ELSEVIER

Microprocessing and Microprogramming 41 (1995) 273–288

Microprocessing and Microprogramming

On synthesizing cube and tree for parallel processing

S.K. Basu [a], J. Datta Gupta [b,*], R. Datta Gupta [c]

[a] *Computer Centre, Banaras Hindu University, Varanasi-221005, India*
[b] *Electronics Unit, Indian Statistical Institute, 203, B.T. Road, Calcutta-700035, India*
[c] *Department of Computer Science & Engineering, Jadavpur University, Calcutta 700032, India*

Received 13 April 1994; revised 19 July 1994; accepted 12 May 1995

Abstract

In this paper we propose a VLSI implementable architecture called Cube Connected Tree having advantageous properties of both tree and hypercube. This structure has a fixed low degree of nodes for any size of the network unlike the hypercube where the node degree is dependent on the size of the hypercube. The degree-diameter product metric [26] of CCT is low compared to that of a hypercube of comparable size. It overcomes the data congestion problem near the root of the binary tree by having multiple roots in the structure, thereby enhancing the I/O bandwidth of the system. The complexity of the VLSI layout of this structure has been addressed within the grid model of Thompson [12]. By using spare links and PEs, fault tolerance capabilities of the system have been enhanced. Easy programmability of this structure has been demonstrated by designing polylogarithmic algorithms for sorting and discrete Fourier transform.

Keywords: Parallel architecture; VLSI layout; Tree; Hypercube; Parallel algorithm; Sorting; Fourier transform

1. Introduction

Tree architecture has a number of good properties such as low degree nodes, efficient VLSI layout, fault-tolerance capability with a reasonable amount of sparing, straightforward mapping of algorithms, and easiness of routing data. Many real-life problems require an array of input and produce an array of output. The performance of the simple binary tree architecture is severely limited by the constraint imposed by the congestion of data near the root processor [11]. Though the hypercube has a good algorithmic performance for some of these problems it requires variable degree nodes dependent on the size of the hypercube. Requirement of large degree nodes is not a desirable feature from the point of VLSI implementation. Such a structure having cube connections and also having properties of trees might

* Communicating author.

prove to be a good VLSI architecture for parallel computation, as it retains good properties from both tree and hypercube.

A number of architectures such as Orthogonal tree [4], Fat trees [9], X-tree [11], Diamond network [29], Compressed tree [22], etc. based on tree structure have been proposed and studied in the literature. Other efforts have sought to exploit new interconnection networks based on the hypercube topology. Cube-Connected Cycles [15] proposed by Preparata and Vuillemin can efficiently solve certain classes of problems. The Folded hypercube [24] by Latifi and El-Amawy is a standard hypercube with extra links that significantly increase the efficiency of the routing algorithms and reduce the network diameter. The Twisted hypercube [23] is the standard hypercube where two or more edges are exchanged. The enhanced incomplete hypercube [25] improves the performance of the incomplete hypercube by addition of extra links between pairs of nodes having unused ports. The Generalized Boolean n-cube [30] has many properties of the Boolean n-cube and has lower degree nodes. The Generalized hypercube and Hyperbus [26] by Bhuyan and Agrawal based on a mixed radix number system results in a variety of hypercube structures for a given number of processors. The Cube-connected-cubes [21] is a hypercube with each node represented as a cube. It was designed for the divide-and-conquer paradigm and in particular for distributed environments with heavy localized communication. The Crossed-cube [20] network has many of the properties of the hypercube but has a diameter only about half as large. Banyan-hypercubes [19] by Youssef and Narahari combine the advantageous features of both Banyans and hypercubes and thus have better communication capabilities. Banyan-hypercubes offer an improvement over hypercubes in diameter, average distance, and embedding of hierarchical structures. The Hypertree [8] has some advantages of cube connections but retains many of the problems of the tree machine. Moreover, programming this structure seems to be complex for many parallel processing applications. Hence a new structure called Cube-Connected-Tree or CCT in short is proposed below which contains advantageous features of both tree and hypercube. This paper is organized as follows: Section 2 defines the architecture, Section 3 deals with its VLSI area complexity, Section 4 with fault tolerance issues, Section 5 with mapping of algorithms on it and Section 6 concludes the paper.

2. Architecture

N^2-pruned full binary trees each with $2 \log N$ leaves [N is a positive integer power of 2] are connected in the way explained below for construction of an $N \times N$ CCT. A pruned full binary tree with $2 \log N$ leaves is obtained as follows: A full binary tree with height $= \lceil \log \log N \rceil + 1$ is taken. If $\log \log N = \lceil \log \log N \rceil$ then this is accepted as a pruned full binary tree with $2 \log N$ leaves. Otherwise, the first $2 \log N$ leaves from the left of this tree are retained and the remaining leaves are pruned. This will create a number of leaf nodes at a lower height. These leaves are also pruned and the process is continued till no leaves remain at lower heights. Illustrative diagrams for pruned full binary trees with 4, 6, 8 and 10 leaves are shown in Fig. (1).

These trees are connected through cube connections via their leaves. N^2 trees are numbered as $0, 1, 2, \ldots, (N^2 - 1)$ from the north to south direction and their leaves are numbered as $0, 1, 2, \ldots, (2 \log N - 1)$ starting from the leftmost leaf. Two trees would be connected by a cube link through the jth leaves if the binary representation of the numbering of the trees differs only in the jth bit position. Thus tree number 0 would be connected to tree number 1 through their leftmost leaves, to tree number 2 through the second leftmost leaves (i.e. leaf number 1), to tree number 4 through the 3rd leftmost leaves (i.e. leaf number 2) and so on. When all such connections have been made, the resulting structure

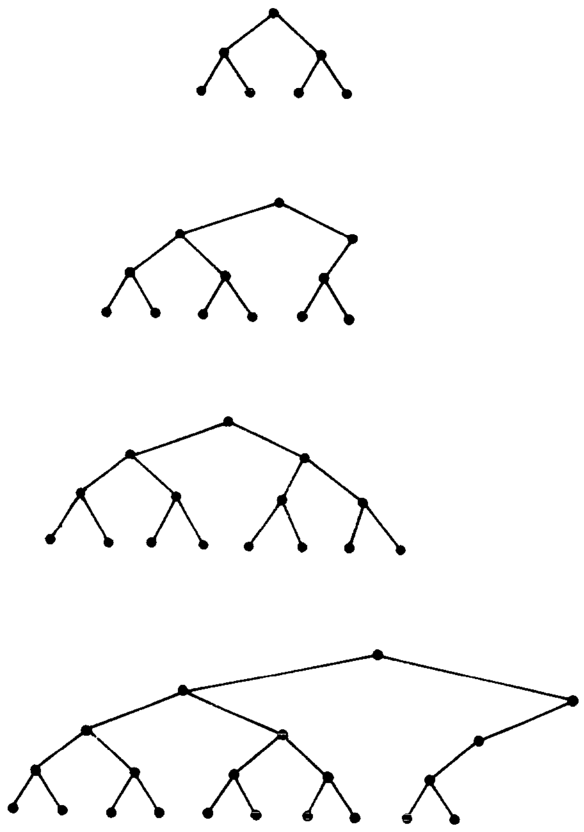
Fig. 1. Pruned full binary trees with different number of leaves.

