A Hexagonal Array Machine for Multilayer Wire Routing

R. VENKATESWARAN AND PINAKI MAZUMDER, MEMBER, IEEE

Abstract—**Maze routing is widely used in both printed circuit board (PCB) and VLSI design. However, for the ever increasing design requirements, this can no longer be done economically without the help of special purpose hardware accelerators. A new hardware accelerator comprised of several fast processors interconnected in the form of a hexagonal mesh with wraparound connections is proposed.**

The novelty of the proposed architecture stems from the fact that it is suitable not only for single-layer routing, but also for routing in parallel on multiple layers. A hexagonal machine of dimension $\sqrt{kG}$, with about $3kG$ processors, can handle a k-layer grid consisting of kG^2 grid points at about the same speed as a *full-grid machine* with kG^2 processors.

A technique for measuring the *performance* of a hardware accelerator, in terms of the average delay incurred over a full-grid machine, is suggested. This has been formalized in case of the hexagonal architecture and is presented for various nets and mesh dimensions. The results have been accurately verified by extensive simulation done in C++ language. It is also demonstrated that the hexagonal mesh, by virtue of its additional links for expansion, is resilient to about 10% of failure in the links and processing elements. A detailed design for a chip implementation of the hexagonal machine is also discussed.

Keywords—**Hexagonal array, multilayer routing, interprocessor cycle period, average delay factor, reconfigurability.**

I. INTRODUCTION

AUTOMATIC LAYOUT of wiring patterns for printed circuit boards (PCB's) and integrated circuits (IC's) have been in vogue for the past several years. For a PCB, the components are IC packages and the electrical connections are made by a metal etching process. Connections between layers are made by drilling holes through the fiberglass and plating them with metal. In an IC, wire lines of polysilicon are fabricated to carry electrical signals between circuits. In addition, one or two layers of metal separated by insulating layers of oxide are deposited and etched above the silicon to form wire lines. Holes are left in the oxide to form interlayer contacts or vias. Thus the routing problem, which is to connect all the points of each net and to ensure that the wiring paths of the different nets do not intersect each other on any layer, is quite similar in both the environments. Furthermore, several constraints, such as the total wire length, number of vias used, critical nets, etc., are imposed on the solution generated. In this paper, we propose a new hexagonal mesh architecture for a parallel multilayered routing algorithm that is applicable in both of these environments.

Several algorithms, such as the channel [4], [23], maze [11], river [15], $\alpha - \beta$ [9], etc., have been proposed in the literature for routing interconnects in IC's and PCB's. Among these, the maze router, originally proposed by Lee [11], uses breadth-first search, and thereby it is *admissible* in the sense that it always finds a shortest-length path, if one exists. This attribute of the maze router is frequently exploited in practice to minimize the total interconnect length and, presumably, the overall chip area. Section III deals with the Lee algorithm in greater depth. Many commercial routers use the Lee algorithm or its variant [7], [6] exclusively, or initially use some other algorithms to rapidly interconnect most of the nets and then utilize the Lee algorithm to interconnect the remaining nets. However, this is achieved by paying a high premium of large storage space (in the worst case, an exponential to the path length L) and expensive runtime (in the worst case, $\Theta(L^2)$ time to find a path of length L). Elegant coding schemes, such as the one suggested by Akers [1], can be used to alleviate the storage space problem. However, time continues to be a severe constraint in a uniprocessor implementation. Two schemes suggested were the pipeline-based approach of Sahni [18] and the raster-based approach of Rutenbar [17]. These approaches, though economical in hardware, often reduce the $\Theta(N^2)$ time complexity by only a small constant factor, and hence, are inadequate for large problem sizes.

For multiple layers, the problem becomes even more acute. The accepted strategy is to route as many nets as possible on each layer independently. A global routing is attempted only for the unfinished nets. However, it is well established that these few remaining nets account for the majority of the time required in routing.

Specially designed multiprocessor-based routing engines or hardware accelerators thus become absolutely necessary [2] for doing the complex routing in the very-large-scale integrated (VLSI) circuits of today. The Lee maze algorithm, by its very nature, offers much potential for parallelization, and hence, is an excellent candidate.

The ideal architecture would be an interconnected $N \times N$ k-layer processor array, where each processor P_{ijk} has

Manuscript received July 21, 1989. This work was supported by the Army Research Office under the URI program under Grant DAAL 03-87-K-0007, by the Digital Equipment Corporation Faculty Award, and by the National Science Foundation Research Invitation Award under Grant MIP-8808978. This paper was recommended by Associate Editor A. E. Dunlop.

The authors are with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109-2122.

IEEE Log Number 9036682.

a one-to-one correspondence with a grid cell a_{ijk} in the layout. Such a *full-grid machine*, however, requires kN^2 processors and $2kN(N-1)+N(k-1)$ links for a k-layer $N \times N$ grid, which is clearly expensive. Breuer and Shamsa's L-Machine [3] is the first published design of this nature. However, it is inflexible in the sense that it is incapable of handling problem sizes larger than the physical size of the processing array. Thus the need for a better architecture, where the grid array can be efficiently mapped onto a much smaller subset of processors, has been widely recognized (the *folding* problem). One of the chief factors affecting the performance of any such architecture implementing the Lee algorithm is the *interprocessor cycle period* (ICP). The ICP is defined as the smallest number of distinct processors that are encountered before one gets repeated, while traveling along any straight line on the grid.

The wire routing machine (WRM) built by Nair *et al.* [8] is probably the precursor of the present trend of *virtual* machines. It consisted of processing elements connected in the form of a square mesh. The chief difference of the WRM was that it used general-purpose microprocessors rather than custom-made hardware for the node elements; thus trading compactness for versatility. Martin [12] has suggested the suitability of the *torus-like mesh* for folding operations. Suzuki and others [21] have built a machine with 64 processors interconnected in the form of a twisted torus. Other similar implementations are described in [19] and [22].

The intention of this paper is to propose a new architecture for the physical implementation of the Lee algorithm, wherein the processors are interconnected in the form of a *C-wrapped hexagonal mesh*. Table I reflects the superiority of the hexagonal interconnection topology to existing ones. The larger ICP value implies fewer conflicts in processor assignments during wavefront expansion, thereby improving the overall performance. Since each processor is connected to six others, the hexagonal machine can do *multilayer* expansion in parallel, unlike existing accelerators. This again results in shorter routing time requirements. Performance results, as obtained from extensive simulation runs and supplemented by analytical derivations, have been very promising.

The basic labeling scheme used in the hexagonal array machine is a very powerful one. It can be used for machines where the number of nearest neighbors of each processor is other than 6. For example, reconfiguring the basic hexagonal machine by deleting all diagonals along any one of the three directions yields a cheaper machine that is still extremely efficient in handling two-dimensional maze routing on a single layer. This is because the ICP property is unaffected by the number of neighbors present. This property also makes the hexagonal array machine resilient to link and processing element failure, so far as its routing capability goes.

The rest of the paper is organized as follows. Section II introduces the architecture, labeling scheme, and other salient properties of the hexagonal array machine. Section III explains how concurrent multilayer wire routing is possible on the hexagonal array machine. In Section IV, an analytical model for estimating the delay is presented. Section V discusses both hardware and software issues for chip implementation of the hexagonal array machine. Performance is analyzed in Section VI. Reconfiguration and fault tolerance aspects are discussed in Section VII.

TABLE I
EVALUATION OF INTERCONNECTION TOPOLOGIES FOR MAZE ROUTING

Interconnection Topology[†]	Interprocessor Cycle Period	Multilayer Routing	Reconfigurability
Square mesh [8]	$\sqrt{N}$	×	×
Twisted torus [21]	$\frac{N}{2}$	×	×
Hexagonal mesh	N	✓	✓

[†]Wraparound topologies with N processing elements. × indicates No and ✓ indicates Yes.

II. HEXAGONAL-ARRAY MACHINE

Definition 1: A C-wrapped hexagonal mesh of dimension e is comprised of $3e(e-1)+1 (=N$, say) processors, labeled from 0 to $3e(e-1)$, such that each processor s has six neighbors $[s+1]_N$, $[s+3e-1]_N$, $[s+3e-2]_N$, $[s+3e(e-1)]_N$, $[s+3e^2-6e+2]_N$, and $[s+3e^2-6e+3]_N$, where $[a]_b$ denotes $a \pmod b$.

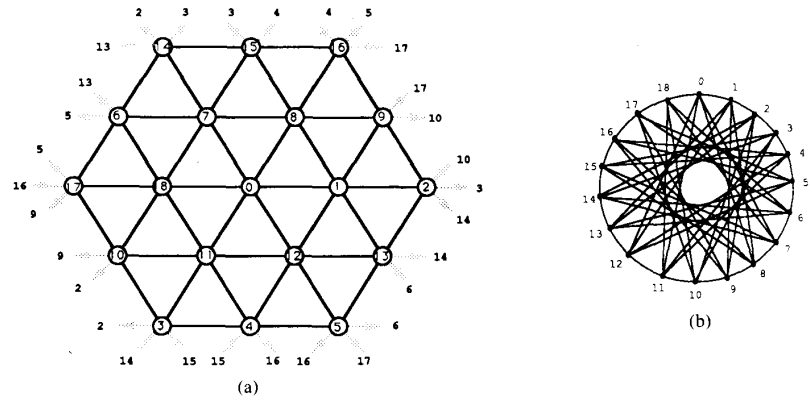
Property 1: An unwrapped hexagonal mesh of dimension e can be partitioned into $2e-1$ rows in three possible ways: along the horizontal direction, along the 60-deg counter-clockwise direction, or along the 120-deg counter-clockwise direction.

Observation 1: Along any of the three directions, let R_0 be the top row, R_1 the second row, and so on until R_{2e-2}. Then a *C-type* wrapping is obtained by wrapping the last processor in R_i to the first processor in $R_{[i+e-1]_{2e-1}}$.

Fig. 1(a) shows an H-mesh of dimension 3 with the wrappings indicated alongside the arrows. Note that in this case there are 19 nodes distributed over five rows, with the wraparounds in the three directions as noted in Observation 1. For example, in Fig. 1, the last processor in R_2 along the horizontal direction, viz., node 2 is wrapped to the first processor in R_4, node 3. The chordal equivalent of the hexagonal mesh, showing all the wraparounds is shown in Fig. 1(b). Such a topology and labeling scheme have also been studied in relation to experimental distributed real-time systems such as HARTS [5] and FAIM [20].

