ISSN:0975-9646

A Δ-Based Linearly Extensible Multiprocessor Network

Manaullah

Department of Electrical Engineering, Faculty of Engineering & Technology,
Jamia Millia Islamia, New Delhi, India

***Abstract*— To speeding up more and more computation power by reducing the size of the system, it is always a trend. In this direction, a novel Triangle based multiprocessor system has been proposed in this paper, which is compact in size (occupy lessor space) and exhibits all the features and characteristics of a commercial available multiprocessor. The proposed network has lessor number of processing elements, smaller diameter and less complex. Its extension requires only one processing element per extension i.e. linearly extensible. Simulation studies show the better performance to other similar systems.**

Keywords: **Multiprocessor Architecture, Linearly Extensible Network, Load Imbalance Factor, Scheduling, and Diameter**

I. INTRODUCTION

In recent years, considerable progress has been made in the design of integrated circuit technology, which has resulted in the emergence of highly powerful processors. Beside that several new parallel architectures have been proposed to increase computing speed to complement the advances in technology [1]. But to this day the problem of interconnecting the processing elements to achieve high computational bandwidth has not been fully solved. Increase parallelism means more communication overheads; internodes distance, message traffic density and fault tolerance are dependent on the diameter of the network and the degree of a node in it [2-7].

An interconnection network with large diameter has very low message passing bandwidth and a network with high degree of node has higher hardware complexity. In addition, computing system should be easily expandable; there should be no changes in the basic node configuration as we increase the number of nodes in a system.

The choice of topology of the interconnection network is critical in design of massive parallel computer system. 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 the networks [8]. In additional to designing an appropriate network, the efficient management of parallelism on an interconnection network involves optimizing conflicting performance indices, like minimization of communication and scheduling overheads and uniform load distribution among the processors. Such issues are addressed at the organizational level by appropriate scheduling mechanisms.

Recently an organizational model has been reported as Linearly Extensible Tree (LET) network with a dynamic scheduling scheme Minimum Distance Scheduling (MDS) [9]. This architecture consists of 6 processors instead of 8 processors as in hypercube or deBruijn architecture. Using the dynamic scheduling scheme, named Minimum Distance Scheduling (MDS); it has been shown that the LET is performing at par with other architectures [10]. Another Linear Extensible Cube (LEC) network has also been reported [11]. This LEC combines the features of LET and hypercube networks. It is shown that the LEC is performing on equal footing or rather better than the remaining similar networks.

In this paper, a new linearly extensible triangle–based architecture has been proposed and its properties have been compared with other similar architectures. Two dynamic scheduling schemes, MDS and Hierarchical Balancing Method (HBM) scheduling, [9,10] are implemented on this architecture and the performance parameters are obtained. These results are compared with other linearly extensible networks with the same scheduling schemes in the following chapters.

2.0 BASIC TOPOLOGICAL PROPERTIES

2.1 Linearly Extensible Tree (LET) Multiprocessor Network

As the proposed network is based on the concept of LET network [8], and LEC network [11] a brief description of LET and LEC networks are given for ready reference to researchers. The LET network combines the properties of linear extensibility withsmall number of processing elements per extension. The network has a small diameter that reduces the average path length traveled by all messages and contains a constant degree per node.

The LET network grows linearly in a binary tree like shape. In a binary tree, the number of nodes at level j is 2^j whereas in LET network the number is (j+1).

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

$$Q' = \{P0 , P1 , ...P N-1 \}$$

The number of processors N in the network is given by

$$N = \sum_{k=1}^{d+1} K \qquad (2.1)$$

where d is the depth of the network. For different depths, networks having 1,3,6,10,15,21,..... processors are possible.