is called a Cube Connected Tree. A pruned tree of height $\lceil 1 + \log\log N \rceil$ contains

$$\sum_{i=0}^{i=\lceil 1+\log\log N\rceil} \lceil \log N/2**(i-1)\rceil$$

nodes and hence a CCT of size $N \times N$ contains in all

$$\sum_{i=0}^{i=\lceil 1+\log\log N\rceil} N^2 * \lceil \log N/2**(i-1)\rceil$$

nodes which is $O(N^2 \log N)$. The number of cube links is $N^2 \log N$ and total number of tree links is

$$N^2 * \sum_{i=0}^{i=\lceil \log\log N\rceil} \lceil \log N/2**(i-1)\rceil$$

and hence the total number of links in a CCT of size $N \times N$ is

$$N^2 \left(\log N + \sum_{i=0}^{i=\lceil \log\log N\rceil} \lceil \log N/2**(i-1)\rceil \right).$$

The diameter of this structure is $2(\log N + \lceil 1 + \log\log N \rceil)$ and its line connectivity is two. The total number of nodes and links present in a 4×4 CCT are 112 and 128 respectively and its diameter is 8. The degree-diameter product [26] metric of CCT is low compared to that of a hypercube of comparable size. Each PE in a CCT has a number of registers and some local memory. Each leaf PE has an additional register for indicating data routing either through tree links or through cube links. Inputs and outputs are made through the roots of the trees. For an $N \times N$ CCT, simultaneously N^2 data items may be input or output through the roots of N^2 trees. CCT creates a parallel computing environment along with a sequential host machine.

3. VLSI layout

Our CCT has all the characteristics suitable for VLSI implementation. It is regular and the maximum degree of any node in CCT is three. We give two layouts of the CCT under the grid model of Thompson [12]. The first layout uses a modified version of the H-tree. The H-tree [2,10] is a compact layout for binary tree. A binary tree of height h may be laid out using H-tree which requires HT(h) horizontal tracks and VT(h) vertical tracks where

$$\begin{aligned} \mathrm{HT}(h) &= 2*\mathrm{HT}(h-1)+1 \\ &\qquad \text{if } h \text{ is an odd number greater than 2} \\ &= 2*HT(\mathrm{h}-2)+1 \\ &\qquad \text{if } h \text{ is an even number greater than 3,} \\ VT(\mathrm{h}) &= VT(\mathrm{h}-1) \\ &\qquad \text{if } h \text{ is an odd number greater than 2} \\ &= 2*VT(\mathrm{h}-1)+1 \\ &\qquad \text{if } h \text{ is an even number greater than 3,} \end{aligned}$$

and HT(2) = VT(2) = 3. The solutions of these recurrence relations are

$$\begin{aligned}\mathrm{HT}(h) &= 2^{(h+3)/2} - 1 \text{ f } h \text{ is odd}\\ &= 2^{(h+2)/2} - 1 \text{ if } h \text{ is even}\\ \text{and } \mathrm{VT}(h) &= 2^{(h+1)/2} - 1 \text{ if } h \text{ is odd}\\ &= 2^{(h+2)/2} - 1 \text{ if } h \text{ is even.}\end{aligned}$$

To use the H-tree for laying out a CCT, the compactness of the H-tree is sacrificed to make room for cube connections. N^2 pruned trees each with 2 log N leaves are laid on the grid using the H-tree layout one after another in the north to south direction. To make room for cube connections, $\sum_{i=0}^{i=2\log N} 2^i = N^2 - 1$ vertical tracks are additionally created in the above layout as described below:

Let VT(h) denote the number of vertical tracks in the H-tree layout of a binary tree of height h as assumed above. Let these tracks be referred to as 1, 2, 3,... in the left to right direction. The number of vertical tracks in the H-tree layout containing leaf nodes is $\lceil \mathrm{VT}(h)/2 \rceil$. The number of leaves in a binary tree of height h is 2^h and hence the number of leaves per vertical track containing leaf nodes is $\mathrm{LN} = 2^h/\lceil \mathrm{VT}(h)/2 \rceil$. The following algorithm generates the required number of additional vertical tracks for laying the cube connections.

```
$j = 0$;
For $i = 1$ to VT($h$), Step 2;
$M = \sum_{k=j*\mathrm{LN}}^{k=(j+1)\mathrm{LN}-1} 2^k$;
If $j$ is even,  Then insert $M$ vertical tracks between existing vertical tracks numbers $i$ and $i+1$; Else insert $M$ vertical tracks between existing vertical tracks numbers $i$ and $i-1$;
$j = j + 1$;
End.
```

Once these additional vertical tracks are created, the cube connections among the leaves of the pruned trees are inserted using these tracks as shown for a 4 × 4 CCT in Fig. (2(a)). The number of horizontal tracks required for an $N \times N$ CCT is

$N^2(4 * 2^{\lceil \log\log N \rceil/2} - 1)$ if $\lceil \log\log N \rceil$ is even,
and $N^2\left((8)^{0.5} * 2^{\lceil \log\log N \rceil/2} - 1\right)$
if $\lceil \log\log N \rceil$ is odd.

The total number of vertical tracks required is
$N^2 + 2 * 2^{\lceil \log\log N \rceil/2} - 2$ if $\lceil \log\log N \rceil$ is even and
$N^2 + (8)^{0.5} * 2^{\lceil \log\log N \rceil/2} - 2$ if $\lceil \log\log N \rceil$ is odd.

(a)

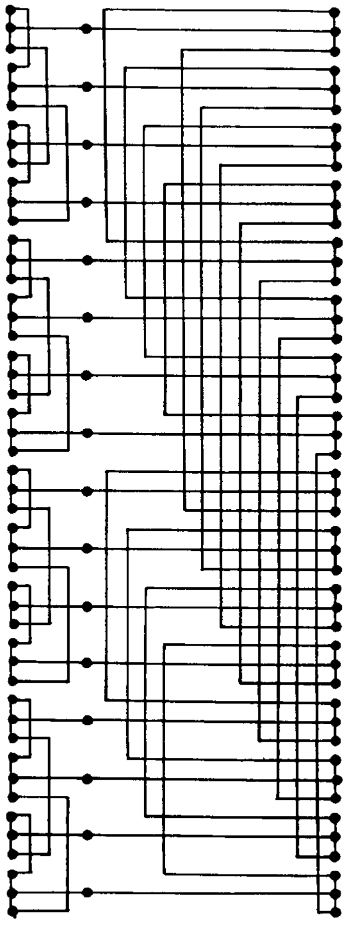

Fig. 2(a). Layout of a 4×4 CCT using H-tree.

