This is to certify that the

thesis entitled

BINARY TREE STRUCTURES

FOR MATRIX MULTIPLICATION

presented by

Abdullah Celik Erdal

has been accepted towards fulfillment
of the requirements for

M.S. degree in Elect. Eng.

David Tsiang
Major professor

Date February 25, 1983

BINARY TREE STRUCTURES FOR MATRIX MULTIPLICATION

BY

Abdullah Celik Erdal

A THESIS

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

Department of Electrical Engineering
and
Systems Science

1983

ABSTRACT

BINARY TREE STRUCTURES FOR MATRIX MULTIPLICATION

By

Abdullah Celik Erdal

Multiple arithmetic processors are interconnected to form a binary tree structure and then utilized as a high performance special-purpose co-processor. Such co-processors may be incorporated into a digital signal or image processing system to perform such tasks as matrix multiplication, which is routinely required to compute convolutions or discrete Fourier transforms. Specifically, this research shows that, any nxn dense matrix multiplication can be performed in [(2n + log(n))] time steps by using n(2n-1) processing elements, where each processing element is designed to be either a multiplier or an adder with some additional registers.

To demonstrate the utility of this binary tree structure, implementation of the convolution and the discrete Fourier transform operations are presented. This binary tree structure may also be applied to other digital image and signal processing algorithms thus establishing the flexibility, generality, and the cost effectiveness of this structure.

To My Parents

ACKNOWLEDGMENTS

I wish to express my sincere thanks and appreciation to Dr. P.D. Fisher for his constant guidance, continuous encouragement and valuable suggestions throughout this study.

Gratitude is also expressed to the members of my thesis committee, Dr. M.A. Shanblatt and Dr. R.G. Reynolds, for their comments and suggestions.

Above all, I especially wish to thank my wife, Nil, for her endless encouragement, patience, and understanding throughout the course of this study.

This research was supported in part by NSF under Grant Number MCS79-09216.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER I

INTRODUCTION

In many areas of computer application, such as image and signal processing, the quality of the answer the computer returns is proportional to the amount of computation performed [1]. The use of general purpose computers has been common for these applications; however, the high computational throughput and data rate demanded by these applications make the conventional computers inefficient and thus impractical for many contemporary applications.

Advances in the design and fabrication of Very Large Scale Integrated (VLSI) circuits will soon make it feasible to implement computers consisting of tens or even hundreds of thousands of computing elements. For this reason, the tendency is to design cost-effective, high-performance, special-purpose devices to meet specific application requirements by employing many processing elements.

Many special purpose devices for applications that require highly parallel computing have been proposed and built. Systolic arrays, introduced by Kung [2-4], and the VLSI computing structures, introduced by Hwang [5], are such devices. Both consist of a set of interconnected cells, each capable of performing some simple operation, where each cell rhythmically computes and passes data through the system. They have simple and regular communication paths and are highly concurrent modular systems. Both authors emphasize procedures for solving linear

systems of equations and some matrix computations, such as L-U decomposition and matrix multiplication. These computations play a vital role in numerous computer applications which require high-speed computer performance. For example, matrix multiplication is an important image and signal processing operation, and it is the subject of this research.

This research develops and investigates a high performance special purpose device for matrix multiplication, which works as a co-processor attached to a host computer. It capitalizes on the properties of VLSI to achieve high throughput rates and high efficiency. Although the algorithm and the structure conform to the restrictions of VLSI technology, they can be easily implemented with pre-VLSI technology without significant performance degradation. A binary tree structure is used to obtain a simple, regular, and short communication geometry, which are considered as some of the most desired attributes of a VLSI implementation. One of the main advantages of using the binary tree structure over either hexagonally or mesh connected structures proposed by Kung [2-4] and Hwang [5], respectively, is that a tree structure uses much fewer connections between the processing elements; also, the longest distance any data must travel is considerably less than any of the other schemes, leading to higher speeds. Furthermore, only two different processing elements, namely a multiplier and an adder, are used for simple, regular, and modular design to yield cost-effective special-purpose systems.

There are many special and general purpose machines that employ binary tree-structured interconnections between the processing elements. For example, the "Inner Product Computer" discussed in a report by Swartzlander [6], is a special-purpose computational unit intended

to be used as a co-processor to compute the inner product of complex vectors by using the binary tree structures, which are very similar to the tree structured arrays investigated in this research. The "X-Tree," University of California, Berkeley, project [7], is a tree structured general purpose multi-processor computer architecture. It has the hierarchical structure of a binary tree, but extra links are employed between the nodes to enhance fault-tolerance and balance uniform message traffic [7]. The California Institute of Technology "Tree Machine," based on Browning's doctoral dissertation [8], is also a binary tree structured multiprocessor system for general purpose applications. However, this machine does not have the extra links between its processors as in the case of "X-Tree." Both of these machines are general purpose computers.

Background information is presented in Chapter II in order to review some alternative highly parallel computing structures for matrix multiplication. Some examples are also given. The third chapter of this research introduces the proposed matrix multiplication algorithm and the structure of the binary tree, where each node of the tree represents a processing element. The chapter closes with an example matrix multiplication and compares the proposed algorithm with the two algorithms introduced in Chapter II, on the basis of speedup and efficiency. The fourth chapter gives application examples for the proposed architecture and describes the processing elements, chips, and the overall system architecture. Lastly, in the concluding chapter, some of the advantages and drawbacks of this architecture are delineated. Possible extensions of this work for future research are also included.

CHAPTER II

BACKGROUND

The purpose of this chapter is to review the matrix multiplication algorithms of the two multiple special-purpose functional units introduced by Kung [2-4] and Hwang [5]. These units employ many tightly interconnected elements and are designed to be used in a highly parallel computing environment for applications such as signal and image processing.

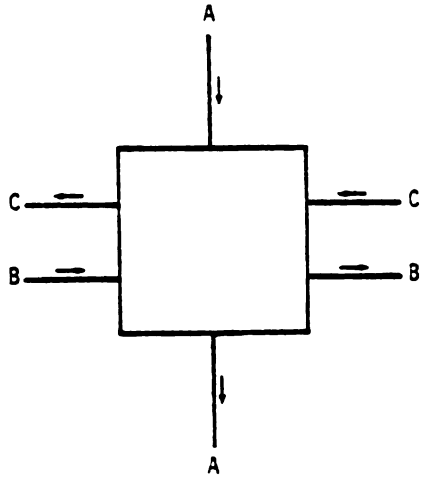
Systolic arrays and cellular arithmetic arrays are introduced in Section 2.1 and Section 2.2, respectively. Sample matrix multiplication algorithms, which may be implemented on these arrays, are also included. The chapter concludes with a discussion of some of the negative aspects of these algorithms.

2.1 Systolic Arrays

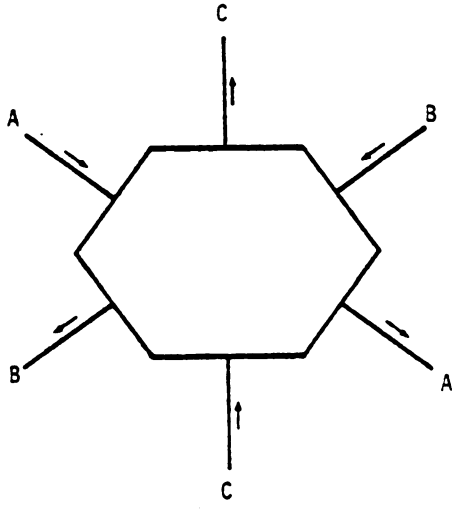
In a systolic system, data flows from the computer memory in a rhythmic fashion, passing through many tightly connected processing elements in a co-processor before returning to memory. Each processing element, called an inner product step processor, performs a single operation, namely the inner product step,

$$C \leftarrow C + A * B.$$

Figure 2.1 shows two types of geometries for this processing element (PE). Each PE has three registers R_A, R_B, and R_C, and each register



(a)



(b)

Figure 2-1: Geometries for the inner product step processor [2].

has two connections, one for input and one for output. In each time step, where one step is defined as the interval of time for any one of the PE's to complete its computation, the PE shifts the data on its input lines denoted by A, B, and C into R_A, R_B, and R_C, respectively, computes

$$R_C \leftarrow R_C + R_A * R_B$$

and makes the input values for R_A and R_B together with the new value of R_C available as outputs on the output lines A, B, and C, respectively [2]. Each processing element is connected to its neighboring PE's as shown in Figure 2.2. Using these different systolic arrays, it is possible to design systolic systems or systolic co-processors for computations for such applications as band matrix multiplication, L-U decomposition, and for solving triangular linear equations [2-4].