Property 2: A C-type wrapping is a homogeneous interconnecton. Any node can be labeled as node 0, that is, as the center of the mesh.

Lemma 1: For a C-wrapped hexagonal mesh of dimension e, the ICP, i.e., the number of distinct processors that one encounters before returning to the same processor



(a) (b)

Fig. 1. (a) A wraparound hexagonal mesh of dimension 3. (b) Its chordal equivalent.

while traveling along any of the three directions, is $p = N = 3e^2 - 3e + 1$.

Proof: From Observation 1, we know that the last node of row R_i is connected to the first node of row $R_{[i+e-1]_{2e-1}}$. This can be interpreted as being a linear congruent sequence of the form mod $(a_x + b)\, m$. In our case, we have $a = 1, b = e - 1, x = i$, and $m = 2e - 1$. From random number theory, this sequence has been shown to have the maximum possible period of m if and only if b is relatively prime to m [10]. This implies, that in our case, the sequence will be of length $2e - 1$ (the number of rows in the unwrapped hexagonal mesh of dimension e) if $(e - 1)$ is relatively prime to $(2e - 1)$.

Hence, our assertion that $p = 3e^2 - 3e + 1$ is true, provided that the node numbering is unique. This is true in the horizontal direction, as we number the nodes with consecutive numbers. However, from Property 1 and Observation 1, it follows that it must also be true in the other two directions. Q.E.D.

III. Routing on the Hexagonal Mesh

The hexagonal array machine is unique in the sense that it is the first to attempt concurrent multiple-layer routing. By cleverly assigning processors to grid points on the active wavefront, the hexagonal array machine considerably reduces the otherwise enormous time requirements. Concurrent multilayer wire routing has several advantages.

- *Via Minimization:* A *via* is a contact used to connect a wire that extends over two physical layers (metal and polysilicon). Vias not only take more area but also reduce the reliability of the circuit. Hence, minimizing the number of vias is an important issue and is usually carried out by a post-processing step. This additional cost is often very high and sometimes unacceptable.

Most routers constrain the polylines to run in one direction (say horizontal) and the metal lines to run in the other (vertical) direction. Thus the two can be superposed and a common wavefront propagated. However, every bend in a wire route causes a switch in the layers, and hence, must be realized using a via. Clearly, a high percentage of these vias are unnecessary and could have been avoided by allowing poly and metal to run in both directions. Relaxing the constraint is made possible in the hexagonal array machine by allowing for multiple wavefronts at the same time. It also implicitly assigns an additional cost to each via during the wave-expansion step. In this way, paths with more vias become less attractive costwise.

- *Increased connectivity:* Usually maze routers proceed by routing one net at a time. The question of net ordering is, therefore, an important one. The hexagonal array machine does not eliminate this problem. However, by treating the metal and poly (and possibly other layers) separately, an obstacle on one layer does not preclude routing on the other. Now, consider a router that does multiple-layer routing by considering one layer at a time in a serial fashion. An important question for such routers is where to introduce the vias? This question is rendered irrelevant by routing on all layers simultaneously.
- *Technology:* Current IC technology is capable of more than two layers: two-metal and two-silicon layer technology is already in production. In PCB's, several layers have traditionally been used. So the need for multiple-layer wire routing is an accepted one.

The rest of this section will explain how the hexagonal array machine can perform concurrent multiple-layer routing. Before proceeding any further, we digress a little to outline the basic Lee maze routing algorithm. The Lee algorithm consists of three distinct phases, namely a) wave expansion, b) backtracing, and c) label clearance. The *wave-expansion* step starts from the source cell/cells by labeling all unoccupied adjacent cells. These newly labeled cells constitute the new wavefront for the next expansion. The process is repeated until the target cell is reached. If we assume that all cells and all nodes have similar behavior, and that the propagation speeds are similar in all directions, even in the case of an asynchronous implementation, then for uniform labeling these wavefronts appear as diamond-shaped fronts. This observation

suggests that any good mapping strategy must ensure *minimum repetition* of processors along any 45- or 135-deg lines. In the *backtracing* step, the labels are traced back from the target to the source, and the shortest path found. In the final *label-clearance* step all extra labels are cleared and the cells on the new net are marked as being occupied for future expansions.

For multiple layers, each cell must now expand the wavefront not only to its four neighbors on the same layer, but also to its neighbors on the adjacent layers. Such an expansion is possible in a single step on the hexagonal array machine, since each node has six adjacent neighbors with which it communicates directly. In a square mesh or torus topology, the usual strategy is to assign the same processor to handle the cell a_{ij} in all the layers. However, expansion can no longer take place in a single step, resulting in longer routing times.

Definition 2: (*Operator*) Let Φ be the mapping operator, such that

$$\Phi: I_x \times I_y \times I_z \rightarrow \mathrm{P}$$

where I_i is $\{k \mid 0 \leq k \leq i$, where i is the dimension of grid$\}$ and P is $\{m \mid 0 \leq m < e^2 - 3e + 1\}$. Then

$$\Phi: \langle x, y, z \rangle \equiv [a + w_z[z]_N + w_y[y]_N + x]_N$$

where a is the processor assigned to the grid cell (0, 0, 0), $w_y = 3e - 1$ and $w_z = 3e - 2$, and $[a]_b$ refers to a mod b.

Φ is derived based on the mapping scheme shown in Fig. 2. For other assignments, the formula for Φ can be similarly derived. Fig. 3 shows the mapping for a two-layer 16 by 16 grid, using a four-dimensional C-wrapped hexagonal array. As can be seen, it consists of repeated *folding* of the tile, corresponding to the unwrapped four-dimensional hexagonal array.

Mapping: Of the six neighbors of each node of the hexagonal mesh, four of them are assigned to cells on the same layer and the remaining two to the up-and-down neighbors on the adjacent layers. Since cells on different layers are reached in subsequent expansions, vias are implicitly given the cost of traveling along an arc. If a greater cost must be attached to them, then we can assume the different arcs to have different weights. For instance, a weight of 3 can be assigned to the z-links to represent the additional cost of the vias. To discourage bends in the wiring path, a cost of 1 or 2 can be assigned to an x/y link emerging from a node, depending on whether the link lies in the same direction or in a different one from that by which that node has been labeled. For an N-node hexagonal mesh, it has been shown in Lemma 1 that all horizontal, vertical, 45- and 135-deg paths are mapped on a cycle containing *all* cells of the machine, i.e., on a cycle of periodicity N. This clearly leads to a processor assignment with the fewest conflicts. The revised Lee algorithm for multiple layers is given in the following.

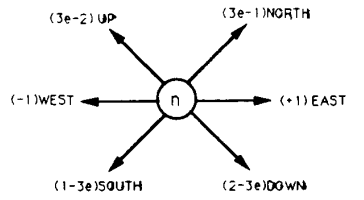


Fig. 2. Processor labeling scheme.

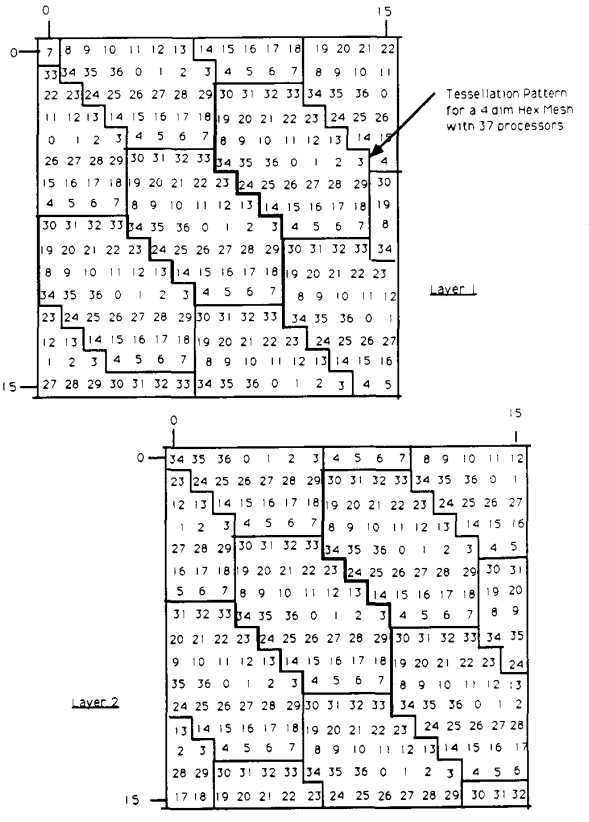


Fig. 3. Mapping for the two-layer 16 by 16 grid using a four-dimensional H-mesh.

A. Modified Lee Algorithm for Multilayer Concurrent Routing

This algorithm is a variant of the Lee algorithm. It makes use of the six links of each processor in the hexagonal array to propagate the wavefront simultaneously over all the layers and find the most optimal path, if one exists.

1) Wave Propogation Phase:

1) Initialization. Set obstruction flags,[1] source flags (SC's), and target flags (TC's) in the appropriate processors. Mark SC as the active processor to be expanded.

2) Expand from the active cells in all six directions

[1]These represent the terminals and the obstructions posed by the nets that have been previously routed.

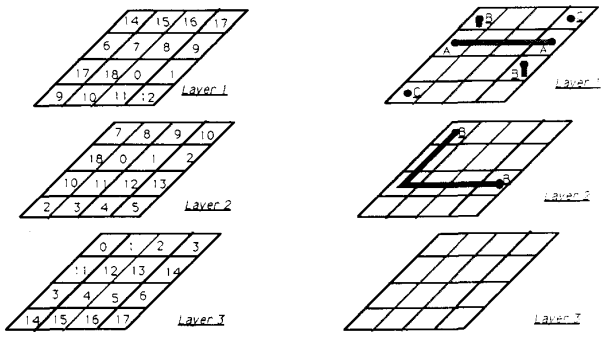


Fig. 4. (a) Processor assignments for a three-layer 4 by 4 grid. (b) Initial netlist.