(b)

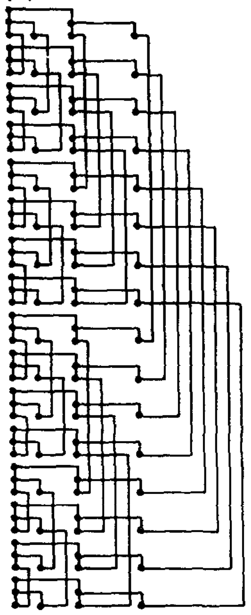

Fig. 2(b). Layout of a 4×4 CCT.

Hence the length of the longest wire and the area of this layout are O(N^2 $\log^{0.5}$ N) and O(N^4 $\log^{0.5}$) respectively.

An alternative layout without using the H-tree is given below. This layout requires O(N^4 log log N) areas and has O(N^2 log log N) as the length of the longest wire. The pruned binary trees in the CCT are numbered in the same way as assumed in Section 2. In each tree, leaves are assumed to be in level number one and root at the highest level. The PEs in each level are numbered 0, 1, 2, ..., from left to right. The PEs of a particular level of a tree are placed on the grid points of the same horizontal line. In general, the ith PE of level l is placed at a distance of

$$\sum_{k=1}^{k=2^{l-1}} \left(1 + 2^{(2-^{l-1})i-k}\right)$$

from the $(i-1)$th PE of the same level of the tree, if it exists. Nodes of successive levels of a pruned tree are placed on consecutive horizontal tracks. In this way, all the nodes of a pruned binary tree are laid out. All the N^2 pruned full binary trees are laid out similarly keeping a vertical distance of unity between the leaves of one tree and the root of the next tree as shown for a 4×4 CCT in Figure (2(b)). By using the increasing number of free vertical tracks available between successive leaves of the trees, the cube connections can be established among the N^2 pruned full binary trees through their leaves as shown in Fig. (2(b)).

An $N \times N$ CCT requires $N^2(2 + \lceil \log \log N \rceil)$ horizontal and $2 \log N + (N^2 - 1)$ vertical tracks for laying on a VLSI chip. Hence the area required is

$$N^2(2 + \lceil \log \log N \rceil) * (2 \log N + N^2 - 1)$$

which is of the order of N^4 log log N. The length of the longest wire is $N^2(4 + \lceil \log \log N \rceil)/2$ and hence O(N^2 log log N).

4. Fault tolerant CCT

Raghavendra et al. [13] suggested methods for enhancement of the reliability of a binary tree machine by systematic use of spare processing elements under the assumption that the links are perfectly reliable but the PEs may fail. We have extended their schemes and applied it to the CCT. With the same assumptions, we consider fault-tolerance of CCT under four different schemes of redundancy and evaluate their relative suitability. The schemes considered here are level sparing 1 (LS-1), level sparing 2 (LS-2), performance degradation (PD) and combined (CBD). Two metrices, system reliability and reliability improvement per spare PE are used for this purpose. The first metric is more important from the point of performance and the second from the point of cost.

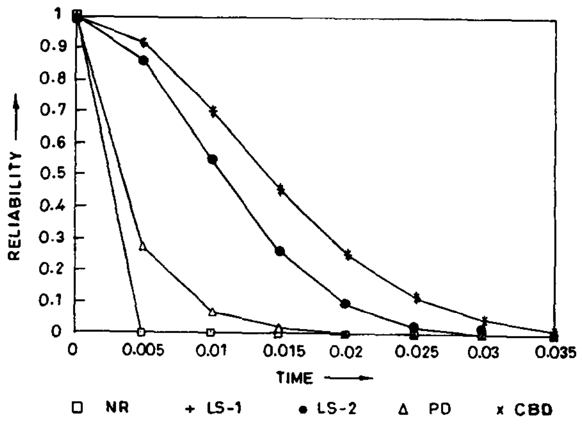

Fig. 3. Reliability of CCT under different schemes of fault-tolerance.

Let us denote $\lceil 1+\log\log N\rceil$ by L, $\lceil(\log N)/2^{i-1}\rceil$ by p. We assume that all the nodes are equally reliable and the reliability of a node is given by $R=e^{-\lambda t}$, where λ is the failure rate and t is time. The reliability expression (A) for a nonredundant $N\times N$ CCT (represented as NR in Fig. 3) may be expressed as

$$A=R^{N^2 * \sum_{i=0}^{i=L} p}$$

In level sparing 1 scheme, one spare PE is used for each level of the pruned trees of CCT. Level i of each pruned tree of CCT has p original PEs and one spare PE. The reliability of this level for a particular tree is given by $R^{p+1}+R^p(1-R)+pCR^p(1-R)$ where C is the coverage factor, i.e. the conditional probability of successful recovery after the occurrence of a fault. The reliability of the system is the product of all such terms for levels 0 through L raised to power N^2 since there are N^2 pruned full binary trees in CCT. Hence the system reliability may be written as

$$\prod_{i=0}^{i=L}\left(R^{p+1}+R^p(1-R)+pCR^p(1-R)\right)^{N^2}.$$

Let this expression be called B. The number of redundant PEs required for this scheme is $N^2(1+L)$ and hence the reliability improvement factor per spare PE under this scheme may be written as $B/(A*N^2(1+L))$.

For a redundant CCT under level sparing 2 scheme, one spare PE is used for each level of the pruned trees having k (k is an integer power of 2) or less PEs but for the other levels one spare PE is used for a maximum group of k consecutive PEs. Let us denote $\lfloor p/k\rfloor$ by Q. The reliability of a level containing k or less PEs is given by $\alpha=R^{p+1}+R^p(1-R)+pCR^p(1-R)$ where $p\le k$. The reliability of a level containing more than k PEs is given by

$$\beta=\left(R^{k+1}+R^k(1-R)+kCR^k(1-R)\right)^Q$$
$$*\left(R^{p-kQ+1}+R^{p-kQ}(1-R)\right.$$
$$\left.+(p-kQ)CR^{p-kQ}(1-R)\right).$$

The first factor is the reliability of Q groups of k PEs each and the second factor is the reliability of the remaining PEs in that level. For each group of k PEs one spare PE and for the remaining $p-kQ$ PEs one spare PE are used. The reliability of the system is the product of α^{N^2} taken over all levels containing upto k PEs and β^{N^2} taken over levels containing more than k PEs. Hence the system reliability becomes

$$\prod_{i=L}^{i=j \text{ s.t. } (\log N)/2^{j-1}\le k}\left(R^{p+1}+R^p(1-R)\right.$$
$$\left.+PCR^p(1-R)\right)^{N^2}$$
$$*\prod_{i=j-1}^{i=0}\left(R^{k+1}+R^k(1-R)+kCR^k(1-R)\right)^{N^2Q}$$
$$*\left(R^{p-kQ+1}+R^{p-kQ}(1-R)\right.$$
$$\left.+(p-kQ)CR^{p-kQ}(1-R)\right)^{N^2}.$$

Let this expression be referred to as D. This scheme would be requiring

$$N^2 \left\{ \sum_{i=L}^{i=\lceil 1+\log(\frac{\log N}{k})\rceil} (p+1) + \sum_{i=\lceil 1+\log(\frac{\log N}{k})\rceil - 1}^{i=0} (P+Q+1-\delta_{p,kQ}) - \sum_{i=0}^{i=L} p \right\}$$

additional PES, where gd is the Kronecker delta function. Let this expression be represented by E. Hence the reliability improvement per spare PE under this scheme is given by $D/(A*E)$.