For example, consider the problem of multiplying a matrix $A = (a_{ij})$ with a vector $x = (x_1,\ldots,x_0)$ (see Figure 2.3). The product term, $y = (y_1,\ldots,y_0)$ can be computed by the following equations [2]:

$$y_i^{(1)} = 0,$$

$$y_i^{(k+1)} = y_i^{(k)} + a_{ik}x_k,$$

$$y_i = y_i^{(n+1)}$$

If A is an nxn band matrix with bandwidth

$$W = P + Q - 1,$$

(See Figure 2.4 for the case when P = 2 and Q = 3.), then the above equations can be evaluated by pipelining the x_i and y_i through a linearly connected systolic array consisting of w inner product step PE's in (2n + w) time steps [2].

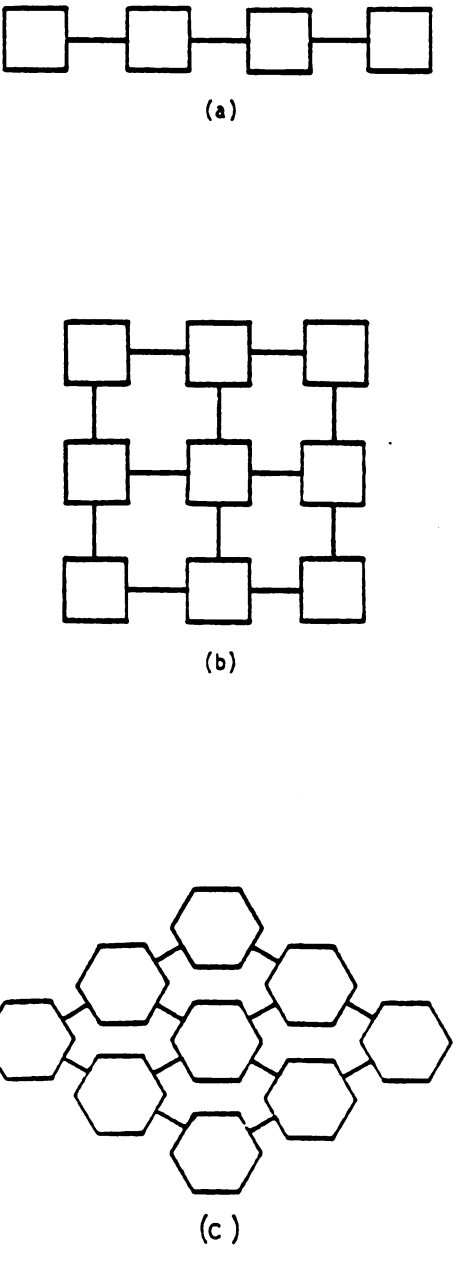


Figure 2-2: Mesh-connected systolic arrays [2].

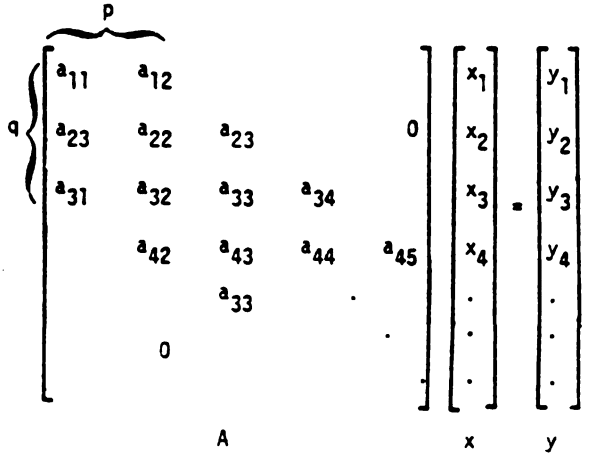


Figure 2-3: Multiplication of a vector by a band matrix with p = 2 and q = 3 [2].

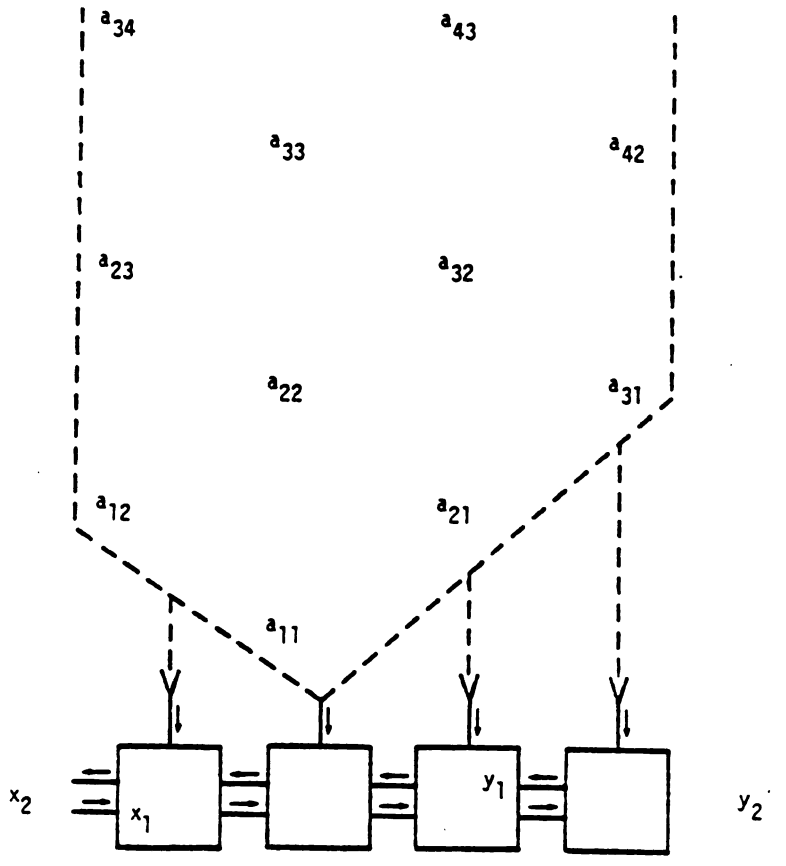


Figure 2-4: The linearly connected systolic array for the vector multiplication problem in Figure 2-3 [2].

Another example is the multiplication of two nxn band matrices with bandwidth w_1 and w_2. The matrix product $C = (c_{ij})$ of $A = (a_{ij})$ and $B = (b_{ij})$ can be computed as follows [2]:

$$c_{ij}^{(1)} = 0,$$

$$c_{ij}^{(k+1)} = c_{ij}^{(k)} + a_{ik}b_{kj},$$

$$c_{ij} = c_{ij}^{(n+1)}.$$

Band matrices A, B, and C, shown in Figure 2.5, are pipelined through a systolic array of $(w_1 * w_2)$ hex-connected PE's. The interconnection network and data flow are shown in Figure 2.6. Each c_{ij} is able to accumulate all of its terms before it passes through the upper boundaries. If A and B are nxn band matrices with bandwidths w_1 and w_2, respectively, then a systolic array of $(w_1 * w_2)$ hex-connected PE's can compute the result C in $[3n + \min(w_1, w_2)]$ time steps. If A and B are nxn dense matrices, then $(3n^2 - 3n + 1)$ hex-connected PE's can compute the result in C in $5(n - 1)$ time steps [4].

As described, the systolic array architectures provide the capability for realizing a number of important matrix operations. In addition to achieving a high computational rate by means of pipelining and concurrent computation, these arrays are characterized by having a simple, regular and short communication geometry.