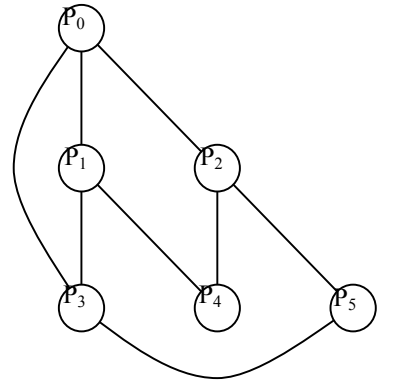
In order to define the link functions, we denote each processor in the set Q as P_{ij}-, j being the level in LET where the processor P_i resides. As per the LET policy, only (j+1) processors exist at level

j. Thus at level 0, P_0 exists and it may be redesignated as P_{00} and so on. The arrangement is shown in the Fig. 2.1

P_{00}
P_{11} P_{21}
P_{32} P_{42} P_{52}
P_{63} P_{73} P_{83} P_{93}
.....
.....

Fig. 2.1 Arrangement of processors in LET

Let Q′ be the set of redesignated processors of Q. Thus,
$Q' = \{ P_{ij} \}$ $0 \le i \le N-1$, $0 \le j \le d$



(a) Linearly Extensible Tree

$$\begin{pmatrix} 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \end{pmatrix}$$

(b) Adjacency Matrix

Fig. 2.2 Network with 6 Processors

The interconnection between processors are governed by two functions L and R and is represented as

$$\left.\begin{aligned} L(P_{ij}) &= P_{(i+j+1) \bmod N} \\ R(P_{ij}) &= P_{(i+j+2) \bmod N} \end{aligned}\right\} \text{ for all } P_{ij} \text{ in } Q' \qquad (2.2)$$

These two functions L & R indicate the links between various processors in the network. Fig. 2.2 shows a LET network for six processors along with its adjacency matrix.

2.2 Linearly Extensible Cube (LEC) Multiprocessor Network

2.2.1 Design and analysis

LEC network grows linearly in a cube like shape. In LET network, the number of nodes at level j is (j+1), whereas in LEC network no addition of nodes is required, at any level, the number of processing elements is fixed i.e 2 at every level. The network itself may be defined through connection functions in a manner similar to that of cube connection

Let Q be a set of N identical processors, represented as
$Q' = \{ P_0, P_1, P_2 ..... P_{N-1} \}$

The number of processors N in the network is given by 2n for n=1,2,3.....,d where d is the depth of the network. For different depths, network having 2,4,6,8,10..... processors are possible.

In order to define the link functions we denote each processor in the set Q as P_{ij}, j being the level in LEC where the processor P_i resides. As per the LEC policy, only two processors exist at level j. Thus at level 0, P_0 and P_1 exist and it may be redesignated as P_{00} and P_{10}, and so on. The arrangement is shown in Fig. 2.3

P_{00} P_{10}
P_{21} P_{31}
P_{42} P_{52}
.....

Fig. 2.3 Arrangement of processors in LEC

Let Q′ be the set of designated processors of Q Thus
$Q' = \{Pi\}$, $0 \le i \le N-1$

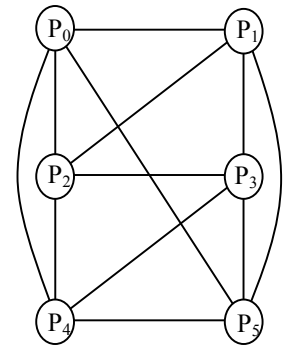
The link function L and R define the mapping from Q′ to Q as
$L(Pi) = P_{(i+1)} \bmod N$
$R(Pi) = P_{(i+2)} \bmod N$ For all P_i in Q′

The two functions **L** and **R** indicate the links between various processors in the network Fig.2.4 shows an LEC network for 6 processors along with its adjacency matrix

2.3 Design of the Proposed Multiprocessor Network

The proposed triangle-based multiprocessor network is basically having the concept of simple geometry and its interconnections topology exhibits, the properties of a linearly extensible multiprocessor architecture. The details of the design of a triangle-based network topology is given below.