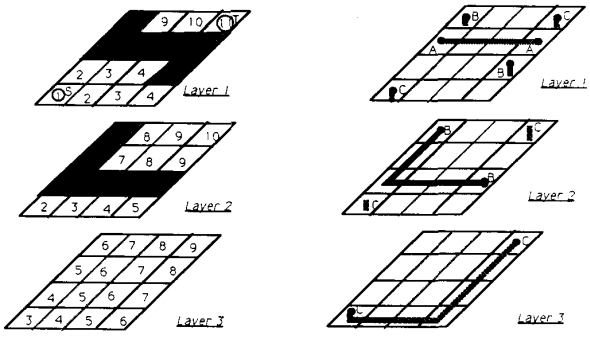


Fig. 5. (a) Snapshots of wavefronts on three layers. (b) Final wiring.

along the links of the hexagonal mesh. Transmit to each processor the following four-tuple:

$$[\lambda, x, y, c]$$

where λ is the layer, x and y are the x and y coordinates of the cell being reached in this expansion, and c represents the cost for reaching the cell along this path and can be omitted if unequal weights are not assigned to the various links.

3) If TC has been marked, go to the backtrace phase. Else repeat step (2).[2]

2) Backtrace Phase:

1) Let l be the final label of the TC. Mark TC as the *cur_node*.

2) Mark all the adjacent (at most six) nodes which have the label $l - 1$. If there is only one such node, go to step 4.

3) Choose the node that is on the same layer as *cur_node* if possible. If there is more than one possible candidate, then pick the one that lies in the same direction as *cur_node* was with its predecessor.

4) Mark the new node chosen in steps 2 or 3 as the *cur_node*. If it corresponds to the source, then quit, as a path has been traced out. Else go to step 2.

Example: We now illustrate the process by means of an example. We consider a 4 by 4 three-layer grid.[3] Fig. 4(a) shows the processor assignment to the 64 grid points, obtained by applying the operator Φ. Fig. 4(b) shows two nets, *A* and *B*, that have been previously routed. Our assignment is to route the third net, *C*, whose endpoints are also shown. At this stage, one may note that it is not possible to complete this interconnection without using the third layer.

[2]It is assumed that all processors can simultaneously send and receive messages from their neighbors. This enables us to mark all the neighbors which need not be expanded in the next phase. To further speedup the process, we may mark the cells in all layers at the same $\langle x, y \rangle$ position as the TC as targets. This is true if we assume accessibility of terminals from all layers.

[3]The three layers could represent a two-metal and one-silicon layer technology.

TABLE II
PROCESSOR ASSIGNMENTS DURING WAVE PROPAGATION CYCLES

Clock	Processor Assignments		
Cycle	Layer 1	Layer 2	Layer 3
1	9	-	-
2	10, 17	2	-
3	11, 18	3	14
4	0, 12	4	3, 15
5	-	5	4, 11, 16
6	-	-	0, 5, 12, 17
7	-	0	1, 6, 13
8	-	1, 8	2, 14
9	15	2, 9	3
10	16	10	-
11	17	-	-

Fig. 5(a) is the snapshot at the culmination of the wave propagation phase of the modified Lee algorithm. The shaded areas indicate the previous obstacles. The number in each square is the label associated with that grid point, i.e., all cells with a label i would lie on the ith wavefront. For this example, the wave propagation phase requires eleven clock cycles. There are also no processor conflicts. Table II shows the active processors in each clock cycle. The distinction between layers is made solely to aid understanding. Thus during clock cycles 3, 4, and 9, a common wavefront exists over all three layers. Fig. 5(b) shows the final route for all three nets.

IV. Delay Modeling for the Hexagonal Array Machine

The performance of the hardware accelerator, using multiple interconnected processors *folded* to yield a larger grid size, is usually measured by how much the mapping minimizes the *additional delay* incurred over the corresponding *full-grid* implementation. For our machine, the maximum additional delay for any given wavefront is determined by the maximum number of cells assigned to a single processor on that wavefront. Thus if c_i is the number of cells assigned to processor P_i on a certain wavefront, then the additional delay involved is given by $[\max(c_i) - 1]$, where $0 \leq i \leq N - 1$. Hence, ideally, we

would like the additional delay to be as close to zero as possible.

The extra delay is usually quite complex to calculate as its depends on a number of factors, such as i) location of source and destination cells on the grid, ii) obstacles on the grid, iii) size of the grid, iv) dimension of the hexagonal mesh used, and v) physical implementation issues such as SIMD/MIMD mode of control. To date, most work in this area has been characterized by an attempt to estimate the delay by simulation alone, which ordinarily requires $\Theta(kG^2)$ computations for a k-layer grid of size $G \times G$. However, based on the nature of wavefront propagation on the hexagonal mesh, we can propose a simpler model that can yield us the same result using only $\Theta(kN^2)$ computations, where N is the number of processors available and $G \gg N$. We believe that similar models can be developed for other topologies as well.

Notation 1: Let D_i be a permutation of the group

$$Z_N = \{[0], [1], [2], \cdots, [N-1]; \oplus\}$$

of congruence classes modulo N.

Theorem 1: The four diagonals of the ith wavefront expansion on the hexagonal mesh machine of dimension e, handling a single-layer obstacle-free grid of dimension $G \times G$, are given by the first $i+1$ columns of the *diagonal matrix*, $\mathfrak{D}$, where

$$\mathfrak{D} = \begin{bmatrix} D_1 \\ D_2 \\ D_3 \\ D_4 \end{bmatrix}$$

and the D_i are as per Notation 1.

Proof: We know that on an obstacle-free grid, the wavefronts are diamond-shaped. Let us label the four diagonals as D_1, D_2, D_3, and D_4. As can be seen from Fig. 6, processors along D_1 and D_3 differ by $[3e]_N$, while those along D_2 and D_4 differ by $[3e-2]_N$. From Lemma 1, we know that both these result in cycles of length N. Hence, indeed the D_i are of the form Z_N.

We claim that to estimate the delay, we need know only the ordinality of the set of cells assigned to each processor and not the identity of the cells or processors themselves. Hence, without loss of generality, we can map the processors appearing along D_1 by the vector $\{0, 1, 2 \cdots N-1\}$.[4] The entries on the other rows of $\mathfrak{D}$ will be based on this mapping.

[4] Henceforth, it is assumed that the numbers 0, 1, etc. refer to the corresponding set of congruence classes modulo N.

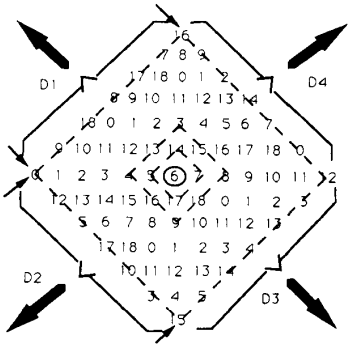


Fig. 6. Wavefront expansion on 6 on a three-dimensional mesh.

It is obvious that D_3 will only be a shifted version of D_1, where the shift factor j is given by

$$j = [N - (2e-1)i]_N$$

where $N = 3e^2 - 3e + 1$.

Furthermore, it can be verified that D_2 can be obtained by applying a mapping function $\Psi: D_1 \rightarrow D_2$, where

$$\Psi: \langle x \rangle \equiv [N - (2e-1)x]_N.$$

Corollary 1 shows that Ψ is an *automorphism*, which is to be expected as D_1 and D_2 are cyclic groups of order N. Like D_3, D_4 will also be a shifted version of D_2, where this time the shift factor k is given by

$$k = [N - (3e^2 - 9e + 4)i]_N. \qquad \text{Q.E.D.}$$

Numerical Example: The D matrix for a hexagonal mesh of dimension 3 and for $i = 6$ is shown in the following. Note that D_3 is D_1 shifted right, $j = [19 - (2*3-1)6] \bmod 19 = (-11) \bmod 19 = 8$, and D_4 is D_2 shifted right, $k = [19 - (3*3^2 - 9*3 + 4)6] \bmod 19 = (-5) \bmod 19 = 14$:

$$\left[\begin{array}{ccccccc|ccccccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 \\ 0 & 14 & 9 & 4 & 18 & 13 & 8 & 3 & 17 & 12 & 7 & 2 & 16 & 11 & 6 & 1 & 15 & 10 & 5 \\ 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 6 & 1 & 15 & 10 & 5 & 0 & 14 & 9 & 4 & 18 & 13 & 8 & 3 & 17 & 12 & 7 & 2 & 16 & 11 \end{array}\right].$$

The first seven columns of D represent the processor assignment on the sixth wavefront expansion. For instance, the fifth entry of D_1 is the same as the fourth entry of D_2 (viz. 4). Referring to Fig. 6, we find that processor 17 is indeed assigned to both these cells. The elegance of this scheme is that, on the basis of the D matrix alone, we can now estimate the delay on any wavefront.

Corollary 1: $\Psi = [N - (2e-1)i]_N$ is an automorphism.

Proof: Ψ is a bijection from D_1 to D_2. This is true as elements of group D_2 differ by $(2e-1)$, which is seen to have a cycle of N.

$\Psi(x \oplus y) = \Psi(x) \oplus \Psi(y)$ for any x and $y \in D_1$ and where $\oplus$ is the modulo N addition operator. Q.E.D.

A. Delay Estimates When e Is 1

This is an interesting case as $e = 1$ corresponds to a uniprocessor environment. The number of cells on the ith wavefront n_i for a grid of size $G \times G$ is given by the following. For G odd

$$n_i = \begin{cases} 4i, & i \le \left\lfloor \frac{G}{2} \right\rfloor \\ 4(G - i), & \left\lceil \frac{G}{2} \right\rceil \le i < G. \end{cases}$$

For G even

$$n_i = \begin{cases} 4i, & i < \left\lfloor \frac{G}{2} \right\rfloor \\ 4i - 2, & i = \frac{G}{2} \\ 4(G - i), & \left\lceil \frac{G}{2} \right\rceil < i < G \\ 1, & i = G. \end{cases}$$

Hence, the maximum delay up to j expansions is given by

$$4 \sum_{i=1}^{j} [\min(i, G - i) - 1], \qquad G \text{ odd or } j < G/2$$

$$4 \sum_{i=1}^{j} [\min(i, G - i) - 1] - 2, \qquad \text{otherwise.}$$

With $j = G$, we get $MD = (G + 1)$, which is as expected.