2.2 VLSI Cellular Arithmetic Arrays

Processing elements, as in the case of systolic arrays, compute the inner product,

$$C \leftarrow C + A * B$$

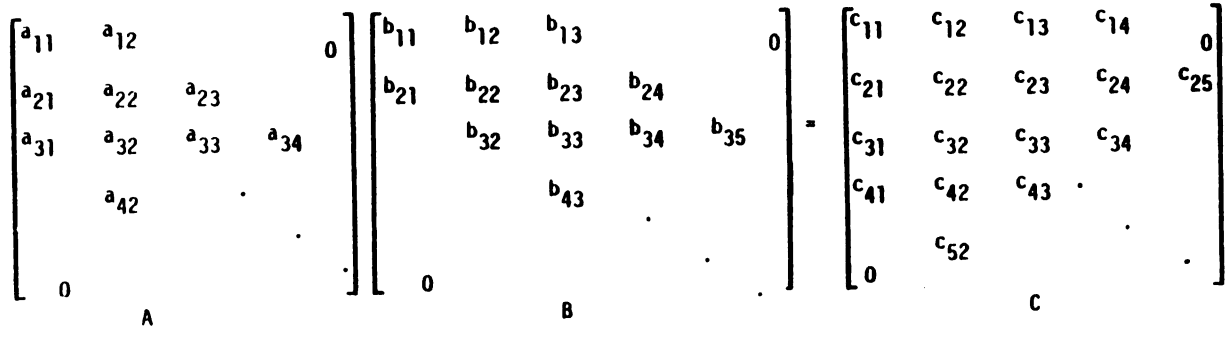


Figure 2-5: Band matrix multiplication [2].

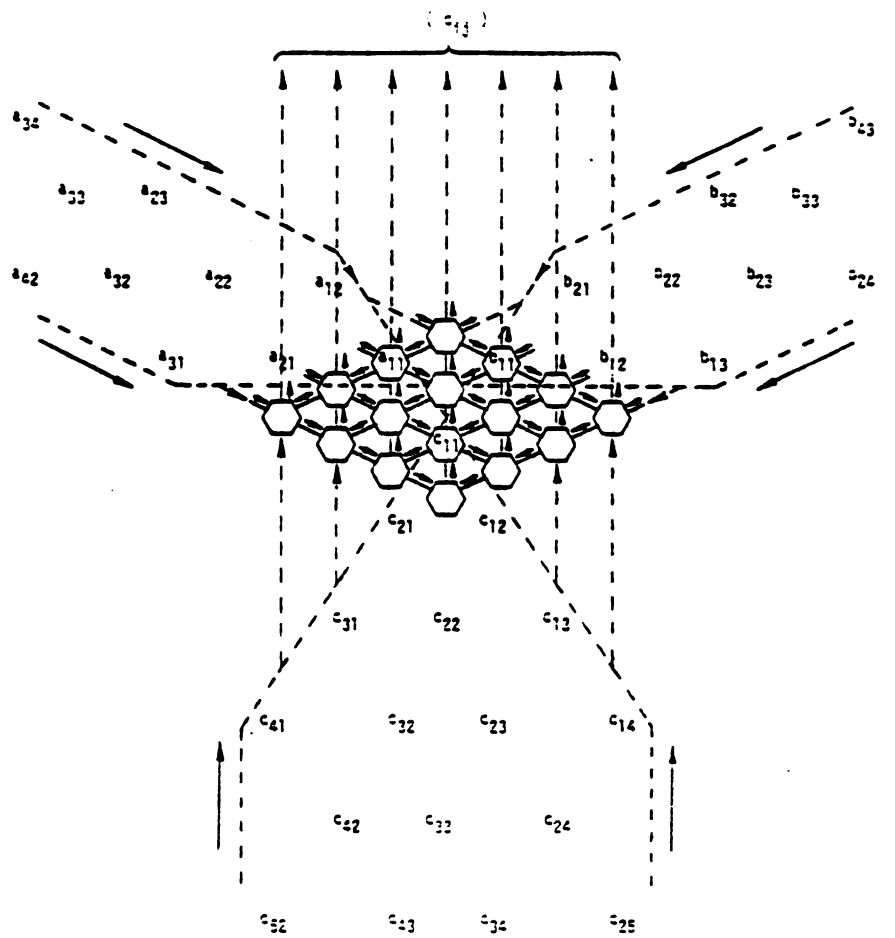


Figure 2-6: The hex-connected systolic array for the matrix multiplication problem in Figure 2-5 [2].

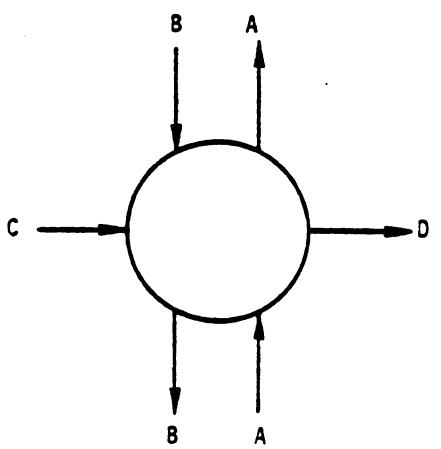


Figure 2-7: The additive multiplication cell [5].

and have three inputs and three outputs as shown in Figure 2.7. Although it is not specified in [5], presumably each processing element has a set of three registers that relates to the input and output lines of A, B, and C. Each processing element is connected to its neighboring PE to form a short communication path.

Consider the problem of multiplying two nxn dense matrices $A = (a_{ij})$ and $B = (b_{ij})$ (see Figure 2.8 for $n = 3$), where the product coefficients

$$c_{ij} = \sum_{k=1}^{n} (a_{ik} b_{kj})$$

for all i and j [5]. The elements of matrices A and B are fed from the lower and upper input lines in a pipelined fashion, one skewed row or one skewed column at a time, as in Figure 2.9 [5]. Some dummy zero inputs are interspaced with the matrix elements to ensure correct results. Multiplying two nxn dense matrices requires $n(2n - 1)$ processing elements and it takes $(4n - 2)$ time steps to produce the product matrix $C = (c_{ij})$.

2.3 Discussion

All the PE's for both algorithms are kept busy all the time, either by performing the inner product computation or by simply passing the data to their neighbors. Other computations, such as matrix inversion and L-U decomposition, can also be implemented by using one or two different PE's and different array architectures. Both of these VLSI arrays are expandable to allow modular growth. However, they both have some drawbacks. For example, they need data skewing, and control of every input and output data, in order to accomplish their correct

$$A \times B = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{32} & b_{32} & b_{33} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{bmatrix} = C$$

Figure 2-8: 3x3 square matrix multiplication.