Draw an isosceles triangle i.e. a triangle whose two sides are equal. Bisect the base angles of this triangle P_0 P_1 P_2. The new isosceles triangle is P_0 P_1 P_3. Connect the vertex of these two triangles right angle upward. It is the proposed triangle based multiprocessor architecture. Extend the vertices upward at any point on this extended vertex when joined to the base angles P_0 P_1, will show the expandability of the proposed architecture as shown in Fig. 2.6



(a) LEC Network

$$\begin{pmatrix} 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 & 0 \end{pmatrix}$$

(b) Adjacency Matrix

Fig. 2.4 Linearly Extensible Cube with 6 processors

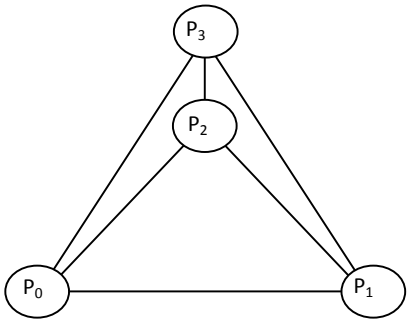
2.3.1 Properties of the proposed network: Some properties of the proposed network have been compared with LET, LEC, de-Bruijn and hypercube networks.

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 nodes, lesser is the systems complexity and hence is more economical. The number of nodes in the LEC network is **N= 2n (**for n>0), whereas the number of nodes in the LET network **N=∑k,** where as the nodes in hypercube and deBruijn networks are $\mathbf{2^n}$. In the proposed network, it is k+1, (for k> 3) i.e. **4**. Thus the proposed network is more economical than other networks of having lesser number of processing elements in it.

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 the 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. The degree of node in the LEC is always 4 or less, same as in LET network. Though in case of deBruijn it is constt. at 4 where as in hypercube, the degree increases with the size of the system. In the proposed network it is (N-1).



(a) Linearly Extensible Triangle

$$\begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

(b) Adjacency Matrix

Fig. 2.5 Network with 4 Processors

c) Extensibility

Extensibility is the property which facilitates large sized system out of small ones with minimum changes in the configuration of the nodes. In the proposed network the extension complexity increases in a constant manner. Each extension requires single layer of two nodes in LEC whereas in the case of LET network the extension complexity increases linearly because each extension requires adding a single layer of (N+1) nodes, where as in hypercube and deBruijn netrworks, the extensibility is exponential. In the proposed network, each extension requires only one node without changing the basic configuration.

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 the network is bound to increase as the size grows unless there is no limit on the number of the links. Ignoring the foldback connections, the diameter of LET network is O√N, whereas in LEC network the diameter is O ([N]), where as in hypercube & deBruijn it is logarithmic. In the proposed network the diameter is only 2 and is constant for all stages of the system with drastically reduced number of processors at different depths in comparison to LET & LEC networks.

A comparative study of number of processors for various depth of a network is shown in table 2.1.

Table 2.2 shows the diameter of different depth. These results have been obtained using shortest path algorithm. It may be seen that the diameter does not always in crease with the addition of a layer of processors.

Table 2.3 shows the summary of these parameters, which is depicted, in a comparative form.

Table 2.1
NUMBER OF PROCESSORS FOR VARIOUS DEPTH

Depth	0	1	2	3	4	5	6
LEC	2	4	6	8	10	12	14
LET	1	3	6	10	15	21	28
Proposed Network	3	4	5	6	7	8	9

Table 2.2
COMPARATIVE DIAMETER OF VARIOUS SIZED NETWORKS

Depth	0	1	2	3	4	5	6
Diameter							
in LEC	0	1	1	2	2	3	3
in LET	0	1	2	3	4	4	5
in Proposed Network(LEΔ)	1	2	2	2	2	2	2

Table 2.3
SUMMARY OF PARAMETERS

Parameters	**Hypercube**	**De-Bruijn**	**LET**	**LEC**	**Proposed Network (LEΔ)**
No. of processor	$N = 2^n$	$N = 2^n$	$\mathbf{N} = \sum_{k=1}^{d+1} \mathbf{K}$	$N = 2^*n$	$N=\sum_{K=3}^{n} k+1$
Degree	N	4	4	4	N-1
Extensibility	2^n	2^n	N+1	2	N+1
Diameter	$O(\log_2 N)$	$O(\log_2 N)$	$O(\sqrt{N})$	O(N)	2

3.0 THE HBM AND MDS SCHEDULING ALGORITHMS

When the problem graph topology is not known a priori, the mapping is done on the fly onto the processors. This dynamic load balancing is essential for efficient utilization of highly parallel systems when solving non-uniform problems with unpredictable load estimates. Our studies show that the network has good load balancing properties when considering problem structures having parallelism but non-uniform growth in various branches.

