

MODELING AND SIMULATION

VOLUME 23

Part 3

**Computers, Computer Architecture,
Microprocessors in Education, Transportation,
Optimization, Education**

Editors

**William G. Vogt
Marlin H. Mickle**



**Proceedings of the Twenty-Third Annual Pittsburgh Conference
Held April 30 - May 1, 1992
School of Engineering - University of Pittsburgh**

**Published and Distributed by
School of Engineering - University of Pittsburgh**

EMS
LIB
TA
343
P702
1992
11.3

ACKNOWLEDGEMENTS

Conference Co-Chairmen

William G. Vogt Marlin H. Mickle

Conference Coordinators

Samer S. Saab Jihan Z. Saab

Conference Committee

O. Abdulkarim	Y.-H. Chang	J.-W. Hong	O. Russian
T. Akgul	T. N. Dentel	N. Jabnoune	S. S. Saab
J. M. Alhumoud	X. Dong	A. H. Keun	S. Tadjudin
A. Ali	A. El Hajj	P. A. Kwan	C. T. Weissman
D. Breitenstein	J. Garcia	B. Pack	S. Zahir

An international conference such as the Modeling and Simulation Conference owes its success to many people. To list each of those who helped and the manner in which they helped would likely require a volume the size of the PROCEEDINGS. But we can mention some who have been of particular help.

The editors wish to express sincere thanks to Professor Marwan A. Simaan, Chairman of the Department of Electrical Engineering, University of Pittsburgh, for his support of this conference.

The School of Engineering, University of Pittsburgh, has continued its support for more than 20 years. Accordingly, the editors wish to thank Dean Charles Sorber.

A special thanks to Katherine Tamenne, Sandra Weisberg, Madeline Nuzzo and Barbara Borzym, each of whom have helped with great enthusiasm and grace. An extra special thanks to Betty Victor and Barbara Dippold for constant help and encouragement.

Thanks to our Conference Coordinators, Conference Committee and all the others who organized sessions, operated audio-visual equipment, made and posted signs, helped with registration and handled efficiently those many situations and problems, some of which could have meant disaster to the Conference, but each of which was satisfactorily resolved in a truly professional spirit.

But, most of all, a final thanks to all of our authors, coauthors, session organizers and session chairpersons, without whom there would be no conference.

William G. Vogt
Marlin H. Mickle
Editors

Tree Structured Dictionary Machines for VLSI

M. R. Meybodi

Computer Science Department
Ohio University

Abstract

Recent advances of VLSI has led to the development of high performance, special purpose hardware to meet specific application requirements. In this report, we review a number of tree structure multiprocessor designs called Dictionary Machines. These designs have been proposed for performing a group of operations including *SEARCH*, *INSERT*, *DELETE*, *XMIN* (Extractmin), *XMAX* (Extractmax), and *NEAR* on a set of keys. A modification to one of the these designs is presented. This modification enables the machine to handle redundant operations.

Keywords and Phrases: Dictionary Machine, VLSI, Parallel Algorithm, Parallel Processing, Special Purpose Machine

1 Introduction

Special classes of computational tasks have led to the development and realisation of special purpose computer systems that most efficiently perform the given tasks. One class of these special purpose architectures is characterized by a strong, inherent parallelism functioning in a systolic manner.

A systolic system is a network of processors which rhythmically computes and passes data through the system [11]. In systolic systems, processors (cells) are typically interconnected to form a systolic array or systolic tree which results in a simple, regular communication and control structure. Data flows between cells in a pipelined fashion and communicate with the outside world only at the boundary processors. Systolic architecture is used to implement variety of computational tasks such as : signal and image processing, matrix arithmetic, graph algorithms, geometric algorithms, relational database operations [4,8,9,10]. This architecture was originally proposed for VLSI implementation of some matrix operations [4].

In this report we study a number of systolic tree designs which have been proposed for performing a group of dictionary operations on a set of keys. These machines known as Dictionary Machines perform some subset of the following operations: *INSERT*, *DELETE*, *SEARCH*, *XMIN*, *XMAX*, and *NEAR*. Before we describe these tree structured Dictionary Machines, we define the effect of the various operations. Our notations follow [3]. Let F denote the set of all key-record pairs stored in the machine and (k, r) denote a pair with key k and record r . For a key k and set F , we say k is stored in the machine if there exists an r such that (k, r) is in F . Define

$$F(k) = \{(k, r) | (k, r) \in F\};$$

so $F(k)$ is a singleton set if k is in F and empty if k is not in F . The effect of dictionary operations are as follows:

INSERT (k, r):
 $F \leftarrow (F - F(k)) \cup \{(k, r)\}$.
Response is null.

