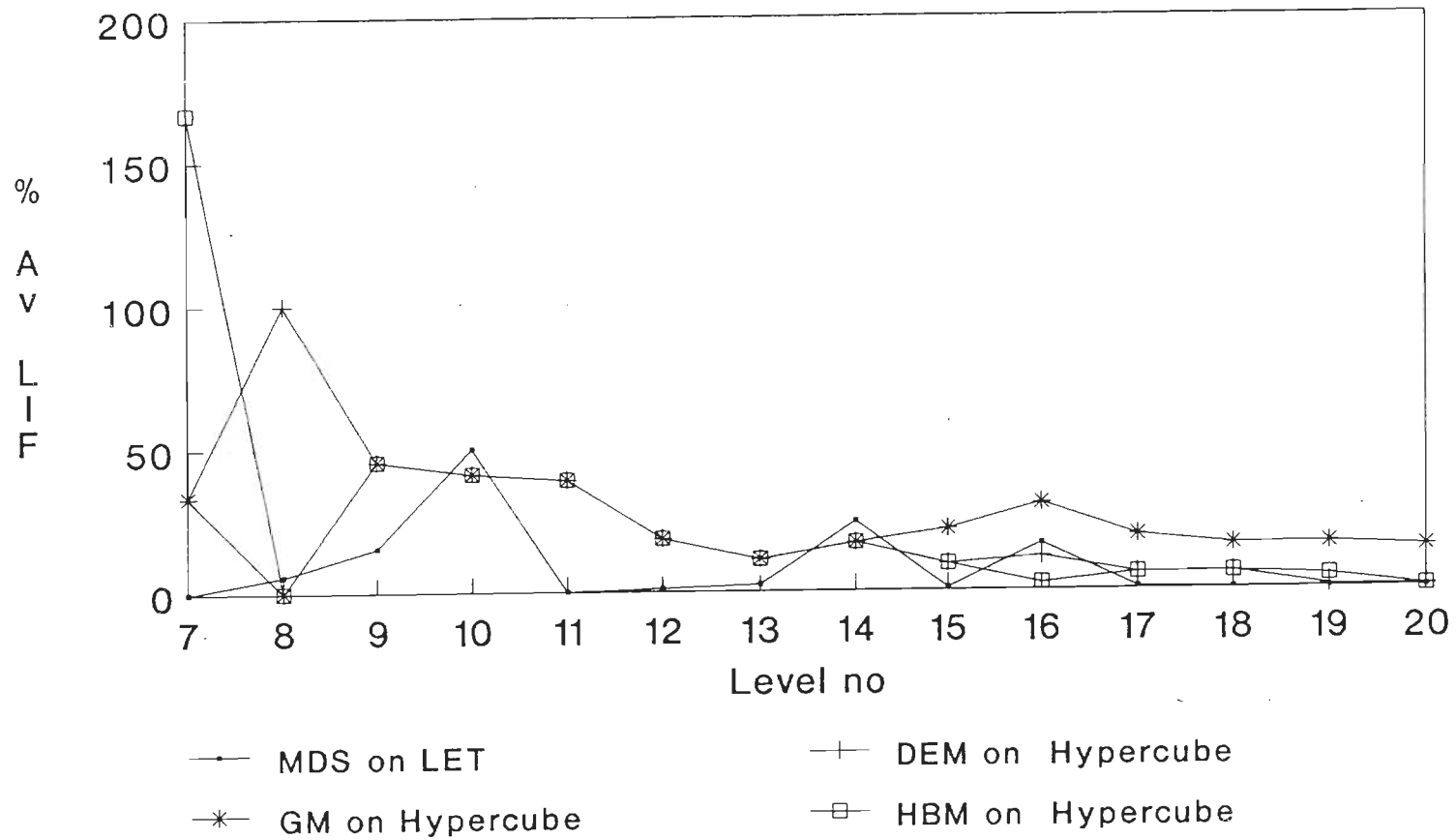


Fig. 5.6 Various dynamic scheduling sch.
on hypercube and MDS on LET for
 arbitrary binary tree

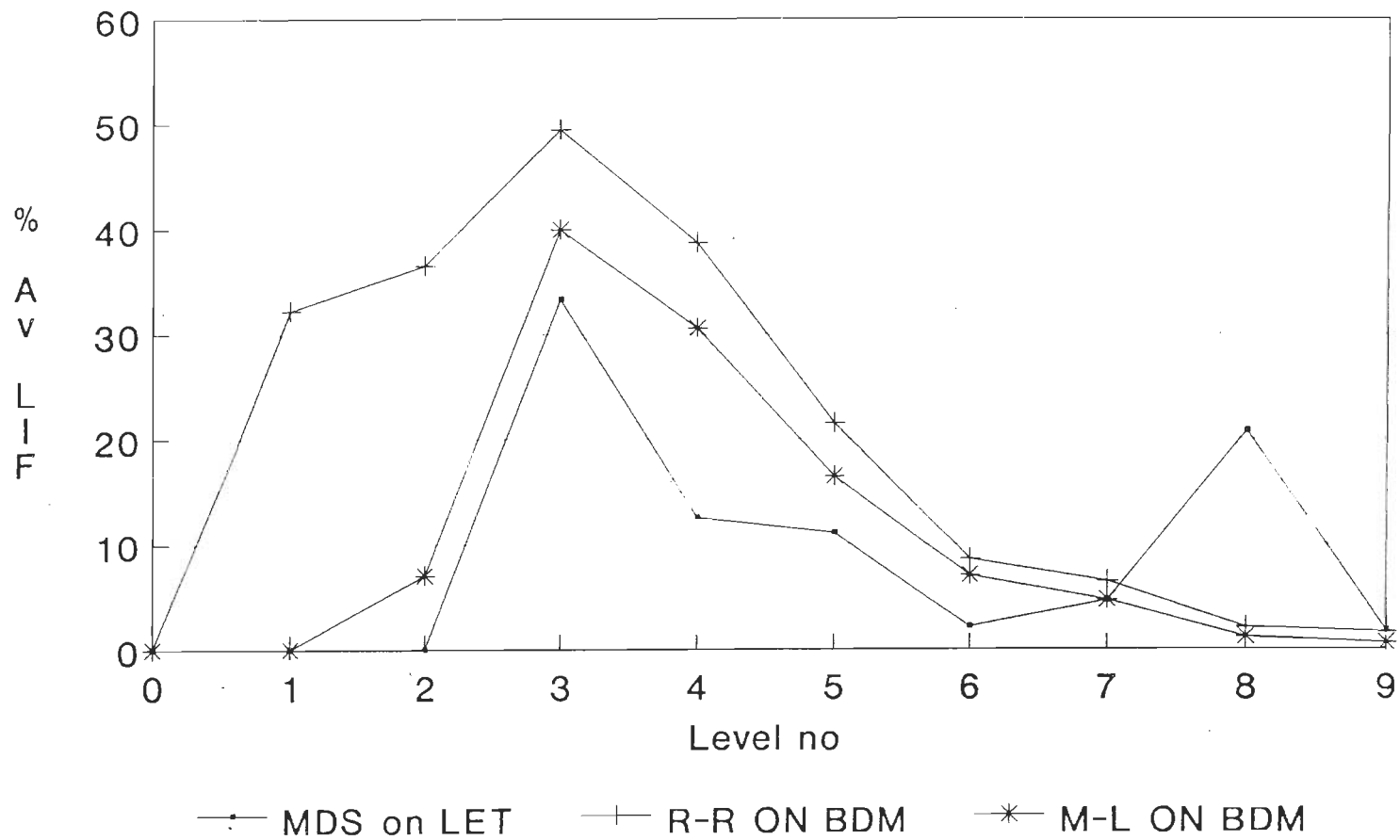


Fig. 5.7 Various static scheduling sch.
on deBruijn net and MDS on LET for
 arbitrary ternary tree

goes down the levels of the task tree. When the size of the task tree is relatively larger than the size of the network, it has been observed that LIF approaches zero much faster. The MDS scheme on the LET network is always having lesser LIF at every level of the task tree in comparison to other organisation models.

The values in Figure 5.8, for arbitrary ternary tree type task graph, show that the proposed organisational model is out performing with the other organisational models at every level of the tree. The LIF is always lesser in comparison to the other models and approaches to zero value quickly.

The curves in Figures 5.6-5.7, show the value of LIF for arbitrary binary task trees for different task levels. This is evident that the number of tasks, in those levels of the graphs where the average LIF shows a rising trend, is less than the number of processors in the network. The load imbalance factor starts falling in these graphs once sufficient number of tasks are available in the network. It is also observed that for problems having a high degree of parallelism the LIF approaches zero quickly. The curve plotted for the proposed study is performing better in comparison to de-Bruijn and hypercube.

It has been observed that the LIF shows a similar behavior in all these cases, rising initially from zero to peak and then reducing to zero asymptotically. For every level of the task tree, average LIF for the proposed study indicate the superior values, i.e. lesser values at every point as well as lesser value of LIF at peaks with respect to other studies. The maximum value of the average LIF is half to the others values. Since the proposed organisation model i.e. the LET network and MDS scheme, is having a closed correspondence to the

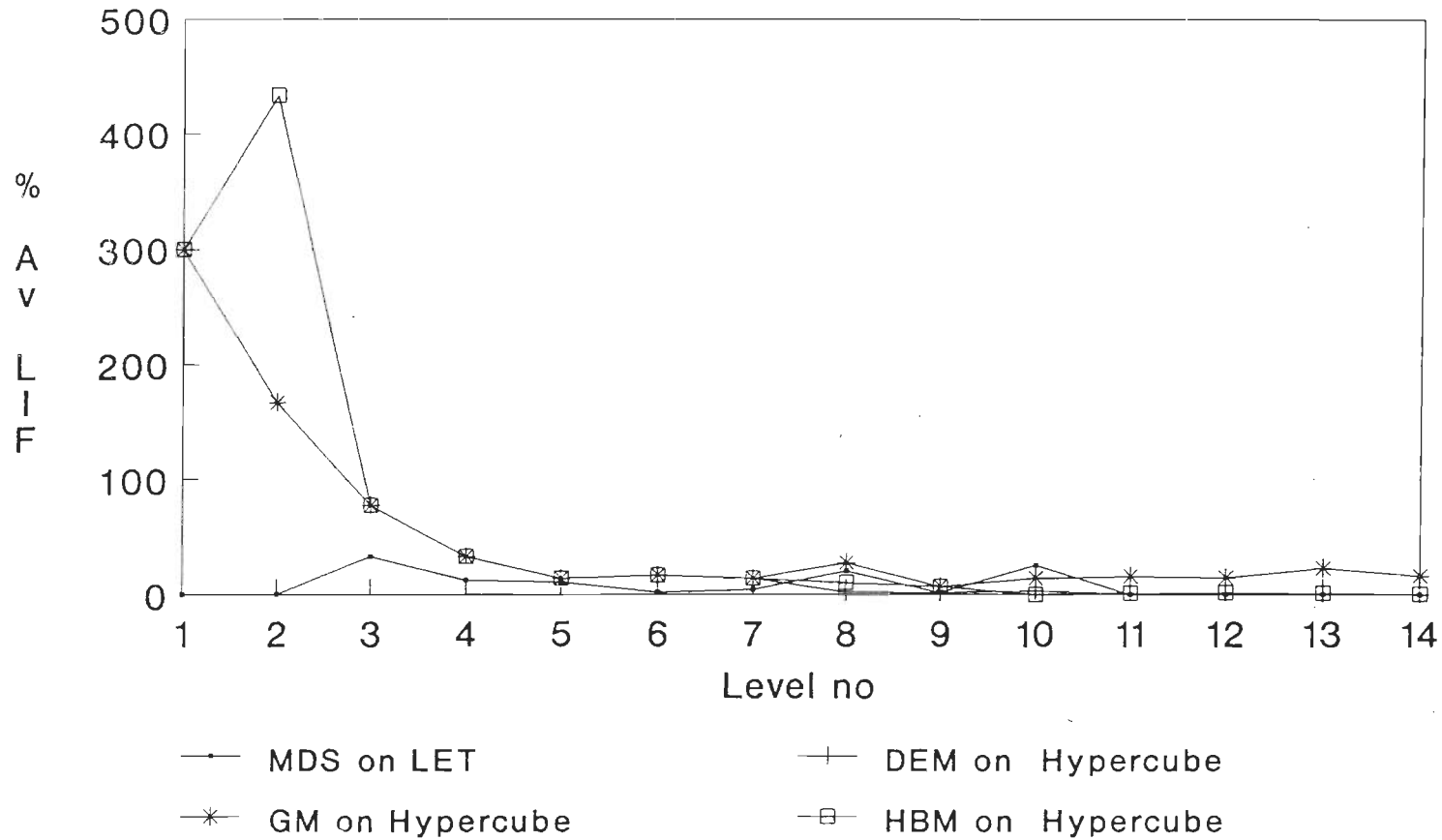


Fig. 5.8 Various dynamic scheduling sch. on hypercube and MDS scheme on LET for arbitrary ternary tree

