TECHNION - Israel Institute of Technology
Computer Science Department

EMBEDDING TREE STRUCTURES IN FAULT- TOLERANT
VLSI HEXAGONAL ARRAY

by

D. Gordon*, I. Koren**, and G.M. Silberman***

Technical Report #265
February 1983

* Dept. of Computer Studies, University of Haifa, Haifa
** Dept. of Electrical Engineering, Technion, Haifa
*** Dept. of Computer Science, Technion, Haifa,
EMBEDDING TREE STRUCTURES IN FAULT-TOLERANT VLSI HEXAGONAL ARRAYS

Dan Gordon
Dept. of Computer Studies
University of Haifa
Haifa 31999, Israel

Israel Koren
Gabriel H. Silberman
Dept. of Electrical Engineering and Computer Science
Technion - Israel Institute of Technology
Haifa 32000, Israel

ABSTRACT

Tree structures have been proposed for special-purpose and general-purpose multiprocessors due to their desirable property of logarithmic path from the root to any leaf element. Since only local communication among processors is needed, in tree structures, they are well suited for the VLSI technology. Such an implementation requires an area-economical mapping of a tree on a plane. Novel mapping schemes for trees onto hexagonal arrays (or grids) and appropriate algorithms are proposed in this paper and shown to be superior over known mappings on square arrays (or grids). Finally, fault-tolerance in hexagonal arrays is introduced in a manner suitable for trees and other configurations.

KEYWORDS: Tree structures, hexagonal multiprocessor array, VLSI, mapping scheme, distributed configuration algorithm, fault-tolerance.
1. INTRODUCTION

Hierarchical tree-structured multiprocessor systems have received recently a great amount of attention [1-5]. Two of the most desirable features of a tree structure are the ability to access any processor in a tree of n processors in at most $\log_2 n$ time and its pipelining capability (e.g. [1]). Tree structures have been shown to be well suited for general-purpose multiprocessors [5,6], as well as for special-purpose devices such as data-base machines [1,2] and other specialized applications [3].

When VLSI implementations of multiprocessor systems are considered, tree configurations look very attractive due to the simple and regular interconnections which are needed among the processing elements. In VLSI technology, computation is cheap but communication is costly [1-4]. Consequently, by adopting a tree configuration in which every processing element (PE) communicates only with its immediate neighbors, the design costs are substantially reduced. To achieve a space-economical implementation of a tree machine on a VLSI chip, an appropriate placement strategy to map the tree structure on a plane is required. One such strategy which uses an area that is linear in the number of PE's has been widely used [1-4], and an appropriate construction algorithm has been devised [7]. It maps a binary tree on a square grid as illustrated in Figure 1.

A different approach was introduced in [8], the binary tree being mapped instead on a square array of PE's. The underlying idea is that by adding the possibility of restructuring the array as a tree, we increase the number of applications utilizing the processor array, thus making the VLSI chip more appealing to the semiconductor industry. Sales volume is the key to success in this industry.

---

*When a tree is embedded in a grid, some PE's function as links between active tree nodes. The latter are shown enlarged.*
and one way to increase it, is to make standard pieces of silicon which can be programmed by on-chip software to perform different functions for different customers.

Various mesh-connected processor arrays have been proposed and numerous appropriate computational algorithms have been devised [3,9,10]. One of the most flexible schemes is the hexagonally mesh-connected array, since each PE in it has direct communication with its six immediate neighbors, thus allowing the array to be configured in several ways. Still, hexagonal arrays as other VLSI configurations, are sensitive to failures, and the failing of a single PE may prevent in some applications the use of the entire array. Consequently, fault-tolerance is a desirable property of processor arrays allowing the array to recover from errors by reconfiguring itself after a faulty processor (or faulty connection) is detected.

The paper is divided into six sections. In section 2 we concentrate on various tree configurations which can be mapped on an hexagonal array. In section 3 we derive area and propagation time expressions, and show the superiority of hexagonal arrays over square arrays. Next, a distributed configuration algorithm for a tree on the hexagonal array is presented in section 4. Section 5 deals with the subject of fault-tolerance in hexagonal arrays.