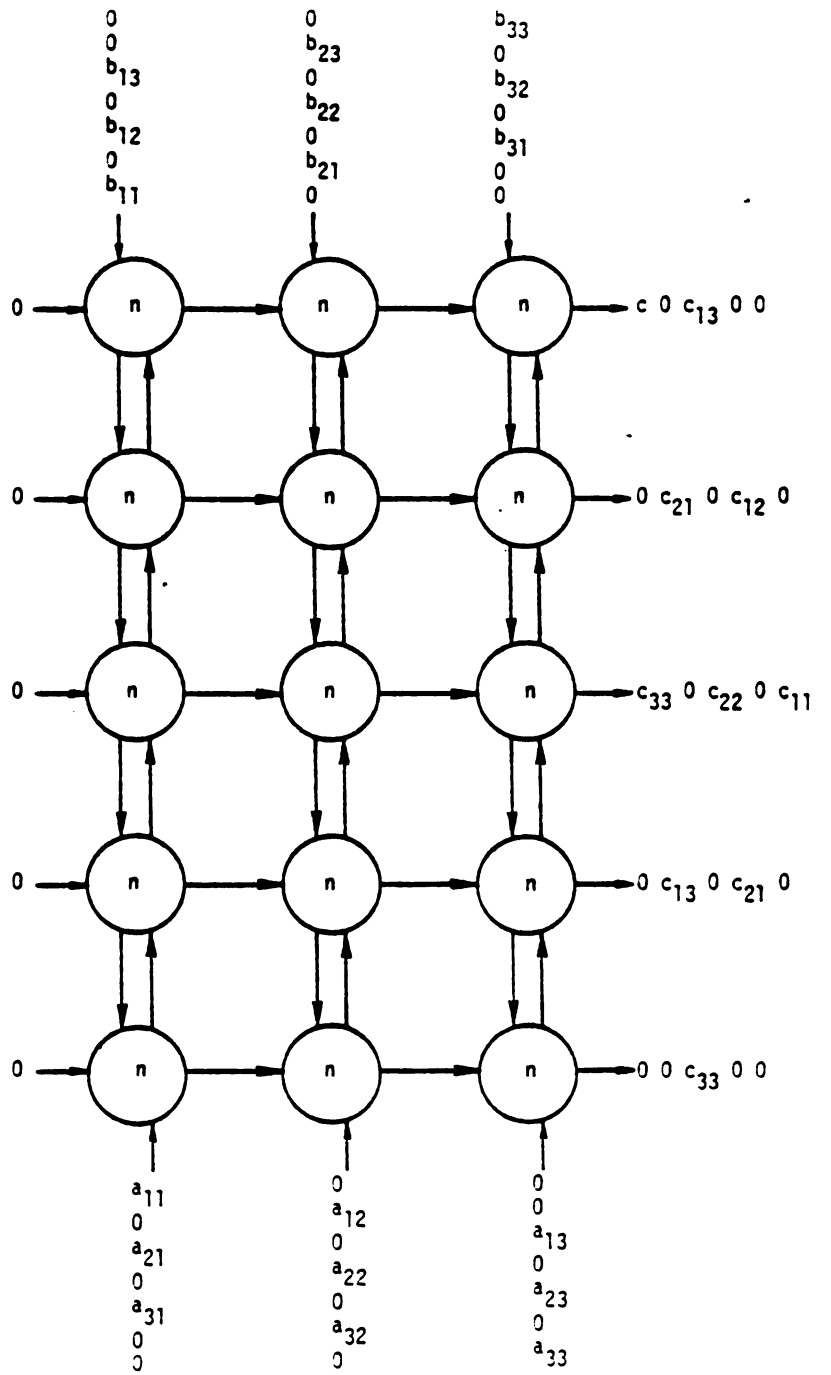


Figure 2-9: VLSI array for pipelined multiplication of two 3x3 dense matrices [5].

timing. This increases the circuit complexity and introduces additional delays. One other negative aspect of these arrays is that to accomplish the multiple use of every element of matrices A and B, the result of each processing element, as well as the elements of A and B, are pipelined through the array. However, this requires the use of more interconnection lines between the processing elements, and hence more chip area. And the number of required registers also increases for the same reason.

CHAPTER III

AN ALGORITHM FOR MATRIX MULTIPLICATION

This chapter provides the requisite groundwork and the algorithm for the development of a highly parallel computing machine. Basically, this machine is a tree structured array and may be classified as a multiple special-purpose functional unit. First, the processing elements are defined and the basic tree structure is outlined in Section 3.1. In the next section, a new matrix multiplication algorithm is introduced, and a sample matrix multiplication is provided. Section 3.3 concludes with a discussion of the space-time complexity of this tree structure and compares its speedup and efficiency with those of the structures reviewed in Chapter 2.

3.1 Functional and Structural Description

* Processing Elements (PE)--As mentioned earlier there are two types of processing elements, namely, a multiplier and an adder. Each of these PE's has three connection lines A, B, and C, as shown in Figure 3.1, where lines A and B are the input lines and the line C is the output line. Each PE performs only a single operation, either

$$C \leftarrow A \times B$$

or

$$C \leftarrow A + B.$$

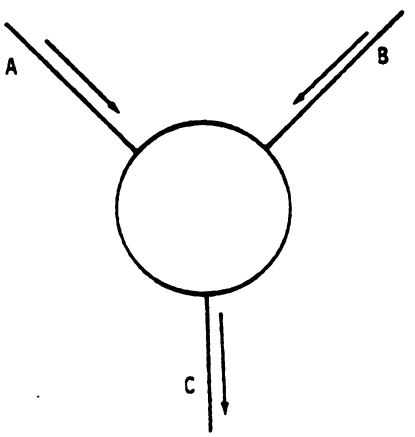


Figure 3-1: A processing element (PE).

In each time step, a processing element takes the data on its input lines A and B, performs its operation, and latches its result into register R_C, where R_C is directly connected to the output line C. All outputs are latched and the logic is clocked so that when one PE is connected to another, the changing output of one during a time step will not interfere with the input to another during this time step.

* Tree Structure--The tree structure, shown in Figure 3.2, has seven nodes and three levels. The first node is called the root node, and it is accepted as the level one of the binary tree. The nodes 4, 5, 6, and 7 are called leaf nodes or input nodes. This tree structure is redrawn in Figure 3.3 to show the actual data flow. As shown in this figure, data is pipelined downward from the leaf nodes to the root node, where the output of the root node is the final result of the computation. Only the input nodes (leaf nodes) are represented by the multipliers; the rest of the nodes are all adders.

3.2 An Algorithm for a Dense Matrix Multiplication on Trees

Let $A = (a_{ij})$, $B = (b_{ij})$, and $C = (c_{ij})$ be nxn dense matrices, where C is the product term, which can be computed in $(1 + \log n)$ time steps by using $n^2(2n - 1)$ PE's, and by employing the type of tree structures shown in Figure 3.3. (Note: the logarithmic function will be used in base two throughout this paper.) All the rows of the matrix A and all the columns of matrix B are multiplied together in one time step to produce all the partial products of the resultant matrix C. Then the produced partial products are added in $(\log n)$ time steps to obtain the final result. Consequently, very high speedup is obtained over the other algorithms. However, it requires too many PE's and

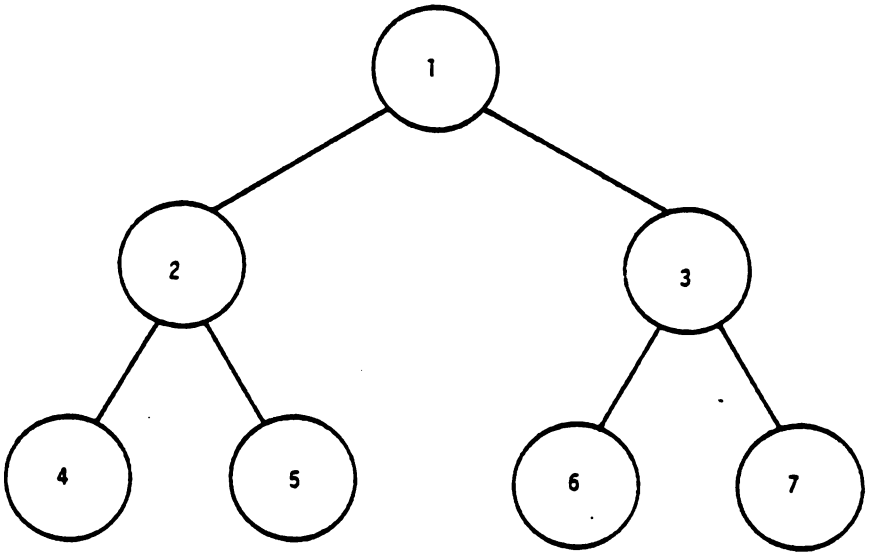


Figure 3-2: The tree structure.

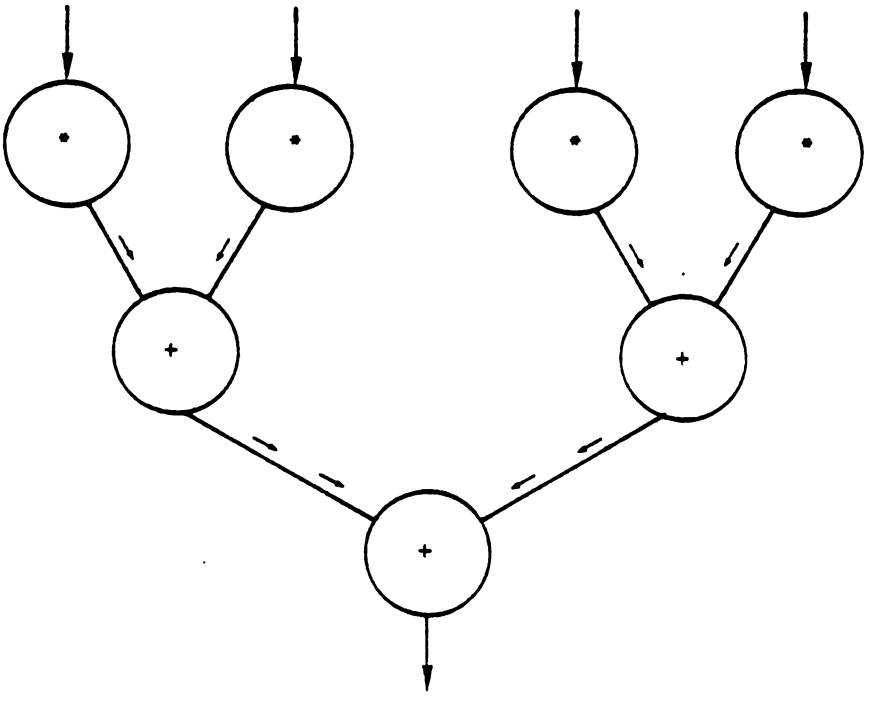


Figure 3-3: The tree structure with PE's and data flow shown.

very large I/O bandwidth even for very small matrix multiplications. Since this would be an unrealistic assumption, a different approach will be taken to obtain high speedup with many fewer PE's.