structure of the problem for which the simulation studies are carried out. Heuristically, a good organizational model (suitable topological layout and appropriate scheduling strategy) is one which has a close correspondence to the structure of the problem. Motivated by this heuristic and dynamic behavior of highly parallelized programs, a network topology has been proposed and considered for performance evaluation. A simple scheduling mechanism has been developed for mapping the problem structures onto the network.

It is necessary to consider the design issues falling in the realm of parallelism, while implementing parallel programs. Efficient management of parallelism involves optimizing conflicting performance indices, like minimization of communication and scheduling overheads and even distribution of load among the processors in the network. A judicious choice of topological layout for independent processing elements and a scheduling strategy improve the performance of a system implementing tree structured programs.

To justify that the choice of proposed architecture is on sound footing, its performance is compared with that of Hypercube and de-Bruijn architectures. Complete, arbitrary binary and ternary task trees have been used as test problems. From simulation results, it has been observed that the proposed architecture with proposed scheduling mechanism performs reasonably well for all classes of task structures.

In the next chapter the performance of LET network for graph structure problems has been shown. Also, a modified form of Dynamic Level Scheduling (DLS) algorithm has been applied on LET, binary deBruijn network and Hypercube networks.

PERFORMANCE OF LET FOR GRAPH-STRUCTURED PROBLEMS

In the previous chapters, the performance of LET network has been discussed for tree structured problems in terms of load balance obtainable. Another important class of problems is graphs. Problems having precedence-graph structures frequently appear in digital signal processing systems. In this chapter, the performance of LET network is evaluated for a specific type of graph problems viz. Acyclic Precedence Graphs (APG). A static scheduling algorithm for APG's called Dynamic Level Scheduling (DLS) was proposed in [95]. In the present work, a modified form of this algorithm has been applied on LET, binary deBruijn and hypercube networks and the results show an overall better performance of LET network in terms of speedup and processor utilisation.

6.1 INTRODUCTION

A parallel program is a collection of separate cooperating and communicating modules called *tasks*. Such a program may be represented as a precedence graph $G = (T, E)$, where $T = \{T_i : i = 1, 2, \dots, n\}$ is a set of nodes (tasks) representing a group of sequential computations and E is the set of directed edges. If there is a directed edge $E(i, j)$ from task T_i to T_j , then it implies that T_j can not execute until T_i completes. Such dependencies are due to data or control. In either case, the task graph establishes precedence relations and data paths among all tasks [95]. Figure 6.1 shows a sample graph. Labels on edges indicate the

amount of data to be sent between the two tasks. The weights of the node would be used to indicate execution time of the corresponding task. In the discussion to follow, the network is assumed to be homogeneous.

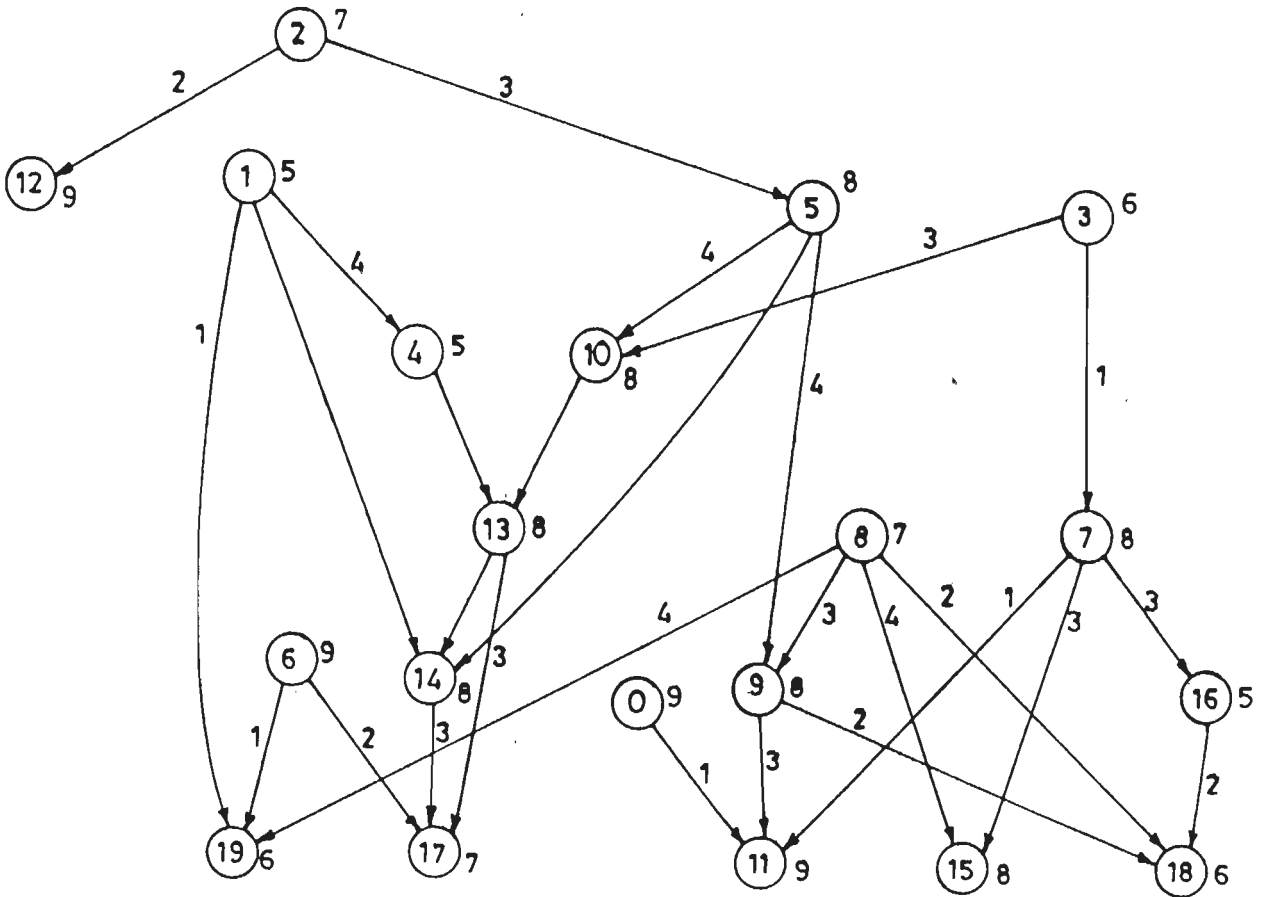


Figure 6.1 Sample Graph

If two communicating tasks are scheduled onto the same processor, the communication delay is zero; otherwise, the delay equals the message start up time, plus the data size divided by the transfer rate. A process start time is dictated by the communication delay, which is also a function of the number of hops from one processor to another along the

interconnection network. The algorithm will again work under minimum distance constraint as in the case of dynamic scheduling of tree structured problems.

A scheduling strategy in this case is usually designed with one or more of the following objectives:

- 1) minimum schedule length,
- 2) minimum inter processor communication (IPC) overheads
- 3) minimum load balance among the processors.

Schedule length is the total time required to execute the program on N processors. Interprocessor communication is required when two tasks having precedence edge are scheduled on two different processors. IPC overheads are a major factor that limits the schedule length. Load balance implies uniform distribution of computation load among processors. It is always better to keep the processors equally busy. These objectives obviously have conflicting demands.

Performance evaluation in this case is being done in terms of speedup and efficiency. Speedup is the ratio of execution time of program on a single processor to the execution time on N processors.

$$S(N) = t_1/t_N$$

Efficiency is a measure of average processor utilisation and hence the load balance. It is defined for a N processors networks as,

$$E(N) = S(N)/N$$

6.2 LATEST PRECEDENCE SCHEDULING STRATEGY

The scheduling scheme described here is primarily based on list scheduling technique and is modification of the work of Sih and Lee

[95], on a compile-time scheduling strategy for APG's. This algorithm runs faster than the one suggested by Sih and Lee but it does the same work. A description of the development follows.

6.2.1 List scheduling

In list scheduling each task node in problem graph is assigned a priority (according to some criterion). The tasks are then arranged in decreasing priority order. A task is considered ready for execution if all its immediate predecessors have been executed. Whenever a processor is available, a ready task with highest priority is assigned to it. In case of a tie, some tie-breaking mechanism is invoked.

Different schemes in this class differ only in the way the priorities are assigned to tasks. HLF algorithms [1,54] have priorities based on static levels. The static level of a task T_i is defined as the largest sum of execution times along any directed path from T_i to a terminal node, the maxima being taken over all terminal nodes (critical path length). Thus if $SL(T_i)$ denote static level of T_i , S_i the set of successors of T_i and E_i the execution time of T_i then

$$SL(T_i) = E_i + \max_{S_i} (SL(\text{succ}(T_i))) \quad \dots(6.1)$$

HLF algorithms have been demonstrated to have near-optimal performance when IPC costs are not included.

6.2.2 HDLF algorithm

The HLF algorithm does two things at each scheduling step. It selects the next task to schedule, and chooses the processor to execute

it on. These selections are done independently on each step based on static levels, causing poor performance in the presence of IPC. Sih and Lee [95] introduced the idea of dynamic level and thus the Highest Dynamic Level First (HDLF) algorithm.

HDLF algorithm not only changes the task selection criterion but goes further to suggest that a busy processor at a scheduling step need not be kept out of the available processor list. In fact it says that all processors be considered as candidates for scheduling. A processor can start execution of a task T_i when it has finished execution of the last node assigned to it or when all the data required for T_i is available at it, whichever is later. This idea is used in expressing the dynamic level $DL(T_i, P_j)$ of a task-processor pair as

$$DL(T_i, P_j) = SL(T_i) - \max\{DA(T_i, P_j), TF(P_j)\} \quad \dots(6.2)$$

Here $DA(T_i, P_j)$ is the earliest time at which data required by T_i from its predecessors is available at P_j and $TF(P_j)$ is the time that P_j finishes execution of its last assigned node. Dynamic level takes IPC into account and indicates the matching of a task and processor. In task selection for scheduling, a large static level is desirable because it indicates a high priority for execution. In selecting processor, an earliest start time (the second term of the right hand side of Eq. 6.2) is desirable irrespective of whether processor is busy or idle. Thus a task should be scheduled on a processor for which dynamic level is highest.