DELETE (k):

```

 $F \leftarrow F - F(k).$ 
Response is null.

SEARCH( $k$ ):
 $F$  is not changed.
Response is  $F[k]$  if  $k$  is in  $F$  and "not in" if  $k$  is not in  $F$ .

XMIN:
 $F \leftarrow F - F(k_{\min}).$ 
Response is  $F[k_{\min}]$  where  $k_{\min}$  is the smallest key in  $F$ .

XMAX:
 $F \leftarrow F - F(k_{\max}).$ 
Response is  $F[k_{\max}]$  where  $k_{\max}$  is the largest key in  $F$ .

NEAR( $k$ ):
 $F$  is not changed.
Response is  $F[k_{\text{near}}]$  where  $k_{\text{near}}$  is the stored key closest to  $k$ .

```

An insertion operation is redundant when the pair requested to be inserted already exists in set F . A deletion operation is redundant when the pair requested to be deleted does not exist in set F . We use n to denote the number of pairs stored in the machine (set F) at a given time, and N to denote the capacity (maximum number of pairs that may be stored in the machine) of the machine.

In section two of this report we present a review of the designs proposed in the literature. In section three we modify one of the existing designs. This modification enables the machine to handle redundant operations. In section four we compare the design proposed in this report with the one proposed by Bentley and Kung.

2 Previous Designs

At least five papers propose tree-structured design for the implementation of dictionary operations. Even though they differ in details they share some basic principles. A machine is a collection of processors connected together so as to form a binary tree. That is, all processors (except the leaves) have two successors and all processors (except the root) have one predecessor. Each processor may hold zero or more pairs, and can communicate with its father, two children and sometimes with some additional processors. The root processor, in addition to storing data is also used as input/output port of the machine. Operations are broadcast from the root and are pipelined along the depth of the tree. Such a tree machine is a completely general purpose parallel processing engine and can be used for problem decomposed in a hierarchical way. Tree machine can solve any problem that can be stated as computing some function over every element in the set (such as equality or absolute value of difference) and then combining the values of those functions by some associative, commutative binary operator [1]. Tree structured systolic arrays, have been exploited to solve a wide range of problems [8,2].

We consider two measurements to compare different designs: *response time*, the time elapsed between the initiation and completion of an operation, and *pipeline interval*, the minimum time needed between the initiation of two separate operations.

2.1 Bentley and Kung's Machine

The basic structure of this machine [1] is illustrated in figure 1. This machine is a mirrored binary tree, containing three kinds of processors: circles, squares, and ovals. The circle processors are used to broadcast data and also to decide where a new pair is to be inserted. The ovals are used to combine their inputs. These processors perform operations such as min, max, and, or, and plus. The operations by this machine supported are *INSERT*, *SEARCH*, and *DELETE*. To illustrate the operation of this machine we consider the problem of performing the member search to answer the query "is (k, r) an element of the set F ?". Let us suppose that each of N square processors holds an element of set F . To determine if (k, r) is a member of F , pair (k, r) is inserted into the root of the tree, and $\log N$ steps later it will arrive at all of the square processors. Then N comparisons are performed in parallel, and a bit is set to one if the pair stored in that square processor is equal to (k, r) and zero otherwise. This is shown in Figure 2. These bits are then combined in $\log N$ steps by letting the ovals compute the logical or of their two inputs. So after a total of $2 \log N$ steps have

passed since the query was posed, a single bit emerges from the output processor telling whether or not (k, r) is an element of set F .

A new pair can be inserted into the machine by placing it in any unused square processor. Such a square processor can be found by having each circle keep track of the number of unused square processors descendants of each of his two children. When a request comes to the root for a new (unused) position, it passes the request to one of its children with unused square processors, and so on. This is done by masking all of the square processors except the one which will hold the pair. This square processor then loaded with the desired data. Note that a single pair can be inserted in $\log N$ steps, and a set of M pairs can be inserted in $M - 1 + \log N$ steps.