The performance degradation scheme uses one spare PE for each of the pruned trees of a CCT and hence a total of N^2 spare PEs is used. For levels containing p PEs, there are $u = \lfloor p/2 \rfloor$ pairs where failure of one PE is tolerated. Reliability of a pair is $R^2 + 2CR(1-R)$ and reliability of all such pairs is $(R^2 + 2CR(1-R))^u$. The reliability of this level is $(R^2+2CR(1-R))^u * R^{p-2u}$ where R^{p-2u} is the contribution from unpaired PEs, if there are any in that level. Reliability of the system is the product of such terms for level 0 through L and then the whole expression raised to power N^2, since there are N^2 pruned full binary trees in the CCT. Hence the system reliability expression may be written as

$$\prod_{i=L}^{i=0} \left((R^2 + 2CR(1-R))^u * R^{(p-2u)} \right)^{N^2}.$$

Let this expression be called F. The reliability improvement per spare PE is then given by $F/(N^2 * A)$.

In a combined scheme, level sparing is employed for level number 0 to level number d (for some suitable d), and performance degradation for other levels near the leaves of the trees of the CCT. The contribution to the system reliability from levels d through L may be written as

$$\prod_{i=d}^{i=L} \left(R^{p+1} + R^p(1-R) + pCR^p(1-R) \right)^{N^2}$$

and those of levels 0 through $d-1$ employing performance degradation as

$$\prod_{i=0}^{i=d-1} \left((R^2 + 2CR(1-R))^u * R^{p-2u} \right)^{N^2}.$$

Hence the overall system reliability may be written as

$$\prod_{i=L}^{i=d} \left(R^{p+1} + R^p(1-R) + pCR^p(1-R) \right)^{N^2} * \times \prod_{i=d-1}^{i=0} \left\{ \left[R^2 + 2CR(1-R) \right]^u * R^{(p-2u)} \right\}^{N^2}.$$

Let us call this expression G. The number of spare PEs required under this scheme is $N^2(L-d+1)$. Hence the reliability improvement per spare PE is $G/(A * N^2(L-d+1))$.

In Table 1, we give the percentage of PE redundancy required under different schemes for a $2^5 \times 2^5$

Table 1
Percentage of processor redundancy and reliability improvement factor per spare PE for $2^5 \times 2^5$ CCT under different schemes of fault-tolerance

Scheme	% Processor redundancy	Reliability improvement
Nonredundant	0	0.0
Level sparing 1	50	0.544E−03
Level sparing 2	50	0.0
Performance degradation	10	0.200E−02
Combined	50	0.544E−03

CCT and the corresponding reliability improvement per spare PE.

In Table 2, we give the reliabilities of CCT under different schemes. We assume that $C = 1$, $\lambda = 1$, $N = 16$, $K = 4$, and $d = 3$.

Among the four schemes of fault-tolerance for CCT, level sparing 1 and combined schemes provide the highest reliabilities. They have become identical in the plot of reliability against time as shown in Fig. (3). The level sparing 2 scheme ranks next to this. The performance degradation scheme ranks last. For binary tree, performance degradation and combined schemes provide nearly the same reliabilities which are much higher than those provided by the level sparing schemes. For CCT, the reliability improvement per spare PE is highest in the performance degradation scheme. Level sparing 1 and combined schemes provide almost identical improvements in reliability per spare PE and these are lower than that provided by the performance degradation scheme. For the binary tree, the reliability improvement per spare PE is highest in the performance degradation scheme followed by that in the combined scheme. The level sparing 1 scheme ranks next to the combined scheme and is followed by the level sparing 2 scheme in terms of reliability improvement per spare PE for binary tree.

5. Mapping of algorithms

CCT is a multipurpose architecture and not tailored to any particular application. We demonstrate programmability of CCT by mapping bitonic sorting and discrete Fourier transform on it. Unless specified otherwise, all log used below stands for logarithm base 2. For these two problems, we assume that the number of data points is $2N^2 \log N$ and these data points are distributed among the leaves of an $N \times N$ CCT, one data point per leaf PE. We assume that the N^2 pruned full binary trees of an $N \times N$ CCT are numbered 1 through N^2 in the north to south direction and the leaves of each tree are numbered 1 through $2 \log N$ in the left to right direction. Since there are $2 \log N$ leaves in each of the trees of an $N \times N$ CCT, for all trees in parallel data may be broadcast from any leaf of the tree to all other leaves of the tree in $O(\log \log N)$ time. Data may also be sent from any leaf j to any other leaf i of the same tree in $O(\log \log N)$ time for all trees in parallel.

5.1. Sorting

We first discuss bitonic sorting on tree because this would be utilized for sorting data using CCT. We assume that the tree has N leaves and the items

Table 2
Reliabilities of $2^4 \times 2^4$ CCT under different schemes of fault-tolerance

Reliabilities Time	Nonredundant	Level sparing 1	Level sparing 2	Performance degradation	Combined
0.000	.1000E + 01	.1000E + 01	.1000E + 01	.1000E + 01	.1000E + 01
0.005	.1284E − 03	.9153E + 00	.8594E + 00	.2728E + 00	.9153E + 00
0.010	.1649E − 07	.7051E + 00	.5499E + 00	.7164E − 01	.7051E + 00
0.015	.2119E − 11	.4599E + 00	.2650E + 00	.1813E − 01	.4599E + 00
0.020	.2722E − 15	.2556E + 00	.9724E − 01	.4422E − 02	.2556E + 00
0.025	.3496E − 19	.1217E + 00	.2745E − 01	.1040E − 02	.1217E + 00
0.030	.4491E − 23	.4990E − 01	.6017E − 02	.2361E − 03	.4990E − 01
0.035	.5768E − 27	.1771E − 01	.1034E − 02	.5175E − 04	.1771E − 01

of the list to be sorted are distributed among the leaves of the tree, one item per leaf in the left to right direction. Two fields are attached to each data item in the leaves. They are: position of the data item in the sequence which is the same as the left to right numbering of the leaf PE containing the data item, and a flag indicating the next direction of motion of the data through the tree, up indicated by '1' and down indicated by '0'.

