MODELING AND SIMULATION
VOLUME 23

Part 3
Computers, Computer Architecture,
Microprocessors in Education, Transportation,
Optimization, Education

Editors
William G. Vogt
Martin 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
ACKNOWLEDGMENTS

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 Q. Russian
T. Akgul T. N. Daniel N. Jabinoun S. S. Saab
J. M. Alhamoud X. Dong A. H. Kurn S. Tadjudin
A. Ali A. El Hajj P. A. Kwan C. T. Weissman
D. Breitenstein J. Garcia G. Peck 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. Smaan, 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 Tamerine, Sandra Welsberg, 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 true 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, XMN (ExtractMin), XMAM (ExtractMax), and NEAR on a set of keys. A modification to one of 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 realization 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 a variety of computational tasks such as signal and image processing, matrix arithmetic, graph algorithms, geometric algorithms, relational database operations [4,9,9,10]. 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, XMN, XMAM, and NEAR. Before we describe these tree structured Dictionary Machines, we define the effect of the various operations. Our notations follow [5]. 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:

\[\text{INSERT} (k, r):\]
\[F = (F - F(k)) \cup \{(k, r)\}.\]
Response is null.

\[\text{DELETE} (k):\]

2569
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 \( F \). We use \( x \) 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 Knuth.

2 Previous Designs

At least five papers propose tree-structured designs for the implementation of dictionary operations. Even though they differ in detail 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 is in addition to storing data is used as input/output port of the machine. Operations are broadcast from the root and brought to it by pipelining along the depth of the tree. Such a tree machine is a completely general purpose parallel processing engine and can be used for problem decomposition in a hierarchical way. Tree machines 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 combining the values of those functions by some associative, commutative binary operator \( [1] \). Tree structured symbolic arrays have been employed to solve a wide range of problems \( [2] \).

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

2.1 Bentley and Knuth's Machine

The basic structure of the machine \( [1] \) is illustrated in figure 1. This machine is a modified binary tree, containing three kinds of processors: circles, squares, and oval. The circular 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. Those processor 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 search for an entry with a given key. At each node we compare the key with the data of the entry, and go down the tree. If the key is smaller than the data of the entry, we go to the left child; if the key is larger than the data of the entry, we go to the right child. This process is repeated until the key is found or the end of the tree is reached.

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 a redundant, a valid pair may be destroyed, a hole will be left in the tree. To avoid the problem of redundant deletion we devised the square processor with key \( k \) simply mark its content as deleted, so that the machine can operate correctly for redundant or non-redundant operations. The problem of redundant deletion is avoided by having the square processor with key \( k \) simply mark its content as deleted,
while the other square processors do not shift their contents, this relation produces holes just as redundant insertion does. The holes are removed by an operation called COMPRESS. Even with these modifications the machine needs $O(q \log N)$ time to respond. The COMPRESS operations are generated by the root processor just like any other operation, but the hole with key $k$ will be exactly one square processor with content $(k_r, r)$. If it is in the machine there will be to the left of any square processor with content $(k_r, r)$. Square processor $S_k$ processes operations as follows.

**SEARCH($k$):**
- if $k_r = k$ and $r_i \neq *$ then answer $(k_r, r_i)$, else answer "not in".

**INSERT($k$):**
- if $k_r = k$ then $(k_r, r_i) = (k_r, r)$
- $(k_r, r_i) = (k_r, r_i, t_i, r_i)$. 

**DELETE($k$):**
- $(k_r, r_i) = (k_r, r), (k_r, r_i) = (k_r, r_i)$. 

**UPDATE($k$):**
- if $k_r = k$ and $r_i \neq *$ then $(k_r, r_i) = (k_r, r)$.

**XMN:**
- if $i = 1$ and $r_i \neq *$ then answer $(k_r, r_i)$, else answer "not in".

**COMPRESS($k$):**
- if $r_i = *$ and $r_i \neq *$ then $(k_r, r_i) = (k_r, r_i, t_i, r_i)$.

Square processor $S_k$ processes COMPRESS by asking $S_{k-1}$ to see if it contains a hole. If $S_{k-1}$ finds a hole in $S_{k-1}$ and the content of $S_{k-1}$ becomes a hole, a hole created by a redundant DELETE is pushed to the right by at most $N-1$ COMPRESS operations. Ottmann, Reinschberg, and Stockmeyer showed that if the dictionary machine executes each DELETE, then after these COMPRESS operations any initial segment of the dictionary of length $c$ never contains more than $k/2$ holes.

This machine can be further improved by using operations SEARCH and COMPRESS into a new operation called CONSEARCH, defined as follows.

**CONSEARCH($k$):**
- if $k_r = k$ and $r_i \neq *$ then answer $(k_r, r_i)$
- if $r_i \neq *$ and $r_i \neq *$ then $(k_r, r_i)$.

If every SEARCH is replaced by a CONSEARCH, an additional COMPRESS is only needed if an INSERT or a DELET is not followed by a SEARCH operation. This new operation will increase the capacity of the machine whenever retrieval occurs significantly more frequently than INSERT, DELETE, or XMN [2].

### 2.3 Ottmann et al. Machine

This machine [2] uses all the processors for storing data, and therefore the response time is reduced to $O(q \log N)$ where $q$ is the total number of pairs present in the machine. The main change in the structure of the machine is that the square processors are now stacked inside the tree as shown in Figure 4. As a result all pairs are stored in square processors whose depth in the tree is $O(q \log N)$ which leads to a response time of $O(q \log N)$. 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 no number of disadvantages.

### 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 $O(q \log N)$ before the next operation can be entered into the machine. With the proposed machine sequences of DELETE operations can be pipelined with constant interval, but INSERT operation needs an interval of $O(q \log N)$. For the insertion operation to operate correctly when asked to insert a pair which is already present, a search for that pair is initially done. The result of search operation will determine whether the pair must be inserted or not. This will increase the pipeline interval for INSERT to $O(q \log N)$.

With this machine redundant operations will not create any problems.

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 binary tree structure rather than a full binary tree structure. The machine contains three types of nodes: circle circular which broadcast data, square which stores data and compute, and oval which combines the inputs. The square 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 retrieve answers from the square processors to the root of the image tree.
3.1 SEARCH Operation

The SEARCH operation determines whether or not a pair \((k,v)\) is stored in the machine. The operation is entered into the root of the forward tree. \(\log N\) steps later it reaches all the square processors that are equal to the root of the image tree. A square processor gives an answer, and the operation is directed toward the root of the image tree. The search answer is the correct way. For example, if a triangle receives a pair from one of the other children, it should transmit that pair to its father. The root of the image tree receives the search to the image tree. 

3.2 DELETE Operation

This operation travels down the forward tree. After \(\log N\) steps it will arrive at all the square processors holding the pair. The response of the square processor is either "not in" or "in" depending on whether or not the square processor contains the pair. It is the square processor number of the deleted pair. The square processor number is the output produced by that processor.

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 an integer \(r\) we want to store that pair in the \(r^{th}\) square processor from the left. The leftmost processor is numbered zero. Let \(N\) be the number of square processors, be \(2^n\) and \(2^n - k\) for some \(k\). The binary representation of \(k\) will be used to find the pair to the \(k^{th}\) square processor on the left. The root of the forward tree is used for the bit pattern of \(k\), and the pair is not found there.

When an INSERT operation is entered into the root of the forward tree, it is analyzed a square processor different level of the machine to store the pair and the square processor number at which it is stored is stored. The square processor number is numbered zero. The square processor number is used to reach the square processor at which the pair is stored.

5. References
