Embedding Tree Structures in VLSI Hexagonal Arrays

DAN GORDON, ISRAEL KOREN, AND GABRIEL M. SILBERMAN

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 here and shown to be superior over known mappings on square arrays (or grids).**

Index Terms—**Distributed configuration algorithm, hexagonal multiprocessor array, mapping scheme, tree structures, VLSI.**

I. 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 [1]-[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 Fig. 1.[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 and the users alike. One way to achieve it, is to make standard pieces of silicon which can be programmed by on-chip software to perform different functions.

Manuscript received July 9, 1982; revised August 15, 1983.

D. Gordon is with the Department of Computer Studies, University of Haifa, Haifa, 32000, Israel.

I. Koren is with the Department of Electrical Engineering, Technion—Israel Institute of Technology, Haifa, 32000, Israel.

G. M. Silberman is with the Department of Computer Science, Technion—Israel Institute of Technology, Haifa, 32000, Israel.

[1] When a tree is imbedded in a grid, some PE's function as links between active tree nodes. The latter are shown enlarged.

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. In the next section we concentrate on various tree configurations which can be mapped on an hexagonal array. In Section III we derive area and propagation time expressions, and show the superiority of hexagonal arrays over square arrays. Finally, a distributed configuration algorithm for a tree on the hexagonal array is presented in Section IV.

II. 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 tree which is replicated to form the full tree (Fig. 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 (Fig. 2) or a parallelogram (Figs. 3-4). The basic tile, containing a full binary tree, 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 Fig. 2. The resulting overall pattern is always rectangular. 2) The *parallel* replication of a parallelogram basic tile is shown in Fig. 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 Fig. 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 Figs. 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, Fig. 2 shows that a level 6 tree need have no links to connect the nodes (compare with Fig. 1). Also, Figs. 2-4 show that in the basic tile almost all PE's are active nodes. In Section III we show that these advantages over the square grid are maintained even when the tree level tends to infinity.

The basic tiles of Figs. 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 Fig. 6.

III. 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 $k = c + i$, we have

$$P_{c+i} = [(A+1)2^{\lfloor i/2 \rfloor} - 1][(B+1)2^{\lfloor i+1/2 \rfloor} - 1]; \; i \geq 0. \tag{1}$$

0018-9340/84/0100-0104$01.00

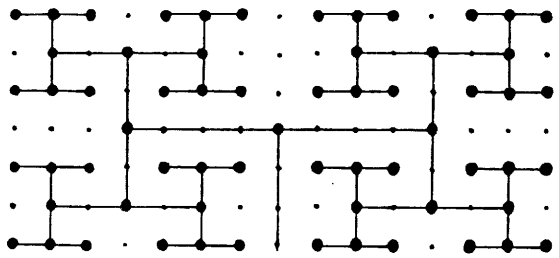


Fig. 1. A six level tree on a square grid.

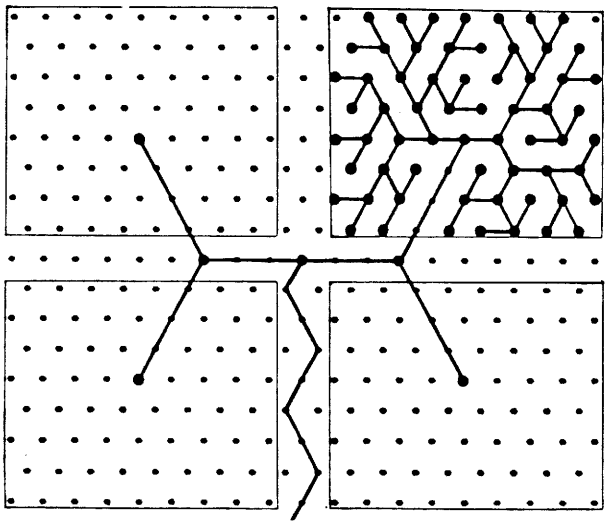


Fig. 2. Rectangular replication of a six level subtree embedded in a rectangular tile.

