Minimum Distance Scheduling Scheme on Linearly Extensible Multiprocessor Network

Manaulah

Department of Electrical Engineering, Faculty of Engineering & Technology, Jamia Millia Islamia, New Delhi – 25

Abstract— In this millennium research is on to speed up more and more computation power by reducing the size of the system. In this direction, a novel Triangle based multiprocessor system has been proposed in this paper, which is compact in size (occupy lesser space) and exhibits all the features and characteristics of a commercial available multiprocessor. The proposed network has lesser number of processing elements, smaller diameter and fewer complexes. Its extension requires only one processing element per extension i.e. linearly extensible. Simulation studies show the better performance to other similar systems. The triangle-based architecture is therefore an economical and compact architecture, which is better suited to the need of the hour.

Keywords— Load Imbalance Factor, Scheduling, Multiprocessor Architecture, Linearly Extensible Network, 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.

Recently an organizational model has been reported as Linearly Extensible Tree (LET) network with a dynamic scheduling scheme i.e. 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].

In this paper, a new linearly extensible triangle-based architecture has been proposed and its properties have been compared with other similar architectures. Dynamic scheduling schemes, MDS [9] is implemented on this architecture and the performance parameters are obtained. These results are compared with other linearly extensible networks with the same scheduling schemes and results are plotted.

II. BASIC TOPOLOGICAL PROPERTIES

2.0 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 with small 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

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

The number of processors N in the network is given by

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

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 renamed as $P_{00}$ and so on. The arrangement is shown in the Fig. 2.1

\[
\begin{align*}
&P_{00} \\
&P_{11} \\
&P_{22} \\
&P_{33} \\
&P_{44} \\
&P_{55} \\
&P_{66} \\
&P_{77} \\
&P_{88} \\
&P_{99}
\end{align*}
\]

**Fig. 2.1 Processors in LET**

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

\[
Q' = \{ P_{ij} \mid 0 \leq i \leq N-1, \ 0 \leq j \leq d \}
\]

\[
\begin{bmatrix}
0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
\]

**Fig. 2.2 (a) Linearly Extensible Tree**

**Fig. 2.2 (b) Adjacency Matrix**

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

\[
\begin{align*}
L\left(P_{ij}\right) &= P_{(i+j+1) \mod N} \\
R\left(P_{ij}\right) &= P_{(i+j+2) \mod N} \quad \text{for all } P_{ij} \in Q'
\end{align*}
\]

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.

### 2.1 Design of the Proposed Multiprocessor Network

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

![Fig. 2.3 (a) Linearly Extensible Triangle](image)

![Fig. 2.3 (b) Adjacency Matrix](image)

Fig. 2.3 Network with 4 Processors

The details of the design of a triangle-based network topology are 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 as shown in Fig. 2.3 along with its adjacency matrix.

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. 2.4

### 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 \( N = 2^n \) (for \( n > 0 \)), whereas the number of nodes in the LET network \( N = \sum k \), where as the nodes in hypercube and deBruijn networks are \( 2^n \). In the proposed network, it is \( k+1 \), (for \( k > 3 \)) i.e. 4.

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 networks, the extensibility is exponential. In the proposed network, each extension requires only one node without changing the basic configuration.

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 constant at 4 where as in hypercube, the degree increases with the size of the system. In the proposed network it is \( (N-1) \).

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.

Table 1: Number of Processors for Various depth

<table>
<thead>
<tr>
<th>Depth</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEC</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>10</td>
<td>12</td>
<td>14</td>
<td>16</td>
</tr>
<tr>
<td>LET</td>
<td>1</td>
<td>3</td>
<td>6</td>
<td>10</td>
<td>15</td>
<td>21</td>
<td>28</td>
<td>36</td>
</tr>
<tr>
<td>Proposed Network(LEΔ)</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
</tbody>
</table>

Table 2: Comparative Diameter of various sized networks

<table>
<thead>
<tr>
<th>Depth</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEC</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>LET</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>Proposed Network(LEΔ)</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>
TABLE 3
Summary of parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Hypercube</th>
<th>LET</th>
<th>LEC</th>
<th>Proposed Network(LEA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of processor</td>
<td>(N = 2^n)</td>
<td>(N = \sum_{k=1}^{d+1} K)</td>
<td>(N = 2^n)</td>
<td>(N = \sum_{k=3}^{n} k + 1)</td>
</tr>
<tr>
<td>Degree</td>
<td>(N)</td>
<td>4</td>
<td>4</td>
<td>(N - 1)</td>
</tr>
<tr>
<td>Extensibility</td>
<td>(2^n)</td>
<td>(N+1)</td>
<td>2</td>
<td>(N+1)</td>
</tr>
<tr>
<td>Diameter</td>
<td>(O(\log N))</td>
<td>(O(\sqrt{N}))</td>
<td>(O(N))</td>
<td>2</td>
</tr>
</tbody>
</table>

III. THE 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.

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 \(\text{LIF}_k\), is defined as:

\[
\text{LIF}_k = \frac{\text{max} \{ \text{load}_k(P_i) \} - \text{ideal-load}_k}{\text{ideal-load}_k}
\]

Where \((\text{ideal-load})_k = \left[ \text{load}_k(P_0) + \text{load}_k(P_1) + \ldots + \text{load}_k(P_{N-1}) \right] / N,\)

And \((\text{load}_k(P_i))\) 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:

```c
mds()
{
  map root_task on P_0;
  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 ++)
    {
      T_c = select (list);
      /* select(list) retrieves a task from the list */
      store(T_c);
      T_f = father(T_c);
      /* father(task) returns the father of the task */
      P_t = processor(T_f);
      /* processor(task) returns the processor on which task is scheduled */
      map T_c on P_t; /* this is zero distance scheduling */
    }
    update (k); /* update (k) modifies the kth row of LT */
    schedule (k);
    k = k+1;
  } while (k < k_max);
}
```

The load balancing profitability for kth level of task tree, denoted as \(\text{LIF}_k\), is defined as:

\[
\text{LIF}_k = \frac{\text{max} \{ \text{load}_k(P_i) \} - \text{ideal-load}_k}{\text{ideal-load}_k}
\]

Where \((\text{ideal-load})_k = \left[ \text{load}_k(P_0) + \text{load}_k(P_1) + \ldots + \text{load}_k(P_{N-1}) \right] / N,\)

And \((\text{load}_k(P_i))\) 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:

```c
mds()
{
  map root_task on P_0;
  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 ++)
    {
      T_c = select (list);
      /* select(list) retrieves a task from the list */
      store(T_c);
      T_f = father(T_c);
      /* father(task) returns the father of the task */
      P_t = processor(T_f);
      /* processor(task) returns the processor on which task is scheduled */
      map T_c on P_t; /* this is zero distance scheduling */
    }
    update (k); /* update (k) modifies the kth row of LT */
    schedule (k);
    k = k+1;
  } while (k < k_max);
}
```

...
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. 2.5 - 2.8.

**IV. 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 MDS scheduling scheme. 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 under loaded 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 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 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.

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.

V. 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