The modified form of HDLF (called Dynamic Level Scheduling-DLS) studied two cases viz.

- 1) Select a ready task with highest static level and schedule it on the processor that maximises its dynamic level.
- 2) Examine all ready task and processor pairs and choose the task-processor pair with highest dynamic level.

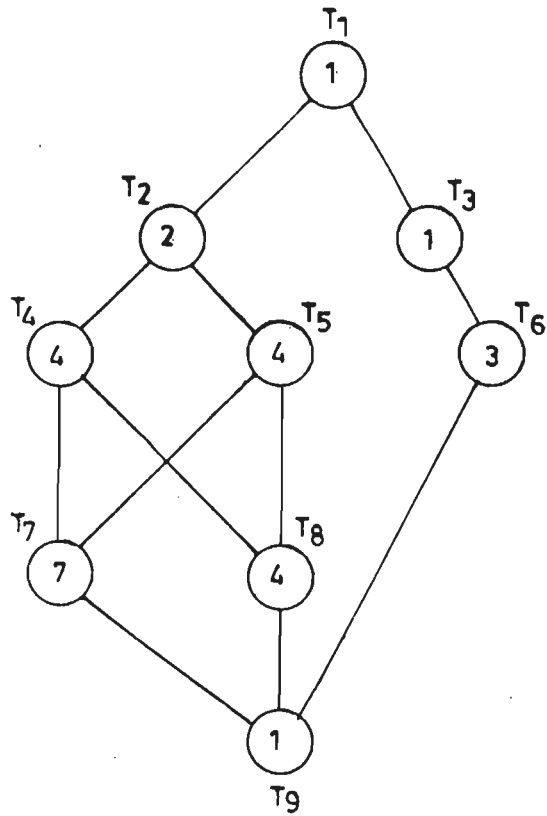
Obviously, computation complexity is much less in case 1) than in case 2). The DLS algorithm has been reported to show better performance compared to HDLF mainly due to greater freedom in processor selection.

6.2.3 Task selection

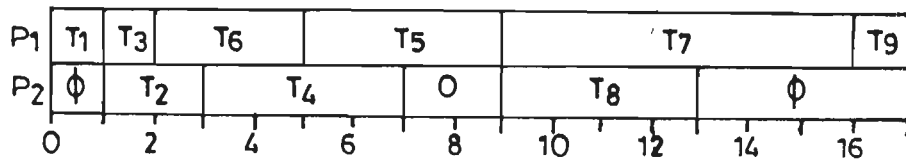
In the study of deterministic scheduling models [46], it has been shown that task selection is an important factor in determining the optimal schedule for a task graph. The obvious choice of taking up a ready task on priority basis does not always yield on optimum schedule. Figure 6.2 (a) shows a task graph. For simplicity, communication costs have been ignored here. The figures inside circles indicate the execution times of the tasks.

If policy of scheduling a ready task, as soon as possible, is followed then the schedule obtained for two processors is shown in Figure 6.2 (b) as a Gantt chart. The schedule length is 17 and processor utilisations are P_1 : 88.3%, P_2 : 70.6%. The speedup obtained is $27/17 = 1.59$ for 2 processors. In contrast to this, if the scheduling of task T_6 is postponed until T_7 and T_8 are scheduled, we get the schedule shown in Figure 6.2 (c). Here the speedup is $27/15 = 1.8$ and processor utilisations are P_1 : 93.3% and P_2 : 86.7%.

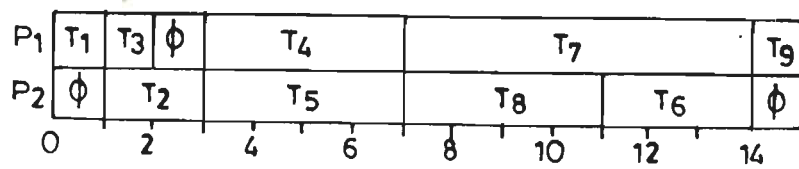
The idea is that the activation of T_6 does not generate any extra work whereas activating T_4 and T_5 can make more tasks ready. Thus the tasks at higher levels (distance from terminal nodes) should get



(a) Task graphs for latest precedence partitioning



(b) Schedule with eager allocation of tasks



(c) Optimal schedule

Figure 6.2 Two processor Gantt Charts for the graph G

preference in scheduling so as to keep on generating maximum possible work at each scheduling step.

From the above discussion, it can be concluded that all tasks at the same level are candidates for simultaneous execution or group scheduling. An appropriate grouping of tasks can be achieved by latest precedence partitioning of the task graph. Latest Precedence Level (LPL) of tasks may be determined as follows:

$$\begin{aligned} \text{LPL}(T_i) &= 1, & T_i \text{ is a terminal task} \\ &= 1 + \max_{S_i} \{ \text{LPL}(\text{succ}(T_i)) \}, & \text{otherwise} \end{aligned} \quad \dots (6.3)$$

where S_i is the set of successors of task T_i .

Sih and Lee [95] have also indicated performance gain through task selection. However, in their approach, task selection is done from among all tasks whereas we propose grouping of tasks through latest precedence partition and then choosing a task from a group only. This makes the algorithm run faster.

Based on the developments described above, a static scheduling algorithm for APG's has been designed which is being called Latest Precedence Scheduling algorithm taking into account its latest precedence partitioning of graphs for task selection. The algorithm described in next section, incorporates minimum distance property also.

6.3 LATEST PRECEDENCE SCHEDULING ALGORITHM

The algorithm incorporates the following :

- i) choosing a task for scheduling from amongst those at latest precedence level, and

ii) choosing a processor, from amongst minimum distance processors irrespective of whether it is busy or idle.

The input is a precedence graph $G = \{T, E\}$. Tasks have arbitrary execution time and edges have labels indicating the amount of data transfer requirements. Static levels as per Eq. (6.1) are assigned to all tasks and the whole graph is divided into, say L , latest precedence levels. Tasks in one group of precedence are sorted in decreasing order of selection priority given by

$$SP(T_i) = SL(T_i) + C(\max_k (D_{ki})) \quad \dots (6.4)$$

Here D_{ki} represents the number of data units to be obtained by T_i from its predecessor before it can execute and $C(D)$ is the cost of IPC in terms of time units needed for this transfer. Selection of ready nodes is based on this priority which takes IPC into account.

Processor selection is done from a set of minimum distance processors (MDP) defined as follows:

If a task T_i is to be scheduled and T_k is its immediate predecessor scheduled on processor P_k , then MDP consists of P_k and all other processors connected directly to it in the multiprocessors network. If T_i has a number of predecessors then MDP is obtained by taking the union of all the MDP's corresponding to each predecessor.

Having decided the members of MDP, a task T_i is scheduled on the processor with earliest start time (second term of the right side of Eq. 6.2). The complete algorithm is described formally below in "C" like notation:

```

for all tasks find static level  $SL_i$ ;
partition input graph giving L levels of latest precedence;
for (level = L; level > 0; level--) {
    Find selection priority  $SP_i$  for each task;
    Sort the tasks in decreasing order of selection priority;
    for each task at current level {
         $T_S$  = task with highest SP and not yet scheduled;
        Find MDP for  $T_S$ ;
        for each processor  $P_i$  in MDP
            Find earliest start time ;
        Schedule  $T_S$  on processor with minimum earliest start time;
    }
}

```

The above algorithm has been implemented for LET, binary deBruijn and hypercube networks. The next sections deal with this experimentation and the performance results for various networks.

6.4 GRAPH GENERATION

This section deals with the generation of an arbitrary graph of the tasks with given specifications. These specifications include the number of nodes N in the graph, maximum in degree and maximum out degree for each node, range of execution time for nodes and the range of weights to be assigned to links. Following is the algorithm for graph generation :

```

Initialize all the N nodes of the graph;
  for each node, {
    execution time of the node    random number in execution time
                                  range;

    if(node has no parents)
      outdegree = random number between 1 and max. outdegree;
    else
      outdegree = random number between 0 and max. outdegree;
    if (outdegree > number of nodes left beyond current node with
        indegree not full)
      adjust outdegree;
    while (outdegree > 0){
      childnode = random number between next node and N-1;
      if (child node is not already a child of current node and
          its parent count < max. indegree){
        make a link between current node and child node;
        link weight of this link = random number in link weight
                                  range;
        increment child count of current node;
        increment parent count of child node;
        decrement outdegree;
      }
    }
  }
}

```


6.5. PERFORMANCE OF LET AND OTHER NETWORKS

The above discussed scheduling scheme i.e. latest precedence scheduling scheme has been implemented on the hypercube, de-Bruijn and the proposed LET networks to obtain various performance parameters viz. LIF, speedup and efficiency. The parameters varied to obtain the results on the networks include number of tasks in a graph, communication cost, and indegree and outdegree of the task nodes. The results are plotted in Figures 6.3 - 6.10 for the graph problems.

Figures 6.3 -6.5 depict the effect of changing the value of the degree of the graph, other parameters remaining constant. The ratio indegree / outdegree is varied. It is observed that LET network gives minimum value of LIF, higher efficiency and comparable speedup as the degree of the graph is increased.

Figures 6.6 - 6.8 show the effect of increasing problem size i.e. the number of tasks in the graph. The result indicates that as the number of task in the graph is increased, the LIF is minimum for LET in comparison to other networks. The efficiency is maximum for LET and reaches to a high value of above 80% only when the number of task in the graph is 50, whereas speedup is increasing steadily.

In other graphs of Figures 6.9 - 6.10, it is observed that as the E/C ratio is increased, both speedup and efficiency improve better in LET network in comparison to other networks. This is due to increased overheads at lower values of E/C ratio. From the above, it can be concluded that for less parallelism, LIF is high and efficiency is low and in contrast to it, with more parallelism, LIF is low and efficiency is high.