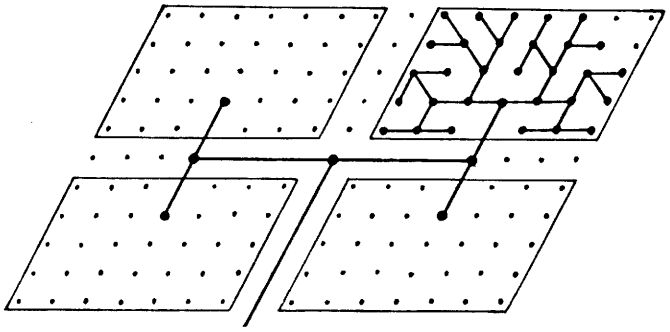


Fig. 3. Parallel replication of a five level parallelogram tile.

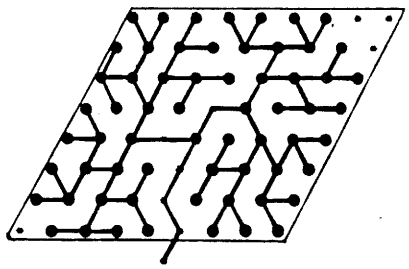
$$Z_{c+i} = \left\lfloor (A+1)2^{\lfloor i/2 \rfloor} + \left\lfloor \frac{B-1}{2} \right\rfloor - 1 \right\rfloor \left[(B+1)2^{\lfloor i+1/2 \rfloor} - 1 \right]; \quad i \geq 0. \quad (2)$$

The derivation of (1) is straightforward. The derivation of (2) can be followed by inserting the parallelogram of Figs. 3 or 4 into Fig. 5. $Z_{c+i} > P_{c+i}$ *but* $\lim_{i \to \infty} Z_{c+i}/P_{c+i} = 1$, i.e. only a vanishing percentage of PE's is wasted by using the zigzag instead of the parallelogram scheme.

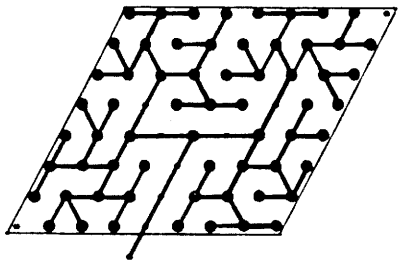
A full binary tree of level k has $2^k - 1$ nodes, so the ratio of PE's used as nodes over the total number of used PE's is $(2^k - 1)/P_k$ and $(2^k - 1)/Z_k$, for the parallel and zigzag schemes, respectively. It is easy to see that both ratios tend to $2^c/(A+1)(B+1)$ *as* $k \to \infty$. For $A =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



(a)



(b)

Fig. 4. Six level subtrees embedded in parallelogram tiles.

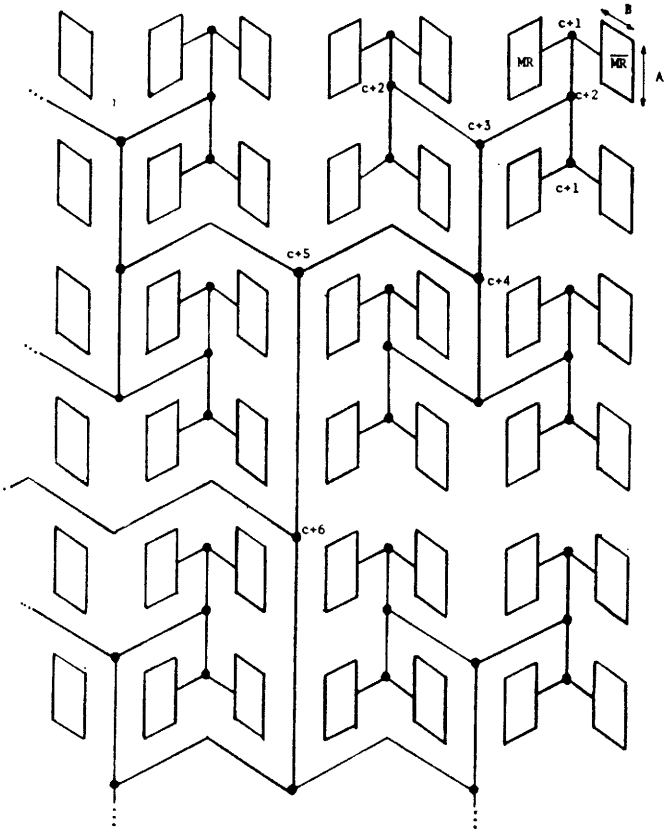


Fig. 5. Zigzag replication of a parallelogram tile.