2. TREE CONFIGURATIONS

In the usual H-type embedding of a binary tree in a square grid, there is a basic unit consisting of a level 3 binary tree which is replicated to form the full binary tree - see Figure 1. The basic unit can be chosen differently, as in the Type 2 tree of [8]. On an hexagonal grid it is possible to choose a basic unit (which we call the "basic tile") in several different ways, and to replicate it in various ways.
The mappings which we have found to be simplest to replicate on an hexagonal grid have a basic tile consisting of a rectangle (Figure 2) or a parallelogram (Figures 3-4). The basic tile, containing a full binary tree (of levels 5 and 6 in Figures 2-4), is replicated in one of three different patterns: (1) The replication of the rectangular tile is called the rectangular pattern and the method of replication is obvious from Figure 2. The resulting overall pattern is always rectangular. (2) The parallel replication of a parallelogram basic tile is shown in Figure 3 and it is probably the simplest scheme as all communication lines are straight. In this scheme, the resulting overall pattern is a large parallelogram, which may be unsuitable if the chip is not of the same shape. This possible problem is remedied in (3) - the zigzag replication pattern outlined in Figure 5. Here, some of the communication lines are straight and some alternate in direction. This scheme is somewhat more complicated than the parallel scheme, but the resulting overall shape is rectangular.

In Figures 2 and 3 only one basic tile is shown in detail and others are only outlined. Note however that some of the outlined tiles are mirror images of the detailed one. Some of the advantages of using the hexagonal grid are immediately evident from the figures: For example, Figure 2 shows that a level 6 tree need have no links to connect the nodes (compare with Figure 1). Also, Figures 2-4 show that in the basic tile almost all PE's are active nodes. In Section 3 we show that these advantages over the square grid are maintained even when the tree level tends to infinity.

The basic tiles of Figures 2-4 try to minimize the area and/or the distance from the root of the tile to the leaves, but it is fairly easy to construct tiles with other objectives, for example minimal inorder or postorder distance between nodes - see Figure 6.
3. TIME AND AREA ANALYSIS

In this section we present formulas for the area and propagation delay of the hexagonal schemes, and compare them with the square grid scheme. In counting the number of PE's used by a scheme we include all the PE's in the smallest rectangle or parallelogram containing all the basic tiles.

**Area Comparison**

Assume we have a parallelogram basic tile with A and B PE's on the sides, and containing a tree of level c. Assume further that the path to the root of the tile is through the side containing A PE's. Denote by $P_k$ and $Z_k$ the number of PE's used by the parallel and zigzag schemes, respectively, to construct a level k tree. Setting $k = c + 1$, we have:

$$P_{c+1} = [(A+1)^2 + \left\lfloor \frac{B-1}{2} \right\rfloor - 1] \left\lfloor \frac{A+1}{2} \right\rfloor - 1]$$ \qquad (1)$$

$$Z_{c+1} = [(A+1)^2 + \left\lfloor \frac{B-1}{2} \right\rfloor - 1] \left\lfloor \frac{B+1}{2} \right\rfloor - 1]$$ \qquad (2)

The derivation of (1) is straightforward. The derivation of (2) can be followed by inserting the parallelogram of Figures 3 or 4 into Figure 5.

$Z_{c+1} > P_{c+1}$ but $\lim_{k \to \infty} \frac{Z_{c+1}}{P_{c+1}} = 1$, i.e. only a vanishing percentage of PE's is wasted by using the zigzag instead of the parallel scheme.

A full binary tree of level k has $2^k - 1$ nodes, so the ratio of PE's used as nodes over the total number of used PE's is $(2^k - 1) / P_k$ and $(2^k - 1) / Z_k$, for the parallel and zigzag schemes respectively. It is easy to see that both ratios tend to $2^k / (A+1)(B+1)$ as $k \to \infty$. For $A=7$, $B=5$ and $c=5$ this ratio is 0.667, and for $A=9$, $B=8$ and $c=6$ it becomes 0.711 (compared to 0.5 for the square grid [7]).

If we wish to compare the actual areas used by hexagonal and square grids, we must take into account the fact that an hexagonal PE of the same
computational power as a square PE probably has a larger area, because it has to monitor more connections. In this comparison we consider the area between two adjacent PE's as part of the area taken up by the PE's, so that the total area is just the number of PE's multiplied by the area of a single one.