To delete a pair a flag in its square processor is set to indicate that it is unused and then the counts in all of the circles above it are adjusted. This can be done either by pushing information backward to the top of the tree, or by doing a dummy reinsertion of that pair, and modifying the counters on the way down. The time for either of these operations is proportional to $\log N$. The pipeline interval for *DELETE* operation because of modifying the counters in the squares is $O(\log N)$ in contrast with the $O(1)$ interval for the *INSERT* and *SEARCH* operations. Since the pairs are stored in the square processors, the response time for all the above mentioned operations is $O(\log N)$.

2.2 Leiserson's Machine

This machine [2] supports the operations *INSERT* and *XMIN* which simultaneously returns and deletes the pair with the smallest key in the machine. The machine operates within time $O(\log N)$ and pipeline interval $O(1)$. In this machine square processors are connected to form a linear chain and it is shown in figure 3. The data elements are stored only at the square processors in increasing order over the linear chain with the pair having the smallest key stored in the leftmost processor. This machine was designed to implement non-redundant *INSERT* and *XMIN* operations only but it can be modified to handle *SEARCH*, *NEAR*, and *DELETE* operations.

Like the Bentley and Kung Machine operations are broadcast from the root and pipelined along the depth of the tree. It takes $\log N$ steps for an operation to reach all the leaf processors. When the leaf processors receive an *INSERT*(k, r), the pair (k, r) is inserted at the proper place in the linear chain, and all pairs with keys greater than k are shifted right to make room. When *DELETE*(k) operation is received, all pairs with keys greater than k are shifted left to fill up the hole created by *DELETE* operation. We now describe more precisely how an square processor S_i processes the various operations. Let (k_j, r_j) be the content of S_j (j^{th} processor from the left). We also let $k_0 = -\infty$, $k_{N+1} = \infty$, and $r_0 = r_{N+1} = \emptyset$. \emptyset is used to denote a null pair. Initially, all the square processors contain (∞, \emptyset) . When an square processor S_i receives an operation, depending on the operation it executes one the following algorithms:

```

SEARCH( $k$ ):
if  $k_i = k$  then answer  $(k_i, r_i)$ ; else answer "not in".

```

```

INSERT( $k, r$ ):
if  $k_{i-1} < k < k_i$  then  $(k_i, r_i) \leftarrow (k, r)$ .
if  $k < k_{i-1}$  then  $(k_i, r_i) \leftarrow (k_{i-1}, r_{i-1})$ .

```

```

DELETE( $k$ ):
if  $k \leq k_i$  then  $(k_i, r_i) \leftarrow (k_{i+1}, r_{i+1})$ .

```

```

UPDATE( $k, r$ ):
if  $k_i = k$  then  $(k_i, r_i) \leftarrow (k, r)$ .

```

```

XMIN:
if  $i = 1$  and  $r \neq \emptyset$  then answer  $(k_i, r_i)$ ; else answer "not in".
 $(k_i, r_i) \leftarrow (k_{i+1}, r_{i+1})$ .

```

This machine does not operate correctly when asked to insert a pair which is already present or delete a pair which is not present. If a delete is redundant, a valid pair may be destroyed, a hole in the data structure is created which wastes a processor. Leiserson machine were later modified so that the machine can operate correctly for redundant operations [3]. The problem of redundant deletion is avoided by having the square processors with key k simply mark its content as deleted,

while the other square processors do not shift their contents, this solution produces holes just as redundant insertion do. The holes are removed by an operation called *COMPRESS*. Even with these modifications the machine needs $O(\log N)$ times to respond. The *COMPRESS* operations can be generated by the root processor just like any other operation. $(k, *)$ denotes a hole with key k . For a given k , there could be several square processors with content $(k, *)$. If k is in the machine there will be exactly one square processor with content (k, r) with $r \neq *$, and this square processor will be to the left of any square processor with content $(k, *)$. Square processor S_i processes operations as follows:

SEARCH(k):
if $k_i = k$ and $r_i \neq *$ then answer $\{k_i, r_i\}$; else answer "not in".

INSERT(k, r):
if $k_{i-1} < k \leq k_i$ then $\{k_i, r_i\} \leftarrow \{k, r\}$.
if $k_{i-1} = k$ then $\{k_i, r_i\} \leftarrow \{k, *\}$.
if $k_{i-1} > k$ then $\{k_i, r_i\} \leftarrow \{k_{i-1}, r_{i-1}\}$.