B. Delay Estimates When e > 1

Notation 2:

- Let $D(k)_{\langle i:j \rangle}$ be the submatrix of $\mathfrak{D}(k)$, the diagonal matrix for the kth layer, comprising of columns i to j, both inclusive.
- Let $EDf_k(i) = [\max t_i]$, where t_i is the number of times processor P_i appears on the active wavefronts of the ith expansion and k is the number of layers. More formally, $EDf_k(i)$ is the number of times P_i appears in $D(1)_{\langle 0:[i+k-1]_N \rangle} \cdots D(j)_{\langle 0:[i+(k-j)]_N \rangle} \cdots D(k)_{\langle 0:[i]_N \rangle}$.
- Let $EDb_k(i) = [\max r_i]$, where r_i is the number of times processor P_i appears on the active wavefronts of the ith expansion, when $i > G/2$. Thus EDb differs from EDf in the sense that it characterizes wavefronts that are cut off by the grid boundaries. More formally, $EDb_k(i)$ is the number of times P_i appears in $D(1)_{\langle [i+k-1]_N : N-1 \rangle} \cdots D(j)_{\langle [i+(k-j)]_N : N-1 \rangle} \cdots D(k)_{\langle [i]_N : N-1 \rangle}$.
- Let D_i^j represent row D_i of the diagonal matrix being repeated j times.

Observation 2: Based on the periodicity of the processor mappings on a diagonal, as given in Lemma 1 and from Theorem 1, it may be observed that the processor-to-cell mapping on the ith wavefront expansion is given by the extended diagonal matrix D', where

$$D' = \begin{bmatrix} D_1^l & D_{1\langle 0:m \rangle} \\ D_2^l & D_{2\langle 0:m \rangle} \\ D_3^l & D_{3\langle 0:m \rangle} \\ D_4^l & D_{4\langle 0:m \rangle} \end{bmatrix}$$

and $i = lN + m$.

Theorem 2: For an N-processor hexagonal machine and a k-layer obstacle-free grid of dimension $G \times G$, where $G \gg N$, the upper bound on the delay factor[5] to route any net is $k((G/N) + 1)$.

Proof: It is obvious that the net that will result in the maximum time is the one spanning from the center of the grid to a corner. This is because this leads to maximum possible conflict in the processor assignment over the four edges of the wavefront expansion.[6]

In the following derivation, we assume that one unit of time corresponds to the time taken by a full-grid machine to perform a single wavefront expansion. Hence, for our machine, we estimate the time taken by counting the multiple cell-to-processor assignments on a wavefront. From Observation 2, the time for the jth wavefront expansion[7] for a single layer, $MT_1(j)$, is given by

$$MT_1(j) = \begin{cases} \left\lfloor \frac{j}{N} \right\rfloor * 4 + EDf_1([j]_N), \\ \qquad 0 < j \le \left\lfloor \frac{G}{2} \right\rfloor \\ \left\lfloor \frac{G - j}{N} \right\rfloor * 4 + EDb_1\left(\left[j - \left\lfloor \frac{G}{2} \right\rfloor \right]_N\right), \\ \qquad \left\lceil \frac{G}{2} \right\rceil \le j < G. \end{cases}$$

Hence, the total time up to the Gth expansion is given by

$$T_1(G) = \sum_{j=1}^{\lfloor G/2 \rfloor} \left(\left\lfloor \frac{j}{N} \right\rfloor * 4 + EDf_1([j]_N) \right) + \sum_{j=\lfloor G/2 \rfloor}^{G-1} \left(\left\lfloor \frac{G - j}{N} \right\rfloor * 4 + EDb_1\left(\left[j - \left\lfloor \frac{G}{2} \right\rfloor \right]_N\right) \right). \tag{1}$$

[5] Delay factor (DF) = time taken by the hexagonal machine/time taken by a full-grid machine.

[6] From Lemma 1, we know that this conflict is minimized over a single edge.

[7] Note that after $G/2$ expansions, the wavefronts are partly cut off by the grid boundaries.

If we let $\lfloor (G/2) \rfloor = aN + b$, then (1) can be reduced to

$$T_1(G) = 8\left[N\frac{a(a-1)}{2} + ab\right] + aM_1 + \sum_{j=1}^{j=b} \left(EDf_1(j) + EDb_1(j)\right) \quad (2)$$

where $M_k = f_k + b_k$ and $f_k = \Sigma_{i=1}^{N-1} EDf_k(i)$ and $b_k = \Sigma_{i=1}^{N-1} EDb_k(i)$.

Note that if b is not negligible with respect to $(a \cdot N)$, then we have to add a correction factor $\delta = aN \cdot EDf_k(b) - 4ab$ to the total time required.

The expressions in the case of more than one layer are also very similar. The total time required to route a net originating from the center of the top layer to a corner at the bottom layer is given by

$$MT_k(j) = \begin{cases} \left\lfloor \dfrac{j-k+1}{N} \right\rfloor * 4k + EDf_k\left([j-k+1]_N\right), \\ \qquad \text{for } k < j \le \left\lfloor \dfrac{G}{2} \right\rfloor + k - 1 \\ \left\lfloor \dfrac{G-(j-k+1)}{N} \right\rfloor * 4 + EDb_k\left(\left[j-k+1-\left\lfloor \dfrac{G}{2} \right\rfloor \right]_N\right), \\ \qquad \text{for } \left\lfloor \dfrac{G}{2} \right\rfloor + k \le j < G + k - 2. \end{cases}$$

On simplification, we find that the total time needed is given by

$$T_k(G) = 8k\left[N\frac{a(a-1)}{2} + ab\right] + aM_k + \sum_{j=1}^{j=b} \left(EDf_k(j) + EDb_k(j)\right). \quad (3)$$

At this point we note that (2) can be obtained from (3) by replacing k with 1. From extensive simulation, M_k is found to be nearly equal to $6kN$. If we also assume that $(G/2) \simeq aN$, then (3) reduces to

$$\begin{aligned} T_k(G) &= 8k \cdot N\frac{a(a-1)}{2} + aM_k \\ &= 4k \cdot [a(a-1)]N + a \cdot 6kN \\ &= 2akN(2a+1) \\ &= kG\left(\frac{G}{N} + 1\right). \end{aligned}$$

Thus the DF is given by

$$\begin{aligned} \text{DF} &= kG\left(\frac{G}{N} + 1\right)\Big/ G \\ &= k\left(\frac{G}{N} + 1\right). \end{aligned} \quad (4)$$

Q.E.D.

Corollary 2: For an N-processor hexagonal machine and a grid of dimension $G \times G$, where $G \gg N$, the DF in routing a net which spans from the center to a corner of the grid is twice as much as that of a net connecting two diagonally opposite corners of the grid, even though the latter net is twice as long.[8]

For a net proceeding from corner to corner, we find that each wavefront consists of only a single segment. For simplicity, we are assuming a single-layer grid. The extensions to the multilayer case is straightforward and is left to the reader. Note that there will be $2G$ wavefront expansions:

$$\begin{aligned} T_1(2G) &= 2\sum_{i=1}^{i=G} \left(\left\lfloor \frac{i}{N} \right\rfloor\right) - \left\lfloor \frac{G}{N} \right\rfloor \\ &\simeq 2Na \cdot (2a-1) + (4b-1)2a \end{aligned}$$

where $\lfloor (G/2) \rfloor = aN + b$.

Thus neglecting b, we get

$$\begin{aligned} \text{DF} &= G\left(\frac{G}{N} - 2\right)\Big/ 2G \\ &= \frac{1}{2}\left(\frac{G}{N} - 2\right). \end{aligned} \quad (5)$$

Hence, from (4) and (5), we have

$$\text{DF(net 1)} = \left(\frac{G}{N} + 1\right) \simeq \frac{G}{N}$$

$$\text{DF(net 2)} = \frac{1}{2}\left(\frac{G}{N} - 2\right) \simeq \frac{G}{2N}.$$

Q.E.D.

Corollary 3: A hexagonal machine of dimension $\Theta(\sqrt{kG})$ can handle a k-layer grid consisting of $\Theta(kG^2)$ cells, at about the same speed as a full-grid machine comprised of $\Theta(kG^2)$ processors.

Proof: From Theorem 2, we know that the delay factor in routing a net is $\Theta(kG/N)$. Also, for a hexagonal mesh, $N = \Theta(e^2)$, where e is the dimension of the mesh. Q.E.D.

Also, a point worth observing is that the only major computations required are for the $2N$ element vectors EDf_k and EDb_k, which take $\Theta(kN^2)$ time at the most.

[8] These two nets represent extreme cases. In practice, we contend that the maximum delay will be less by a factor of 4–5 for most other nets.

V. Chip Implementation Issues

A. Architectural Issues

The hexagonal array machine, as we envision, will consist of a C-wrapped hexagonal mesh of processing elements which are under the control of an array control unit (ACU). The machine operates under an SIMD computer organization wherein each processing element executes the same instruction globally broadcast by the ACU.

The hexagonal array machine will act as a coprocessor, or "routing accelerator," to a workstation or serial mainframe computer. Our design philosophy is oriented towards a compact design suitable to packing in a VLSI chip. Some researchers [13], [21] have advocated general-purpose processing elements made up of commercial chips and microprocessors. Their rationale is that they offer flexibility in employing various variants of the maze algorithm. They further believe that such a machine can be used for design tasks other than routing. Agreeably, this can be quite attractive in an experimental setup. However, the low speedup factors of around 3–4 achieved by the WRM can be attributed precisely to this lack of dedicated hardware, interchip communication overhead, and lack of stress on performance, among others. Hence, we believe that a routing accelerator must be dedicated and hardwired to do its job as fast and efficiently as possible. It is also our feeling that any routing accelerator, at least any in the near future, must considerably outperform its serial version in order to be practically viable. In today's ever-increasing design environment, the routing accelerator also must be capable of handling problem sizes much larger than the physical array size. More importantly, it must be able to address the issue of multilayer routing, not only for PCB's, but also for future IC design. It has been amply demonstrated previously that the hexagonal mapping appropriately answers all of these problems. Hence, we employ this interconnection topology in our goal to come up with a design of a compact, fast machine that can handle problem sizes much larger than the physical size of the processing array.