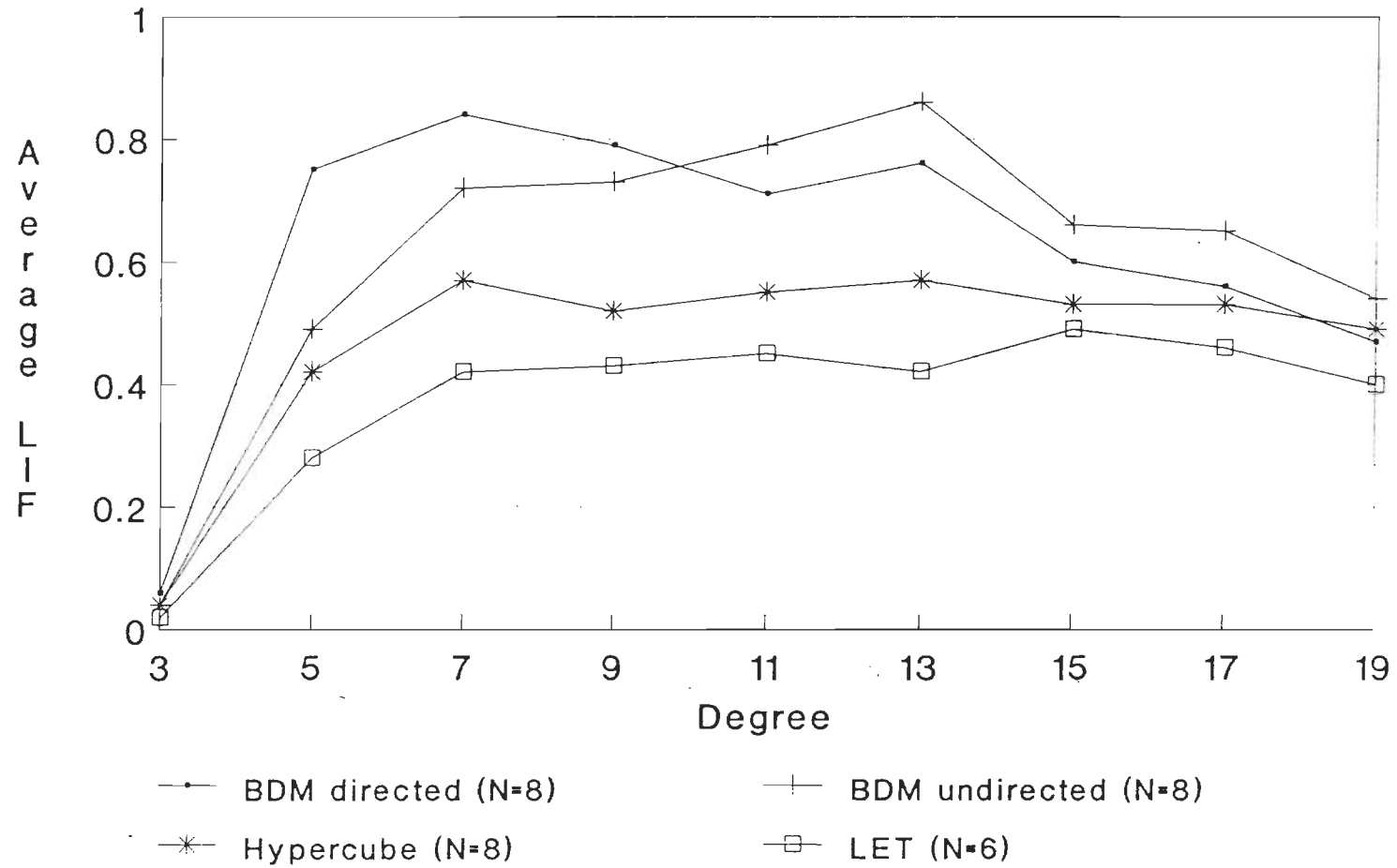


Fig. 6.3 LIF of various networks for graph structured problems

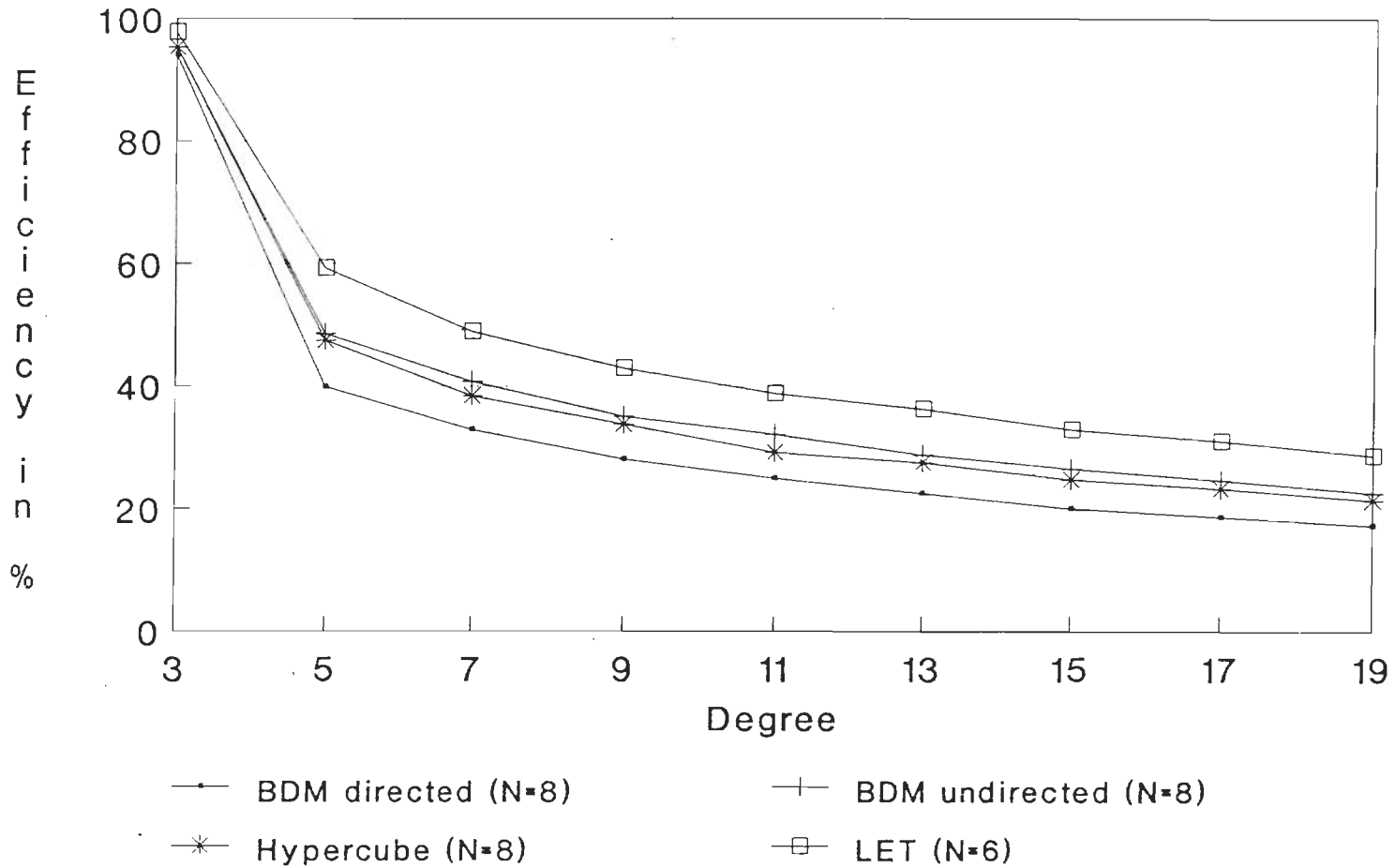


Fig. 6.4 Efficiency characteristics of various networks

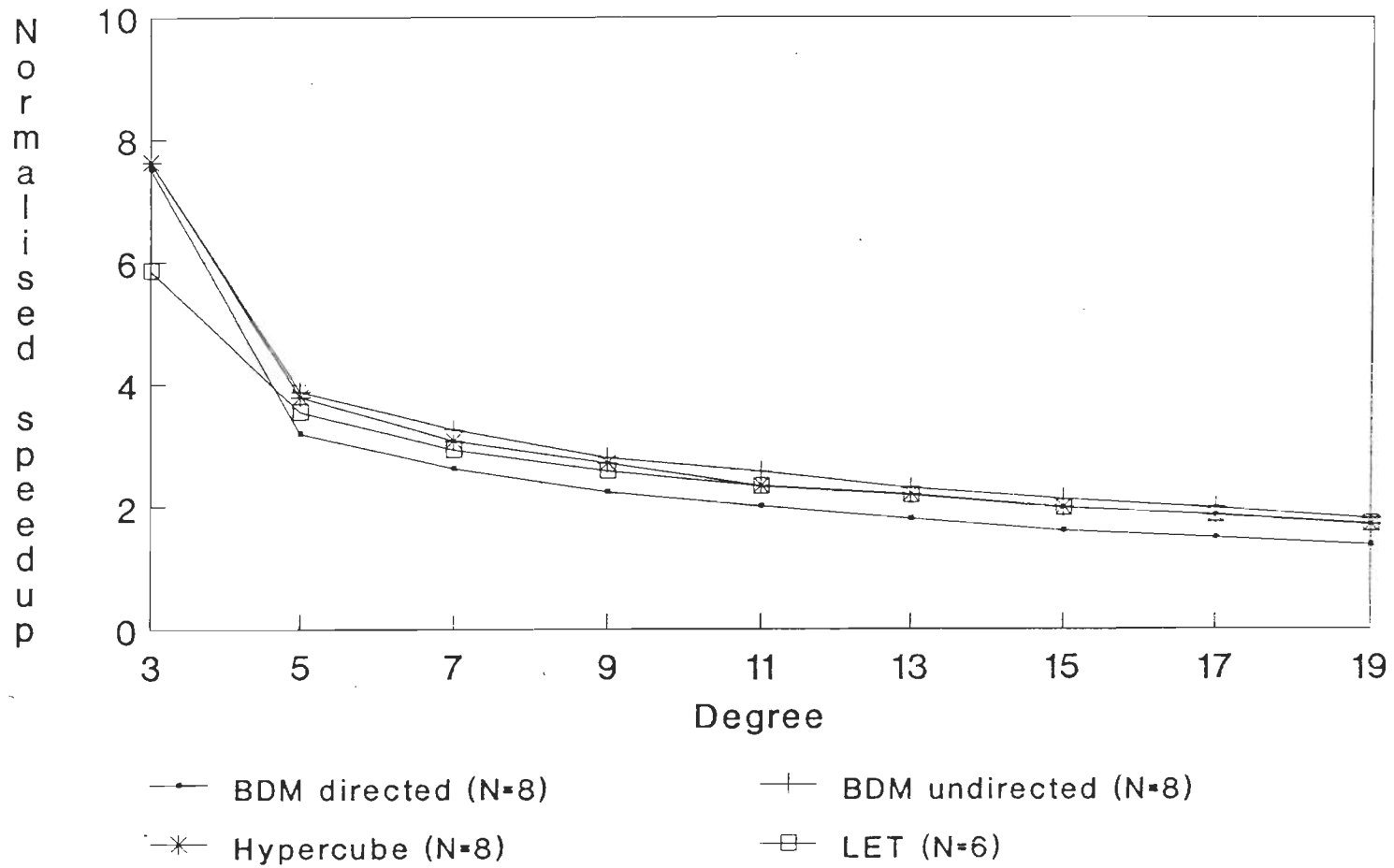


Fig. 6.5 Speedup characteristics of various networks

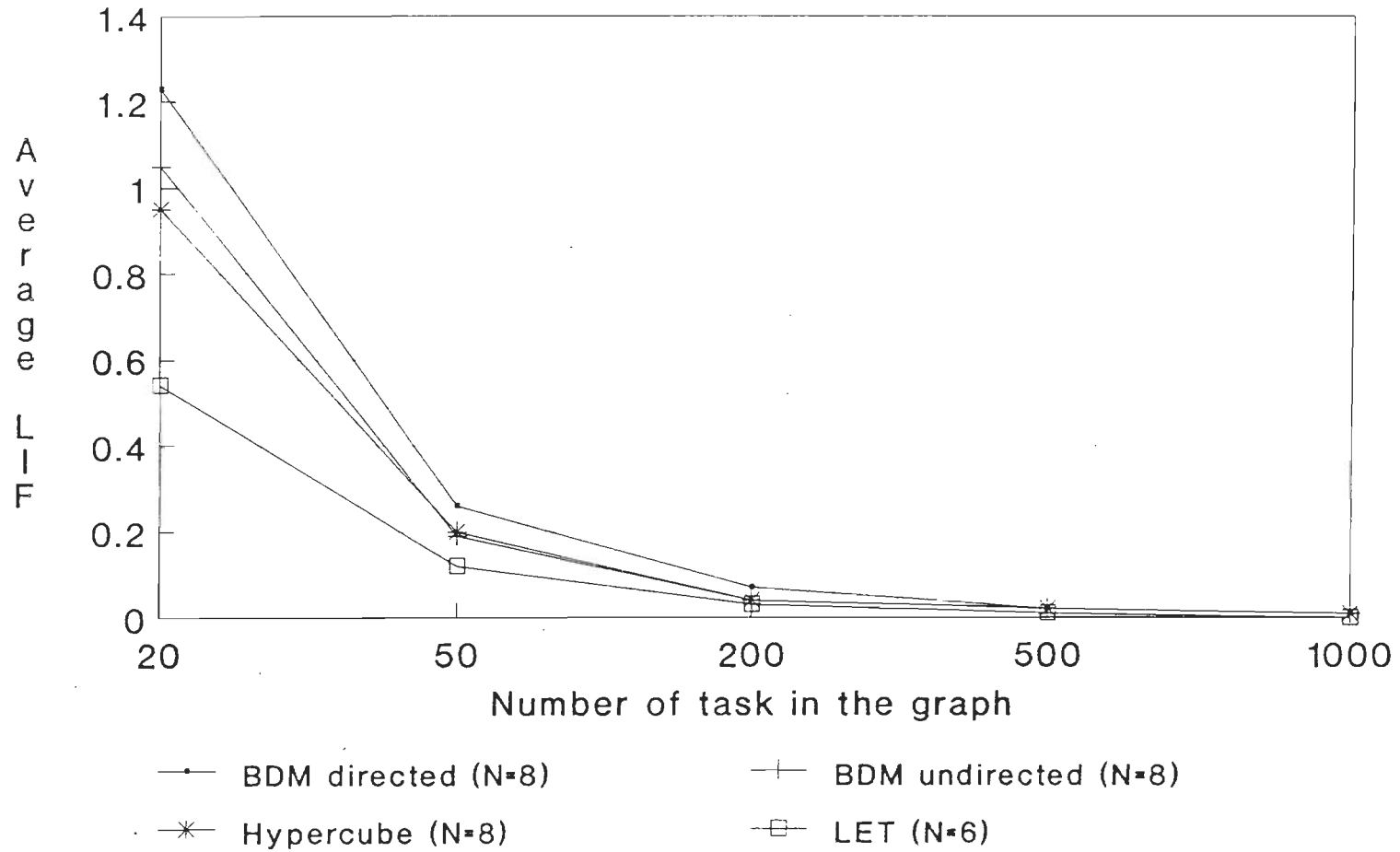


Fig. 6.6 LIF of various networks for graph structured problems

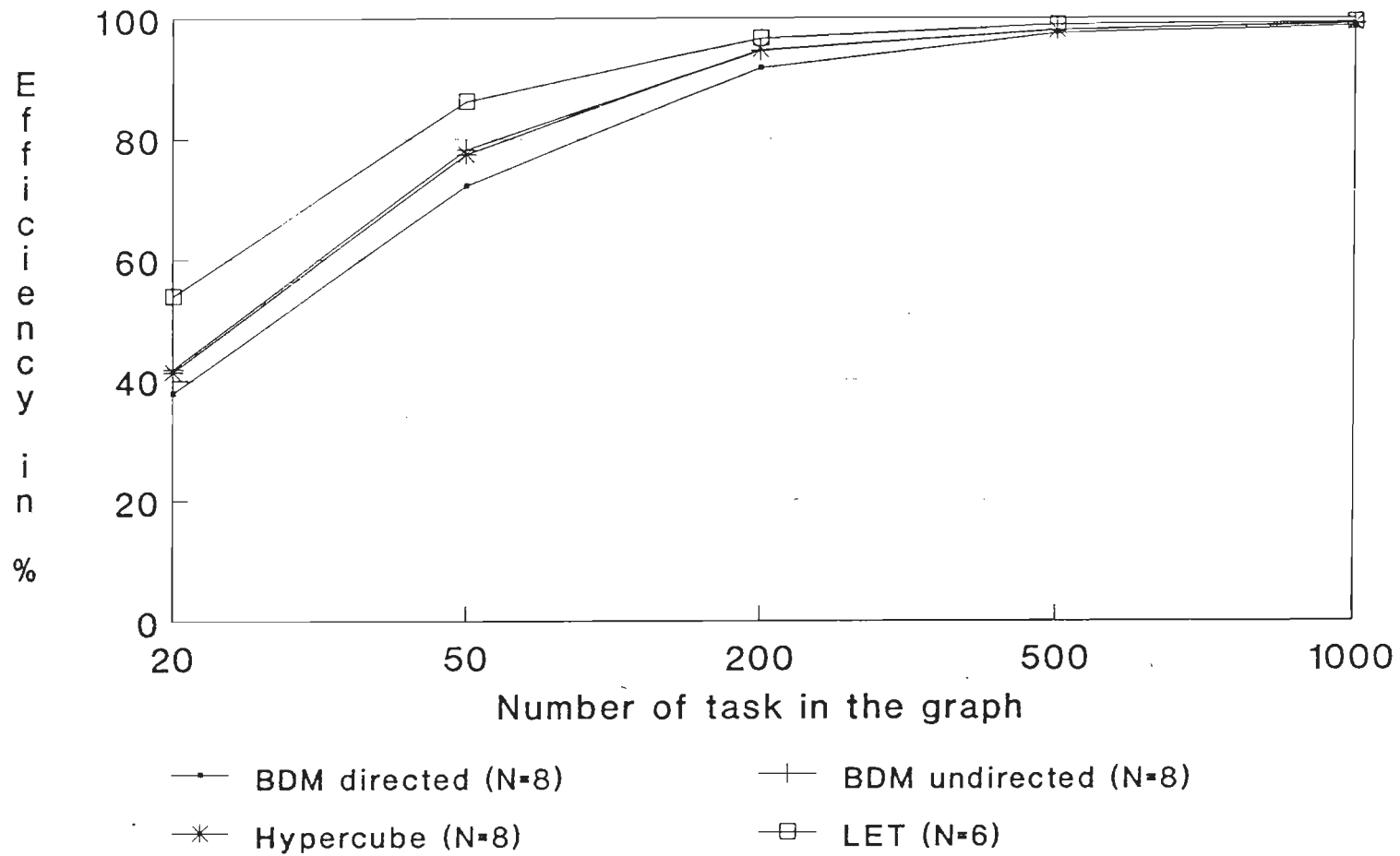


Fig.6.7 Efficiency characteristics of various networks

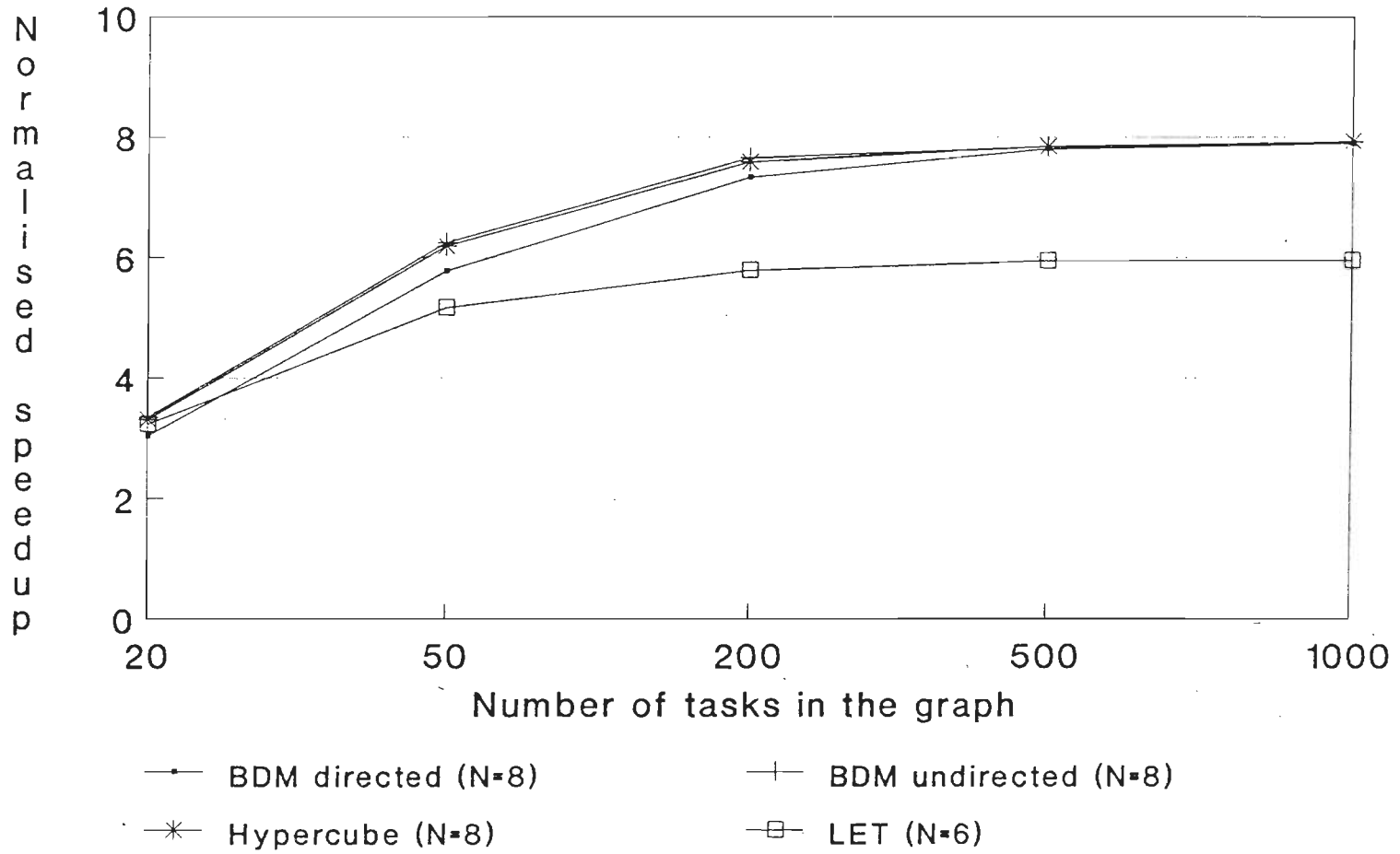


Fig 6.8 Speedup characteristics of various networks

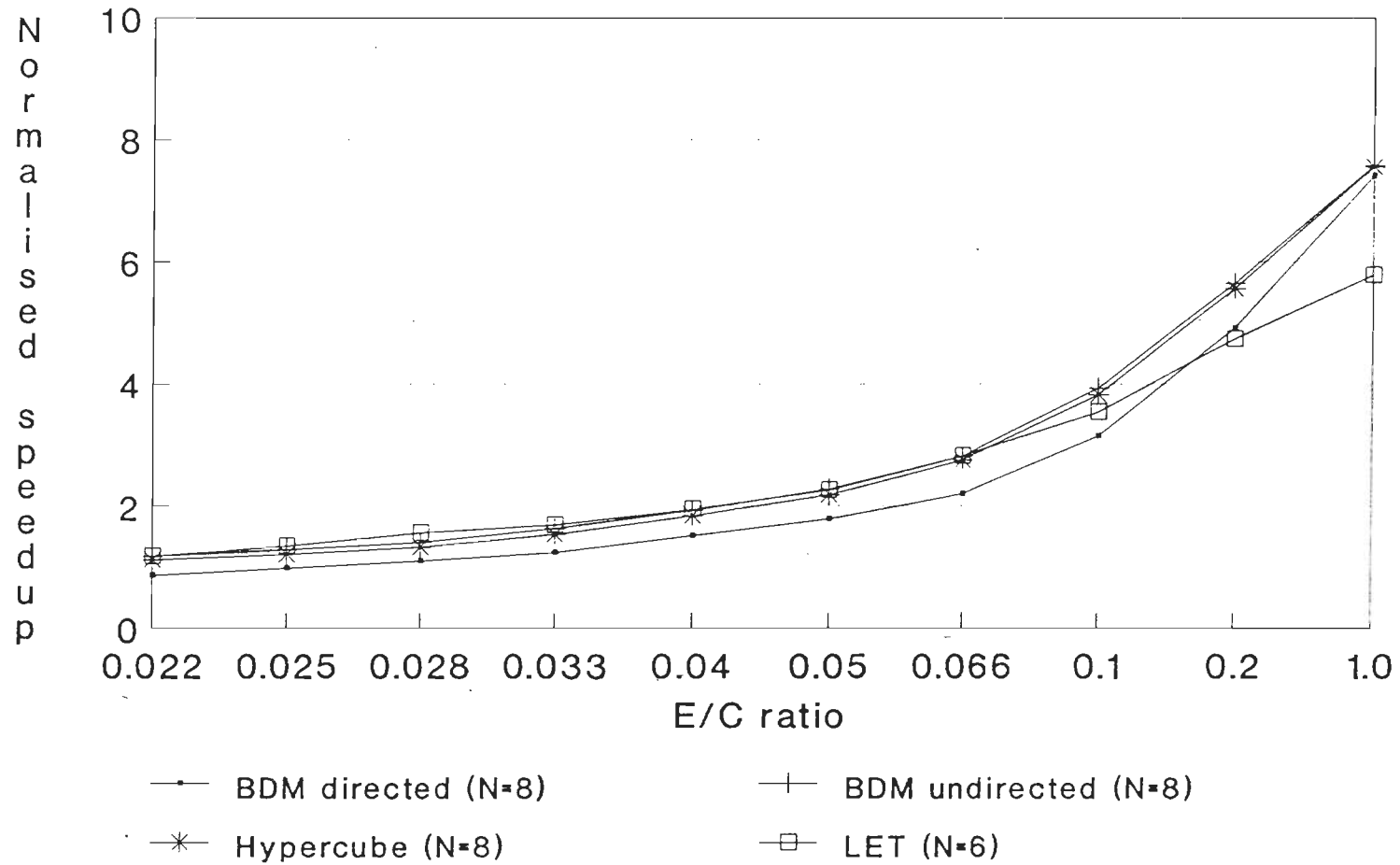


Fig. 6.9 Speedup characteristics of various networks

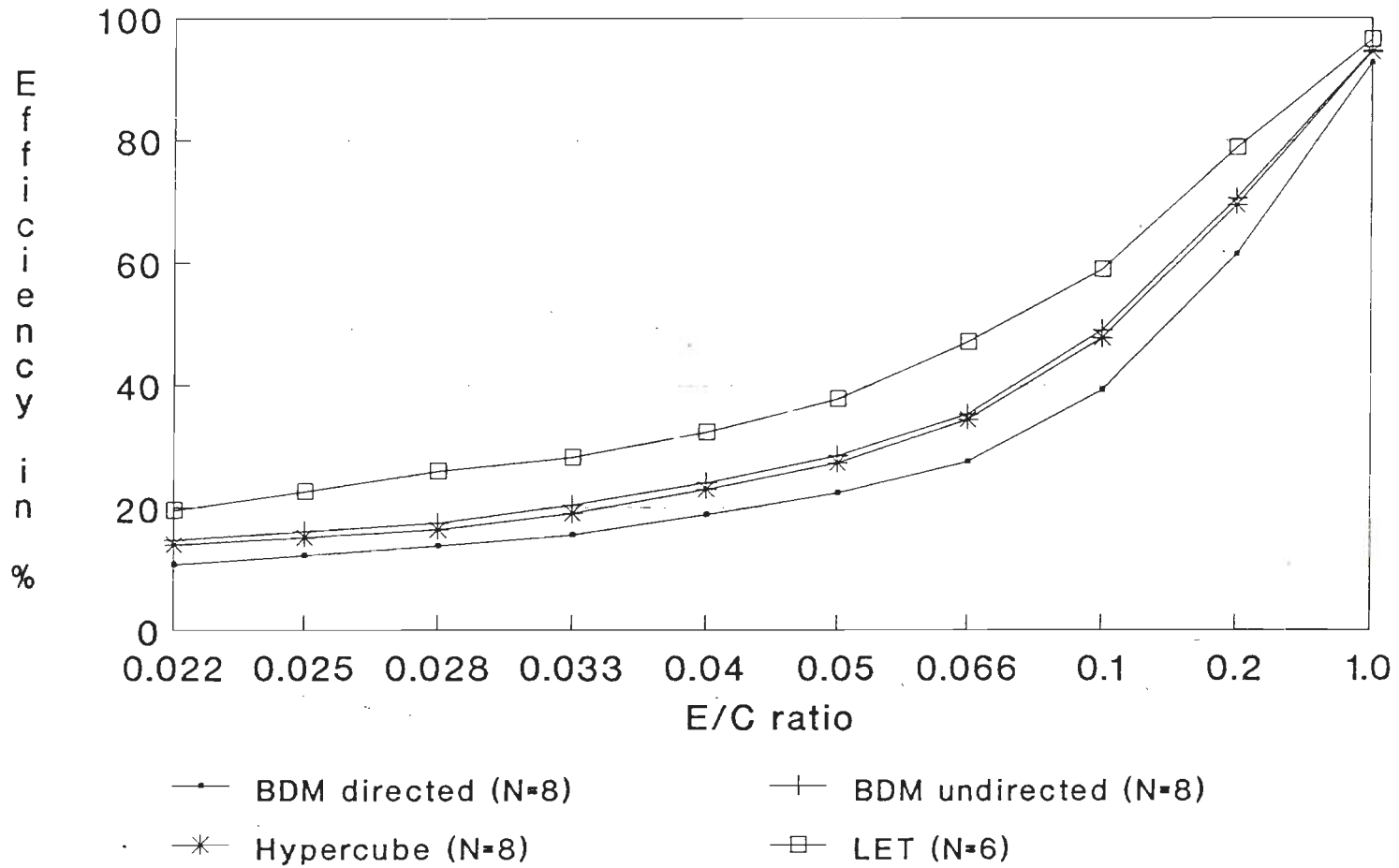


Fig. 6.10 Efficiency characteristic of various networks

All the results discussed above underline one common thing in all the networks that they are able to exploit parallelism whenever available, though, LET network shows better capability for all cases. It may be expected that hypercube will show better results as the network size increases, because in hypercube the number of immediate neighbours available to a processor increases with size. However, this advantage would be accompanied by increased number of connections required at each processor. LET network has constant degree and it shows better results than those in the category of fixed degree networks.

CONCLUSIONS AND FUTURE WORK

Quest for higher and higher computation speed has been a main endeavour of computer research. The trend in the computer systems has been towards distributing processing power over a number of identical processors executing in parallel and connected via a communication medium. The multiprocessing (where different instructions on different data can be executed asynchronously by each processor - Multiple Instruction Multiple Data (MIMD)) approach to parallelism is the most generalized and flexible one. However, the problem of interconnecting processors to achieve high computational bandwidth, and increased parallelism has not been fully solved.

An efficient management of parallelism involves optimizing conflicting performance indices, like the minimization of communication and scheduling overheads, and even load distribution among the processors. Such issues are addressed at the organizational level by designing a suitable topological layout of the network and an appropriate scheduling mechanism.