How much larger can an hexagonal PE be, in order that the total area taken up by a tree of level $k$ be no larger than that taken up by a tree of the same level in a square layout? We denote by $S_k$ the number of PE's taken by a tree of level $k$ in a square grid. Its expression is [6]:

$$S_k = \begin{cases} \frac{k+1}{2} \cdot \frac{k+1}{2}, & \text{if } k \text{ is odd} \\ \frac{k+2}{2} \cdot \frac{k}{2}, & \text{if } k \text{ is even} \end{cases}$$

(3)

Let $1+x$ be the factor by which an hexagonal PE is larger than a square PE. The hexagonal layout takes less total area if $P_k(1+x) < S_k$ or $1+x < S_k/P_k$ (replace $P_k$ with $Z_k$ for the zigzag scheme). Both $S_k/P_k$ and $S_k/Z_k$ tend to $2k+1/(A+1)(B+1)$ as $k \to \infty$. For the 8x9 parallelogram this limit is 1.422 and $S_k/P_k > 1.422$. This means that the hexagonal PE can be larger than the square PE by as much as 42% and still save area. The same is true if the rectangular pattern of Figure 2 is used, because it takes up even less area than the parallel layout based on the 8x9 parallelogram.

The exact ratio of areas between square and hexagonal PE's depends on the computational vs. communication requirements of the PE's, and these depend on what applications the designer of the VLSI chip has in mind. The figure of 42% allows the designer to decide, for a given application, which of the two schemes—square or hexagonal, would use less area.

**Propagation Delay Comparisons**

We count the number of connections a signal has to cross in order to reach a leaf from the root of a tree. Assuming a parallelogram basic tile, the distance
from level \(c+i\) to level \(c+i-1\) is:

\[
\Delta_{c+i} = \begin{cases} 
1, & \text{if } i=1, \\
(A+1)2^{\frac{t-1}{2}}, & \text{if } i \geq 2 \text{ and even}, \\
(B+1)2^{\frac{t-9}{2}}, & \text{if } i \geq 3 \text{ and odd.}
\end{cases}
\]  

From this, the total propagation delay is

\[
D_{c+i} = \begin{cases} 
D_c + 1, & \text{if } i=1; \\
D_c + 1 + \frac{A+1}{2} + (A+1)(2^{\frac{k-2}{2}} - 1) + (B+1)(2^{\frac{t-1}{2}} - 1), & \text{if } i \geq 2.
\end{cases}
\]  

where \(D_c\) is the internal propagation delay of a parallelogram tile. \(\Delta_k\) and \(D_k\) are the same for both the parallel and zigzag schemes.

The delay for the rectangular pattern of Figure 2 turns out to be exactly equal to \(D_{c+i}\) for \(c=6, A=9, B=8\) and \(D_c=9\) (i.e., same as for the parallelogram tile of Figure 4(a)).

The propagation delay from the root to the leaves in the square grid is

\[
R_k = \begin{cases} 
3 \cdot 2^{\frac{k+1}{2}} - 2, & \text{if } k \text{ is even} \\
2 \cdot 2^{\frac{k+1}{2}} - 2, & \text{if } k \text{ is odd.}
\end{cases}
\]  

This expression is easily derived from [8].

Assuming an 8x9 parallelogram as in Figure 4(a) (i.e., \(c=6\) and \(D_c=9\)), we find that for "practical" values of \(k\), say \(6 \leq k \leq 10\), (i.e., a tree of up to 1024 nodes) \(R_k/D_k\) is between 1.43 and 2. The advantage of the hexagonal scheme is maintained for all values of \(k\), and the ratio \(R_k/D_k\) approaches 1.26 and 1.4 for even and odd values of \(k\), respectively, as \(k \to \infty\).

Denoting by \((1+x)\) the factor by which the signal delay through an hexagonal PE is larger than that of a square PE, we find that even if \(x > 0\) (which is not necessarily true), the total time delay in the hexagonal scheme can be smaller
than that of the square scheme. The "break-even" value of $x$ depends on $k$, the level of the tree. For $6 \leq k \leq 10$, $x$ can be as large as 43%, and the asymptotic ($k \to \infty$) break-even values of $x$ are 26% and 14% for even and odd values of $k$, respectively.