The general model of the dynamic load balancing is mainly based on the load balancing profitability determination at various sites in a multiprocessor network [10]. Whenever profitable, a balancer is invoked which migrates tasks to achieve a more uniform distribution of load on 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.

3.1 The HBM Algorithm

The HBM strategy organizes the multi-computer system into a hierarchy of balancing domains, thereby decentralizing the balancing process. Specific processor are designated to control the balancing operations at different levels of the hierarchy. In this case, processors incharge of the balancing process at a level I_i, receive load information from both level, $_{Ii-1}$, domains. Global balancing is achieved by ascending the net and balancing the load between adjacent domains at network level in the hierarchy. This procedure is asynchronous, however, where balancing is invoked within a domain whenever an imbalance is detected by the domain's designated controller. For a binary hierarchical configuration, the size of the balancing domains double from one level to the next.

The hierarchical balancing scheme functions asynchronously. The balancing process is triggered at different levels in the hierarchy by the receipt of load update messages indicating an imbalance between lower level domains. All load levels are initialized with each processor sending its load information up the network. To implement HBM in the proposed network, this scheme has been modified. The implementation of this algorithm in C-like notation is given below:

```
Hier ( )
{
/* Generate the first task at the PMAX –2 and PMAX –1 processors
*/
ZDS [PMAX-1] = 1 ; ZDS [PMAX-2] = 1
While (it _ count < L )
{
/* calculate IL & RIL */
IL = CIL (ZDS);
RIL = Ceil (IL);
Printf (ZDS);
For (j = 0; j <N; ++j)
{
flag = 0;
/* find the table no of jth processor */
for (m = 0; m <N/2; ++m)
for (n=0; n <^t2; ++n)
if (1[m] [n] = = j) label_No = m;
```

```
/* find where to migrate */
if (label_No = = (N/2 –1) to_migrate = 0; else to_migrate = label_No +1;
while (true)
{
while (true)
{
for (m=0;m <2++m)
if (ZDS [1 [to_migrate] [m] ] < RIL) flag = 1;
if (flag = = 1) break;
else
{
to_migrate ++;
if (to_migrate > = N/2) to_migrate = 0;
}
}
if (flag = = 1) migrate (j, to – migrate);
if (ZDS[1] < = RIL)break;
}}}
/* calculate LIF */

LIF = (maximum (ZDS) –IL)/ IL;
Printf (ZDS);
/* double the task */
ZDS = 2 * ZDS;
It_count ++;
}}
/* FUNCTIONS USED BY algorithm */
/*CIL & maximum are identical to the previous algorithms */
Migrate (P_No, to_migrate)
{
/* get minimum loaded processes at the level where migration is being done */
small = ZDS [1 [to_migrate] [0] ];
if (ZDS [1 {to_migrate] [1] ])< ZDS [1 [ to_migrate] [0]; small = 1 [to-migrate [1];
/* transfer the load */
ZDS [P_No] --;
ZDS [small] + = 1;
}
```

3.2 The MDS Algorithm

The scheme has been developed for a tree type problem structure. The approach tries to maximize 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 [10].