This thesis sets out to design a multiprocessor architecture model for parallel evaluation of tree-structured problems. A dynamic scheduling strategy has been proposed and tested for the designed network model. Finally, a comparative study has been made to show the superiority of the proposed model and the scheduling scheme for tree-structured as well as for general types of problems.

7.1 CONCLUSIONS

The overall performance of the multiprocessor system is affected by a number of factors, such as communication delays, imbalance of load among the processors and scheduling overheads (problem partitioning and task allocation). A close correspondence between the structure of the problem and processor interconnection is desired in order to minimize these factors. Scheduling plays a vital role to improve the performance, which is guided by some constraints, which may differ from application to application.

The basic skeleton of the proposed multiprocessor network, named as Linearly Extensible Tree (LET) whose size grows linearly and can provide more uniform load distribution as compared to BDM and Hypercube networks for graph problems in general and tree-structured problems in particular has been developed. Some of the properties observed for the LET network as compared to deBruijn and Hypercube networks are as follows:

- 1) In LET network, the number of nodes grow linearly with increase in level whereas in a Hypercube or in deBruijn, it grows as 2^n .
- 2) The degree of a node in the proposed model is always 4 or less. The connectivity of the Hypercube is equal to the number of dimensions in the cube i.e. n , while in case of deBruijn it is 4.
- 3) In LET network, the complexity of extension increases linearly because each extension requires adding a single layer of $(n+1)$ nodes. Hypercube and deBruijn networks though are extensible but the complexity increases exponentially by the power of 2.

A dynamic scheduling scheme known as Minimum Distance Scheduling (MDS) has been developed and implemented onto the LET network, for tree-structured problems. In this direction, investigations have been

made through simulation. The MDS scheme uses two steps. In the first step, problem graphs (task graphs) are mapped on to processors (network) with zero distance and in the second step, the ideal load is calculated and donor and acceptor processors are identified. The tasks are diffused from donor to acceptor using the minimum distance constraint. The behavior of the scheduling rules is evaluated in terms of the performance index called *Load Imbalance Factor* (LIF)_k, which represents the deviation of load among processors and achieves the optimal performance.