4. DISTRIBUTED PLACEMENT ALGORITHM

To take advantage of the fault-tolerance capacity of the hexagonal array (introduced in the next section) we wish to be able to dynamically place a tree on an hexagonal array. Thus, a faulty PE or connection causing the array to reconfigure, will be followed by reexecution of the placement algorithm, resulting in a possibly smaller size tree.

The placement algorithm may be implemented either in a centralized manner, i.e., totally from the outside (by some host), or in a distributed fashion, i.e., mostly internally performed (by the PE's). The first approach relies on a "configuration string" externally generated, containing setup instructions that are distributed to all relevant PE's.

In the second approach, the process is initiated externally, but the exact configuration is determined internally by the PE's. This requires that each processor contain (or obtain through broadcasting from the outside) the entire configuring information. This requirement might increase the size of the local memory, which in turn may increase the physical size of the basic PE, thus limiting the number of such elements for a given chip area. On the other hand, feeding-in a configuration string from the outside minimizes the amount of information required at each processor, at the expense of increasing the time needed to complete the configuration process.

Another important advantage of a distributed placement algorithm is that it is independent of the exact size of the available hexagonal array. As will be
shown in the next section, the testing and reconfiguration steps change the size of the hexagonal array. The distributed placement approach is thus more appealing, since it does not require any changes in the placement algorithm.

The solution we have chosen to present is a compromise between the two approaches, i.e., only a limited size configuration string (describing the basic tile) is generated externally and then distributed. The rest of the configuration is internally controlled by an appropriate algorithm residing locally at each PE. This solution benefits from the regularity exhibited at those levels in the resulting tree which are outside the basic tile (see for example Figure 3). The presented algorithm is also independent of the exact size of the hexagonal array as long as it is larger than a single basic tile.

As a result, each PE has two possible modes of operation during the configuration process, one for the tree levels within the basic tile, in which the externally-supplied configuration string is used, and one for the levels outside the basic tile, where the internal algorithm at each PE is applied. For convenience, we include both modes of operation in a single configuration algorithm, as described below.

The algorithm can be outlined as follows: A message is sent from the outside towards the PE which will function as the tree root. This PE determines the directions of its two subtrees and sends appropriate messages in these directions. This process is repeated at each tree node, until the leaves of the tree are reached. While the algorithm is being executed, the PE's communicate using messages of the form:

\[ M (LV, MR, CN, ST) \]

where \( LV \) is the level number within the tree, \( MR \) is a "mirror flag" indicating (if "true") that a mirror image of the basic tile is to be used, \( CN \) is the number of
PE's which function as links before the next tree node is reached, and ST is the configuration string for the basic tile.

A PE receiving such a message from its d-neighbor (i.e., its immediate neighbor in direction d [8]) executes the appropriate part of the algorithm (according to the values of LV, MR and CN) and transmits similar message(s) to its neighbor(s). Each PE may receive or transmit a message in any one of six possible directions, numbered for convenience 0 through 5, as shown in Figure 7.

The directions of the outgoing messages are determined relative to d - the direction of the incoming message (0 ≤ d ≤ 5). Thus, (d+1) mod 6 (or (d-5) mod 6) is the direction next to d when moving clockwise (Figure 7). Similarly, (d+3) mod 6 is the opposite direction. Since the calculations related to directions use modulo 6 arithmetic throughout, we omit in the following the reference to the modulus and write "(d+i)" instead of "(d+i) mod 6".

A distributed algorithm for a zigzag replication of the parallelogram tile shown in Figure 3 is depicted in Figure 8. The first part of the algorithm deals with tree nodes outside the basic tile, i.e., $LV > c$, where c is the number of levels in the subtree embedded in the basic tile. In this part, the number of PE's behaving as links between the present node and its sons, is first determined using Equation (3). Then, the directions of outgoing messages and their appropriate mirror flags are calculated while configuring in conformance with Figure 3, and the messages are sent.