Simulation studies have indicated that speedup versus mesh-dimension curve tends to level out with the mesh dimension around 8. Therefore, we believe that an eight-dimensional processing array, consisting of 169 processing elements, is ideal for the problem size that can typically be expected.

Fig. 7 gives us an overview of the hexagonal machine while Fig. 8 shows the block diagram of a typical processing element. Thus the two important components of the hexagonal array machine are i) the ACU and ii) the array of interconnected processing elements. The ACU is responsible for the interface with the host computer. To begin with, the host computer would pass on to the ACU information regarding the size of the grid, position of obstacles on the routing surface, if any, and coordinates of the terminals of the various nets. The other functions of the ACU will become evident shortly.

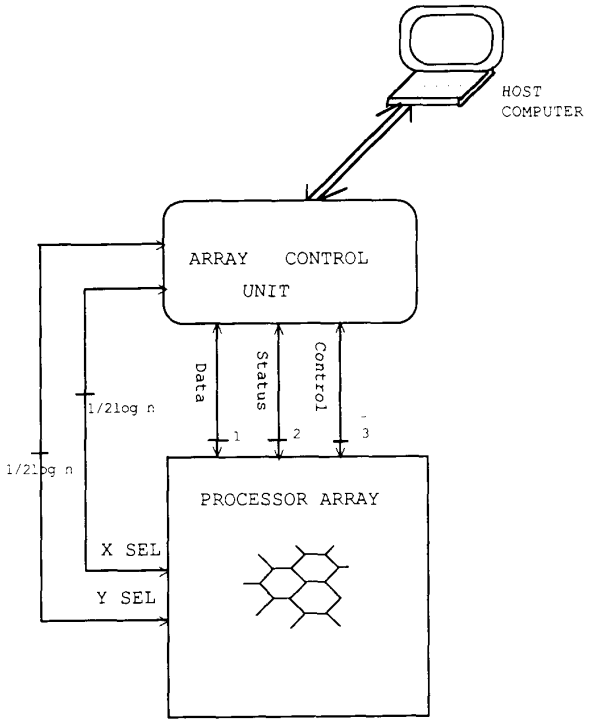


Fig. 7. Overview of the hexagonal array machine.

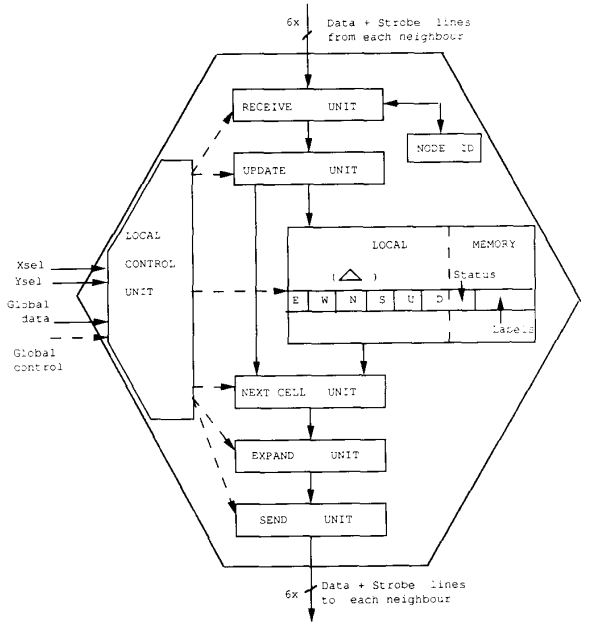


Fig. 8. Block diagram of a single processing element.

The processing array is the actual workhorse of the hexagonal array machine. Each processing element has special-purpose hardware for receiving and propagating labeling information with its six neighbors.

From Fig. 8, it is obvious that the major bottleneck, both in terms of space and performance criteria, is the local memory. The local memory must store pertinent information regarding the various cells that have been mapped onto this processing element. Each entry can be assumed to consist of two parts, namely, the $\langle x, y, \lambda \rangle$ coordinates of the cell (fixed data) and a variable part for maintaining the labeling information for the backtrace phase; status of the cell, i.e., whether it has been expanded already or whether it is a terminal (source or destination) or whether it is blocked, etc.

In a full-grid machine, such as the L-Machine, the information that one processor must send to its neighbor during wavefront expansion can be as simple as a 1-b token. This is possible as each processor is mapped to only a single point on the grid. In any virtual machine, information must also be passed on to the neighbor identifying the cell instance that corresponds to that expansion. For our machine, each processor based on the knowledge of its position in the grid can calculate the $\langle x, y, \lambda \rangle$ coordinates of its neighbors and send it. This scheme, though simple, is also very inefficient. Even for a four-layer grid of dimension 256 by 256, this would entail having to send 18 b each time to each neighbor. More importantly, a processing element upon receiving the token must extract information regarding that cell from its local memory. So, unless the memory is organized as an associative memory, there will be a tremendous overhead in processing.

The alternative would seem to be that each cell pass to its neighbor, during wavefront expansion, the memory address rather than the cell contents. We shall now discuss one scheme which can do precisely that. After the initial mapping using the operator Φ (cf. Definition 1) has been calculated, the ACU generates a new array INDEX, defined as follows:

$$\text{INDEX}[x, y, \lambda] = \{\ i\text{—this is the } i\text{th occurrence of } n \text{ in layer } \lambda \text{ proceeding in a row-major fashion}\ \}$$

where n is the processing element to which cell $\langle x, y, \lambda \rangle$ is mapped.

Each processing element then calculates the difference Δ between its INDEX value and the INDEX values of its neighbor in each direction. In case the grid boundaries are met along any direction, an illegal value, X, is entered, implying the expansion is not possible in that direction. From Lemma 1, it is evident that the maximum absolute value of Δ is going to be $\lceil (G/N) \rceil$. For a 256 by 256 grid and $N = 169$, this works out to be 2. Note that this value is independent of the number of layers in the grid. Hence, the Δ value for each direction can be stored in a 3-b field. This in turns means that the fixed part of each entry would be 18 b wide. The entry for cell $\langle x, y, \lambda \rangle$ is stored in the local memory of the mapped processing element at address $\langle \lambda_k \cdots \lambda_0, b_g \cdots b_0 \rangle$, where $\lambda_k \cdots \lambda_0$ is the binary representation of λ and $b_g \cdots b_0$ is the binary representation of INDEX$[x, y \lambda]$.

Now let m_a be the address of the cell currently being expanded. Then the information passed to the neighboring processing element in direction d is the value $(m_a + \Delta_d)$, where Δ_d is the difference stored at m_a for direction d. For a 256 by 256 four-layer grid, this entails sending only eleven bits of information, as opposed to 18 in the previous case. However, the vital gain is in the fact that the cell information received is in the form of an address. Hence, it results in much speedier retrieval of data from the local memory.

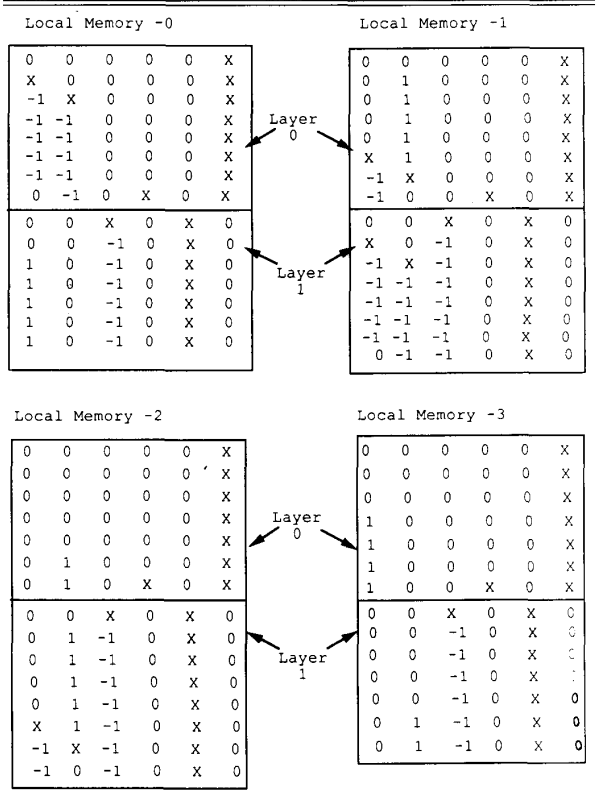
The cost we pay for the new scheme is the additional time spent in generating INDEX, even though this will be more than compensated for by the faster processing. Also, as the information is static for a given mapping, it can be precomputed and loaded in the local memories once and for all. However, the indexes will no longer be consecutive for smaller grids.

The memory contents for the first three processing elements for the mapping shown in Fig. 2 are shown in Table III. The entries marked X indicate that no expansion is possible in those directions because of the grid dimensions.

The *send* and *receive* units in Fig. 8 could be as simple as shift registers with appropriate status flags to indicate if any new data have arrived. The presence of a bank of registers, one per neighbor, ensures complete parallelism in the expansion process. The *update* unit, as the name suggests, would update the pertinent cell status and labeling information. It could store these cells in a stack-like structure for faster retrieval by the *next-cell* unit. Also, the *update* unit informs the local control unit of new data by raising a BSY (busy) line, upon new data arrival, or the DST line, if the cell corresponds to the target. The *next-cell* unit provides the *expansion* unit with one of the possibly several active cells mapped on this processing element. This unit also lowers the BSY line if it finds no more cells remaining to be expanded by the processing element in the given cycle. The *expansion* unit for direction d adds the value Δ_d to the address of the cell to form the message for the neighbor along direction d. This is then sent by the *send* unit, provided the Boolean condition $(\Delta_d \neq X \wedge cell\_state = to\ be\ expanded)$ is satisfied.

The preceding operations are all carried out under the supervision of the local control unit. The ACU broadcasts the commands globally to all the local control units. Also, the ACU can individually access each processing element, using the X and Y select lines, and initialize the processing elements at the start, as to the locations of the initial obstacles and start and end of the nets. During backtrace the processing elements on the net can raise the same lines to indicate the route to the ACU. A point to note is that in this scheme, some processors may be forced to idle if others have multiple assignments on the current wavefront (Procedure *expand_if_busy*). Additional speedup could be achieved if the processors were permitted to continue to expand. This would correspond to an MIMD mode of operation. The cells being expanded during any cycle now would no longer correspond to any par-

TABLE III
CONTENTS OF THE LOCAL MEMORY FOR PROCESSORS 0 TO 3



ticular wavefront. Procedures 1–5 describe the various software routines in pseudocode.