The computation takes place in a pipelined fashion and requires log N phases. Phase 1 merges $N/2$ bitonic sequences each of length 2 to produce $N/2^2$ bitonic sequences each of length 2^2. Phase 2 merges $N/2^2$ bitonic sequences of length 2^2 to produce $N/2^3$ bitonic sequences each of length 2^3. In general, phase $i(1 \leq i < \log N)$ merges $N/2^i$ bitonic sequences of length 2^i to produce $N/2^{i+1}$ bitonic sequences each of length 2^{i+1}. The last phase, i.e. the log Nth phase merges a bitonic sequence of length N to produce a sorted sequence. Merging in phase $i(1 \leq i \leq \log N)$ first utilizes $N/2^i$ disjoint subtrees each of height i in parallel, then $N/2^{i-1}$ disjoint subtrees each of height $i-1$ in parallel and ultimately with $N/2$ disjoint subtrees each of height 1 in parallel. Disjoint subtrees are obtained successively by the logical removal of the root of the previous subtree. Thus by removing the root of the original tree, we get two disjoint subtrees of height $\log N - 1$. Then by logically removing the two roots of the two subtrees of height $\log N - 1$, we get 4 disjoint subtrees of height $\log N - 2$ and so on. These subtrees are referred to by numbers in the range 1 to $N/2$ from left to right. Roots of the subtrees behave in two different ways. Either they send the larger of the two data items to the right child and the smaller to the left child or they send the smaller data to the right child and the larger to the left child. These two situations are shown in the attached diagrams by horizontal arrows piercing the roots of the subtrees. These arrows are directed either left to right representing the situation where larger data goes to the right child and smaller goes to the left child or right to left representing the reverse situation. We now discuss how to set the arrow directions in the roots of the subtrees. In any merging phase, say i, the roots of the initial $N/2^i$ disjoint subtrees each of height i are programmed to have arrows directed left to right and right to left alternately starting from the leftmost subtree. Then we split the $N/2^i$ disjoint subtrees of height i into $N/2^{i-1}$ disjoint subtrees of height $i-1$. The roots of these $N/2^{i-1}$ subtrees will have arrow directions the same as it were with their respective parent node in the previous step. This continues till all disjoint subtrees each of height 1 are obtained. Then the next merging phase starts and the setting of arrows follows the same pattern. The algorithm for sorting on the tree follows:

```
DO for $i = 1$ to log $N$
    DO for $j = i$ to 1 Step $-1$
      DO in parallel for all $N/2^j$ subtrees of
      height $j$ each
       { For $k = 1$ to $2^{j-1}$
          For Leaf PEs DO {
    /* Leaves of the disjoint subtrees are num-
    bered separately by 1, 2, 3,.... in the left to
    right direction */
                Set Direction = '1';
                Set the position of data among
                the leaves of this subtree in a left
                to right numbering;
                Send the record from PE($k$) and
                PE($k + 2^{j-1}$) to the respective
                parents;
                Set direction = '0' for the data
                received from the parent;
                  }
      For intermediate PE DO
                     {
                      The received data from
                      child is sent to its parent
                      PE;
                      The received data from the
                      parent PE is transmitted to
                      its child PE according to
```

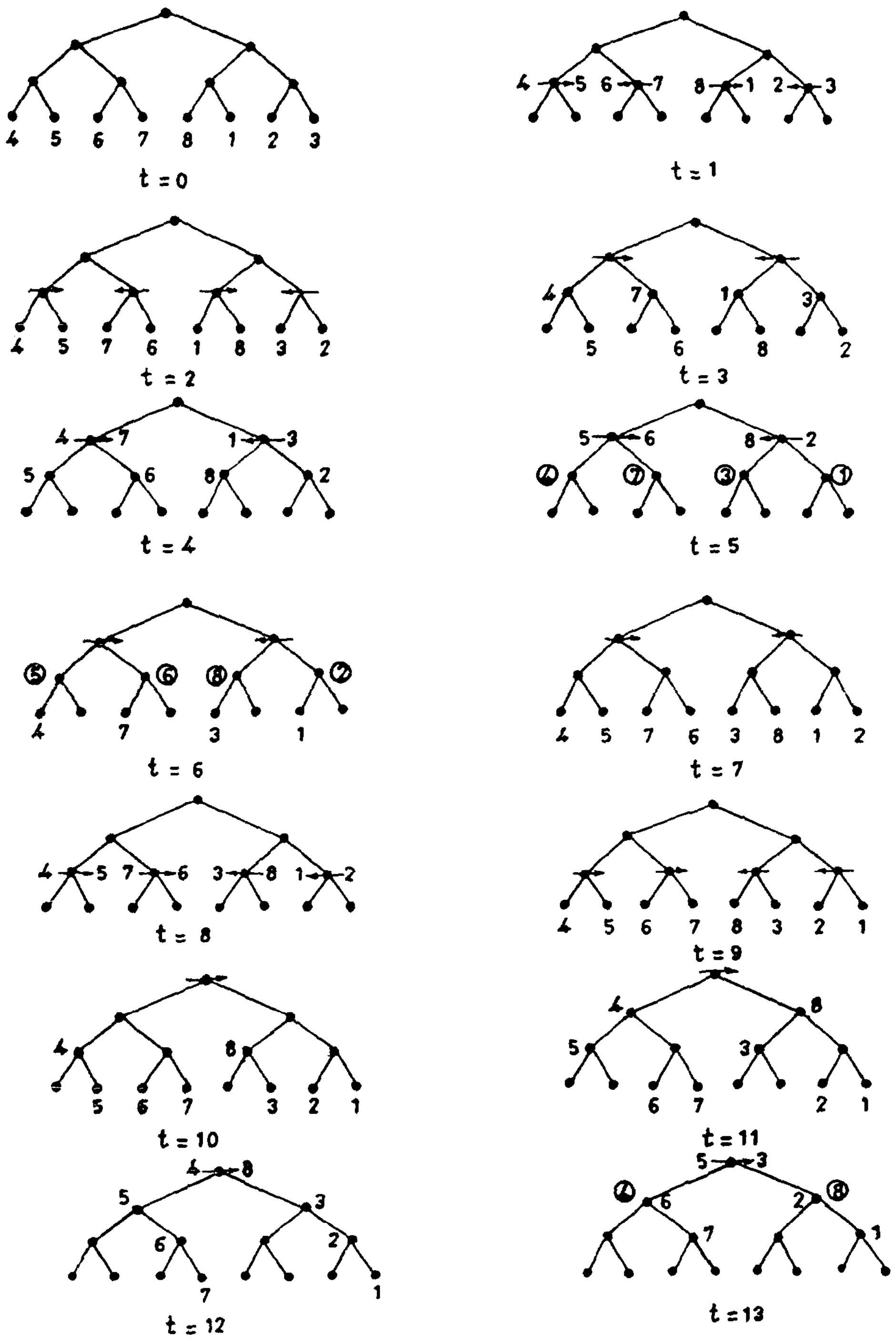

Fig. 4. Sorting on tree.