In the light 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 :

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

where $(ideal\text{-}load)_k = [load_k(P_0) + load_k (P_1)+...+load_k(P_{N-1})] / N$,
and max $(load_k (P_i))$ denotes the maximum load pertaining to level k of the task tree on a processor P_i due to kth level of the task tree. The whole algorithm in the modified form 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 ≤n;  count ++)
     {
        Tc = select (list) ;
        store(TC) ;
        Tf = father(Tc);

 /* father(task) returns the father of the task */
        Pf = processor(Tf);
       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 < kmax);
       }
       schedule ( int k )
      {
       IL = ideal_load(k);
       for (itno = 0; itno < 2; into ++)
          {        /* 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) puts the processor Pi in acceptor priority queue aQ */
        }
                  while(dQ not empty) {
                     Pi = delete(dQ);
            si = MDA(pI);
/* si is the set of minimum distance acceptors for Pi */
        assign(Pi);
/* assign(Pi) tries to transfers a load equal to excess of Pi from Pi to the highest priority acceptor from si.  If not successful, pi continues to be a donor with reduced overload.*/
            }
        update(k);
     }
}
```

4.0 SIMULATION RESULTS

4.1 Dynamic Load Model

The scheduling performance of various strategies in the modified form has been tested for the proposed 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 sub-tasks, 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.

Using the above-simulated dynamic load, the performance of the proposed network is being tested for HBM & MDS scheduling schemes. The performance is measured in terms of load imbalance left after a balancing action. A constraint that has been forced in the scheduling to maintain minimum distance i.e. task do not migrate to underloaded processors in a way so as to make the distance from parent task more than one hop in the processor network. Under this constraint, the network gets fully balanced at all levels for different types of trees. Its comparison with the LET & LEC networks shows better load balancing. The performance of the network becomes more attractive considering the fact that the diameter of the network remains fixed and having the lesser number of processors unlike other same type of networks.