B. Procedures for an SIMD Model of Computation on the Hexagonal Machine

Procedure 1: *PROC_ELEM main loop*
repeat
Process CU command
Execute the appropriate routine, viz.,
EXPAND, *EXPAND_IF_BSY*, or *BACKTRACE*
until *OVER*.

Procedure 2: *EXPAND*
for each R_i **do**
if new data received
Check with cell bank if this is a new cell
if *new*
cnt++
Update the labels for the cell
Set DST flag if target reached
Expand one unexpanded cell by sending message to the applicable neighbors
cnt−−
if *cnt* > 0
set BSY flag ON
else
reset BSY flag
end.

Procedure 3: *EXPAND_IF_BSY*
if BSY flag is ON
Perform Algorithm *EXPAND*
else
skip
end.

Procedure 4: *BACKTRACE*
if new data received or if DST flag set
if neighbor on *cur_dir* has correct label
Next = *cur_dir*
else if neighbor in direction *n* on same layer has correct label
Next = *n*
else if neighbor in direction *n* on adjacent layer has correct label
Next = *n*
cur_dir = *Next*
Send message to neighbor in direction *Next*
end.

Procedure 5: *CONTROL_UNIT main loop*
Compute the mapping of cells to processors
Load mapping information onto the cell banks of the processors
repeat
Broadcast the source and target for this net
EXPAND
Read DST vector and go to next net if set or if route not possible
while BSY vector ≠ 0
EXPAND_IF_BSY
until all *nets* over.

VI. PERFORMANCE ANALYSIS

A simulation has been made based on the preceding architecture and the multilayered routing algorithm proposed. The simulation program has been coded in C++.

We ran several simulations, varying different parameters, such as the hexagonal mesh dimension, the grid size, and the number of layers to be routed. Thus we were able to investigate their impact on the routing time. We varied the mesh dimension from 2 to 32 (i.e., from seven processing elements to 2977 for the 32-dimensional mesh). We performed sample routing on grids of sizes ranging from 16 by 16 to 1024 by 1024 and comprised of 1–8 layers. All these routing problems were considered on an empty grid (i.e., with no previous blockages). A net originating from the center of the grid and proceeding to a corner, along with another running between two diagonally opposite corners, were always considered. Our motivation here was simply to get a feel for the order of delays that we may expect from a hexagonally interconnected machine. The aforementioned two nets, as we have already shown, require the most time to route.

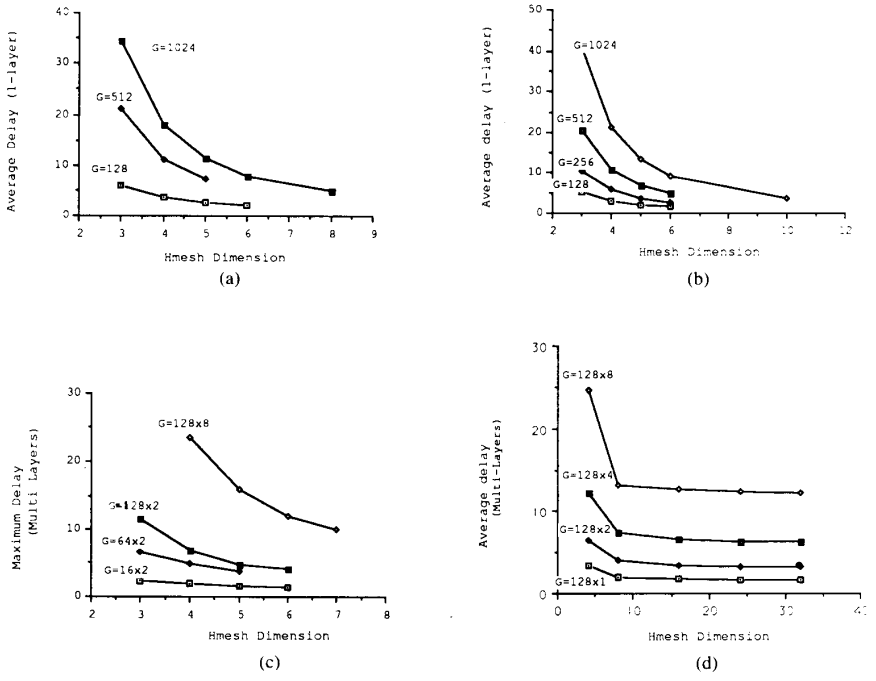


Fig. 9. Simulation and analytical results of routing time on the hexagonal array machine. (a) Simulation results (one layer). (b) Analytical results (one layer). (c) Simulation results (multilayers). (d) Analytical results (multilayers).

TABLE IV
ANALYTICAL RESULTS OF DF FOR A K-LAYER 128 BY 128 GRID

Mesh Dim	No. of PEs	$128^2 \times 1$		$128^2 \times 2$		$128^2 \times 4$		$128^2 \times 8$	
		M_1	DF_1	M_2	DF_2	M_3	DF_3	M_4	DF_4
2	7	31	13.8691	73	28.3184	163	57.5332	392	118.5470
4	37	211	3.3750	400	6.5273	741	12.2754	1537	24.6211
8	169	1021	2.0039	1967	4.0137	3677	7.4297	6342	13.1484
16	721	4447	1.7578	8836	3.4395	16967	6.6621	30838	12.8145
24	1657	10273	1.6582	20697	3.2402	40452	6.3633	75167	12.4043
32	2977	18499	1.5820	37549	3.1816	74085	6.3457	140376	12.3281

The graphs in Fig. 9 show the timing characteristics of the hexagonal array machine as obtained from simulation. Also shown are the upper bounds on the time as projected by the analytical model of Section IV. Both single-layer and multiple-layer routing results are plotted. Each plot is of the time needed to route *versus* the dimension of the mesh used. The unit for the y axis is taken as the time taken by a processor to perform one expansion. Table IV shows the analytical results for M_k and the DF, which are as expressed in Section IV.

It is seen that the graphs are all exponential in nature and of the form $y = b * 10^{-cx}$, where b and c are positive numbers. As is intuitively evident, the graphs show that for a very small dimension, such as 2 or 3, there are many multiple assigments of cells to the processing elements per wave expansion. This does not mean a lack of parallel processing. It only implies that since the same work is to be handled by fewer processors, the overall time required to complete the routing is more. As the number of processing elements available increases, we need lesser and lesser time to perform the same routine. Note that the quality of routing obtained is not dependent on the number of processors used. It is determined by the routing algorithm, like the cost functions employed during wave expansion or the manner in which backtracing is performed. After a mesh dimension of around 8, all of these curves seem to flatten out, in the sense that increasing the number of processors available does not provide any significant reduction in routing time. Hence, we feel that a practical implementation which would be required to tackle problems of these specifications could ideally be built as an eight-dimensional mesh comprised of 169 processing elements. In this way, we will be able to do routing nearly as fast as a full-grid implementation made up

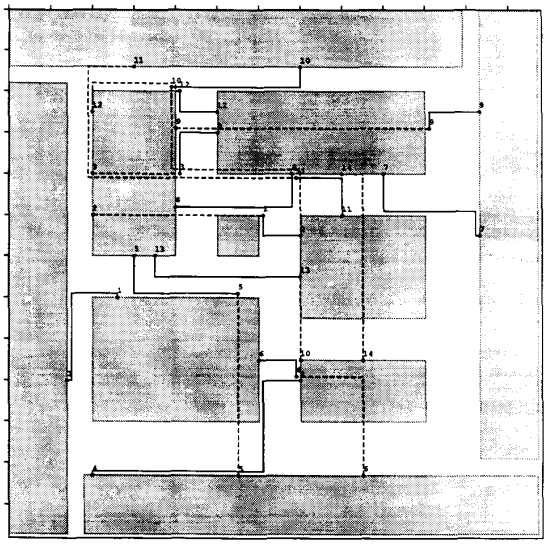


Fig. 10. Sample two-layer routing-example.

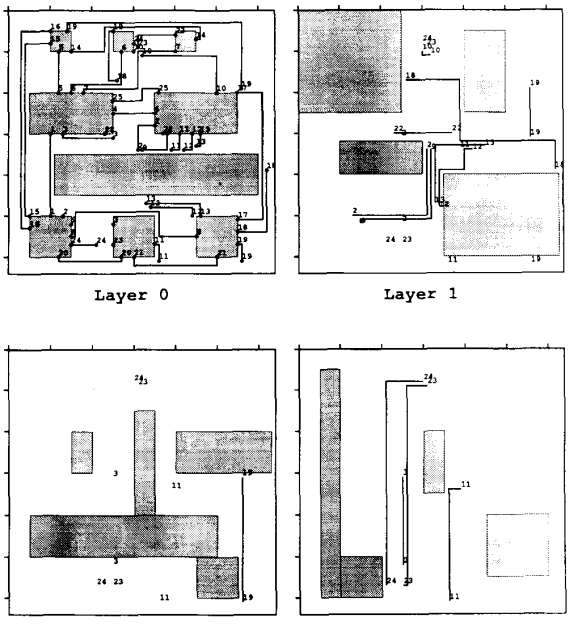


Fig. 11. Sample four-layer PCB routing example.

of thousands of processing elements. Another conclusion that we were able to draw was that the time required to route for most nets is about 0.65–0.85 times the upper bound on the delay, as predicted by Theorem 2. Thus this measure of the delay can be very simply used for evaluating the suitability of a mesh of a certain dimension for solving a given class of problems.

The rest of the section shows the routing produced for two, more complex problems considered. Fig. 10 shows the layout obtained for a sample problem involving 14 nets on a two-layer grid of size 128 by 128. All the blocks and terminals of all nets lie on the top layer only. Fig. 11 shows the initial blockages and the wiring obtained for a four-layer PCB problem. In this case, 25 nets were successfully routed. Table V summarizes the performance results for the two problems for meshes of dimensions 4 and 8, respectively.