Next, the case of a tree node inside the basic tile ($LV \leq c$) is dealt with. To determine the directions of the outgoing messages in this case, we use a configuration string - ST. Each element of the string indicates the outgoing directions, relative to the direction of the incoming message, for a specific PE in the basic tile.
In order to encode this information into the string elements, we use a five-bit binary code \( X = (x_4x_3x_2x_1x_0) \), where \( x_j = 1 \) indicates that an outgoing message should be transmitted in direction \( d + (j + 1) \) (i.e., the weights of the digits \( x_4, x_3, x_2, x_1, x_0 \) are 5, 4, 3, 2, 1, respectively).

For example, the code word \( 9 = 01001 \) means that two outgoing messages are to be transmitted, one in direction \( d + 4 \) and the other in direction \( d + 1 \). Mirroring the basic tile is accomplished by assigning the negative weights \(-5, -4, -3, -2, -1\) to the digits \( x_4, x_3, x_2, x_1, x_0 \) respectively. Thus, if mirroring is needed (according to the value of MR), the code \( 9 = 01001 \) means that the outgoing directions are \( d - 4 \) and \( d - 1 \), which are the same as \( d + 2 \) and \( d + 5 \) in our residue arithmetic system.

The first element \( X \) of \( ST \) is dropped after being used to determine the outgoing direction(s), and the rest of the string is transmitted to the next PE if a single bit in \( X \) is set (i.e., the present PE performs as a link), or to the two sons if two bits in \( X \) are set (i.e., the PE is a tree node).

The order in which the elements in the string \( ST \) are organized is determined by the "even-odd" numbering scheme [7]. Consequently, if two bits \( x_j \) and \( x_k \) \((j > k)\) in \( X \) are set, all odd-positioned (even-positioned) elements in the rest of the string \((1ST \) in APL notation) are transmitted to the neighbor specified by \( x_j \) \( (x_k) \). For example, the configuration string for the basic tile shown in Figure 4 is

\[
ST = 4, 9, 6, 20, 18, 6, 6, 10, 9, 6, 12, 17, 24, 6, 9, 9
\]

The first element in \( ST \) (i.e., 4) corresponds to the first and only link element in the basic tile. When the next processor inside the tile is reached, the \( 9 = 01001 \) is dropped and two sub-strings are formed. If the mirror flag is \( MR = \text{false} \) (\( MR = \text{true} \)) then the odd sub-string \( 6, 18, 6, 9, 12, 24, 9 \) is transmitted in direction \( d + 4 \) \( (d - 4) \), and the even sub-string \( 20, 6, 10, 6, 17, 6, 9 \) is transmitted in
direction $d+1$ ($d-1$).

Finally, the last part of the algorithm deals with PE's acting as link elements between tree nodes, outside the basic tile. If $L^V - c$ is even for a certain node (at level $L^V$), then straight lines connect that node to its sons (see Figure 3). Otherwise, zigzagging is needed, and the direction is changed every $B+1$ linking PE's, where $B$ is the number of processors on the smaller side of the basic parallelogram.

Note that the algorithm in Figure 3 is the same for any parallelogram basic tile (i.e., values for $A$, $B$ and $c$). Similar algorithms can be devised for parallel and rectangular replications of basic tiles.

5. FAULT TOLERANCE

One of the major drawbacks of tree structures is their sensitivity to malfunctions. The path between any two nodes in a tree is unique, hence any fault along the path will cause certain nodes (within a subtree) to become unreachable. A solution to this problem has been suggested, adding extra connections to form the half ring or full ring schemes [5,6]. The difficulty with these schemes however, is their non-planarity, making the task of embedding them in a single chip substantially harder.

Our approach is to introduce fault-tolerance into the hexagonal array which embeds the tree structure. This is done as follows: When some PE's or connections become faulty, the other PE's can reconfigure into an hexagonal array (of smaller size). The importance of such a mechanism is that the failure of some of the PE's does not preclude the usage of the remaining PE array for trees (and other structures) which do not require more PE's than are currently usable.
Fault tolerance is achieved in two basic stages, the testing stage and the reconfiguration stage. In the testing stage the PE's test their neighbors and themselves, in order to identify faulty PE's or connections. In the reconfiguration stage, the PE's with neighboring faults turn into connecting elements (CE's) and initiate messages which turn some other PE's into CE's. These CE's cease to perform processing per se and behave like connectors (not to be confused with "links"! - see last Section).