In order to confirm the performance of the LET network, several other static/dynamic scheduling schemes, such as *round robin* (RR), *minimum load* (ML), *declustering scheme*, *dimension exchange method* (DEM), *hierarchical balance method* (HBM) and *gradient model* (GM) have been compared to the MDS scheme on LET network as well as onto their respective networks for which these schemes have been designed.

Simulation results show that all the above mentioned scheduling schemes, when applied onto LET network, considering identical parameters, the proposed scheduling scheme performs better or equal for all types of problems.

Other simulation results indicate that the static scheduling scheme i.e. the latest precedence scheme, when implemented on the LET, Hypercube, directed BDM and undirected BDM networks for acyclic precedence graphs, gives the best performance on the LET network. As the number of task grows, the efficiency is higher than 80 percent for LET network of six processors.

A comparison of the LET multiprocessor network and its inherent qualities reveal that this network is reasonably comparable with the

existing multiprocessor networks. From the simulation studies, it has been found that the new dynamic scheduling scheme is performing better on the LET network in comparison to other static/dynamic scheduling schemes. Therefore, it can be concluded that the Linearly Extensible Tree (LET) multiprocessor network with the new dynamic scheduling scheme is a better organization model for parallel evaluation of all types of problem graphs, particularly tree-structured problems.

7.2 FUTURE WORK

The following extensions are recommended to the work presented in the thesis.

- 1) The LET network and the scheduler be studied for real life problems, since the tree-structured problems may not specify the whole concept of real problems. It is recommended that the network may be simulated and the actual program traces are used in evaluating the performance of the network scheduling rules.
- 2) The fold-back connections in LET network provide a short-cut between leaves and top processors around the root. A different network providing fold-back connections somewhere in the middle of the network may be studied from the point of view of its effect on diameter, average leaf to leaf distances and LIF obtainable.
- 3) In the application of Latest Precedence Scheduling the simulation study may be extended to study the effect of increasing number of processor in LET network on speedup and LIF.