The results are very promising indeed. For a problem similar to the two-layer case, Suzuki *et al.* [21] had obtained an average of 3.4 cells assigned per processor (total of 64 processors) over each wavefront. This is nearly twice the value that we get with a five-dimensional mesh (61 processing elements). We contend that this is a direct reflection of the superiority of our mapping, viz., the same processor appears at every N processors along any direction, instead of the smaller periodicity of $N/2$ or less obtained by others. Also, our solution is not unnecessarily constrained by horizontal–vertical restrictions on wire directions. This offers scope for a greater number of wires being routed.

Table VI shows the netlist used for the two-layer sample problem, whose solution is given in Fig. 10. Average MAF refers to the multiple assignment factor of cells to processing elements on a given wavefront average over all wavefront expansions. Total time gives an estimate of the amount by which our machine might be slowed down as compared to a full-grid implementation. It is based on an SIMD mode of computation wherein some processing elements may remain idle (similar to masking) during the period when others are handling multiple expansions for that particular wavefront. Avg PrUt refers to the average processor utilization percentage.

Table VII shows the netlist for the four-layer PCB example of Fig. 11. All columns have the same implications as before.

VII. Reconfigurability and Fault-Tolerance Issues

A. Reconfiguration

The ability of the hexagonal array machine to perform multilayer routing stems from the fact that each processing element is directly connected to six other neighbors. Obviously, this implies an increase in the total number of links required. In fact, for an n node machine, a square mesh topology requires $2n$ links as compared to the $3n$ links required by the hexagonal machine. However, it is now possible to perform multilayer routing very efficiently. Currently, most routers tackle a multilayer problem one layer at a time. This necessitates the onerous task of finding out the optimal point for the vias, to optimize the routing criteria as well as to maximize the connectivity ratio. These routers, therefore, adopt heuristic measures to simplify the problem, as a result of which, we may frequently obtain unacceptable results. However, for two-layer horizontal-and-vertical routing, a four-neighbor interconnection topology may suffice. For this purpose, it

TABLE V
PERFORMANCE RESULTS ON THE HEXAGONAL ARRAY MACHINE

Grid Size	No. of Layers	Mesh Dim.	No. of Nets	Total Wire Length	Avg. MAF	Total Time	Avg. Pr. Util.	Completion Ratio
128 × 128	2	4	14	877	2.9872	5.0735	50	100%
128 × 128	2	8	14	877	1.3063	2.8302	20	100%
64 × 64	4	4	25	1048	2.5573	4.7643	40	100%
64 × 64	4	8	25	1048	1.2539	2.6411	15	100%

TABLE VI
NETLIST AND PERFORMANCE DATA FOR SAMPLE TWO-LAYER ROUTING PROBLEM

Net	Source		Target		Wire Length	No. of Vias	No. of Bends	Avg. MAF		Total Time		Avg. Pr. Ut.	
	x	y	x	y				4D	8D	4D	8D	4D	8D
1	14	90	26	70	34	0	3	2.1930	1.2789	4.1471	2.6471	42	14
2	20	50	70	55	57	2	1	3.2904	1.3373	5.2456	2.8421	55	22
3	20	40	50	30	42	2	1	2.6543	1.3175	4.6191	3.0238	48	16
4	20	113	70	90	73	0	3	2.5556	1.1665	4.1644	2.2055	56	24
5	55	113	30	60	80	2	1	3.6429	1.4512	5.8000	3.5750	57	20
6	85	113	60	85	55	2	2	2.9429	1.3024	5.0546	2.6546	52	22
7	113	55	90	40	38	0	3	2.4581	1.2258	4.2368	2.3947	49	19
8	68	40	40	48	36	0	1	2.7963	1.2779	4.6667	2.6944	49	19
9	40	29	113	25	79	2	1	3.7639	1.4903	6.3544	3.6962	53	20
10	70	85	70	14	135	2	3	4.1843	1.4472	6.7704	3.5259	58	24
11	30	14	80	50	110	2	3	2.9041	1.2473	5.0000	2.6636	52	21
12	20	25	50	25	46	2	4	2.1899	1.1811	4.3696	2.3261	42	17
13	35	60	70	65	40	0	1	2.6402	1.2038	4.8500	2.4500	45	19
14	85	85	80	40	52	2	1	3.6048	1.3611	5.7500	2.9231	54	23

TABLE VII
NETLIST AND PERFORMANCE DATA FOR SAMPLE FOUR-LAYER PCB ROUTING PROBLEM

Net	Source		Target		Wire Length	No. of Vias	No. of Bends	Avg. MAF		Total Time		Avg. Pr. Ut	
	x	y	x	y				$4d$	$8d$	$4d$	$8d$	$4d$	$8d$
1	10	50	10	30	20	0	0	2.3827	1.2001	4.5500	2.3000	40	18
2	13	50	35	28	46	2	2	3.2050	1.3398	6.2609	3.0435	45	20
3	25	52	13	30	40	6	1	3.5517	1.4173	6.7750	3.4250	47	20
4	25	25	35	25	10	0	0	1.4895	1.0886	2.7000	1.7000	31	11
5	12	20	12	10	10	0	0	1.0427	1.0000	1.3000	1.0000	19	5
6	15	20	27	10	22	0	2	1.3010	1.0433	2.0909	1.3636	26	8
7	18	19	40	10	31	0	2	2.5292	1.2163	4.0968	2.4516	37	13
8	15	55	45	55	42	0	4	2.8576	1.2596	5.4524	2.7857	47	20
9	15	52	38	30	47	2	3	3.1545	1.3338	5.9149	2.9787	47	20
10	30	10	50	20	32	2	2	3.4796	1.4337	6.2813	3.4688	49	19
11	35	57	41	30	47	6	3	2.5776	1.2290	4.5745	2.6170	45	17
12	44	50	44	30	42	2	5	3.0325	1.3446	5.2381	2.9286	38	14
13	46	50	46	30	48	2	3	3.1453	1.3235	5.1875	2.7083	36	15
14	45	7	15	10	41	0	4	3.0886	1.3088	6.0732	3.1219	43	18
15	5	50	10	8	49	0	2	2.7349	1.2993	5.6122	3.2449	44	16
16	5	53	10	5	57	0	2	2.7076	1.3271	5.3684	3.0351	46	17
17	55	51	55	20	43	0	2	2.1820	1.1427	4.2326	2.3256	36	14
18	55	54	24	5	96	2	5	2.9367	1.2719	5.5417	2.8438	45	19
19	55	57	14	5	111	4	3	2.6883	1.4638	5.0360	3.1441	46	15
20	12	60	27	60	17	0	2	1.8967	1.2126	3.9412	2.6471	34	11
21	30	60	50	60	24	0	2	1.9835	1.1372	3.7917	2.2083	43	16
22	23	30	37	30	16	2	0	1.8767	1.1365	3.3750	2.1875	41	13
23	25	57	40	6	72	6	3	3.3784	1.3911	6.5139	3.4306	48	19
24	15	57	30	8	70	6	1	3.0558	1.2962	6.0000	3.0000	47	20
25	25	22	36	20	15	0	3	1.6553	1.1296	3.2000	2.0667	33	11

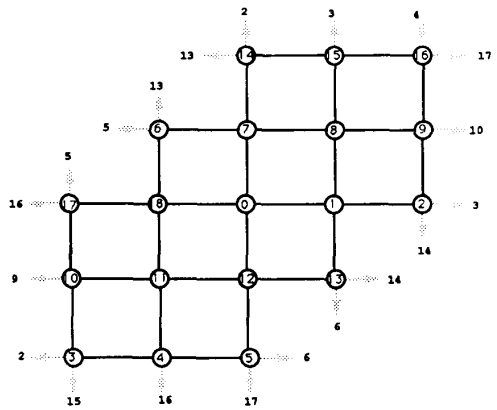


Fig. 12. Four-dimensional hexagonal array with diagonals links removed.

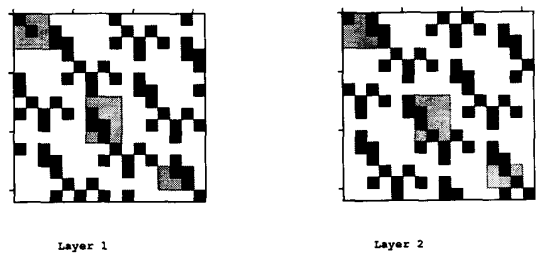


Fig. 13. Additional blockages caused by faulty processing elements.

TABLE VIII
EFFECTS OF FAULTY LINKS ON ROUTING

# of Faulty Links	Average Time	Total Wirelength	Average Via per Net	Average Bends per Net	Average Pr. Util.
0	5.0735	877	1.2857	2.0000	50
2	5.1231	879	1.4286	2.3571	49
3	5.2181	823	1.4286	2.2143	49
4	5.1558	877	1.2857	2.4286	50
5	5.2059	861	1.7143	3.4286	49
6	5.1369	877	1.2857	2.4286	49
7	5.2001	879	1.2857	2.5000	49
8	5.3131	829	1.5714	2.4286	48
9	5.2753	837	1.5714	2.9286	49
10	5.4035	797[†]	1.3846	3.8462	49

[†] 1 net could not be routed in this case.

is possible to simplify the basic hexagonal mesh topology by deleting the two additional links of each processing element. One way to achieve this is by removing all the links along one of the three main directions. Fig. 12 shows the modified architecture for a four-dimensional mesh, where all the links along the 60-deg direction have been removed. Since the interprocessor cycle period is still n for an n-node mesh, this mapping would result in fewer conflicts in processor assignment than a comparable square-mesh or toroidal interconnection.

B. Fault Tolerance

Can routing machines tolerate some amount of failure in the links or processing elements? This is a very important question, especially for WSI or VLSI technology, where such faults could arise due to imperfections in the manufacturing process. Unfortunately, this issue is often unaddressed. What then are the ramifications of faulty links and processors on routing on the hexagonal array machine? Clearly these faulty elements will hinder wavefront expansion and backtracing. In fact, a faulty processing element can be modeled as a case where all the six incoming links are treated as being faulty.