In the remainder of this paper, it is assumed that A, B, and C are all nxn dense matrices, where $n = 2^k$ for k = 1,2,3,..., although this is not an essential assumption. With this assumption, we can define some terms that will be used throughout this paper as follows:

p = Total number of PE's that will be used for a given nxn matrix multiplication.

t = Total amount of time it takes for a given nxn matrix multiplication to be computed.

Then, it will be shown in this section that

$p = n(2n - 1)$;

$t = (2n + \log n)$;

where

$n = 2^k$ for k = 1,2,3,...

for nxn dense matrix multiplication.

The first modification needed is to reduce the number of input lines for the input nodes (multipliers) from two to one. This will not only reduce the I/O bandwidth requirements, but also reduce the number of pins for a chip implementation. However, the price that we pay is the reduction of the overall speed. The second modification is to include one extra register, R_B, for each multiplier to use as a buffer to hold the elements of the matrix B. This register will be implemented inside the multiplier PE. Finally, it will be assumed that n-way memory interleaving is incorporated in the host computer.

With this memory model, one word can be fetched from each of the n memory units at once in one clock period, then the effective memory bandwidth is n [9].

With all the given modifications and the assumptions, it is now possible to load all the elements of the matrix B into the register R_B of n^2 multiplication PE's in n time steps, one column at a time, where each column has n elements. Note that if the I/O bandwidth is larger, more data can be accessed in one time step. Also, it is assumed that memory access time is equal to one time step, although it may be possible to access the memory more than once in one time step, which will speed up the overall computation time. Furthermore, for simplicity, it will be assumed that each of the PE's, multiplier or adder, have equal computation times.

Consider a 4x4 matrix multiplication example to illustrate this algorithm, where matrices A, B, and C are shown in Figure 3.4. Individual elements of the matrix C are shown in Figure 3.5a.

Each row vector will be computed separately. For this reason n Column Vector Computation Units (abbreviated as CVCU) will be used, where each CVCU will contain a total of (2n - 1) PE's bringing the total number of PE's to n(2n - 1). These (2n - 1) PE's in CVCU's consist of n multiplier PE's and (n - 1) adder PE's. Sepcifically, for this example, 4 CVCU's have been used. Each CVCU contains 7 PE's, where 4 of them are multiplier and 3 of them adder PE's. First consider the CVCU-I which produces column vector-I of resultant matrix. As shown in Figure 3.5a, the equations C_{11}, C_{21}, C_{31}, and C_{41} have four elements in common to all of them. For this reason, first these common elements, namely b_{11}, b_{21}, b_{31}, and b_{41}, are loaded into the

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}$$

$$B = \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{bmatrix}$$

$$C = \begin{bmatrix} c_{11} & c_{12} & c_{13} & c_{14} \\ c_{21} & c_{22} & c_{23} & c_{24} \\ c_{31} & c_{32} & c_{33} & c_{34} \\ c_{41} & c_{42} & c_{43} & c_{44} \end{bmatrix}$$

Figure 3-4: Matrices A, B, and C where A x B = C.

Column I.

$$c_{11} = a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} + b_{14}b_{41}$$

$$c_{21} = a_{21}b_{11} + a_{22}b_{21} + a_{23}b_{31} + b_{24}b_{41}$$

$$c_{31} = a_{31}b_{11} + a_{32}b_{21} + a_{33}b_{31} + b_{34}b_{41}$$

$$c_{41} = a_{41}b_{11} + a_{42}b_{21} + a_{43}b_{31} + b_{44}b_{41}$$

Column II.

$$c_{12} = a_{11}b_{12} + a_{12}b_{22} + a_{13}b_{32} + b_{14}b_{42}$$

$$c_{22} = a_{21}b_{12} + a_{22}b_{22} + a_{23}b_{32} + b_{24}b_{42}$$

$$c_{32} = a_{31}b_{12} + a_{32}b_{22} + a_{33}b_{32} + b_{34}b_{42}$$

$$c_{42} = a_{41}b_{12} + a_{42}b_{22} + a_{43}b_{32} + b_{44}b_{42}$$

Column III.

$$c_{13} = a_{11}b_{13} + a_{12}b_{23} + a_{13}b_{33} + b_{14}b_{43}$$

$$c_{23} = a_{21}b_{13} + a_{22}b_{23} + a_{23}b_{33} + b_{24}b_{43}$$

$$c_{33} = a_{31}b_{13} + a_{32}b_{23} + a_{33}b_{33} + b_{34}b_{43}$$

$$c_{43} = a_{41}b_{13} + a_{42}b_{23} + a_{42}b_{33} + b_{44}b_{43}$$

Column IV.

$$c_{14} = a_{11}b_{14} + a_{12}b_{24} + a_{13}b_{34} + b_{14}b_{44}$$

$$c_{24} = a_{21}b_{14} + a_{22}b_{24} + a_{23}b_{34} + b_{24}b_{44}$$

$$c_{34} = a_{31}b_{14} + a_{32}b_{24} + a_{33}b_{34} + b_{34}b_{44}$$

$$c_{44} = a_{41}b_{14} + a_{42}b_{24} + a_{43}b_{34} + b_{44}b_{44}$$

Figure 3-5a: Individual elements of the matrix C.

$$c_{11} = x_{11} + x_{12} + x_{13} + x_{14}$$

$$c_{21} = x_{21} + x_{22} + x_{23} + x_{24}$$

$$c_{31} = x_{31} + x_{32} + x_{33} + x_{34}$$

$$c_{41} = x_{41} + x_{42} + x_{43} + x_{44}$$

Figure 3-5b: Redefined version of the first row of matrix C.

R_B registers as shown in Figure 3.6a. Also, for simplicy, C_{11}, C_{21}, C_{31}, and C_{41}, are redefined as shown in Figure 3.5b.

At time steps 5, 6, 7, and 8, matrix A is pipelined one row at a time, where each row has n elements, and for an nxn dense matrix there are n rows. So, it takes n time steps to pipeline all the rows of nxn dense matrix. After the pipelining of the last row, it takes (log n) time steps to merge the partial results of each PE. At each time step each PE takes its operands, does computations on these operands, and stores the result into the R_C. Like a systolic array each PE takes data in and pumps data out at every time step.

Only the CVCU-I is shown in Figure 3.6. CVCU's-II, -III, and -IV, also perform the same computations at time steps 1 through 10. Note that the same row elements are needed to do computations on these CVCU's. For example, at time step 5, the first row of matrix A is inputted to CVCI-I, -II, -III, and -IV simultaneously. At time steps 6, 7, and 8, row II, III, and IV of matrix A are pipelined through every CVCU, respectively. From this, it is concluded that, only n elements are needed at each time step to do all the computations as shown in Figure 3.7.

Note that band matrix multiplications and matrix-vector multiplications can easily be implemented on these tree structures with the same algorithm. For example, an nxn and n-element vector multiplication can be computed using n(2n - 1) PE's in (n + 1 + log n) time steps.