Each remaining PE is not aware of the presence of the CE's and continues to communicate with six neighbors as it did before the reconfiguration occurred, using the same links as it did before (i.e. the neighbor in a given direction is still accessed in that direction). It is possible though, that some of its neighbors are not physically the same as before, and the PE reaches them through some CE's. This concept first appeared in [8], where it was applied to rectangular arrays.

A Distributed Testing Procedure

We propose a distributed testing procedure in which every PE tests all its neighbors [8]. In this way, faulty PE's and faulty connections between PE's are detected by the adjacent PE's.

The procedure first partitions all the PE's into 7 disjoint testing groups, $T_0, T_1,..., T_6$. After this partitioning, there are seven phases of testing, where at phase $i$ ($0 \leq i \leq 6$), the members of $T_i$ test all their neighbors.

The partition is such that (1) every PE is surrounded by PE's of other groups, and (2) no PE has two neighbors belonging to the same group. These two properties guarantee that for every $i$, no two members of $T_i$ will test each other, or try simultaneously to test a third PE. It can easily be seen that 7 groups are both necessary and sufficient for a partition with the above properties.
An example of one such partitioning is given in Figure 9. This particular method was chosen because it leads to a very concise algorithm for assigning group numbers.

The testing procedure for this particular partitioning is initiated externally by assigning the group number 0 to the left-upper corner PE in the whole array. Every other PE, after being assigned a group number $i$, assigns group numbers to its neighbors in directions 1, 2 and 3 as follows:

- Assign $(i+2) \mod 7$ to PE in direction 1.
- Assign $(i+3) \mod 7$ to PE in direction 2.
- Assign $(i+1) \mod 7$ to PE in direction 3.

Notice that if each element has been assigned matrix indices $(i,j)$, then their group number is equivalent to $(i+2j) \mod 7$.

After waiting a suitable period of time, phase 0 of the testing is externally triggered, and broadcast to all PE's. All those PE's in $T_0$ then start testing all of their neighbors. The transition from phase to phase is always externally controlled, after a suitable period of time has elapsed.

The grouping stage is necessary every time before the testing stage due to possible reconfigurations resulting from earlier testings.

Reconfiguration

In the reconfiguration stage, every PE is assumed to know the status (faulty/not-faulty) of its six connecting links or neighboring PE's. There is no difference in the action taken by the PE if the actual fault is in the neighboring PE proper, or in the link leading to it.
The basic unit for reconfiguration purposes is the CE (connecting element). This element is a PE which simply acts as a connector between pairs of neighboring PE's. Out of all possible configurations of a connector, only the three shown in Figure 10 are actually used in the following algorithm. Any one of these is completely defined by the link which connects opposite directions.

For reasons which will become clear later, we find it convenient to label the three types of CE's by adding (modulo 6) the numbers of the opposite directions which characterize them. Thus CE's of type 1, 3 and 5 link the opposite pairs 2-5, 0-3 and 1-4, respectively. Note that if we disregard absolute directions, there is only one type of CE.

**Single Faults**

A single fault is either a faulty connection or a faulty PE. We begin with the description of the action taken by a PE which discovers that it has a single faulty connection (or faulty neighbor). In the following figures, a CE is represented by a stroke through the processor, the slope of this stroke indicating the link which characterizes the type of the CE.

The PE initiates the transmission of two messages, P. (for Primary) and S. (for Secondary) Assume that the fault is in direction \(d\) \((0 \leq d \leq 5)\) from the PE. Message P. is always sent in the opposite direction to \(d\) \(((d+3) \mod 6)\), and message S. is sent in direction \(s(d) = (d - (-1)^d) \mod 6\). Thus, in the case of a faulty connection between two PE's, two P. messages will be initiated in opposite directions and two S. messages will be received by a neighboring PE.

After transmitting P. and S., the PE becomes a CE of type \((d+s(d)) \mod 6\). When a PE receives P. from direction \(d\), it retransmits P. in the opposite direction and becomes a CE of type \((d+s(d)) \mod 6\). The P. message thus travels in a (virtual) straight line from the fault, turning all PE's in its path to CE's of identi-
The action taken by a PE on receiving an \textit{S} message is different. The PE acts only after it has received two \textit{S} messages from two directions which differ by 1 (modulo 6), i.e., two \textit{S} messages coming from two PE's which are adjacent to each other. If and when this happens, exactly one of these two directions will be odd (i.e., 1, 3 or 5). The PE then behaves as though it had received a \textit{P} message from this odd numbered direction.