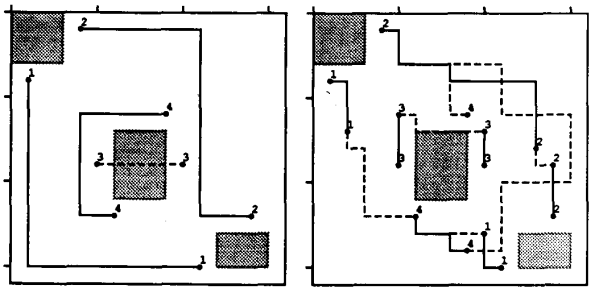
From the point of view of grid layout, these faults, however, can be conceptualized as creating additional obstacles on the grid surface. The darker squares in Fig. 13 show these additional blockages caused by ten out of the 37 processing elements (27%) becoming faulty. Consequently, 138 additional cells (26.75%) get blocked. In spite of this high failure rate, it is heartening to note that all wires could still be routed for a sample problem as described in Section VII-B-2).

1) Faulty Links: We took the routing problem of Table VI and randomly assigned faulty links. This information is easily handled by setting the appropriate Δ entries in the local memory of the two processing elements connected by the faulty link to the illegal X value. Table VIII summarizes the results obtained.

We observe that the time needed to route, as well as that for the processor utilization, remains practically the same. The quality of routing in terms of the number of bends and number of vias used per net deteriorates slightly. An interesting observation is that in some instances with faulty links, there is a reduction in the total wire length needed (chiefly for nets 10 and 11). There are a couple of reasons for this. Firstly, we employ a very simple backtracer which favors a reduction in the number of bends and vias to an increase in the wire length. Secondly, it is another indicator to the effect of net ordering on performance. Our conclusion, therefore, is that a small amount of link failure, say about 5% (in this instance, 9%), will not be catastrophic to the hexagonal machine. Part of this assurance is based on the additional 2 deg of freedom available to each processing element for expansion.

2) Faulty Processors: To study the effect of processor failure on routing, we considered a grid corresponding to the mapping shown in Fig. 2. Our objective was to route four nets in the presence of 0–10 faulty processing elements, out of a total of 37. Although the processing elements were randomly set faulty, we ensured that none of the source or target cells were mapped onto them. A faulty processing element can be represented either by setting the appropriate six Δ entries in the local memories of the six neighbors to an illegal value or by considering all cells mapped to it as being blocked. The results are summarized in Table IX.

Our conclusion here, also is that the hexagonal machine can survive a large amount of processor failure, but at the



(a) (b)

Fig. 14. Wiring results in the presence of faulty processors. —— Layer 1; ---- Layer 2.

TABLE IX
EFFECTS OF FAULTY PROCESSING ELEMENTS ON ROUTING

# of Faulty Processors	Total Wirelength	Average Time	Avg. Pr. Util.	Avg. of Vias per Net	Avg. of Bends per Net
0	62	2.131	26	0.50	1.25
1	64	2.0873	27	0.50	1.50
2	64	2.0976	24	0.50	2.00
3	62	2.1245	23	0.50	2.75
4	66	2.0461	22	1.00	1.75
5	66	1.9603	24	0.50	4.00
6	90	1.8454	19	1.50	5.00
7	74	2.0227	20	1.00	3.75
8	66	2.1460	18	1.00	3.25
10	94	1.6962	16	2.00	6.50

cost of additional vias and bends. Fig. 14(a) shows the wiring when no processor is faulty and Fig. 14(b) shows the wiring when ten of the 37 processing elements are faulty. All the blocks shown are on the top layer, as are the terminals of all nets. Also note that net 1 running on layer 2 is partly covered by net 4 running on layer 1. So is net 4 partially covered by net 2. Although the routing in this case is of poor quality, it is worth noting that this is an instance where nearly 27% of the processing elements are faulty.

VIII. CONCLUSION

The popularity of hardware routers can be attributed to their utmost importance in any automated design environment. Powerful processors are required to achieve faster turnaround times even though design requirements are steadily increasing. Full-grid designs like the L-machine are no longer possible. The question of net ordering in achieving 100% connectivity with a minimum (or absence) of overflow nets continues to be a vexing issue for automatic routers. Some amount of rip-up and rerouting will inevitably be required in the absence of a scheme for determining the optimal net order *a priori*. This rerouting phase often accounts for the bulk of the total time required. Allowing for multiple layers is one approach to reduce the number of overflow nets and thereby the rerouting overheads. Concurrent multiple-layer routing also minimizes the number of vias introduced, thereby improving reliability of the circuit. More nets can be connected because of the additional routing space available. However, existing accelerators based on square-mesh or torus topologies are not successful when it comes to multiple layers. We believe that this is a restriction of any topology wherein each processing element has only four nearest neighbors.

In this paper, we have investigated the hexagonal mesh architecture for the physical implementation of the Lee algorithm. We have shown the high promise of such a machine in handling routing on single as well as on multiple layers. The mapping, corresponding to a C-wrapped hexagonal interconnection of N processing elements, results in an interprocessor cycle length of N. This is much superior to the $N/2$ results obtained by other researchers [13], [21]. Consequently, fewer conflicts arise during wavefront expansion and a good quality routing can be achieved in a much shorter period.

We have shown that a mesh of dimension $\sqrt{Gk}$ can do routing on k-layer grids with kG^2 grid points at speeds comparable to the full grid machine. For example, we estimate that a four-dimensional hex mesh, with 37 processors, will take about $((256/37) + 1) \cdot 0.6 \sim 5$ times more than the *full-grid* implementation having 65 536 processors. We have discussed some of the major issues involved in physically implementing such a machine. An SIMD-type model, along with a unique labeling scheme to lower the local memory requirements, has been proposed. We have also shown the ability of the hexagonal mesh to withstand considerable link and processing element failure and still produce acceptable wiring.

ACKNOWLEDGMENT

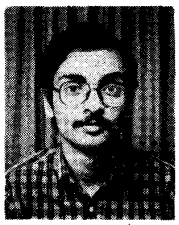
The authors would like to sincerely thank Prof. Kang G. Shin, Director of the Real-Time Computing Laboratory for his stimulating discussions about the hexagonal mesh with wraparound topology.

REFERENCES

[1] S. Akers, "A modification of Lee's path connection algorithm," *IEEE Trans. Electron. Comput.*, pp. 97-98, Feb. 1967.
[2] T. Blank, "A survey of hardware accelerators used in CAD," *IEEE Design Test Comput.*, pp. 21-39, Aug. 1984.
[3] M. A. Breuer and K. Shamsa, "A hardware router," *J. Digital Syst.*, vol. 4, no. 4, pp. 393-408, 1980.
[4] M. Burstein and R. Pelavin, "Hierarchical channel router," in *Proc. Design Automation Conf.*, 1983, pp. 591-596.
[5] J. W. Dolter, P. Ramanathan, and K. G. Shin, "A microprogrammable VLSI routing controller for HARTS," Tech. Rep. CSE-TR-12-89, Univ. of Michigan, Dep. of EECS, 1989.
[6] J. M. Geyer, "Connection routing algorithms for printed circuit boards," *IEEE Trans. Circuit Theory*, vol. CT-18, pp. 95-100, 1971.
[7] F. O. Hadlock, "A shortest path algorithm for grid graphs," *Networks*, vol. 7, pp. 323-334, 1977.
[8] S. J. Hong, and R. Nair, "Wire-routing machines—New tools for VLSI physical design," *Proc. IEEE*, vol. 71, pp. 57-65, Jan. 1983.
[9] Hu and Shing, "The alpha-beta routing," in *VLSI Circuit Layout: Theory and Design*, T.C. Hu and E. S. Kuh, Eds. New York: IEEE Press, 1985, pp. 139-144.
[10] D. E. Knuth, *The Art of Computer Programming, 2nd Ed.* Reading, MA: Addison Wesley, 1973, vol. 2.

[11] C. Y. Lee, "An algorithm for path connections and its applications," *IRE Trans. Elec. Comput*, pp. 346-365, 1961.
[12] A. J. Martin, "The torus: An exercise in constructing a processing surface," in *Proc. Second CalTech Conf. on VLSI*, Jan. 1981, pp. 527-537.
[13] R. Nair, S. J. Hong, S. Liter, and R. Villani, "Global wiring on a wire routing machine," in *Proc. Design Automation Conf.*, June 1982, pp. 224-231.
[14] T. Ohtsuki, "Maze running and line-search algorithms," in *Layout Design and Verification*. Amsterdam, The Netherlands North-Holland, 1986, chap. 3, pp. 99-132.
[15] R. Y. Pinter, "River routing: Methodology and analysis," in *Proc. Third CalTech Conf. VLSI*, Mar. 1983, pp. 141-163.
[16] B. Preas and M. Lorenzetti, *Physical Design Automation of VLSI Systems*. Menlo Park, CA: Benjamin/Cumming, 1988.
[17] R. A. Rutenbar, "A class of cellular computer architectures to support physical design automation," Ph.D. dissertation, Univ. of Michigan, 1984.
[18] S. Sahni and Y. Won, "A hardware accelerator for maze routing," in *Proc. Design Automat. Conf.*, 1987, pp. 800-806.
[19] S. Sastry and R. Kumar, "Parallel placement on reduced array architecture," in *Design Automat. Conf.*, 1988.
[20] K. S. Stevens, "The communication framework for a distributed ensemble architecture," AI Tech. Rep. 47, Schlumberger Res. Lab., Feb. 1986.
[21] K. Suzuki, Y. Matsunaga, M. Tachibana, and T. Ohtsuki, "A hardware maze router with application to interactive rip-up and reroute," *IEEE Trans. Computer-Aided Design*, pp. 466-476, Oct. 1986.
[22] T. Watanabe, H. Kitazawa, and Y. Sugiyama, "A parallel adaptable routing algorithm, and its implementation on a two dimensional array processor," *IEEE Trans. Computer-Aided Design*, vol. CAD-6, pp. 241-250, Mar. 1987.
[23] T. Yoshimura and E. S. Kuh, "Efficient algorithms for channel routing," *IEEE Trans. Computer-Aided Design*, pp. 180-190, Jan. 1982.

*



R. Venkateswaran received the B.Tech. degree in computer science from the Indian Institute of Technology, Bombay, in 1988. Currently, he is working towards the Ph.D. degree at the University of Michigan.

His areas of interest include physical design, fault tolerance, neural networks, and high-level synthesis.

*

Pinaki Mazumder (S'84-M'87) for a photograph and biography, please see page 511 of the May 1990 issue of this TRANSACTIONS.