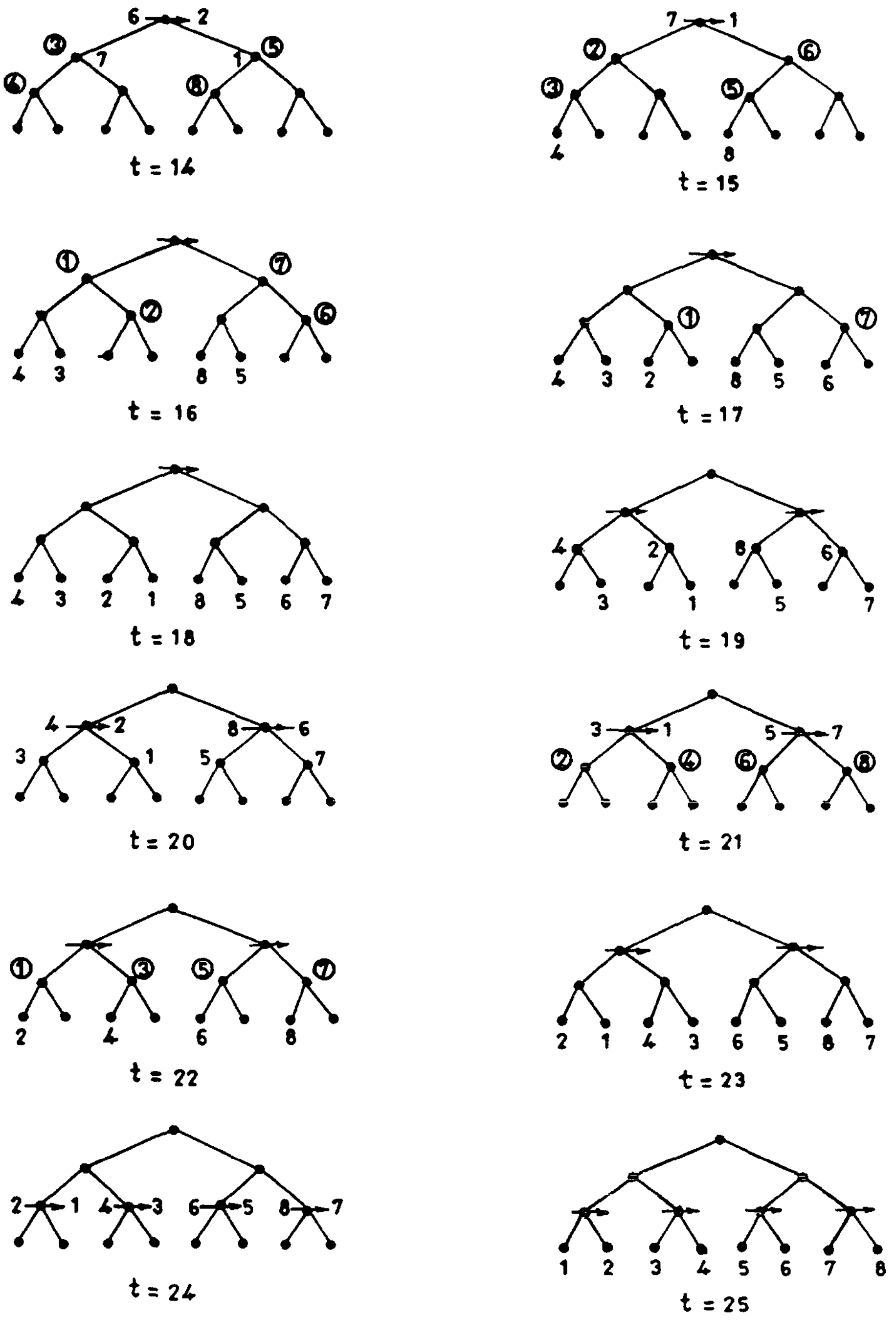

Fig. 4 (continued).

```
                    the content of the location
                    field;
                  }
      For root PE DO
                  {
                    Set the direction fields of
                    the two data records from
                    its two children to '0';
                    Compare the data fields and
                    if the data received from
                    the left child is greater than
                    that received from the right
                    child and the arrow direc-
                    tion of the root PE is left to
                    right or the data received
                    from the right child is
                    larger than that received
                    from the left child and the
                    arrow direction is right to
                    left, then exchange their
                    data fields;
                    Send these records to the
                    children according to their
                    location fields;
                  }
      }
      END
    END
END
```

A snapshot of the algorithm is shown in Fig. (4) where the tree has 8 leaves. Data items moving down the tree are shown by encircling them. Uncircled data items are moving up the tree. In analyzing the algorithm, we note that phase i $(1 \le i \le \log N)$ utilizes $N/2^i$ disjoint subtrees of height i in parallel, then $N/2^{i-1}$ disjoint subtrees of height $i-1$ in parallel and ultimately $N/2$ disjoint subtrees of height 1 each in parallel. Computation for phase i requires

$$\sum_{i=i}^{i=1} (2i + 2^{i-1} - 1)$$

time steps which is equal to $i^2 + 2^i - 1$. Hence the total time required for sorting on tree is

$$\sum_{i=1}^{i=\log N} (i^2 + 2^i - 1)$$

$$= (1/6)(2\log N + \log N)(\log N + 1)$$

$$+ 2(N-1) - \log N \quad \text{which is } O(N).$$

We now use the above algorithm to sort $2N^2 \log N$ data items using an $N \times N$ CCT. Let us assume that the data items are distributed one per leaf PE of a CCT. Initially, using the above algorithm, the data items of N^2 trees are sorted simultaneously, the data items of the 1st tree are sorted in increasing order, the data items of the 2nd tree are sorted in decreasing order and so on. This involves $1 + \log\log N$ phases and requires $0(\log N)$ time. This will produce $N^2/2$ bitonic sequences each of length $4 \log N$. Each bitonic sequence comprises the data items in the leaves of two consecutive trees in row-major way starting from the 1st tree. These $N^2/2$ bitonic sequences each of length $4 \log N$ are merged successively to form a single bitonic sequence of length $2N^2 \log N$. This will require in all $2 \log N - 1$ merging phases. Ultimately in the $(2 \log N - 1)$th phase the single bitonic sequence of length $2N^2 \log N$ is merged into a sorted sequence of length $2N^2 \log N$. In the ith phase $(1 \le i \le 2 \log N - 1)$, $N^2/(2^i)$ bitonic sequences each of length $2^{i+1} \log N$ are merged to produce $N^2/(2^{i+1})$ bitonic sequences each of length $2^{i+2} \log N$.

In phase $i(1 \le i \le 2 \log N - 1)$, simultaneously for $N/(2^i)$ groups:

Trees $1, 2, \ldots, 2^i$ are merged forming an increasing sequence.