An example is given in Figure 11. The link between processors A and B is assumed faulty. A and B initiate \textit{P} messages in opposite directions and send \textit{S} messages to processor C which in turn sends \textit{P} in direction 0. This results in three "rays" of \textit{CE} emanating from the area of the fault. Consider now the path of a connection between two PE's around the newly formed \textit{CE}'. It either goes straight through (as between D and E), or makes two "turns" through two \textit{CE}'s of the same type, and comes out in the same direction it started (as between F and G). This fact is important because a message sent \textit{out} by a PE in direction \textit{d} has to be received by its (logical) neighbor as if coming \textit{from} the opposite direction.

In Figure 12 we see an example of a single faulty PE. All its six neighbors initiate \textit{P} messages away from the fault and six rays of \textit{CE}'s are thus formed. It can be checked that each \textit{CE} at the beginning of a ray receives a single \textit{S} message, and therefore no action is taken as a result of these messages.

\section*{Multiple Faults}

When a PE detects more than one fault, it transmits \textit{P} messages to all its non-faulty neighbors. If the faults are only in the connections, this has the same effect as if all the neighboring PE's had detected that particular PE as being faulty. When one (or more) of the neighboring PE's is faulty, we get a situation similar to the one depicted in Figure 13.
are faulty). In such a situation it might happen that a PE receives first an S-message and then either a P-or another S-from a direction not matching (adjacent, as explained above) to the first S-(i.e. the two S-messages do not cause the PE to become a CE).

In fact, it is possible for a PE to receive as many as three S-messages before receiving either a P-or a matching S. This problem is solved by making the PE remember the directions from which S-messages were received. If, as a consequence of a P-or a complementary S-message, the PE turns into a CE, it transmits all non-matched S-messages in the direction it would have sent them if it had been a CE before it received them.

The effect obtained is the same as altering the time sequence of the messages. In the example of Figure 13, processor E receives an S-message from C and D, and a P-message from B (which sees A as faulty).

Since the whole process is asynchronous, E may receive the messages in any of the following sequences: (1) P, S, S; (2) S, S, P; (3) S, P, S. The first two cases pose no problem since E turns into a CE at a certain stage and will then redirect the other messages according to its type. Our solution to the problem in case (3) is simply to transform it into case (1). The resulting configuration of CE's is shown in Figure 13.

A program simulating the effect of single and multiple faults has been written. All the cases examined show that the remaining PE's form and hexagonally connected array (of a smaller size). Figures 11-13 are sample outputs from the simulating program. Notice that in Figure 11 the resulting configuration contains a 9x9 parallelogram array, while in Figure 12 with a faulty PE, the resulting configuration contains an 8x8 parallelogram only. These and other examples lead us to the following conjecture: Assuming an initial nxn parallelogram (or other shapes), if p connections and q PE's become faulty (simultaneously or
sequentially), the resulting configuration will contain a parallelogram of size \((n-p-2q) \times (n-p-2q)\) at least.

The same simulation program was used to derive some statistics concerning the proportion of cells employed as CE's due to faulty connections or faulty PE's. For example, in a 20x20 hexagonal array an average of 6% and 12% cells turned into CE's for a faulty connection and a faulty PE, respectively. It is easily seen that the maximum number of CE's due to a single faulty connection in an \(n\times n\) array is \(2n-1\), so the percentage of CE's vanishes as \(n \to \infty\). The same is true for a faulty processor.

6. CONCLUSIONS

Mapping schemes for trees on hexagonal arrays have been introduced in this paper. They were shown to be superior to the known mapping of a tree on a square array (or grid), when area and propagation time are considered. Next, a distributed algorithm for placing a tree on a given hexagonal array was presented. Finally, a fault-tolerance strategy for hexagonal arrays was introduced.

Appropriate mapping schemes and placement algorithms for other multiprocessor structures are needed in order to increase the number of applications utilizing the hexagonal array implemented in VLSI.
7. REFERENCES


Fig. 1: A six level tree on a square grid.

