F. Dehne and N. Santoro, "Optimal VLSI dictionary machines on meshes", in Proc. *Int. Conf. on Parallel Processing*, St. Carles, Ill., 1987, pp. 832-840.

OPTIMAL VLSI DICTIONARY MACHINES ON MESHES

FRANK DEHNE and NICOLA SANTORO
Distributed Computing Group
Carleton University
Ottawa, K1S 5B6
Canada

Abstract

In this paper we contend that a systolic mesh is an efficient architecture for implementing dictionaries. Several implementations are considered and their complexity analysed, the difference being in the trade-off offered between performance (i.e., response time, period) and link complexity (i.e., linear, quadrilateral, hexagonal, octagonal).

The proposed VLSI dictionary machines consist of two structures, a *snake* and a *broadcast net*, which are both embedded in and operate simultaneously on the same mesh. The problems arising from having these two concurrent structures on the mesh are discussed and solutions presented. All proposed implementations are asymptotically optimal. In particular, all operations (Insert, Delete, Search, Extract Min, and Find Min) can be performed with O(1) period, and the response time for Search and Find Min operations is $O(\sqrt{n})$ and O(1), respectively. Furthermore, the proposed solutions are capable of handling duplicate insertions and redundant deletions. The difference in performance between the proposed machines rests solely in the size of the constant, which depend on the simultaneous embedding of the two structures in the mesh; in particular the best performance is achieved by using a novel disjoint embedding.

1. Introduction

A *dictionary* is a basic data type which allows for update and retrivial operations. Because of its general and fundamental capabilities, several researchers have studied the problem of designing special-purpose VLSI chips implementing dictionaries or more restricted types (e.g., priority queue).

Almost all proposed implementations are based on the *complete binary tree* structure: Atallah & Kosaraju [AK85], Leiserson [L79], Ottman *et al.* [ORS82], Somani and Agarwal [SA85], and more recently Chang *et al.* [CCIR86]. The reason for the popularity of this structure rests in the fact that in a complete binary tree of n nodes, the maximum distance between any two nodes is O(log n); thus, from a *theoretical* viewpoint, the time to perform most of the dictionary operations when using such a structure is also O(log n).

Unfortunately, the *practical* layout of a binary tree leads to the presence of "long" wires (of lenght at least $\Omega((n/\log n)^{1/2})$ [PRS81]); depending on conditions, models ranging from O(1) to $O(\text{lenght}^2)$ may be appropiate for the delay to traverse such a wire [MC80], [CM81a]. Furthermore, if the tree is embedded in a systolic mesh [BC86, GKS82] or in a programmable grid [S82] of size $\sqrt{n} \times \sqrt{n}$, the maximum delay is at least $\Omega(\sqrt{n})$ since the "long" wires are implemented by actual processors. In other words, the advantages of an O(log n) delay, which was the main reason for choosing binary trees, is often lost in the layout process, and it is definitely lost if the tree structure is to be embedded in a mesh.

Also for these reasons, research on dictionary machines has focused on using trees with fewer but more complex nodes (e.g., Carey and Thompson [CT84], Fisher [F84]) as well as on employing different structures (e.g., the cube-class architectures studied by Schwartz and Loui [SL87]). In this paper, this line of research is followed and the *systolic mesh* (or *systolic array*) is considered. The use of a mesh as a dictionary machine was first proposed by Chazelle and Monier [CM81b] whose solution, however, does not handle duplicate insertions and redundant deletions.

Unlike trees, theoretical bounds on systolic meshes always correspond directly to the actual performance of the VLSI implementation. Because of its simplicity and its inherent correspondence with VLSI chips, the mesh has been the natural choice, as a VLSI machine, for a variety of applications : geometric problems (e.g., [MS84]), numeric computations (e.g., [KL80]) etc. .