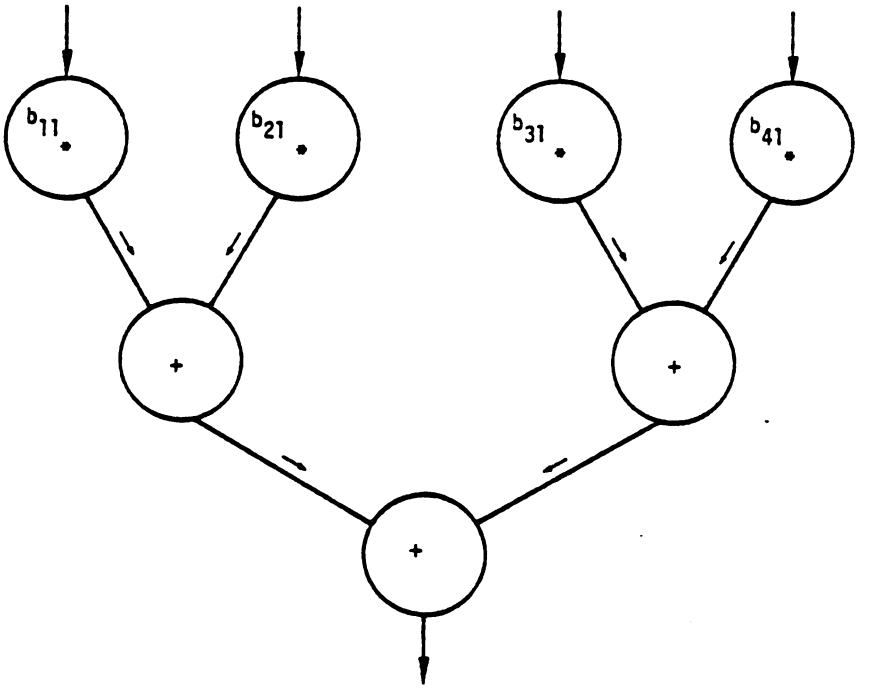


Figure 3-6a: Time step 4.

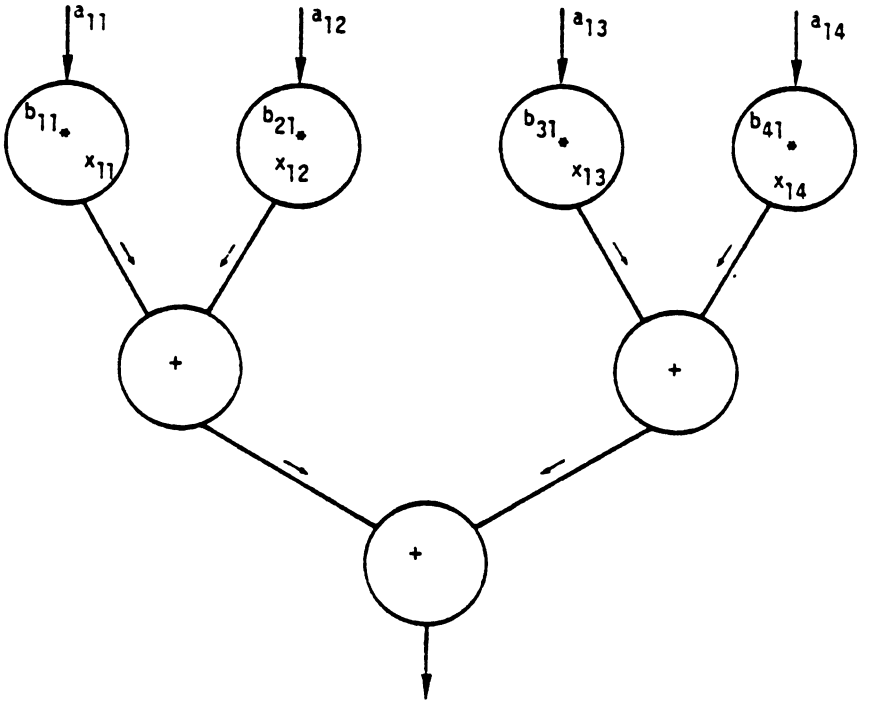


Figure 3-6b: Time step 5.

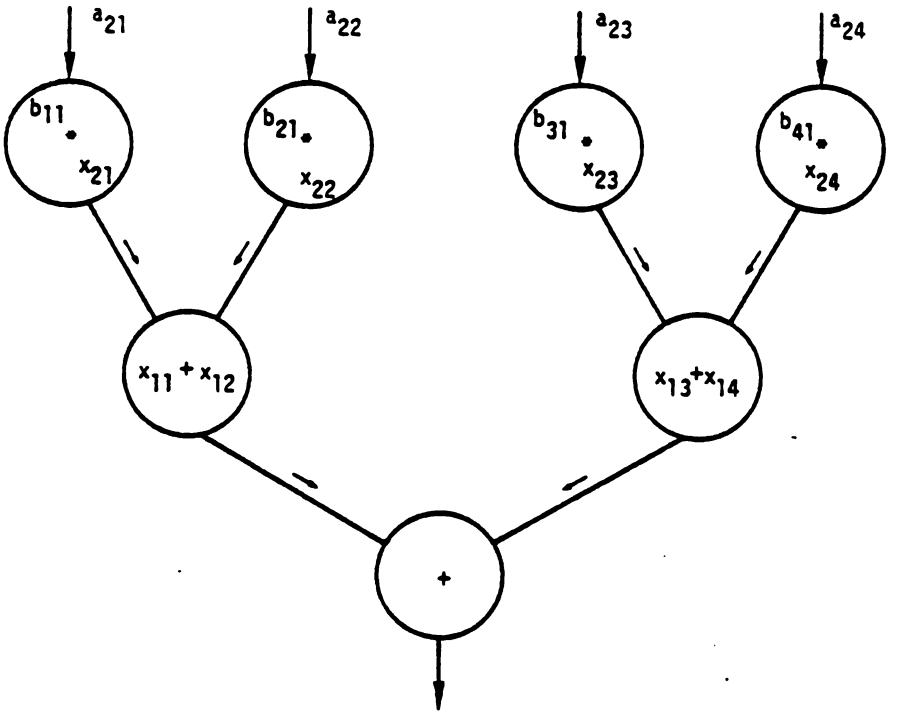


Figure 3-6c: Time step 6.

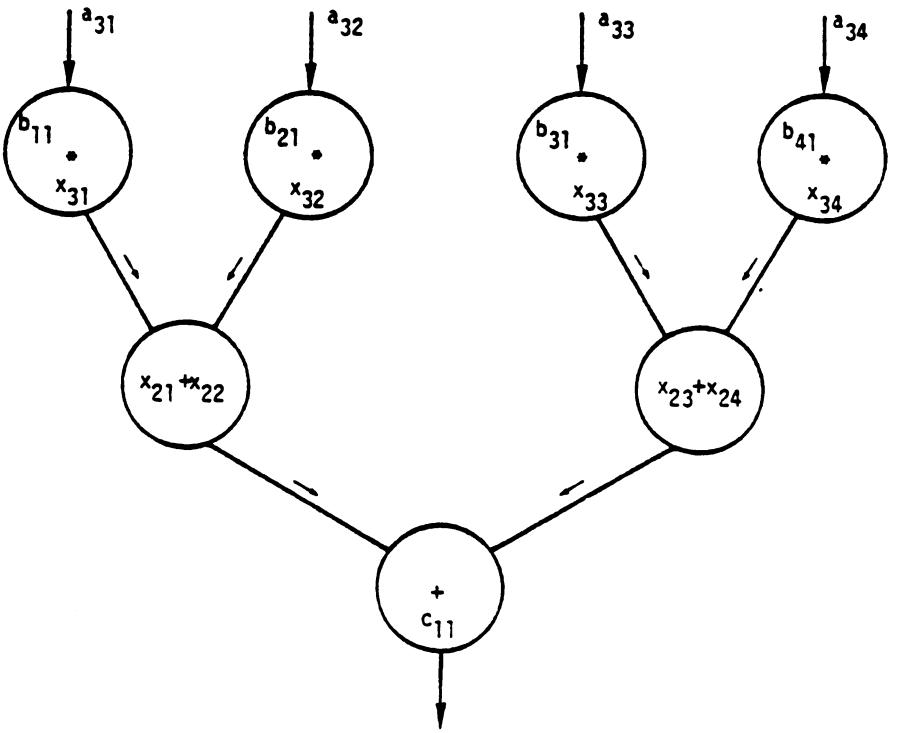


Figure 3-6d: Time step 7.