Fig. 2: Rectangular replication of a six level subtree embedded in a rectangular tile.
Fig. 3: Parallel replication of a five level parallelogram tile.

Fig. 4: Six level subtrees embedded in parallelogram tiles.
Fig. 5: Zigzag replication of a parallelogram tile.
Fig. 6: Five level tiles minimizing (a) inorder, (b) postorder.

Fig. 7: Absolute and relative to d numbering of directions.
A message $M(LV, MR, CN, ST)$ is received from $d$-neighbor

- **if** $CN = 0$ **then**
  - Entering a node in the tree

- **if** $LV > c$ **then**
  - A node outside the basic tile
  - $CN' := LV$
  - $i := LV - c$

- **if** $MR \in (i \leq 2)$ **then**
  - Mirrored tile image, except for first
  - $2$ levels outside basic tile, where
  - Outgoing directions are reversed

- **if** $i$ odd **then**
  - transmit $M(LV-1, MR, CN', ST)$ to $(d+1)$-neighbor
  - transmit $M(LV-1, MR, CN', ST)$ to $(d-1)$-neighbor

  **else**
  - transmit $M(LV-1, MR, CN', ST)$ to $(d-1)$-neighbor
  - transmit $M(LV-1, MR, CN', ST)$ to $(d+2)$-neighbor

- **endif**

- **else**
  - Straight tile image, except for first
  - $2$ levels outside tile (see above)

- **if** $i$ odd **then**
  - transmit $M(LV-1, MR, CN', ST)$ to $(d+2)$-neighbor
  - transmit $M(LV-1, MR, CN', ST)$ to $(d-2)$-neighbor

  **else**
  - transmit $M(LV-1, MR, CN', ST)$ to $(d-2)$-neighbor
  - transmit $M(LV-1, MR, CN', ST)$ to $(d+1)$-neighbor

- **endif**

- **else**
  - $LV = c \Rightarrow$ a node inside basic tile
  - $X :=$ first element in $ST$
  - $X = x_0 x_1 x_2 x_3$

- **if** MR then
  - $1$ **endif**

- **if** a single bit $x_i$ in $X$ is set **then**
  - **Drop first element in ST ($qST$ in APL)**

- transmit $M(LV, MR, 0, ST)$ to $d+(i+1)$ $INV$

- **else**
  - Two bits $x_j$ and $x_k$ are on, $j > k$

- $ST_{odd} :=$ all odd-positioned elements in $ST$
- $ST_{even} :=$ all even-positioned elements in $ST$

- transmit $M(LV-1, MR, 0, ST_{odd})$ to $d+(i+1) \cdot INV$
- transmit $M(LV-1, MR, 0, ST_{even})$ to $d+(i+1) \cdot INV$

- **endif**

- **else**

- **if** $CN \neq 0 \Rightarrow$ this is a link element
  - between tree nodes outside basic tile

- **if** $i$ even **then**
  - Straight-line links at this level

- transmit $M(LV, MR, CN, ST)$ to $(d+3)$-neighbor

  **else**
  - Zigzagging links at this level

- **if** $CN \mod (B+1) = 0$ **then**
  - Change in direction is needed

- **if** $MR \in \left( \frac{CN}{B+1} \text{ odd} \right)$ **then**
  - transmit $M(LV, MR, CN-1, ST)$ to $(d+2)$-neighbor

  **else**
  - transmit $M(LV, MR, CN-1, ST)$ to $(d+4)$-neighbor

  **endif**

- **else**

- transmit $M(LV, MR, CN-1, ST)$ to $(d+3)$-neighbor

- **endif**

- **endif**

Figure 8: A Distributed Placement Algorithm for the Zigzag Replication of a Parallelogram Tile.
Fig. 9: An example of partitioning the PE's into seven testing groups.

Fig. 10: The three types of connecting elements:
(a) type 3 (0+3), (b) type 5 (1+4), (c) type 1 (2+5).
Fig. 11: CE's formed by a faulty connection between A and B.
Fig. 12: CE's formed by a faulty PE.
Fig. 13: The effect of multiple faults; A and the connection between C and D are faulty.