DELETE(k):
if $k_i = k$ then $\{k_i, r_i\} \leftarrow \{k, *\}$.

UPDATE(k, r):
if $k_i = k$ and $r_i \neq *$ then $\{k_i, r_i\} \leftarrow \{k, r\}$.

XMIN:
if $i = 1$ and $r_i \neq \emptyset$ then answer $\{k_1, r_1\}$; else answer "not in".
 $\{k_i, r_i\} \leftarrow \{k_{i+1}, r_{i+1}\}$.

COMPRESS:
if $r_i = *$ and $r_{i+1} \neq *$ then $\{k_i, r_i\} \leftarrow \{k_{i+1}, r_{i+1}\}$.
if $r_i \neq *$ and $r_{i-1} = *$ then $\{k_i, r_i\} \leftarrow \{k_i, *\}$.

Square processor S_i processes *COMPRESS* by asking S_{i-1} to see if it contains a hole; if so, S_i shifts its content to S_{i-1} and the content of S_i becomes a hole. A hole created by a redundant *INSERT* or by a nonredundant *DELETE* could be pushed completely to the right by at most $N - 1$ *COMPRESS* operations. Ottmann, Rosenberg, and Stockmeyer shown that if the dictionary machine executes one *COMPRESS* operation after each *INSERT* or *XMAX* and two *COMPRESS* operations after each *DELETE*, then after these *COMPRESS* operations any initial segment of the dictionary of length c never contains more than $\lfloor c/2 \rfloor$ holes.

This machine can be further improved by combining operations *SEARCH* and *COMPRESS* into a new operation called *COSEARCH*, defined as follows:

COSEARCH (k):
if $k_i = k$ and $r_i \neq *$ then answer $\{k_i, r_i\}$.
if $r_i \neq *$ and $r_{i-1} = *$ then $\{k_i, r_i\} \leftarrow \{k_i, *\}$.
if $r_i = *$ and $r_{i+1} \neq *$ then $\{k_i, r_i\} \leftarrow \{k_{i+1}, r_{i+1}\}$.

If every *SEARCH* is replaced by a *COSEARCH*, an additional *COMPRESS* is only needed if an *INSERT* or an *XMIN* is not followed by a *SEARCH* or if a *DELETE* is not followed by two *SEARCH* operations. This new operation will increase the capacity of the machine whenever retrieval occur significantly more frequently than *INSERT*, *DELETE*, or *XMIN* [12].

2.3 Ottmann et al. Machine

This machine [3] uses all the processors for storing data, and therefore the response time is reduced to $O(\log n)$ where n is the total number of pairs present in the machine. The major change in the structure of the machine is that the square processors are now snaked through the tree as shown in figure 4. As a result all pairs are stored in square processors whose depth in the tree is $O(\log n)$ which leads to response time of $O(\log n)$. The operations are executed in the same way as in Leiserson's machine. The pipeline interval for this machine is $O(1)$.

There are two kinds of links connecting the square processors: tree links used to broadcast operations, and array links for communication between an square processor and its neighbors. This machine has number of disadvantages:

1. One corner processor at each level is needed to store its old contents as the reference data for the successful execution of an operation at the next level.
2. Different numbers of an additional operation, called *COMPRESS*, are sent from the root processor after every regular operation to remove the holes (processor without an element) created by a nonredundant *DELETE* or a redundant *INSERT* operation which results in a non-uniform output pipeline interval.
3. The last level in this machine is divided on two physical levels. Therefore, the response from two sons of a node is generated at different times and a processor is required to wait for response from both of its sons before generating the response for its own father. This requires additional delay buffers at each processor which complicates the control algorithm, and increases the pipeline interval by one cycle.

2.4 Atallah and Kosaraju's Machine

This machine [7] performs all the dictionary operations in $O(\log n)$ time, and at pipeline interval $O(1)$. Each processor (except the last active processor) holds three pairs. They are stored in the machine in such a way that a preorder traversal of the tree is isomorphic to the sorted sequence of the pairs, as shown in figure 5.

If no redundant operation is allowed, the pairs with minimum and maximum key are available at the root processor. If redundant operations are allowed, holes may be created in the machine and the pair with maximum key may not be available at the root processor. Like the Ottmann et al. Machine different compress and cleartail operations are broadcast after every regular operation to remove holes. This results in a non-uniform output interval.