REFERENCES

- [1] Adam, T., Chandy, K. and Dickson, J.A., "Comparison of list schedulers for parallel processing systems," *Comm. ACM*, vol. 17, no.12, pp. 685-690, Dec. 1974.
- [2] Akyildiz, I.F., "Performance analysis of a multiprocessor system model with process communication," *The Computer Journal*, vol. 35, no.1, pp. 52-61, 1992.
- [3] Alijani, G.S. and Wedde, H.F., "A task scheduling scheme for real time multi-robotics systems," *IEEE special issue on Parallel Processing*, pp. 111-117, 1991.
- [4] Arya, S., "An optimal instruction scheduling model for a class of vector processors," *IEEE Trans.on Computers*, vol. C-34, no.11, pp.981-995, November 1985.
- [5] Baccelli, F. and Liu, Z., "Extremal scheduling of parallel processing with and without real-time constraints," *Journal of Association for computing Machinery*, vol. 40, no.5, pp. 1209-1237, Nov. 1993.
- [6] Backus, J., "Can programming be liberated from the von Neumann style ? A functional style and its algebra of programs," *Comm. ACM*, vol. 21, no.8., pp. 613-641, August 1978.
- [7] Barmon, C., Faruqui, M.N. and Bhattacharjee, G.P., "Dynamic load balancing algorithm in distributed system," *Microprocessing and Microprogramming*, Vol. 29, pp. 273-285, North Holland, 1990/91.
- [8] Baruah, S., Rosier, L. and Varvel, D., "Static and Dynamic scheduling of sporadic task for single processor system," *IEEE special issue on Parallel Processing*, pp. 110-115, 1991.

- [9] Berkling, K.J., "A computer machine based on tree structures," IEEE Trans. on Computers, vol. C-20, pp. 404-418, Jan. 1974.
- [10] Bhuyan, L.N. and Agrawal, D.P., "Generalized hypercube and hyperbus structure," IEEE Trans. on Computers, vol. C-33. no.4, pp. 323-333, April 1984.
- [11] Blake, B. A., "Assignment of independent tasks to minimize completion time," Software Practice and Experience, vol. 22(9), pp. 723-734, Sept. 1992.
- [12] Blake, B. A. and Schwan, K., "Experimental evaluation of real time scheduler for a multiprocessor system," IEEE Trans. on Software Engineering, vol. 17, no.1, pp. 34-44, Jan. 1991.
- [13] Blaszewicz, J., Drabowski, M. and Weglarz, J., "Scheduling multiprocessor tasks to minimize schedule lengths," IEEE Trans. on Computers, vol 35, no.5, pp. 389-393, May 1986.
- [14] Bokhari, S.H., "Partitioning problems in parallel, pipelined and distributed computing," IEEE Trans. on Computers, vol. 37, no.1, pp. 48-57, Jan. 1988.
- [15] Bokhari, S.H., "A shortest tree algorithm for optimal assignments across space and time in a distributed processor system," IEEE Trans. on Software Engineering, vol. SE-7, no. 6, pp. 583-589, Nov. 1981.
- [16] Bokhari, S.H. "A network flow model for load balancing in circuit-switched multicomputers," IEEE Trans. on Parallel and Distributed systems, vol. 4, no.6, pp. 649-657, June 1993.
- [17] Bokhari, S.H., "On the mapping problem," IEEE Trans. on Computers, vol. C-30, no.3, pp. 207-214, March 1981.

- [18] Bonomi, F., "On job assignment for parallel system of processor sharing queues," IEEE Trans. on Computers, vol. 39, no. 7, pp. 858-869, July 1990.
- [19] Born, R.G. and Kenevan, J.R., "Theoretical performance based cost effectiveness of multicomputers," The Computer Journal, vol.35, no. 1., pp.62-70, 1992.
- [20] Boudillet, O., Gupta, J.P. and Winter, S.C., Implementation of functional multiprocessors. Research Studies press Ltd.: England, 1991.
- [21] Browne, J.C., and Lan, J., "The interaction of multiprogramming, job scheduling and CPU scheduling," Fall Joint Computer Conference, pp. 13-21, 1972.
- [22] Burks, A.W., Goldstine, H.H. and Von-Neumann, J., "Preliminary discussions for the design of an electronic computing instrument," Computer design and development : Principal papers, by EE-Schwartzlander JR. (ed.), pp. 221-259, Hayden Book Co., 1976.
- [23] Casavant, T.L. and Kuhl, J.G., "A taxonomy of scheduling in general purpose distributed computing systems," IEEE Trans. on Software Engineering, vol. 14, no.2, pp. 141-154, Feb. 1988.
- [24] Casey, L.M., "Decentralized Scheduling," Australian Computer J., vol. 13, pp. 58-63, May 1981.
- [25] Chaudhary, V. and Aggarwal, J.K., "A generalized scheme for mapping parallel algorithms," IEEE Trans. on Parallel and Distributed systems, vol. 4, no. 3, pp. 328-346, March 1993.
- [26] Chou, T.C.K. and Abraham, J.A., "Load balancing in distributed systems," IEEE, Trans. Software Engg., vol. SE-8, no.4. pp. 401-412, July 1982.

- [27] Coffman, E.G. and Graham, R.L., "Optimal scheduling for two processor systems," Acta Inform., pp. 200-213, 1972.
- [28] Coffman, E.G.(Jr) and kleiurock, L., "Computer scheduling methods and their counter measures," Spring Joint Conference, pp. 11-21, 1968.
- [29] Conway, R.W., Maxwell, W.L. and Miller, L.W., Theory of scheduling. Reading, MA : Addison - Wesley, 1967.
- [30] Cvetanovic, Z., "The effects of problems partitioning, allocation and granularity on the performance of multiple processor systems," IEEE Trans. on Computers, vol. C-36, no.4, pp. 421-432, April 1987.
- [31] Das, R., Fay, D.Q.M. and Das, P.K., "Allocation of precedence-constrained tasks to parallel processors for optimal execution," Microprocessing and Microprogramming, vol. 35, pp. 237-244, North Holland, 1992.
- [32] Deo, N., Graph Theory with Applications to Engineering and Computer Science. Prentice-Hall Inc.: N.J., 1983.
- [33] Du, H.C., "On the performance of synchronous multiprocessors," IEEE Trans. on Computers, vol. C-34, no.5. pp. 462-466, May 1985.
- [34] Feng, T.Y., "A survey of interconnection networks," IEEE Computer Journal, pp. 12-27, Dec. 1981.
- [35] Flynn, M.J., "Very high speed computing systems," Proc. IEEE, vol. 54, pp. 1901-1909, Dec. 1966.
- [36] Fox, G.C., et.al., Solving problems on concurrent processors (Vol.I). Prentice-Hall International, Inc.: U.S.A., 1988.

- [18] Bonomi, F., "On job assignment for parallel system of processor sharing queues," IEEE Trans. on Computers, vol. 39, no. 7, pp. 858-869, July 1990.
- [19] Born, R.G. and Kenevan, J.R., "Theoretical performance based cost effectiveness of multicomputers," The Computer Journal, vol.35, no. 1., pp.62-70, 1992.
- [20] Boudillet, O., Gupta, J.P. and Winter, S.C., Implementation of functional multiprocessors. Research Studies press Ltd.: England, 1991.
- [21] Browne, J.C., and Lan, J., "The interaction of multiprogramming, job scheduling and CPU scheduling," Fall Joint Computer Conference, pp. 13-21, 1972.
- [22] Burks, A.W., Goldstine, H.H. and Von-Neumann, J., "Preliminary discussions for the design of an electronic computing instrument," Computer design and development : Principal papers, by EE-Schwartzlander JR. (ed.), pp. 221-259, Hayden Book Co., 1976.
- [23] Casavant, T.L. and Kuhl, J.G., "A taxonomy of scheduling in general purpose distributed computing systems," IEEE Trans. on Software Engineering, vol. 14, no.2, pp. 141-154, Feb. 1988.
- [24] Casey, L.M., "Decentralized Scheduling," Australian Computer J., vol. 13, pp. 58-63, May 1981.
- [25] Chaudhary, V. and Aggarwal, J.K., "A generalized scheme for mapping parallel algorithms," IEEE Trans. on Parallel and Distributed systems, vol. 4, no. 3, pp. 328-346, March 1993.
- [26] Chou, T.C.K. and Abraham, J.A., "Load balancing in distributed systems," IEEE, Trans. Software Engg., vol. SE-8, no.4. pp. 401-412, July 1982.

- [27] Coffman, E.G. and Graham, R.L., "Optimal scheduling for two processor systems," Acta Inform., pp. 200-213, 1972.
- [28] Coffman, E.G. (Jr) and Kleiurock, L., "Computer scheduling methods and their counter measures," Spring Joint Conference, pp. 11-21, 1968.
- [29] Conway, R.W., Maxwell, W.L. and Miller, L.W., Theory of scheduling. Reading, MA : Addison - Wesley, 1967.
- [30] Cvetanovic, Z., "The effects of problems partitioning, allocation and granularity on the performance of multiple processor systems," IEEE Trans. on Computers, vol. C-36, no.4, pp. 421-432, April 1987.
- [31] Das, R., Fay, D.Q.M. and Das, P.K., "Allocation of precedence-constrained tasks to parallel processors for optimal execution," Microprocessing and Microprogramming, vol. 35, pp. 237-244, North Holland, 1992.
- [32] Deo, N., Graph Theory with Applications to Engineering and Computer Science. Prentice-Hall Inc.: N.J., 1983.
- [33] Du, H.C., "On the performance of synchronous multiprocessors," IEEE Trans. on Computers, vol. C-34, no.5, pp. 462-466, May 1985.
- [34] Feng, T.Y., "A survey of interconnection networks," IEEE Computer Journal, pp. 12-27, Dec. 1981.
- [35] Flynn, M.J., "Very high speed computing systems," Proc. IEEE, vol. 54, pp. 1901-1909, Dec. 1966.
- [36] Fox, G.C., et.al., Solving problems on concurrent processors (Vol.I), Prentice-Hall International, Inc.: U.S.A., 1988.

- [37] Ganeshan, E. and Pradhan, D.K., "The hyper-deBruijn networks : scalable versatile architecture," IEEE Trans. on Parallel and Distributed Systems, vol. 4, no.9, pp, 962-978, Sept. 1993.
- [38] Ghosal, D., Serazzi G. and Tripathi, S.K., "The processor working set and its use in scheduling multiprocessor systems," IEEE Trans. on Software Engineering, vol. 17, no.5, pp. 443-453, May 1991.
- [39] Gonzalez, M.J., "Deterministic processor scheduling," ACM Computer Surveys, vol. 9, no.3, pp. 173-204, 1977.
- [40] Goodman, J.R. and Sequin, C.H., "Hypertree: A multiprocessor interconnection topology," IEEE Trans. on Computers, vol. C-30, no. 12, pp 923-933, Dec. 1981.
- [41] Ha, S. and Lee E.A., "Compile time scheduling and assignment of data-flow program graphs with data dependent iteration," IEEE Trans. on Computers, vol. 40, no.11, pp. 1225-1238, Nov. 1991.
- [42] Hamdi, M. and Hall, R.W., "An efficient class of interconnection networks for parallel computation," The Computer Journal, vol. 37, no.3, pp. 206-218, 1994.
- [43] Hanan, M. and Kurtzberg, J.M., "A review of placement and quadratic assignment problems," SIAM Rev., vol. 14, pp. 324-342, April 1972.
- [44] Harget, A.J., and Johnson, I.D., Load Balancing Algorithms in Loosely Coupled Distributed Systems: A Survey. (Distributed Computer Systems - Theory and Practice, HSM Zedan Ed.) Butterworths, 1990.
- [45] Hawang, J.J., Chow, Y.C., Anger, F.D. and lee, C.Y., "Scheduling precedence graphs in systems with interprocessor communication times," SIAM Journal of Applied Maths, pp. 244-257, April 1989.