Trees $2^i + 1, \ldots, 2^{i+1}$ are merged forming a decreasing sequence.

Trees $N^2 - 2^i + 1, \ldots, N^2$ are merged forming a decreasing sequence.

Let us now discuss how to merge the data items of 2^i trees of a group comprising, say, Tree(k), Tree($k+1$), ..., Tree($k+2^i-1$) to produce a sorted sequence. This group is first subdivided into two subgroups of trees:

Tree(k), Tree($k+1$), ..., Tree($k+2^{i-1}-1$) comprises one group and Tree($k+2^{i-1}$), Tree($k+2^{i-1}+1$), ..., Tree($k+2^i-1$) comprises another group. Each group contains 2^i log N data items. Items in the corresponding positions of these two groups are compared and exchanged if they are found out of place using the cube connections in the base of CCT. Simultaneously 2^{i-1} comparisons can take place between two subgroups since there are 2^{i-1} trees in each subgroup. At a time only one element of each tree (M) of a subgroup can be compared with the corresponding element of tree ($M+2^i$ log N) of the other subgroup. Since there are 2 log N leaves in each tree, all comparisons between two subgroups can be completed in O(log N) time. Comparison between corresponding elements of two trees are done through the cube link connecting these two trees. Next, each subgroup is further subdivided into two subgroups, each containing 2^{i-1} log N data items and the corresponding elements between the two subgroups are compared and exchanged if they are found out of place. This is done simultaneously for all pairs of subgroups resulting from the same group. This will also require O(log N) time steps. Ultimately, we will reach a situation when each subgroup contains log N elements and both the subgroups belong to a tree of a CCT. We then sort these two subgroups using the sorting algorithm for a tree as given above and this requires O(log N) steps. Merging phase i requires $(i+1) \times \mathrm{O}(\log N) + \mathrm{O}(\log N)$ time. Hence the total time required for sorting is

$$\sum_{i=1}^{i=2\log N} \{(i+1) \times \mathrm{O}(\log N) + \mathrm{O}(\log N)\}$$

$$+\,\mathrm{O}(\log N) \quad \text{which is } O(\log^3 N).$$

5.2. Fourier transform

We assume that the number of inputs is $2N^2$ log N and that these data points are distributed among the leaves of CCT in column-major way, i.e. the first N^2 data items are distributed among the leftmost leaves of the N^2 pruned full binary trees one item per leaf, the next N^2 data items are similarly distributed among the second leftmost leaves of all the pruned full binary trees and so on.

For $2N^2$ log N data points, we need to simulate 2 log N + log log N + 1 ranks of N^2 log N butterfly operations in each rank. We assume that during computation, appropriate power of ($2N^2$ log N)th primitive root of unity is available at the PEs. First 2 log N ranks of butterfly operations are done by using the cube connections. For butterfly operations belonging to rank i, $1 \le i \le 2 \log N$, cube links connecting the ith leftmost leaves of the N^2 pruned full binary trees are utilized. In each time step, these links can do $N^2/2$ butterfly operations of rank i. But we need to do N^2 log N butterfly operations of rank i, so to do all the butterfly operations of rank i, we need to use these links 2 log N times. To do all the butterfly operations of rank i, we have to successively bring other columns of data to column i where the cube connections corresponding to rank i are available. This amounts to bringing the data from the leaves of pruned trees successively to the particular leaf of the tree where the cube connection corresponding to this rank is available and this has to be done in parallel for all the N^2 pruned trees. Using pipelining in each pruned tree, data from the leaves can be brought successively to a particular leaf of the tree for butterfly operation in time proportional to the number of leaves in each tree, i.e. in O(log N) time since there are 2 log N leaves in each tree. Assuming that the links are bidirectional, the column can be sent back to its original position after the butterfly operation in O(log N) time. Operations of rank i, $1 \le i \le 2 \log N$ can be done in O(log N) time. So the time required for 2 log N ranks of butterfly operations using cube connections is $2 \log N * O(\log N) = O(\log^2 N)$. The remaining $1 + \log \log N$ ranks of the butterfly operations are done by using the tree links of the CCT. For rank $2 \log N + 1$, all the $N^2 \log N$ operations can be done simultaneously utilizing $N^2 \log N$ subtrees each of height 1, logically constructed out of N^2 pruned full binary trees. Each pruned full binary tree provides $\log N$ disjoint binary trees of height 1 each. For rank $2 \log N + 2$, all the $N^2 \log N$ operations can not be done simultaneously because of conflict in using some tree links by two operations at the same time. So these operations are staggered over two time slots. In each time slot we use $(N^2 \log N)/2$ subtrees of height 2 each. Each pruned full binary tree provides $(\log N)/2$ disjoint binary trees of height 2 each. Thus, for rank $2 \log N + 1 + \log \log N$, we use N^2 binary trees each of height $\log \log N$ for $\log N$ times to complete all the butterfly operations of this rank. In general, to compute butterfly operations of rank $2 \log N + i$ where $1 \leq i \leq 1 + \log \log N$, we need to use disjoint trees of height i for 2^{i-1} times. The time for this step is proportional to $i2^{i-1}$. So the total time required for computation of butterfly operations of these $1 + \log \log N$ ranks may be written as proportional to the sum of the following series:

$$2^0 x 1 + 2^1 2 + 2^2 x 3 + \cdots + 2^{\log \log N} x (1 + \log \log N).$$

The sum of the above series is $(2 \log N) * \log \log N + 1$. Hence the total time required for fourier transform of $2N^2 \log N$ data points on a $N \times N$ CCT is $O(\log^2 N) + O(\log N \ \log \log N)$, which is $O(\log^2 N)$.

6. Conclusion

We have demonstrated that CCT is an efficient VLSI architecture and has polylogarithmic performance for ascend/descend type of computation by mapping the problems of sorting and Fourier transform. Though for these problems the performance of CCT is lower than those of hypercube by a logarithmic factor, CCT has the major advantage of having a maximum node degree of three compared to size dependent variable degree nodes in the hypercube. The degree-diameter product metric of CCT is low compared to that of a hypercube of comparable size. Having low node degrees in a network enables one to construct larger and larger sizes of network using the contemporary VLSI technology. When the number of nodes in a hypercube is very large, it would be impossible to implement the network in VLSI because of technological constraints. Moreover, the data routing is simpler in CCT than in hypercube because the number of directions in which data may be sent from a node is two in CCT (through cube links or tree links) whereas it is a logarithmic function of number of nodes in the hypercube. We have also shown how to make it fault-tolerant by suitable use of spare PEs and links. Hence CCT is a potential parallel processing architecture for VLSI implementation.

Acknowledgment

The authors would like to thank the anonymous referees for their thoughtful comments on an earlier version of this paper which helped enhancement of the quality of this presentation.

References

[1] K. Hwang and F.A. Briggs, *Computer Architecture and Parallel Processing* (McGraw-Hill, 1984).