4.2 HBM Scheme on the proposed network and its comparison with other networks

To study the behavior of the Hierarchical Balancing Method (HBM) Scheduling Scheme on the proposed networks, LEC, LET and Twisted N-cube networks, the LIF's 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) as shown in Fig. 4.1 - 4.4.

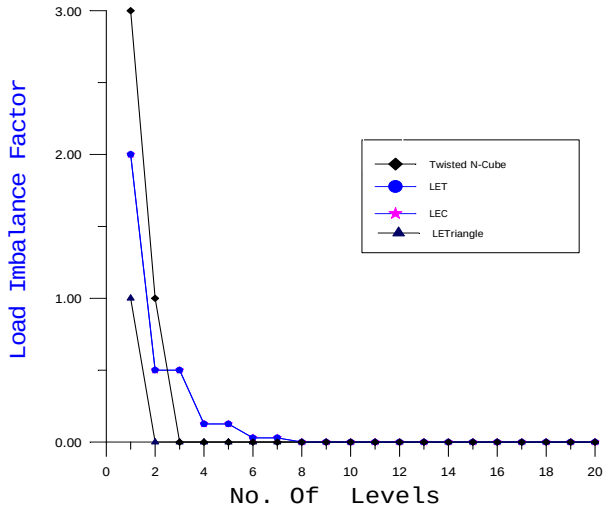


Fig. 4.1 LIF for complete Binary Tree

The trend of the curves obtained, indicates the average behaviour of the load imbalance factor (LIF) with respect to the level in the task tree for different randomly generated task tree structures when HBM scheme is implemented on different networks. It has been observed that LIF shows a similar behaviour in both the cases of binary & ternary tree task structures, decreasing rapidly to zero after attaining sufficient number of tasks. In case of complete binary task trees, this reduction is earlier than in case of ternary task trees and complete ternary task trees.

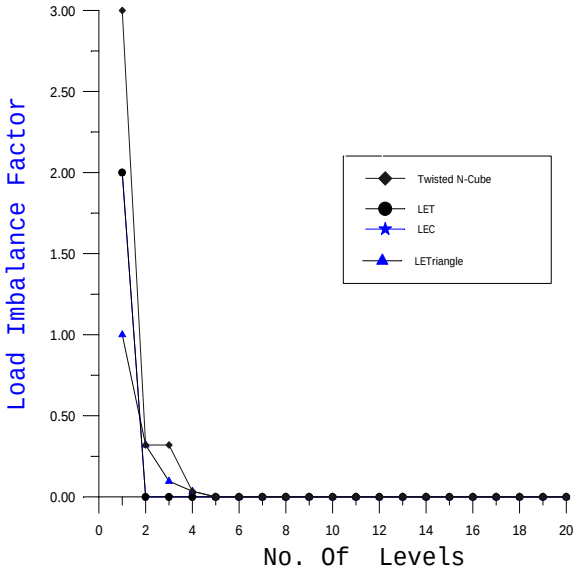


Fig. 4.2 LIF for complete Ternary Tree

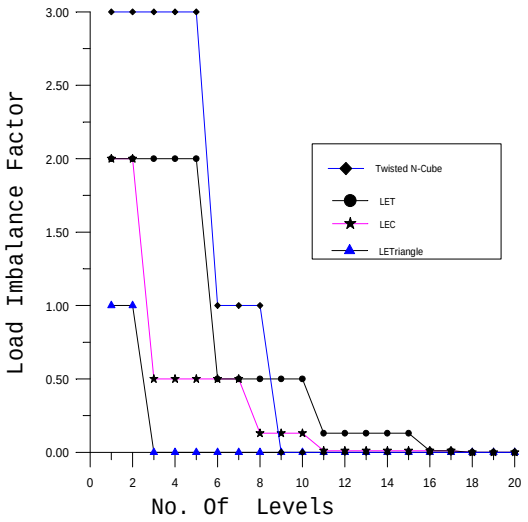


Fig. 4.3 LIF for Arbitrary Binary Tree

The same HBM scheme is implemented on the Twisted N cube, LET & LEC, for the same class of task tree. The simulation study indicates that the HBM scheduling is performing poorly but comparatively for all class of task tree in comparison to proposed network. The performance results indicate that the LIF is always higher for twisted N cube and same in case of LET & LEC in comparison to the proposed network.

The reason for better load balance in the proposed network for all class of tree-structured problems is that the diameter of the network is very small & constant having small number of processors. It may be argued that to obtain better load balance means higher performance, there should be a lessor diameter and better connectivity in the system.

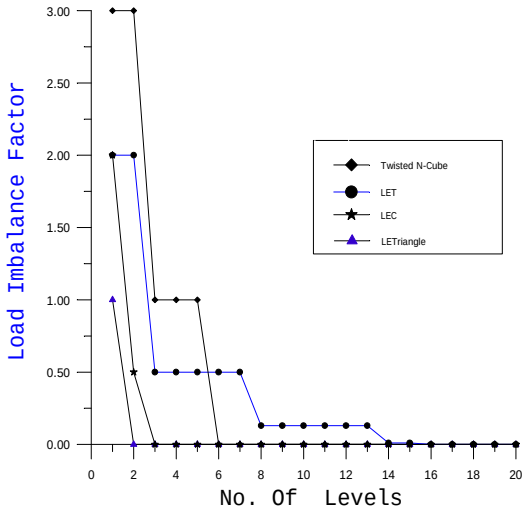


Fig. 4.4 LIF for Arbitrary Ternary Tree on HBM Scheme on different networks for various class of task structure

4.3 MDS Scheme on the proposed network and its comparison with other networks

The above-mentioned scheduling scheme is implemented on the above networks in the same environment. The simulation run consists of generating various classes of task trees and executing them on to the proposed network and on LEC, LET & Twisted N-cube networks. The estimation of LIF is obtained and the curves are plotted as the LIF against the problem size (in terms of the task tree depth) shown in Fig. 4.5 - 4.8.