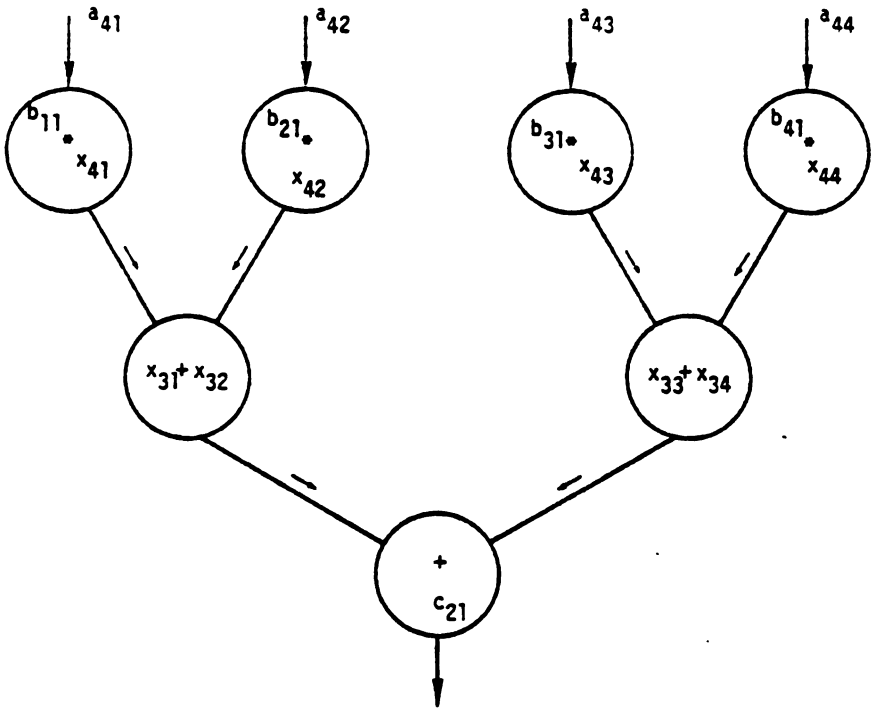


Figure 3-6e: Time step 8.

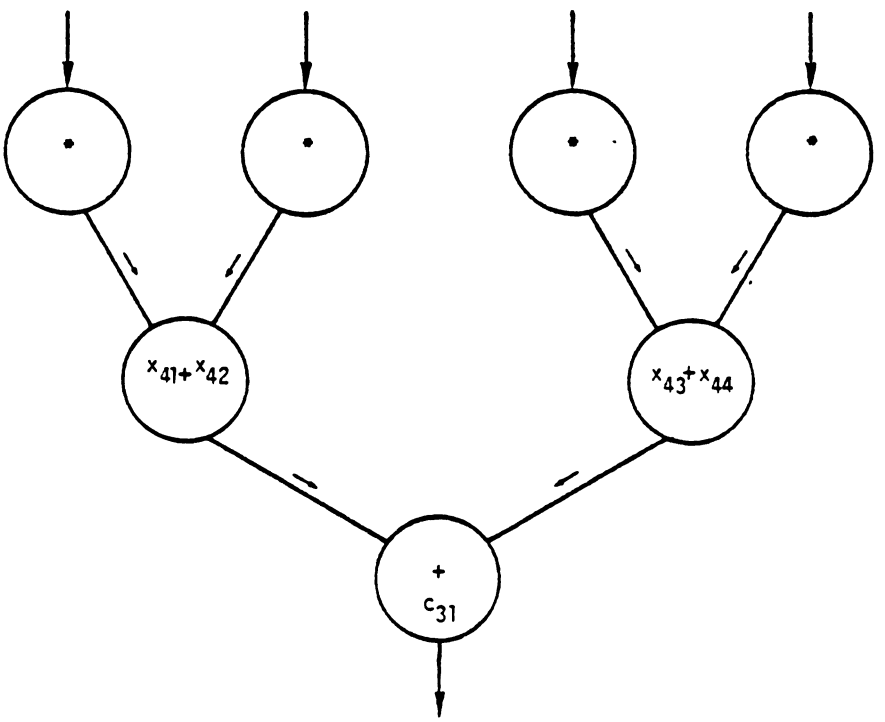


Figure 3-6f: Time step 9.

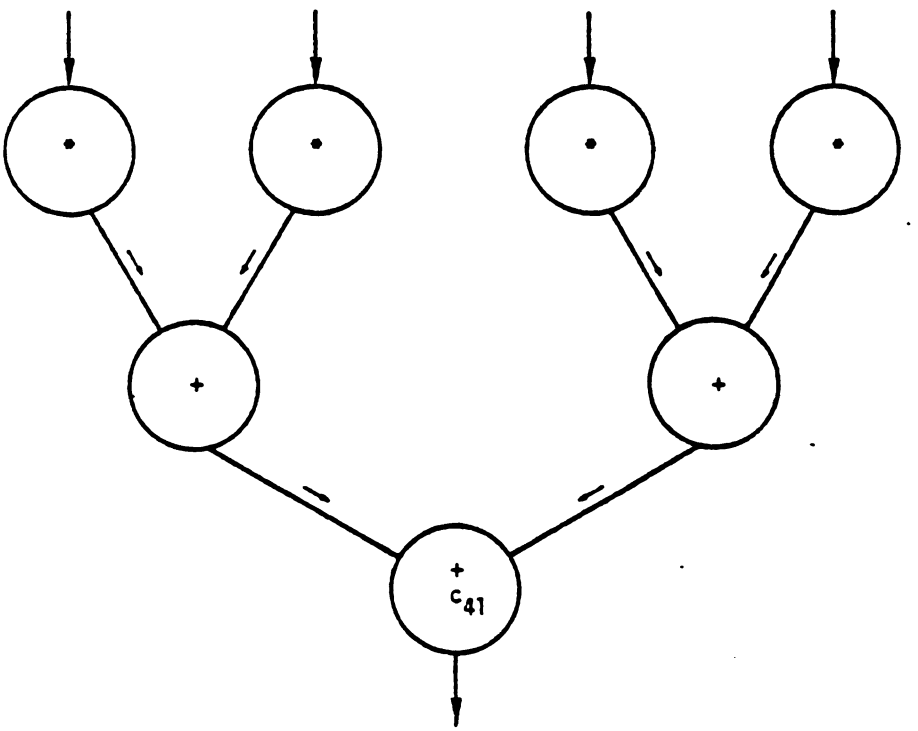


Figure 3-6g: Time step 10.