2.5 Somani and Agarwal's Machine

This machine [5,6] has a pure binary tree structure. Each processor communicates with its father and two children only. The pairs are stored in an unsorted manner which simplifies the algorithms in some respect. This machine supports *INSERT*, *DELETE*, *SEARCH*, *XMIN*, *XMAX*, and *NEAR* operations in time $O(\log n)$. In addition to the above mentioned operations this machine executes one *COMPRESS* operation after every regular operation in order to balance the tree at each node with respect to the number of pairs stored in the left and right subtrees of the node (processor) and to fill the holes created by the execution of either a redundant *INSERT* or *DELETE* operation. The pipeline interval of this machine is just two steps including the additional operation required to maintain the structure which is much better than design of [7] and [3] where the pipeline interval has to be at least 3 steps excluding additional operations. This results in three to four times improvement in throughput when the additional operations are considered. The operations for this machine is rather complicated and will not be explained here. This machine requires the smallest amount of internode I/O among the designs and wastes the fewest processors among the machine performing redundant operations [13].

3 Proposed Machine

The proposed machine is a modified version of Bentley and Kung machine. In Bentley and Kung machine the *SEARCH* and *INSERT* operations can be pipelined with constant interval, but a *DELETE* operation will in general require an interval of $O(\log N)$ before the next operation can be entered into the machine. With the proposed machine sequences of *DELETE* operations can be pipelined with constant period, but *INSERT* operation needs an interval of $O(\log N)$. For the insertion operation to operate correctly when asked to insert a pair which is already present, a search for that pair initiated before inserting it. The result of search operation will determine whether the pair must be inserted or not. This will increase the pipeline interval for *INSERT* to $O(\log N)$. With this machine redundant operations will not create any problem.

The basic structure of Modified Bentley and Kung machine(MBK) is the same as Bentley and Kung machine with the exception that MBK machine has a full k-ary tree structure rather than a full binary tree structure. The machine contains three types of nodes: circle which broadcast data, squares which stores data and compute, and ovals which combine the inputs. The squares are the most complicated nodes. Each may be a processor with a small amount of main memory. The top tree is called forward tree and bottom tree is called image tree. The purpose of forward tree is to broadcast a given operation to all the square processors and the purpose of image tree is to transmit answers from the square processors to the root of the image tree. In other words, the MBK machine like Bentley and Kung machine has a two part structure, one part handles the

dictionary operations and the other does computations on the outcome of the dictionary operations. A stream of operations enter the root of the forward tree in a pipeline fashion, and travels down the forward tree for execution after some preprocessing at the root processor. After the execution for an operation is complete the response for this operation is directed toward the root of the image tree.

3.1 SEARCH Operation

The *SEARCH* operation determines whether or not a pair (k, r) is stored in the machine. The operation is entered into the root of the forward tree. $\log_2 N$ steps later it reaches all the square nodes, and then N comparisons performed in parallel. When a square processor gives an answer, that answer is passed to the root of the image tree. Of course the oval processors of the image tree must combine the answers in the correct ways. For example if a triangle receive a pair from one child and "not in" from the other children, it should transmit that pair to its father. The root of the image tree receives the answer to the query $2 \log_2 N$ steps after it was posed.

3.2 DELETE Operation

This operation travels down the forward tree. After $\log_2 N$ steps it will arrive at all the square processors holding the pairs. The response of the square processor is either "not in" or $(\text{deleted}, i)$ depending on whether or not the square processor contain the pair. i is the square processor number of the deleted pair. Oval processors then combine the response received as follows: if the input of an oval processor are all "not in", the output produced by that processor is "not in", and if one of the inputs is $(\text{deleted}, i)$, the output produced is $(\text{deleted}, i)$. Upon receiving $(\text{deleted}, i)$ by the root of the image tree, it is sent to the root of the forward tree. The root of the forward tree saves i (the square processor number at which the deleted pair used to reside) in a list L . This list at any time contains the numbers of all the square processors which are empty and are to the left of the rightmost processor whose content has not been deleted since it was inserted. We call this processor boundary square processor. The size of the list must be equal to the capacity of the machine.