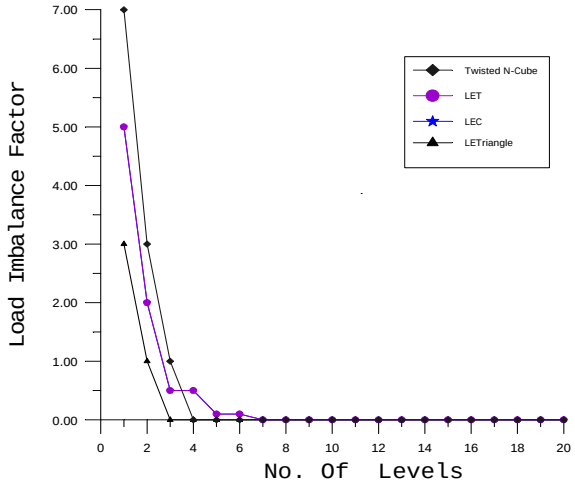


Fig. 4.5 LIF for Complete Binary Tree

The trend of the curves obtained, indicates same pattern of the LIF in case of complete binary and complete ternary task trees, starting from a peak value and then reducing to zero level, as the fair number of tasks are available. Where as, in case of arbitrary binary & ternary task trees, the MDS schemes implementation shows slow reduction from the initial peak as the random tree fails to get sufficient number of tasks. Once good numbers of tasks are available, the LIF reduces to its lowest values and the scheduling deviates the balancing trend again and again as it gets sufficient number of tasks. The complete binary & ternary task trees are performing better on the proposed network than other networks.

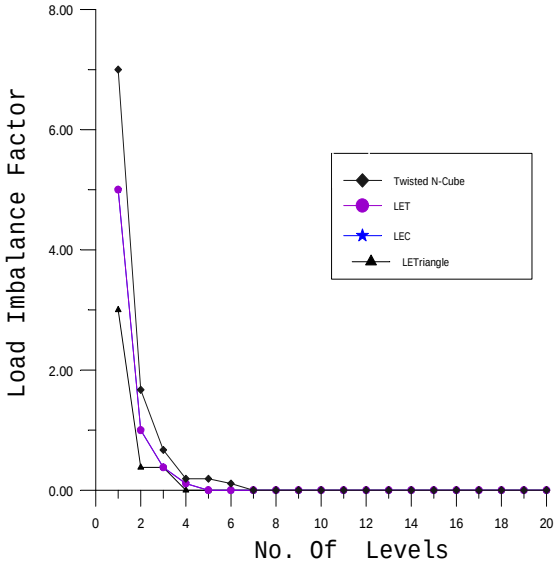


Fig. 4.6 LIF for Complete Ternary Tree

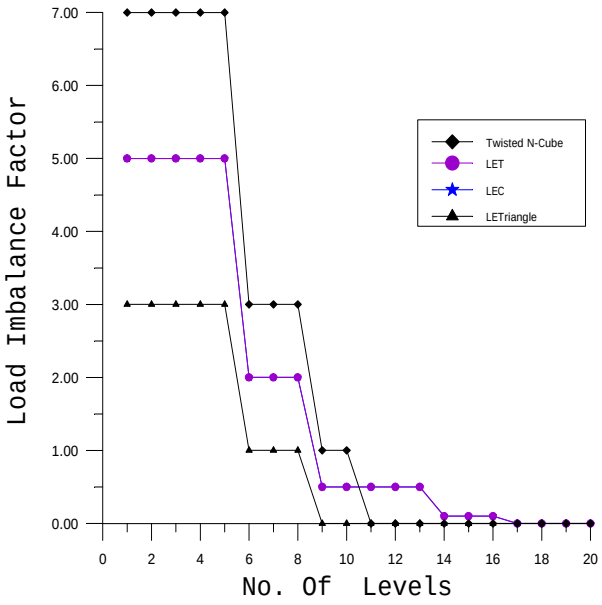


Fig 4.7 LIF for Arbitrary Ternary Tree

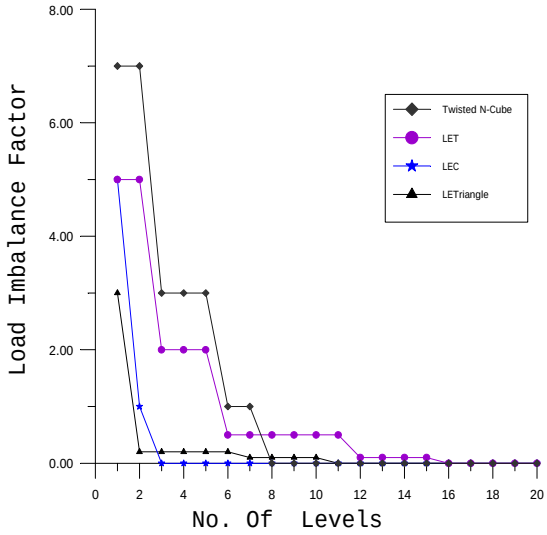


Fig 4.8 LIF for Arbitrary Binary Tree

MDS scheme on different networks for various class of task structures.

The curves also indicate that when both the scheduling schemes (HBM and MDS) are implemented on the proposed network for the same environment, both the schemes are performing equally good and giving optimal balancing. The curves shows that in case of complete binary and complete ternary task trees, the LIF is always reducing more rapidly than in case other networks.

The better performances of the proposed network for complete binary & complete ternary task trees may be attributed for the lesser diameter and lesser number of processing elements in comparison to other networks.

CONCLUSION

It may be concluded that the proposed triangle based interconnection topology with only 4 processors, is also a complete multiprocessor network, which exhibits the better performance in comparison to other similar networks. It is always possible to obtain a better load balance in a smaller and compact network. Hence the proposed network is on equal footing for comparison with other similar linearly extensible multiprocessor networks.