In this paper, it is shown that systolic meshes are an efficient architecture for implementing dictionaries (even with added priority queue operations) which can handle the problem of duplicate insertions and redundant deletions. The proposed VLSI dictionary machines consist of two structures, a *snake* and a *broadcast net*, which are both embedded in the same mesh and operate on it simultaneously. The problems encountered in designing such an architecture are studied. Several implementations and their complexity are analysed, the difference being in the trade-off between performance (i.e., latency and period) and link complexity (i.e., linear, quadrilateral, hexagonal, octagonal). It is shown that all the proposed implementations have the following performance: all operations (Insert, Delete, Search, Extract Min, Find Min) can be performed with O(1) period, and the response time for Search and Find Min is $O(\sqrt{n})$ and O(1), respectively; since in a mesh the maximum distance is at least $\Omega(\sqrt{n})$, it follows that the proposed solutions are asymptotically optimal.

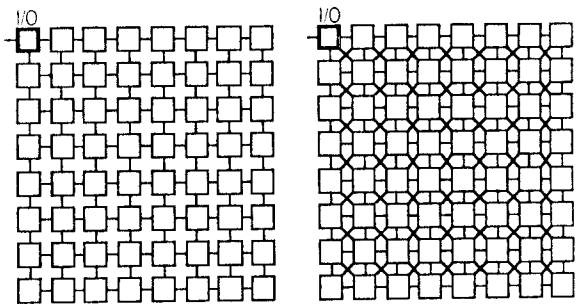
The paper is organized as follows. In the next section, basic definitions and terminology are introduced. In section 3, the two basic structures of the proposed design are described and the problem arising from having them both operating simultaneously on the same mesh are solved. Finally, in section 4, several dictionary machines corrsponding to different layouts of the two structures are described and their complexity analyzed.

2. Mesh Capabilities and Dictionary Performance

The underlying architecture is the mesh of processors: a set of n syncronized processing elements (PEs) arranged on a $\sqrt{n} \times \sqrt{n}$ grid, with each processor being connect by bidirectional communication links to its direct neighbors. In figure 1, two arrangements are shown where each PE is connected to its four (quadrilateral) or eight (octagonal) direct neighbors, respectively.

Each processor has a constant number of registers and within one *time unit* it can simultaneously send an output to and receive an input from each of its communication links.

For simplicity it is assumed that the upper left PE serves as the I/O port to accept requests and deliver answers; this apperant restriction can be easiliy lifted without offending the results presented here.



(a) (b)

Figure 1: quadrilateral (a) and octagonal (b) mesh of processors

A *dictionary* is a device which stores a set of *records* and provides a set of operations on these records, called *dictionary operations*. With each record there is associated a unique *key* k from a totally ordered set K. For simplicity, the record whose associated key is k, will be denoted by *record k*. The standard dictionary operations are

(1) Insert(k) : insert record k in the dictionary
(2) Delete(k) : delete record k from the dictionary
(3) Search(k) : retrieve record k if currently stored or return a negative response otherwise.

In addition to the above operations, the following *priority queue operations* might also be supported:

(4) Find Min : retrieve the current minimum record
(5)Extract Min : delete the current minimum record

Note that only some of these operations require an answer be produced: Search(k) and FindMin; these operations will be called *query operations*.

A *VLSI dictionary machine* receives a sequence of instructions (i.e., requests to perform dictionary or priority queue operations), executes the corresponding operations in a *pipelined* fashion, and in the case of query operations reports the responses via the I/O port. The *latency* of the machine for a query instruction is the

number of time units elapsed from the time the query arrived to the time a response is produced at the I/O port. The *delay* of an instruction is the mimimum number of time units, after the instruction has arrived to the dictionary, necessary before a next instruction can be sent to the machine; the *period* of the machine is defined the maximum of all instruction delays.

The proposed implementations of a dictionary and priority queue of size n on a mesh of size $\sqrt{n} \times \sqrt{n}$ have the asymptotically optimal performance listed in Table 1.