3.3 INSERT Operation

Before describing the *INSERT* operation, let us mention an operation that we can perform on tree machines. Given a pair and integer i we want to store that pair in the i^{th} square processor from the left. The leftmost processor is numbered zero. Let N , the number of square processors, be 2^k and express i as k bit binary $a_1a_2a_3\dots a_k$, also assume that the number of children of each processor is 2^h for some h . The binary representation of i will be used to lead the pair to the i^{th} square processor when traveling down the forward tree. The root of the forward tree uses the bit pattern $a_1a_2\dots a_h$ to select a processor at the next level which will receive the pair. In general, a processor at level p which has received a pair from its father uses the $(p+1)^{\text{th}}$ bits of binary representation of i to choose the receiving processor at the next level. Figure 8 illustrates the association of bit patterns to the children of a node when $h = 2$. If the machine has a full binary tree structure, routing procedure can be described as follows: If $a_1 = 0$, pass the pair and $a_2\dots a_k$ to the left child of the root; if $a_1 = 1$, pass the pair and $a_2\dots a_k$ to the right child. Repeat the same process at each level until an square processor is reached.

When an *INSERT* operation is entered the root of the forward tree, it is assigned a square processor. The binary representation of the number indicating that processor will be used by processors at different level of the machine to lead the pair to the assigned processor when traveling down the forward tree. Routing procedure described in the previous paragraph may be used for directing the pair through the machine to the assigned square processor. One possible algorithm for assigning square processor to a pair is given below.

Square Processor Assignment Algorithm:

```

if list L is empty then
    assign the square processor which immediately follows
    the boundary square processor.
else
    assign an arbitrary square processor from list L

```

Step 1 of above algorithm can be implemented by maintaining a counter at the root of the forward tree. This counter is increased each time a square processor following the boundary square processor is assigned to a pair. Initially, the boundary square processor is numbered zero.

4 Conclusion

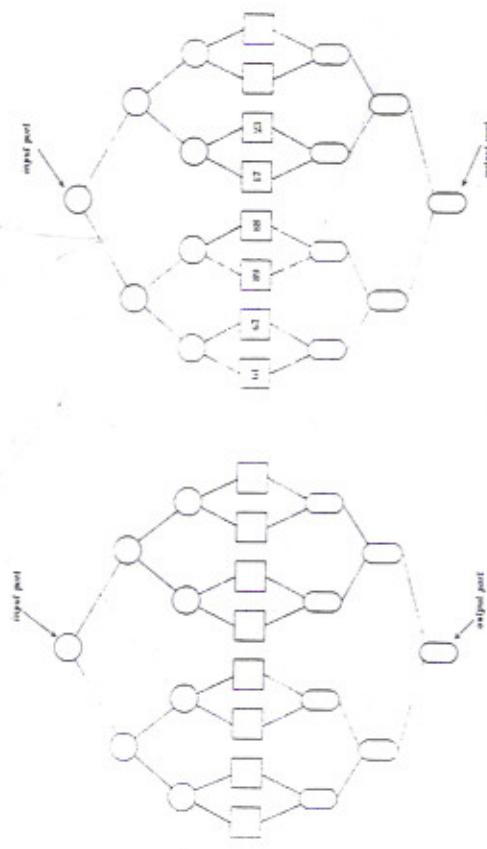
We conclude this report by comparing the Bentley and Kung machine with the Modified Bentley and Kung Machine (MBK). This comparison for $k = 2$ is given in the following table.

Operation	Bently-Kung	MBK
Pipeline Interval	Insert $O(1)$ Delete $O(\log N)$ Search $O(1)$	Insert $O(\log N)$ Delete $O(1)$ Search $O(1)$
Redundant Operations	No	Yes