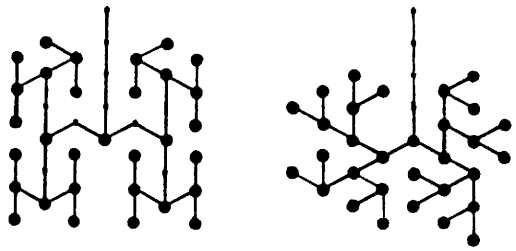


Fig. 6. Five level tiles minimizing (a) inorder, (b) postorder.

$$S_k = \begin{cases} (2^{(k+1)/2} - 1) * (2^{(k+1)/2} - 1), & \text{if } k \text{ is odd;} \\ (2^{(k+2)/2} - 1) * (2^{k/2} - 1), & \text{if } k \text{ is even.} \end{cases} \quad (3)$$

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 $2^{c+1}/(A + 1)(B + 1)$ *as* $k \to \infty$. For the 8×9 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 percent and still save area. The same is true if the rectangular pattern of Fig. 2 is used, because it takes up even less area than the parallel layout based on the 8×9 parallelogram.

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:

$$\Delta_{c+i} = \begin{cases} 1, & \text{if } i = 1; \\ \lfloor (A + 1)2^{(i-4)/2} \rfloor, & \text{if } i \geq 2 \text{ and even;} \\ (B + 1)2^{(i-3)/2}, & \text{if } i \geq 3 \text{ and odd.} \end{cases} \quad (4)$$

From this, the total propagation delay is

$$D_{c+i} = \begin{cases} D_c + 1, & \text{if } i = 1; \\ D_c + 1 + \left\lfloor \frac{A+1}{2} \right\rfloor + (A + 1)(2^{\lfloor (i-2)/2 \rfloor} - 1) + (B + 1)(2^{\lfloor (i-1)/2 \rfloor} - 1), & \text{if } i \geq 2. \end{cases} \quad (5)$$

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

The delay for the rectangular pattern of Fig. 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 Fig. 4(a)].

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

$$R_k = \begin{cases} 3 \cdot 2^{(k-2)/2} - 2, & \text{if } k \text{ is even;} \\ 2^{(k+1)/2} - 2, & \text{if } k \text{ is odd.} \end{cases} \quad (6)$$

This expression is easily derived from [8].

Assuming an 8×9 parallelogram as in Fig. 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 1.14 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 percent, and the asymptotic ($k \to \infty$) break-even values of x are 26 percent and 14 percent for even and odd values of k, respectively.

IV. Distributed Placement Algorithm

The placement of a tree on an hexagonal array may be done 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 PE contain (or obtain through broadcasting) 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 PE, at the expense of increasing the time needed to complete the configuration process.

An important advantage of a distributed placement algorithm is that it is independent of the exact size of the available hexagonal array. It *dynamically* places a tree on the array and consequently, a change in the size of the array 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 Fig. 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.

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 the 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 Fig. 7.

The directions of the outgoing messages are determined relative to d—the direction of the incoming message ($0 \leq d \leq 5$). Thus, $(d + 1) \bmod 6$ (or $(d - 5) \bmod 6$) is the direction next to d when moving clockwise (Fig. 7). Similarly, $(d + 3) \bmod 6$ 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 "$(d + i) \bmod 6$."

A distributed algorithm for a zigzag replication of the tile shown in Fig. 3 is depicted in Fig. 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 (3). Then, the directions of outgoing messages and their appropriate mirror flags are calculated while configuring in conformance with Fig. 3.