Operation	Performance
Insert	O(1) delay
Delete	O(1) delay
Search	O(1) delay, $O(\sqrt{n})$ latency
Find Min.	O(1) delay, O(1) latency
Extract Min.	O(1) delay

Table 1. Performance of Proposed Dictionary Machines

3. General Structure of the Dictionary Machine

The proposed VLSI dictionary machine consists of two logical structures, a *snake* and a *broadcast net*, which are both embedded in the same physical mesh structure and operate on it simultaneously. In this section, the snake and broadcast net are described separately; and then the problems arising from having both structures operating simultaneously on the same mesh are discussed and solved.

3.1. The Snake

The snake is basically a linear array implementation of a systolic priority queue (e.g., [L79], [KL84]). The records are stored in increasing sorted order without gaps starting from the I/O port, see figure 2a.

The snake is embedded in the mesh such that it contains each mesh processor exactly once, and the leftmost I/O processor which contains the minimum element is coincident with the upper left I/O processor of the mesh (a possible embedding is shown in figure 2b).

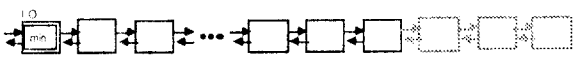


Figure 2a: Priority Queue Structure

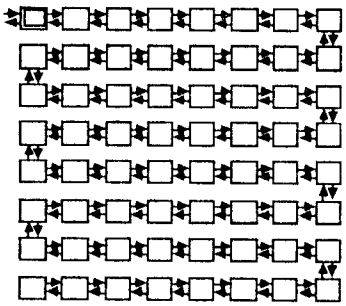


Figure 2b: A Snake Embedding in the Mesh

Obviously, if only the snake is operated on the mesh, the latency for FindMin queries is one time unit.

An insertion instruction Insert(k) is processed in such a way that a message "Insert(k)" is shifted through the snake until it encounters the first PE storing a record k' with $k' \geq k$. Record k' is replaced by record k and, in case $k' > k$, a message "Insert(k')" continues to travel through the snake in the same fashion. When a message "Insert(k^*)" arrives at the end of the snake (last PE that contains a record) record k^* is stored in the next (currently empty) PE. Note, that Insert(k) instructions are processed in such a way that duplicates are avoided.

Upon a Delete(k) instruction, a message "Insert(k)" is shifted through the snake until it encounters the PE P' storing record k (if any). Record k is deleted in P' and a "shift leftwards" message is sent to all subsequent PEs resulting in all subsequent records being shifted one PE leftwards (towards the I/O port).

To ensure that no subsequent "Insert(.)" or "Delete(.)" message encounters a gap, a delay of 2 time units is necessary after each Delete(k) instruction.

The same holds for an ExtractMin instruction which results in the record previously stored in the I/O PE being deleted and all records in subsequent PE being shifted one PE leftwards.

3.2. The Broadcast Net

The broadcast net is a systolic structure whose function will be the handling of Search instructions. Its topology consists of two acyclic directed graphs, G_1 and G_2, where

(a) G_1 is a spanning graph of the mesh with only one *source*;

(b) G_2 is a subgraph of the mesh with only one *sink* and whose *sources* coincide with the *sinks* of G_1.

(c) The I/O processor of the mesh is the source of G_1 and the sink of G_2.

A Search(k) instruction is processed by the broadcast net as follows. The I/O port (the sole source in G_1) will "broadcast" a message <"broadcast",k,v,r> through G_1, where v is a Boolean value denoting whether record k has been found, and r is a field containing record k if found. The sinks of G_1 will then start a "reverse broadcast" process by sending <"reverse broadcast",k,v,r> messages through G_2; this process has the final effect of collecting at the I/O port (the sole sink in G_2) either record k (if it is in the dictionary) or a negative acknowledgment (v=0). A detailed description of the search algorithm can be found in [DS87].

An additional constraint on the structure of the broadcast net is that, if more than one message is received on the same graph by the same processor at the same time, they must all contain the same search key k.

An example of a broadcast net is given in figure 3, where the dark lines are the edges of G_1 and the shaded lines are the edges of G_2.