5 References

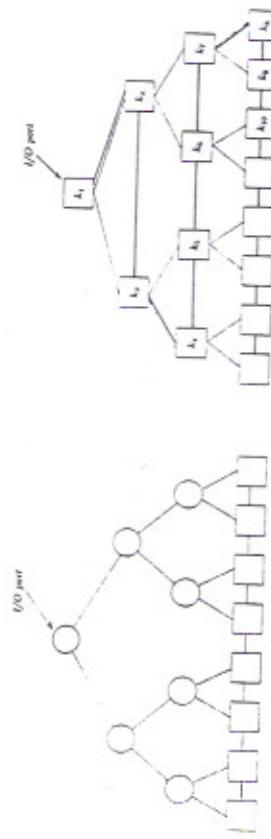
1. J. L. Bentley and H. T. Kung, "A tree Machine for Searching Problems," Proceeding of the International Conf. on Parallel Processing, August 1979.
2. C. E. Leiserson, "Systolic Priority Queues," Dept. of Computer Science, Carnegie Melon University, Pittsburgh, PA, Report CMU-CS-115, 1979.
3. T. A. Ottmann, A. L. Rosenberg, and L.J. Stockmeyer, "A Dictionary Machine for VLSI," IEEE Transaction on Computers, vol. c-31, No. 9, Sept. 1982, pp. 892-897.
4. H. T. Kung and C. E. Leiserson, "Systolic Arrays [for VLSI]," Proc. Symp. Sparse Matrix Computations and their Applications, Nov. 2-3, 1978, pp. 256-282.
5. A. K. Somani and V. K. Agarwal, "An Unsorted VLSI Dictionary Machine," Proceedings of 1983 Canadian VLSI Conference, University of Waterloo, Waterloo.
6. A. K. Somani and V. K. Agarwal, "An Efficient VLSI Dictionary Machine," 11th Annual International Symposium on Computer Architecture, University Of Michigan, Ann Arbor, Michigan, pp.142-150.
7. M. J. Atallah and S. R. Kosaraju, "A Generalized Dictionary Machine for VLSI," IEEE transactions on Computers, Vol C-34, No. 2, Feb. 1985, pp.151-155.
8. L. J. Guibas, H. T. Kung, and C. D. Thompson, "Direct VLSI Implementation of Combinatorial Algorithms," Proc. Conf. Very Large Scale Integration: Architecture, Design, Fabrication, California Institute of Technology, Jan. 1979, pp. 509-525.
9. H. T. Kung, "Special Purpose Devices for Signal and Image Processing : An Opportunity in VLSI," Proc. SPIE, Vol. 241: Real- Time Signal Processing III, Society of Photo-Optical Instrumentation Engineers, July 1980, pp.76-84.
10. H. T. Kung and P. L. Lehman, "Systolic Arrays for Relational Operations," Proc. ACM-SIGMOD 1980 Int. Conf. Data, ACM, MAY 1980, pp. 105-116.
11. H. T. Kung, "Why Systolic Architectures?" IEEE Comut. Mag. 15(1), Jan. 1982, pp. 37-46.
12. H. Schmeck and H. Schroder, "Dictionary Machines for Different Models of VLSI," IEEE transaction on computers, vol. C- 34, No. 5, May 1985, pp. 472-475.

13. A. L. Fisher, "Dictionary Machines with Small Number of Processors," Proc. Int. Symp. on Computer Architectures, Ann Arbor, June 1984, pp. 151-156.



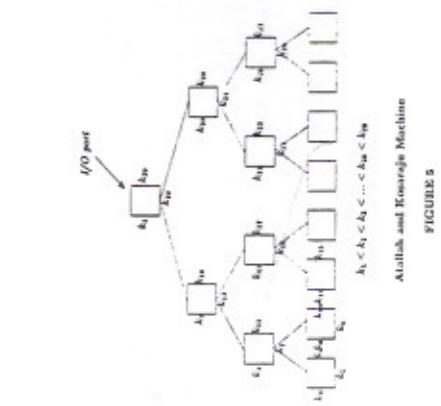
Brantley and Kung Machine

FIGURE 2



Orlitzky et al. Machine

FIGURE 4



Amitha and Kuroda's Machine

FIGURE 5

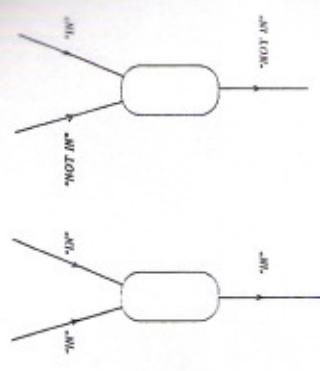
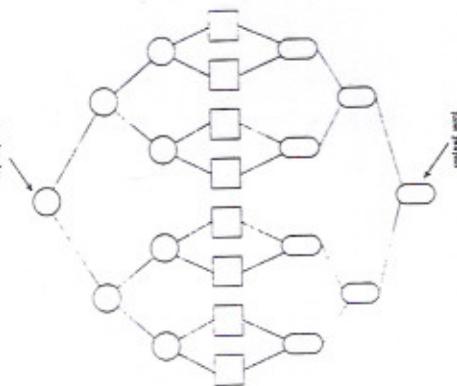
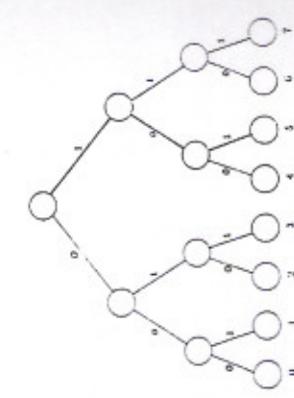