[2] J.D. Ullman, *Computational Aspects of VLSI* (Computer Science Press, 1984).

[3] C.D. Thompson, Fourier Transform in VLSI, *IEEE Trans. Comput.* (11) (Nov. 1983) 1047–1057.

[4] D. Nath, S.N. Maheshwari and P.C.P. Bhatt, Efficient VLSI networks for parallel processing based on orthogonal trees, *IEEE Trans. Comput.* C-32 (6) (June 1983) 569–581.

[5] A.V. Aho, J.E. Hopcroft and J.D. Ullman, *The Design and Analysis of Computer Algorithms* (Addison-Wesley, 1974).

[6] C.D. Thompson, A complexity theory for VLSI, Ph.D Thesis, Department of Computer Science, University of California, Berkeley, 1980.

[7] G. Bilardi and F.P. Preparata, An architecture for bitonic sorting with optimal VLSI performance, *IEEE Trans. Comput.* C-33 (7) (July 1984) 646–651.

[8] J.R. Goodman and C.H. Sequin, Hypertree: A multiprocessor interconnection topology, *IEEE Trans. Comput.* C-30 (12) (Dec. 1981) 923–933.

[9] C.E. Leiserson, Fat-Trees: Universal networks for hardware efficient supercomputing, *IEEE Trans. Comput.* C-34 (10) (Oct. 1985) 892–901.

[10] E. Horowitz and A. Zorat, The binary tree as an interconnection network: applications to multiprocessor systems and VLSI, *IEEE Trans. Comput.* C-30 (4) (April 1981) 247–253.

[11] A.M. Despain and D.A. Patterson, X-Tree: A tree structured multiprocessor computer architecture, in *Proc. Fifth Computer Architecture Symp.* (April, 1978) 144–151.

[12] C.D. Thompson, Area-time complexity for VLSI, Technical Report, Division of Computer Science, University of California, Berkeley, Jan. 1984.

[13] C.S. Raghavendra et al., Fault tolerance in binary tree architectures, *IEEE Trans. Comput.* C-33 (6) (June 1984) 568–572.

[14] M.C. Pease, The indirect binary N-Cube microprocessor array, *IEEE Trans. Comput.* C-26 (5) (May 1977) 458–473.

[15] F.P. Preparata and J.E. Vuillemin, The Cube-connected cycles: A versatile network for parallel computation, *Commun. ACM* (May 1981) 300–309.

[16] C.H. Sequin and R.M. Fujimoto, X-tree and Y-components, in: *VLSI Architectures*, B. Randell and P.C. Treleaven, eds. (Prentice-Hall, 1983) 299–326.

[17] G. Bongiovanni, Two VLSI structures for the Discrete Fourier Transform, *IEEE Trans. Comput.* C-32 (8) (Aug. 1983) 750–753.

[18] R. Duncan, A survey of parallel computer architectures, *IEEE Computer* (Feb. 1990) 5–16.

[19] A.S. Youssef and B. Narahari, The Banyan-hypercube networks, *IEEE Trans. Parallel Distributed Syst.* 1 (2) (April 1990) 160–169.

[20] K. Efe, The crossed cube architecture for parallel computation, *IEEE Trans. Parallel Distributed Syst.* 3, (5) (Sep. 1992) 513–524.

[21] J. Wu and M.M. Larrondo-Petrie, Cube-connected-cubes network, *Microprocessing and Microprogramming* 33 (1991/92) 299–310.

[22] S.Q. Zheng, Compressed tree machines, *IEEE Trans. Comput.* 43 (2) (Feb. 1994) 222–225.

[23] S. Abraham and K. Padmanabhan, An analysis of the twisted cube topology, in *Proc.* 1989 *Int. Conf. on Parallel Processing*, Vol I, 116–120.

[24] S. Latifi and A. El-Amawy, On folded hypercubes, in *Proc.* 1989 *Int. Conf. on Parallel Processing*, Vol. I, 180–185.

[25] H.L. Chen and N.F. Tzeng, Enhanced incomplete hypercubes, in *Proc.* 1989 *Int. Conf. on Parallel Processing*, Vol I, 270–277.

[26] L.N. Bhuyan and D.P. Agrawal, Generalized hypercube and hyperbus structures for a computer network, *IEEE Trans. Comput.* C-33 (4) (April 1984) 323–333.

[27] D.T. Barnard and D.B. Sillicorn, Pipelining tree-structured algorithms on SIMD architectures, *Information Processing Letters* 35 (1990) 79–84.

[28] E. Schwabe, On the computational equivalence of hypercube derived networks, in *Proc. 2nd Annual ACM Symp. on Parallel Algorithms and Architectures* (July 1990) 388–397.

[29] N.S. Woo and A. Agrawala, A symmetric tree structure interconnection network and its message traffic, *IEEE Trans. Comput.* C-34 (8) (Aug. 1985) 765–769.

[30] C.S. Yang, S.Y. Wu and K.C. Huang, A reconfigurable modular fault tolerant generalized Boolean n-Cube network, *Microprocessing and Microprogramming* 32 (1991) 589–592.

[31] S. Srinivas and N.N. Biswas, Design and analysis of a generalized architecture for reconfigurable m-ary tree structures, *IEEE Trans. Comput.* 41 (11) (Nov. 1992) 1465–1478.

Swapan K. Basu received the M.Sc degree in Physics from the University of Calcutta, India in 1977 and the M. Tech. degree in Computer Science from the Indian Statistical Institute, Calcutta in 1981. He has submitted his doctoral thesis to the Jadavpur University, Calcutta in 1994. Presently, he is teaching Computer Science at the Banaras Hindu University, Varanasi, India and working on a university level text book on parallel and distributed processing under a support from the University Grants Commission, India. His research interest is Parallel Processing, architectures and algorithms. He is a member of the Computer Society of India.

J. Dattagupta received the BE. Tel. E. and P.G. Dip. in Computer Science from Jadavpur University. She received her M. Phil. degree from Brunel University, UK. She had also received her PhD degree from Calcutta University, India. She was a Visiting Scientist at G.M.D., Bonn, F.R.G., during 1983–84. She is currently Associate Professor of Indian Statistical Institute, Calcutta. Her research interests are fault diagnosis, interconnection networks, parallel processing and VLSI layout.

R. Dattagupta received the B.E. & Tel. E, M.E & Tel. E and PhD degrees from Jadavpur University. He had also received a PhD degree from Brunel University, UK, where he was a Commonwealth Scholar during the period 1974 to 1977. He was a Visiting Scientist at GMD, Bonn, FRG during 1983–1984.

He is currently Professor and Head of the Computer Science and Engineering Department, Jadavpur University, Calcutta, India. He is a Fellow of the Institute of Engineers (India), Senior Member of IEEE, Computer Society (IEEE) and Computer Society of India.

His research interests include VLSI, parallel processing and microprocessor-based system design.