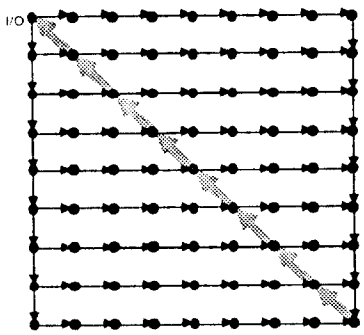


Figure 3: A Broadcast Net

A special class of the broadcast nets is the one of *broadcast trees* : in a broadcast tree, G_1 is a directed binary tree rooted at the I/O port where all the leaves have the same height, and G_2 is coincident with G_1 except for the direction of the edges which is reversed. In a broadcast tree, each query is broadcasted down the tree (broadcast) and then, starting from the leaves, the partial results move upwards towards the I/O port where the final result is computed (reverse broadcast).

3.3. Integrating Snake and Broadcast Net

In the proposed dictionary machine, both a snake and a broadcast net are embedded in the same mesh and operate on it simultaneously. All incoming search instructions are handed over to the broadcast net, whereas all other instructions are executed by the snake. When embedding and operating both structures on the mesh, several factors must be taken into consideration; in fact, most embeddings would not archieve the desired performance and, even worse, would not correctly perform the desired operations.

In this section, these factors are identified and conditions are established for a correct implementation.

3.3.1: Concurrent execution of different processes by a processor

Since each processor belongs to both the snake and the broadcast net on which different processes are executed, a processor might have to execute different processes in parallel creating the possibility of execution conflicts. Such conflicts are avoided provided that each direction of each communication link is used by at most one process at a time. In fact, in this case, the information sent to each output line is still a function of the values of the received inputs, the system time, and the current register contents. Depending on these values, the output for a line might be with respect to a process in either the snake or the broadcast net. Thus, each PE is totally described by the set of such output functions, which can be easily and directly implemented in hardware.

3.3.2: Using the same line for broadcast and snake communication

If the embeddings of the snake and the broadcast net are not disjoint (see figure 4), contention for the same line in the same direction by the two processes may occur. This

problem can be easily solved by splitting each *step* of the dictionary machine into two distinct *phases* of one time unit each, and executing snake operations only in the first phase and broadcast operations only in the second phase. This solution does, however, slow down the time performance of the system by a factor of two. An alternative solution which avoids this slow down consists in finding an embedding where the snake and the broadcast net are edge disjoint (as discussed in section 4); this solution, however, requires a more complex link structure in the mesh.

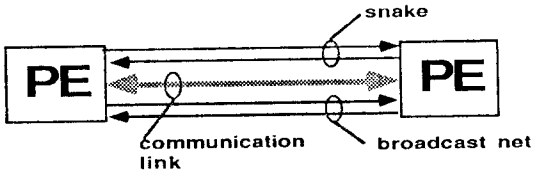


Figure 4: Boadcast Net and Snake Using the Same Line in Both Directions

3.3.3: Insertion/deletion instructions not yet executed

When an Insert(k) [Delete(k)] instruction is handed to the snake, the insertion (deliton) is not performed until its final destination processor P is found; this might require O(n) time. This fact might conflict with the requirement of O(1) period. In fact, it is possible that a subsequent Search(k) operation (started on the broadcast net) reaches P before the Insert(k) [Delete(k)] command (travelling on the snake) does; in this case it will be incorrectly reported that record k is not in the dictionary.

The situation can be even worse. Consider the following instruction sequence (left to right):

Insert(k), Delete(k), Insert(k), Search(k), Delete(k) .

In this case, the search has to retrieve the record inserted by the second insert operation although this might have not yet been completed. It also has to ignore the last Delete(k) that, although started after the search, could be encountered by the Search broadcast.