- [46] Hwang, K. and Briggs, F.A., Computer Architecture and Parallel Processing. McGraw-Hill International Editions: Singapore, 1989.
- [47] Hayes, J.P., Computer Architecture and Organization (2nd Ed.). McGraw-Hill: N.Y., 1988.
- [48] Hockney, R.W. and Jesshope, C.R., Parallel Computers. Adam Hilger Ltd.: Bristol, 1981.
- [49] Hu, T.C., "Parallel sequencing and assembly line problem," Operation Research, vol. 9, pp. 841-848, 1961.
- [50] Jongkim, Das C.R. and Lin, W., "A top-down processor allocation scheme for hypercube computers," IEEE Trans. on Parallel and Distributed systems, vol. 2, no.1, pp. 20-30, Jan. 1991.
- [51] Kain, R.Y., Computer Architecture, Software and Hardware (Vol.II). Prentice-Hall: Englewood Cliffs, New Jersey, 1989.
- [52] Keqin, L. and Cheng, K.H., "Job scheduling in a partitionable mesh using a two dimensional buddy system partitioning scheme," IEEE Trans. on Parallel and Distributed Systems, vol.2, no.4, pp. 413-422, Oct. 1991.
- [53] Killer, R.M. and Lin, F.C., "Simulated performance of a reduction-based multiprocessor," IEEE Computer, vol. 7, no. 17, pp. 70-82, 1984.
- [54] Kohler, W.H., "A preliminary evaluation of critical path method for scheduling tasks on multiprocessor systems," IEEE Trans. on Computers, vol. C-15, no.12, pp. 1235-1238, Dec. 1975.
- [55] Kumar, J.M. and Patnaik, L.M., "Extended hypercube: A hierarchical interconnection network of hypercubes," IEEE Trans. on Parallel and Distributed Systems, vol. 3, no.1, pp. 45-57, Jan. 1992.

- [56] Lee, C.S.G., and Chen, C.L., "Efficient mapping algorithms for scheduling robot inverse dynamics computation on a multiprocessor system," IEEE Trans. on Systems, Man and Cybernatics, vol. 20, no.3, pp.582-594, May/June 1990.
- [57] Lee, S.Y. and Aggarwal, J.K., "A mapping strategy for parallel processing," IEEE Trans. on Computers, vol. 36, no.4, pp. 433-441, April 1987.
- [58] Leighton, F.T., Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes. Morgan Kaufmann Publishers: San-Manteo, California, 1992.
- [59] Lewis, T.G. and Rewini, H.E., Introduction to Parallel Computing. Prentice-Hall: U.K., 1992.
- [60] Lewis, T. and Rewini, H.E., "Parallex : A tool for parallel program scheduling," IEEE Parallel and Distributed Technology, pp. 62-72., May 1993.
- [61] Li, H.F., "Schedulling trees in parallel/pipelined processing environments," IEEE Trans. on Computers, vol. C-26, no.11, pp. 1101-1112, Nov. 1977.
- [62] Lin, H.C. and Raghuvandera, C.S., "A dynamic load balancing policy with a central job dispatcher (LBC)," IEEE Trans. on Software Engineering, vol. 18, no.2, pp. 148-158, Feb. 1992.
- [63] Lin, R.C., Wang, S.D., "Performance modelling and analysis of load balancing policies with priority queuing," J. System Softwares, vol. 20, pp. 169-187., U.S.A., 1993.
- [64] Lin, W. and Wee, C.L., "A distributed resource management mechanism for partitionable multiprocessor system," IEEE Trans. on Computers, vol. 37, no.2, pp. 201-210, Feb. 1988.

- [65] Livny, M. and Melmen, M., "Load balancing in homogeneous broadcast distributed systems," in Proc. Computer Network Performance Symp., pp. 47-58, 1982.
- [66] Lo, V.M., "Heuristic algorithms for task assignment in distributed system," IEEE Trans. on Computers, vol. 37, no.11, pp. 1384-1397. Nov. 1988.
- [67] Ma, P.Y.R., Lee, E.Y.S. and Tsuchiya, J., "A task allocation model for distributed computing systems," IEEE Trans. on Computers, vol. C-31, no.1, pp. 41-47, Jan. 1982.
- [68] Martin, A.J., "The TORUS: an exercise in connecting a processor surface," in Proc. of VLSI Confer., 1981.
- [69] McDowell, C.E. and Appelbe, W.F., "Processor scheduling for linearly connected parallel processors," IEEE Trans. on Computers, vol. C-35, no.7, 1986.
- [70] McHugh, J.A., Algorithmic Graph Theory. Prentice-Hall: Englewood Cliffs, New Jersey, 1990.
- [71] Moldovan, D.I., Parallel Processing - from Applications to Systems, Morgan Kaufmann Publishers: San-Manteo, California, 1993.
- [72] Murthy, C.S., Murthy, K.N.B. and Srinivas, A., "Scheduling of precedence-constrained parallel program tasks on multiprocessors," Multiprocessing and Multiprogramming, vol. 36, pp. 93-104, 1992/93.
- [73] Nicol, D.M. and O'Hallaron, D.R., "Improved algorithm for mapping pipe-lined and parallel computation," IEEE Trans. on Computers, vol. 40, no.3, pp. 295-305, March 1991.

- [74] Nicol, D.M., and Reynolds, P.F.(Jr.), "Optimal dynamic remapping of data parallel computation," IEEE Trans. on Computers, vol. 39, no.2, pp. 206-219, Feb. 1990.
- [75] Nicol, D.M. and Saltz, J.H., "An analysis of scatter decomposition," IEEE Trans on Computers, vol. 39. no.11, pp. 1337-1345, Nov. 1990.
- [76] Parhi, K.K. and Messerschmitt, D.G., "Static rate-optimal scheduling of iterative data - flow programs via optimum unfolding," IEEE Trans. on Computers, vol. 40, no.2, pp. 178-195, Feb. 1991.
- [77] Polychronopoulos, C.D., Parallel Programming and Compilers, Kenwar Academic Publishers: Norwell, U.S.A., 1989.
- [78] Rabhi, F.A. and Manson, G.A., "Divide-and-conquer and parallel graph reduction," Parallel Computing, vol. 17, pp. 189-205, North Holland, 1991.
- [79] Rabhi F.A. and Manson, G.A., "Experiments with a transputer-based parallel graph reduction machine," Concurrency: Practice and Experience, vol. 3(4), pp. 413-422, Aug. 1991.
- [80] Ramamoorthy, C.V., Chandy, K.M. and Gonzalez, M.J. (Jr.), "Optimal scheduling strategies in a multiprocessor system," IEEE Trans. on Computers, vol. C-21, no.2, pp. 137-147, Feb. 1972.
- [81] Ramamritham, K., Stankovic, J.A. and Zaho, W., "Distributed scheduling of tasks with dead-line and resource requirements," IEEE Trans. on Computers, vol. 38, no.8, pp. 1110-1123, Aug. 1989.

- [82] Ravikanth, K., Sastry, P.S., Ramakrishnan, K.R. and Venkatesh, Y.V., "A reduction architecture for the optimal scheduling of binary trees," *Future Generation Computer Systems*, pp. 225-223, 4 (1988).
- [83] Ravikanth, K., Sastry, P.S. and Venkatesh, Y.V., "Simulation studies on the performance on an organizational model for graph reduction," *Future Generation Computer Systems*, pp. 163-180, 6 (1990).
- [84] Rawlins, G.J.E., *Compared to What? An Introduction to The Analysis of Algorithms*. Computer Science Press: New York, 1991.
- [85] Reddaway, S.F., "DAP - a distributed array processor," in *Proc. 1st Annual symposium on Computer Architecture*, pp. 61-65, Florida, Dec. 1973.
- [86] Reddy, A.V., "Design of parallel reduction model for the implementation of functional languages," Ph.D. Thesis, UOR, Roorkee, Dec. 1993.
- [87] Rommel, C.G., "The probability of load balancing success (PLBS) in a homogeneous network," *IEEE Trans. on Software Engineering*, vol.17, no.9, pp. 922-933, Sept. 1991.
- [88] Rosberg, Z., "Process scheduling in a computer system," *IEEE Trans. on Computers*, vol.C-34, pp. 633-645, July 1985.
- [89] Saad, Y. and Shultz, M., "Topological properties of hypercubes," *IEEE Trans. on Computers*, vol. C-37, no.7, pp 867-871, July 1988.
- [90] Samathan, M.R. and Pradhan, D.K., "The deBruijn multiprocessor network : A versatile parallel processing and sorting network for VLSI," *IEEE Trans. on Computers*, vol. 38, no.4, pp. 561-581, April 1989.

- [91] Sequin, C.H., "Doubly twisted torus networks for VLSI processor arrays," Eighth Annual Symp. on Computer Architecture, pp. 471-480.
- [92] Shang, W. and Fortes, J.A.B., "Time optimal linear schedules for algorithms with uniform dependencies," IEEE Trans. on Computers, vol. 40, no.6, pp. 723-742, June 1991.
- [93] Shen, C. and Tsai, W., "A graph matching approach to optimal task assignment in distributed computing systems using a minimax criterion," IEEE Trans. on Computers, vol. C-34, no.3, pp. 179-203, March 1985.
- [94] Shepard, T. and Gagne, J.A.M., "A Pre-run-time scheduling algorithm for hard real-time systems," IEEE Trans. on Software Engineering, vol.17, no.7, pp. 669-677, July 1991.
- [95] Sih, G.C. and Lee., E.A., "Declustering: A new multiprocessor scheduling technique," IEEE Trans. on Parallel and Distributed systems, vol. 4, no.6, pp. 625-637, June 1993.
- [96] Smith, R.G., "The contract net protocol: High-level communication and control in distributed problem solver," IEEE Trans. on Computers, vol. C-29, no.12, pp. 1104-1113, Dec. 1980.
- [97] Stone, H.S., "Multiprocessor scheduling with the aid of network flow algorithms," IEEE Trans. on Software Engineering, vol. S-3, no.1, pp. 85-93, 1977.
- [98] Suen, T.T.Y. and Wong, J.S.K., "Efficient task migration algorithm for distributed systems", IEEE Trans. on Parallel and Distributed systems, vol. 3, no.4, pp. 488-499, July 1992.
- [99] Turner, D.A., "A new implementation technique for applicative languages," Software-Practice and Experience, vol. 9, pp. 31-49, Sept. 1979.

- [100] Vairvan, K. and DeMillo, R.A.; "On the computational complexity of a generalized scheduling problem," IEEE Trans. on Computers, vol.C-25, no.11, pp. 1067-1073, Nov. 1976.
- [101] Wang, C.M. and Wang, S.D., "Efficient processor assignment algorithms and loop transformation for executing nested parallel loops on multiprocessors," IEEE Trans. on Parallel and Distributed Systems, vol. 3, no.1, pp. 71-82, Jan. 1992.
- [102] Wanganv, Y.T., Morris, R.J.T., "Load sharing in distributed systems," IEEE, Trans. on Computers, vol. C-34, no.3, pp. 204-217, March 1985.
- [103] Willebeek - Le Mair, M.H. and Reeves., A.P., "Strategies for dynamic load balancing on highly parallel computers," IEEE Trans. on Parallel and Distributed Systems, vol. 4, no.9, pp. 979-993, Sept. 1993.