FIGURE 7



Modified Brantley and Kung Machine

FIGURE 6



Leiserson Machine

FIGURE 3

FIGURE 8

13. A. L. Fisher, "Dictionary Machines with Small Number of Processors," Proc. Int. Symp. on Computer Architectures, Ann Arbor, June 1984, pp. 151-156.

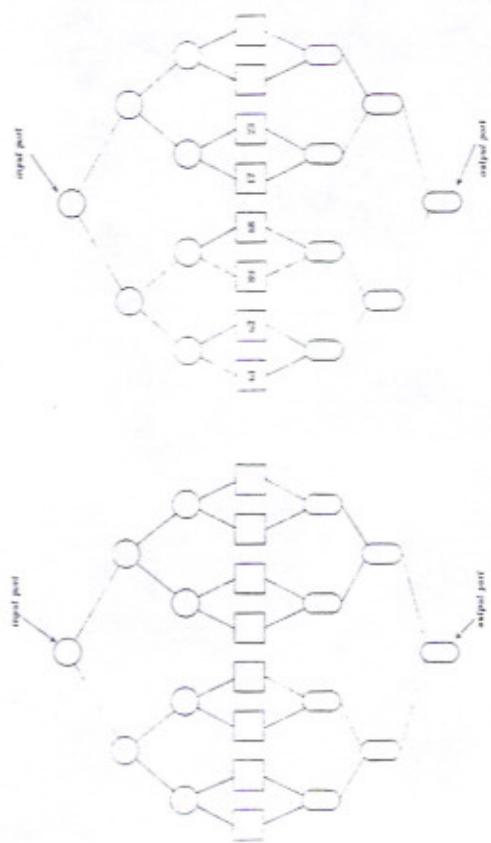


FIGURE 3

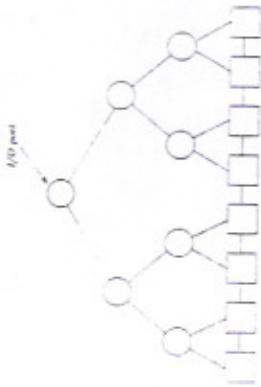


FIGURE 4

FIGURE 2

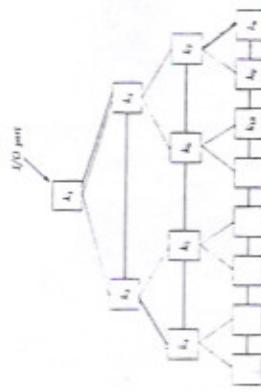


FIGURE 5

FIGURE 2

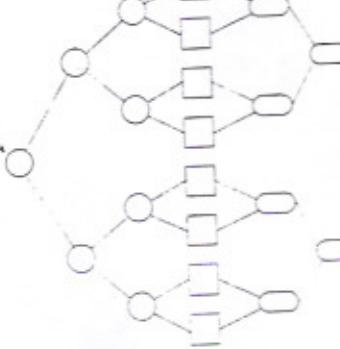


FIGURE 6

FIGURE 1



FIGURE 5

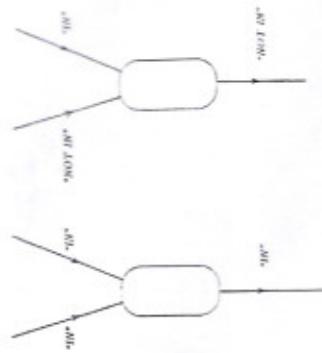


FIGURE 1

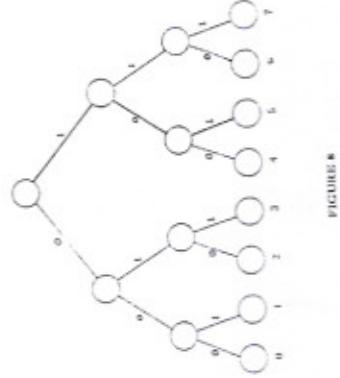


FIGURE 8