All these problems are however solved by simply associating to each incoming instruction a *time stamp* (an integer modulo 2n) and generalizing the broadcast process for Search(k) as follows. Message are now a 7-tuples

<"Broadcast", k, v,r ,t,U,tU>

<"Reverse Broadcast",k,v,r ,t,U,tU >

where

- k: search key
- v: boolean (0,1) indicating whether record k has been found
- r: record k (if found)
- t: timestamp associated to the Search(k) instruction
- U: most recent update on k (Insert(k) or Delete(k,r))encountered by the process and started before Search(k)
- tU: time stamp of most recent update on k

Execution of a Search(k) process on the snake is modified in such a way that the additional fields U and tU are updated during the search process (i.e., broadcast and reverse broadcast) to mantain the condition that U is the most recent update instruction on k not yet executed by the snake which was encountered during the search process and started before Search(k). When the messages arrive back at the I/O port, the final decision is made there on whether record k is in the dictionary (and, thus, is to be reported) or not. A detailed description can be found in [DS87].

3.3.4: Hidden information

In the previous section, it was assumed that the key k (if in the dictionary) as well as every Insert(k) and Delete(k) instruction (still in transit) will be found by a subsequent Search(k) process. To ensure that this condition is met, it is however necessary to consider and solve yet another problem which might occur. In fact, it is possible for the search process not to find the sought record even when it is stored in the dictionary. This is due to the fact that deletion of a record causes all the following records (in the snake) to be shifted one PE towards the I/O port in order to maintain the priority queue structure. Consider the following situation depicted in figure 5:

Within the broadcast net, the search message arrives simultaneously at processors A and B at time i, and is forwarded by them to processors C and D at time i+1. Assume also that at time i record k is shifted backwards within the snake from processor D to processor A following a Delete(k') instruction, k≠k'. Hence, the seach process does not find record k neither at processor A nor at processor D, possibly

resulting in an incorrect answer.

This particular problem may occur only if the snake directly connects processors P and P' such that $|t(P)-t(P')| \geq 1$, where t(P) denotes the time steps necessary for a search message to travel from the I/O port to P (in case of a broadcast tree this is the height of P). This problem is resolved by

1. having each PE remembering the record k* (if any) held in the previous time step as well as the update instruction (if any) performed in the previous time step. This information shall be referred to as *history record*.

2. modifying the search process in such a way that each search message is not only compared to the current record but also to the history record.

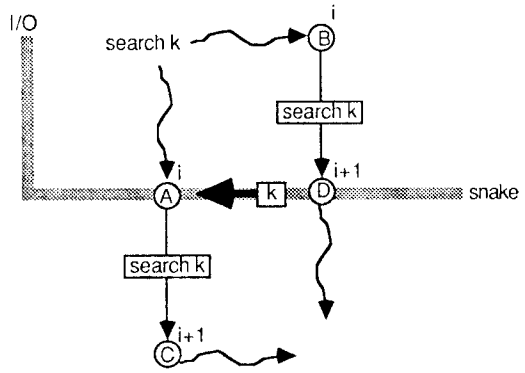


Figure 5: Hidden Information

Similarly, in case of $|t(P)-t(P')| = d>1$, the problem can be solved by keeping at each processor the history records for the previous d time units.

Let Δ denote the maximum $|t(P)-t(P')|$ for all pairs of PEs which are directly connected in the snake; the value of Δ depends on how the snake and the broadcast net are simultaneously embedded in the mesh. Note that the constraint on PE's having a constant number of registers implies that the only feasible embeddings (for our technique) are the ones with $\Delta \in O(1)$. The embeddings described in the following section guarantee $\Delta=1$; thus, only one history record needs to be stored in each PE to solve the problem.

4. Embedding the Broadcast Net and the Snake in a Mesh

4.1. Non-Disjoint Embeddings

The problem which now remains to be solved is how to simultaneously embed the snake and broadcast net on a mesh. It is relatively simple to find non-disjoint embeddings as the ones described in figure 6, where simple arrows denote edges in G_1-G_2, double arrows denote edges in $G_1 \cap G_2$, and the shaded lines denote the snake.