Linearly Extensible Triangle is a linearly growing structure with every extension requires only one processor thereby reducing the number of processors at various depths drastically. Hence the triangle-based network is a compact and economical architecture, which exhibits better properties.

From simulation studies, it has been found that for the same environment, both dynamic scheduling schemes are performing better on the proposed network in comparison to other similar networks particularly for tree structured problems. Therefore, it can be concluded that the Linearly Extensible Triangle-based multiprocessor network is a novel interconnection model for parallel evaluation of all types of problem graphs, particularly for tree-structured problems.

REFERENCES:

[1] Khaled D and Abdel, A A E "The Hyperstar Interconnection Network" Journal of Parallel and Distributed Computing 48, 64-98, 2008

[2] Min, W Y and Wei S " DDE: A Modified Dimension Exchange Method for Load Balancing in k-array n-cubes" Journal of Parallel and Distributed Computing 44, 88-96, 1997.

[3] Ganeshan E and Pradhan, D K "The hyper-de Bruijn networks: Scalable versatile architecture," IEEE Trans. on PDS, Vol.4, No.9, pp 962-978, Sept. 1993.

[4] 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, 1997.

[5] A.H.Esfahanian, L.M.Ni, and B.E. Sagan, "The Twisted N-Cube with application to multuiprocessing", IEEE Trans. on Computers, Vol 40, No.4, pp 88-93, Jan 1993.

[6] Bhuyan, L N and Aggarwal, D. P. "Design and analysis of general class of interconnection networks," in Proc. IEEE Int. Conf. Parallel Processing, Baltimore, MD, Aug.1982.

[7] Mohan Kumar J, and Patnaik, L.M., "Extended Hypercube: A hierarchical interconnection network of hypercube", IEEE Trans on PDS, vol. No.1, pp 45-57, Jan 1992.

[8] Rafiq, M.Q. "A Linearly Extensible Tree Network: Scalable Versatile Architecture" sent for publication in IEEE PDS (No 109754)

[9] Rafiq M.Q., Padam Kumar & Gupta, J.P, "A Novel Tree-Structured Multiprocessor Network" International Conference on Robotics Vision and Parallel Processing for Automation, Malaysia, July 16 - 18, 1999.

[10] Marc H. Willebeek-LeMair and Anthony P. Reeves; "Strategy for dynamic load balancing on highly parallel computers," IEEE Trans. on PDS, Vol. 4, No. 9, pp 979-993, Sept. 1993.

[11] Rafiq M.Q., M. Ba-Ru-K & A.Samad, " A Linearly Extensible Cube Network" sent for publication in IEEE, Jan. 2001