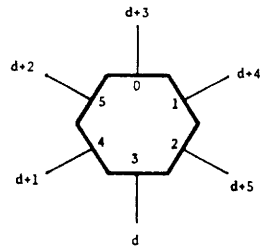


Fig. 7. Absolute and relative to d numbering of directions.

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 outging 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 ($1\downarrow ST$ 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 Fig. 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 processor inside the tile is reached, the 9 = 01001 is dropped and two substrings are formed. If the mirror flag is MR = false (MR = true) then the odd substring 6, 18, 6, 9, 12, 24, 9 is transmitted in direction $d + 4(d - 4)$, and the even substring 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 LV-c is even for a certain node (at level LV), then straight lines connect that node to its sons (see Fig. 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 Fig. 8 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.

V. 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.

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.

```
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' := Δ_LV                   { Refer to Equation (3)                }
  i := LV - c
  if MR⊕(i≤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, MR̄, CN', ST) to (d+1)-neighbor
      transmit M (LV-1, MR, CN', ST) to (d-1)-neighbor
   else
      transmit M (LV-1, MR̄, 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, MR̄, CN', ST) to (d+2)-neighbor
      transmit M (LV-1, MR, CN', ST) to (d-2)-neighbor
   else
      transmit M (LV-1, MR̄, CN', ST) to (d-2)-neighbor
      transmit M (LV-1, MR, CN', ST) to (d+1)-neighbor
   endif
 endif
else                            { LV ≤ c => a node inside basic tile   }
 X := first element in ST       {        X = x4x3x2x1x0                }
 INV := if MR then -1 else 1 endif
 if a single bit x_j in X is set then
                                { Drop first element in ST (1↓ST in APL)}
  transmit M (LV, MR, 0, 1↓ST) to d+(j+1)·INV
 else                           { Two bits x_j and x_k are on, j>k     }

  ST_odd := all odd-positioned elements in 1↓ST
  ST_even := all even-positioned elements in 1↓ST
  transmit M (LV-1, MR, 0, ST_odd ) to d+(j+1)·INV
  transmit M (LV-1, MR, 0, ST_even) to d+(k+1)·INV
 endif
endif
else                            { CN≠0 => 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-1, 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⊕(CN/(B+1) odd) 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
endif
```

Fig. 8. A distributed placement algorithm for the zigzag replication of a parallelogram tile.

References

[1] J. Bentley and H. T. Kung, "A tree machine for searching problems," in *Proc. 1979 IEEE Int. Conf. Parallel Processing*, Aug. 1979, pp. 257–266.

[2] S. W. Song, "A highly concurrent tree machine for database applications," in *Proc. 1980 IEEE Int. Conf. Parallel Processing*, Aug. 1980, pp. 259–268.

[3] C. Mead and L. Conway, *Introduction to VLSI Systems*. Reading, MA: Addison-Wesley, 1980.

[4] C. E. Leiserson, "Systolic priority queues," in *Proc. Caltech Conf. VLSI*, Jan. 1979, pp. 199–214.

[5] D. A. Patterson, E. S. Fehr, and C. H. Sequin, "Design considerations for the VLSI processor of X-tree," in *Proc. 6th Annu. Symp. Comput. Arch.*, Apr. 1979, pp. 90–101.

[6] C. Sequin, "Single chip computers, the new VLSI building blocks," in *Proc. Caltech Conf. VLSI*, Jan. 1979, pp. 435–446.

[7] E. Horowitz and A. Zorat, "The binary tree as an interconnection network: Applications to multiprocessor systems and VLSI," *IEEE Trans. Comput.*, vol. C-30, pp. 247–253, Apr. 1981.

[8] I. Koren, "A reconfigurable and fault-tolerant VLSI multiprocessor array," in *Proc. 8th Annu. Symp. Comput. Arch.*, May 1981, pp. 425–442.

[9] H. T. Kung, "The structure of parallel algorithms," *Advances in Computers*, vol. 19, M. C. Yovits, Ed. New York: Academic, 1980, pp. 65–112.

[10] L. Snyder, "Introduction to the configurable highly parallel computer," *Computer*, pp. 47–56, Jan. 1982.