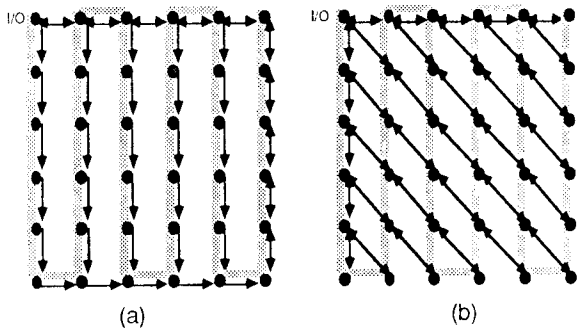


Figure 6: Two Non-Disjoint Embeddings

Period :

Each step of the dictionary machine consists of two phases of one time unit each (see section 3.3.2).The period of the snake is two steps and the period of the bradcast net is one step; hence, the period of both dictionary machines (shown in figure 6) is 4 time units.

Latency :

The latency for search instructions is $8\sqrt{n}$ time units ($4\sqrt{n}$ steps) for embedding (a) and $4\sqrt{n}$ time units ($2\sqrt{n}$ steps) for embedding (b). The latency for Find Min operations is 2 time units for both embeddings.

Number of history records:

It is easy to see that for both embeddings $\Delta=1$; hence, only one history record is required for each PE.

Link complexity:

Embedding (b) can be archieved on quadrilateral meshes. Embedding (a) requires only a c-linear mesh; i.e., a mesh where each PE, except in the top and bottom rows, is connected to just two neighbors.

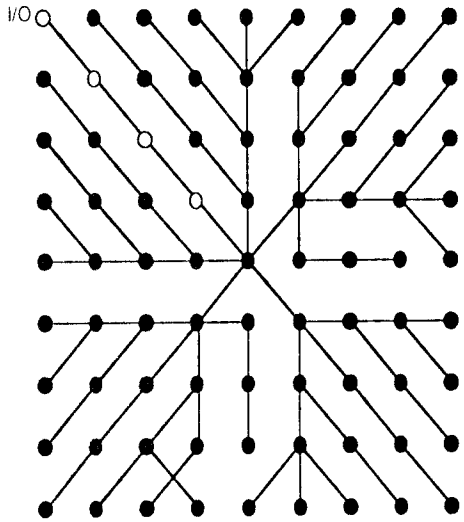
4.2. Disjoint Embeddings

Since non-disjoint embeddings slow down the time performance of the dictionary machine by a factor of two (see section 3.3.2) it is highly desirable to find disjoint embeddings of the broadcast net and snake. Disjoint embeddings are not as simple to derive and do not exhibit a regular pattern easily scaled to meshes of arbitrary size.

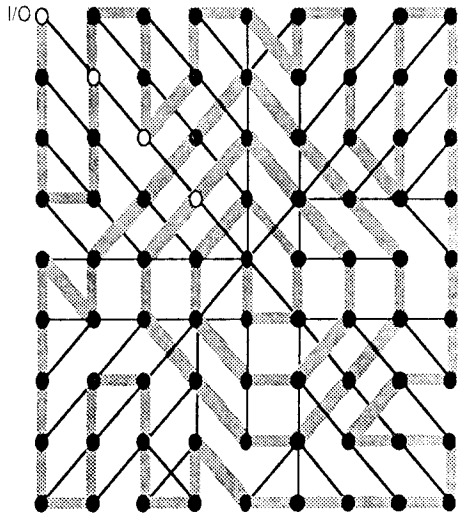
In this section we present a technique for computing a disjoint embedding in $\sqrt{n}$ x$\sqrt{n}$ meshes when $\sqrt{n}$ is odd; a full description of the algorithm can be found in [DS87].

The broadcast tree:

The initial part of the broadcast tree is a linear chain from the I/O port to the center of the mesh; the center is then connected to the four quadrants. The structure of the broadcast subtree in each quadrant is different (so to facilitate the disjoint embedding of the snake). An example of the embedding obtained by the algorithm for a 9x9 mesh is shown in figure 7(a); the regularity of the structure can be observed by comparing it with the embedding obtained for a 11x11 mesh shown in figure 8 .



(a)



(b)

Figure 7: Disjoint Embeddings on a Mesh of Size 9x9,
(a) broadcast net, (b) broadcast net and snake

The snake:

The snake (in reverse order) starts from the center of the mesh and moves along a spiral until it has distance one from all the borders; it then winds through the remaining four quadrants.

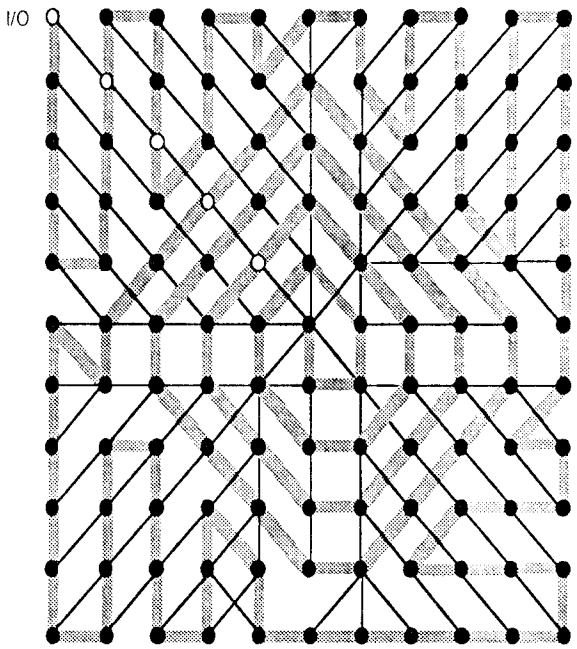


Figure 8: Disjoint Embedding of Broadcast Net and Snake in a Mesh of Size 11x11

In this embedding, the structure is different depending on whether $\lceil \sqrt{n} / 2 \rceil$ is even or odd. If $\lceil \sqrt{n} / 2 \rceil$ is odd, the structure is shown in figure 7b (just enlarged for $\sqrt{n} > 9$); otherwise, the structure has the form shown in figure 8 (again, just enlarged for $\sqrt{n} > 11$).

The relayers:

The PEs on the diagonal from the I/O port to the center of the mesh are distinguished in the sense that they only act as relayers in the search process and do not store records.

The need for relayers derives from the fact that at least one such processor R is directly connected in the snake to a processor P with $|t(P)-t(R)|=\Omega(\sqrt{n})$; hence, R cannot be used to store records (see section 3.3.4). In order to maintain the correct operation of the snake and preserve period 2 for the dictionary, the data storage and processing activities which would be carried out if each relayer R were a normal processor will be performed cooperatively by its predecessor P and its successor S in the snake. In particular, P will also store the record which would have been stored in R; and both P and S will store (for one time unit only) a copy of each message sent through the snake to R. This will ensure that Delete(k), Insert(k) messages and the current record (which would have been stored at a relayer) are not missed by the search process, and that the period for the snake is still 2.

Taking the relayers out of consideration, it is not difficult to verify that in our disjoint embedding $\Delta=1$; hence, every PE has to store one history record only (see section 3.3.4).

Summarizing, we note that the disjoint embedding introduced above induces a VLSI dictionary machine with a period of 2 time units and a latency of $2\sqrt{n}$ and 1 time units for Search and Find Min operations, respectively. However, to obtain these improved performance results, an octagonal mesh is needed in contrast to a c-linear and quadrilateral mesh employed for non-disjoint embeddings.

The following table 2 summarizes all performance results stated in this paper.

performance (time units) \ layout	non-disjoint layout (a)	non-disjoint layout (b)	disjoint
period	4	4	2
latency: find min	2	2	1
latency: search	$8\sqrt{n}$	$4\sqrt{n}$	$2\sqrt{n}$
link structure	c-linear	quadrilateral	octagonal

Table 2: Performance Results

REFERENCES

[AK85] M.J. Atallah and S.R. Kosaraju, "A generalized dictionary machine for VLSI", *IEEE Trans. on Computers C-34*, 2 (Feb. 1985), 151-155.

[BC86] D.A. Bailey and J.A. Cuny, "An efficient embedding of large trees in processor grids", *Proc. 1986 Int. Conf. on Parallel Processing*, St. Charles, Aug. 1986, 819-822.

[BK79] J.L. Bentley, and H.T. Kung, "A tree machine for searching problems", *Proc. 1979 Int. Conf. on Parallel Processing*, Aug. 1979.

[CM81a] B.M. Chazelle, L.M. Monier, "A model of computation for VLSI with related complexity results" *Proc. 13th ACM Conf. on Theory of Computing*, May 1981.

[CM81b] B.M. Chazelle, L.M. Monier, "Optimality in VLSI", Tech. Rep. CMU-CS-81-141, Department of Computer Science, Carnegie-Mellon University, Sept. 1981.

[CT84] M.J. Carey and C.D. Thompson, "An efficient implementation of search trees on [lg N +1] processors", *IEEE Trans. on Computers C-33*, (1984), 1038-1041.

[CCIR86] J.H. Chang, M.J. Chung, O.H. Ibarra, K.K. Rao, "Systolic tree implementation of data structures", *Proc. 1986 Int. Conf. on Parallel Processing*, St. Charles, Aug. 1986, 669-671.

[DS87] F. Dehne, N. Santoro, "Optimal VLSI dictionary machines on meshes", Report SCS-TR-106, Carleton University, January 1987.

[F84] A.L. Fisher, "Dictionary machines with small number of processors", *Proc. Int. Symp. on Computer Architecture*, Ann Arbor, June 1984, 151-156.

[GKS82] D. Gordon, I. Koren, G. Silberman, "Embedding tree structures in VLSI hexagonal arrays", *IEEE Trans. on Computers C-31*, 9 (sept. 1982), 892,897.

[KL80] H.T.Kung, C.E.Leiserson, *Systolic Arrays (for VLSI)*, chapter 8 in [MC80]

[KL84] M.R.Kramer, J. v.Leeuwen, "Systolische Berechnungen und VLSI", *Informatik Spektrum* 7, 1984, pp.154-165

[L79] C.E. Leiserson, "Systolic priority queues", Report CMU-CS-79-115, Carnagie-Mellon University, April 1979.

[L83] C.E. Leiserson, *Area Efficient VLSI Computation*, MIT Press, 1983.

[MC80] C.A. Mead and L.A. Conway, *Introduction to VLSI Systems*, Addison-Wesley, 1980.

[MS84] R. Miller, Q.F. Stout, "Computational Geometry on a Mesh-Connected Computer", *Proc. 1984 Int. Conf. on Parallel Processing*, 1984, pp.66-73

[ORS82] T.A. Ottman, A.L. Rosenberg, and L.J. Stockmeyer, "A dictionary machine for VLSI", *IEEE Trans. on Computers C-31*, 9 (Sept. 1982), 892-897.

[PRS81] M.S. Paterson, W.L. Ruzzo, and L. Snyder, "Bounds on minimax edge lenght for complete binary trees", *Proc. 13th ACM Symp. on Theory of Computing*, May 1981, 293-299.

[S82] L. Snyder, "Introduction to the Configurable Highly Parallel Computer", *Computer Journal*, Jan. 1982.

[S85] A.M. Schwartz, "Dictionary machines for cube-class networks", Rep. ACT-53, University of Illinois at Urbana, 1985.

[SA85] A.K. Somani, and V.K. Agarwal, "An efficient unsorted VLSI dictionary machine", *IEEE Trans. on Computers C-34*, 10 (Sept. 1985), 841-852.

[SL87] A.M. Schwartz and M.C. Loui, "Dictionary machines on cube-class networks", *IEEE Trans. on Computers C-36*, 1 (Jan. 1987), 100-105.

[SS85] H. Schmeck and H. Schröder, "Dictionary machines for different models of VLSI", *IEEE Trans. on Computers C-34*, (1985), 472-475.