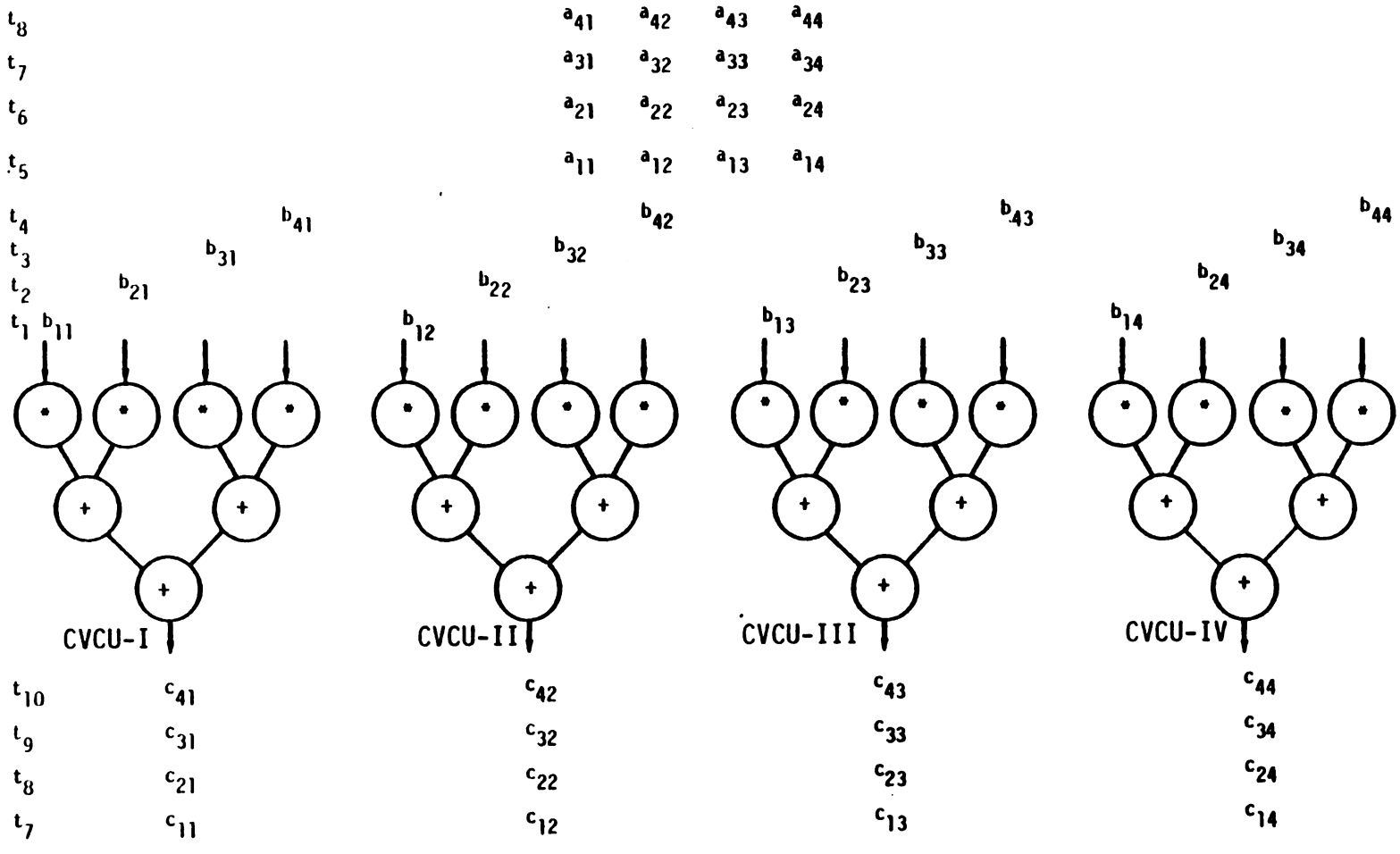


Figure 3-7: Computation table.

3.3 Discussion and Performance Analysis

As shown in Section 3.2, nxn dense matrix multiplication can be computed using n(2n - 1) PE's in (2n + log n) time steps. Note that for the algorithm proposed in this paper, there is no need for data skewing or a data alignment network because the algorithm requires only row access to the matrices A and B for the matrix multiplication of AxB.

The number of processing elements and the total time steps required for a given nxn dense matrix are shown in Figures 3.8 and 3.9, respectively. Figures 3.10a and 3.10b show the comparison of this algorithm with Kung's 2-4 and Hwang's [5] algorithms in terms of number of required PE's for the range of n = 2 to n = 2048. Figure 3.11a, b, and c depicts the comparison of these algorithms in terms of time steps that are necessary for any given nxn dense matrix for the same range. As shown in Figures 3.10 and 3.11, the algorithm proposed in this paper obtains higher speeds with fewer processing elements over the other two algorithms. Note also, it is possible to use fewer PE's to compute a given nxn dense matrix, however, this causes an increase in total time steps. For example, 1 CVCU can be employed instead of n to compute a given nxn matrix using (2n - 1) PE's in (2n + n + log n) time steps, which is still faster than a sequential processor. Combinations of the number of PE's and the total time steps can be easily arranged so that the computation rate of the overall system matches with the I/O bandwidth of the host computer.

Speedup and efficiency provide more insight into the performance of these algorithms, where speedup S, is defined as the ratio of the serial computation time to that of parallel computation time; efficiency,

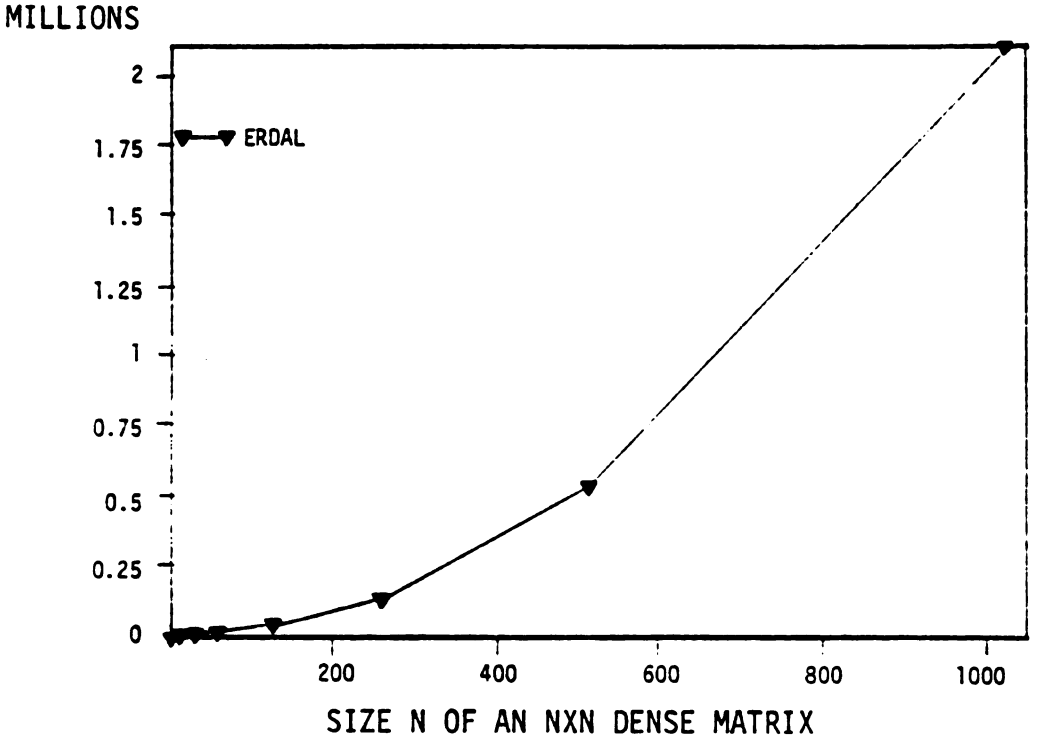


Figure 3-8: Number of required processing elements vs. N.

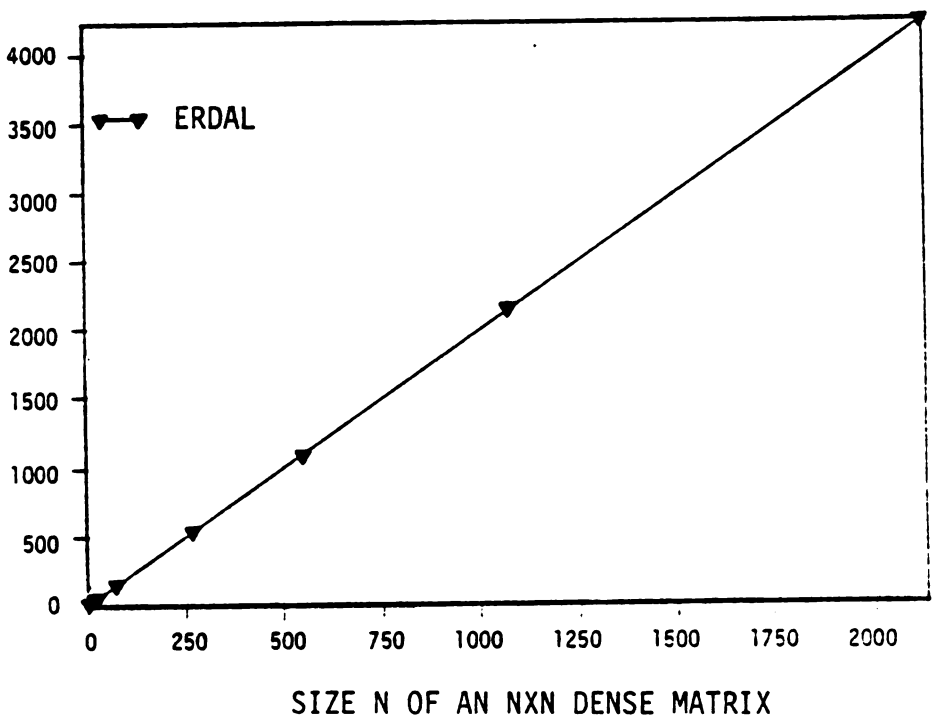


Figure 3-9: Total computation time of an NxN dense matrix.

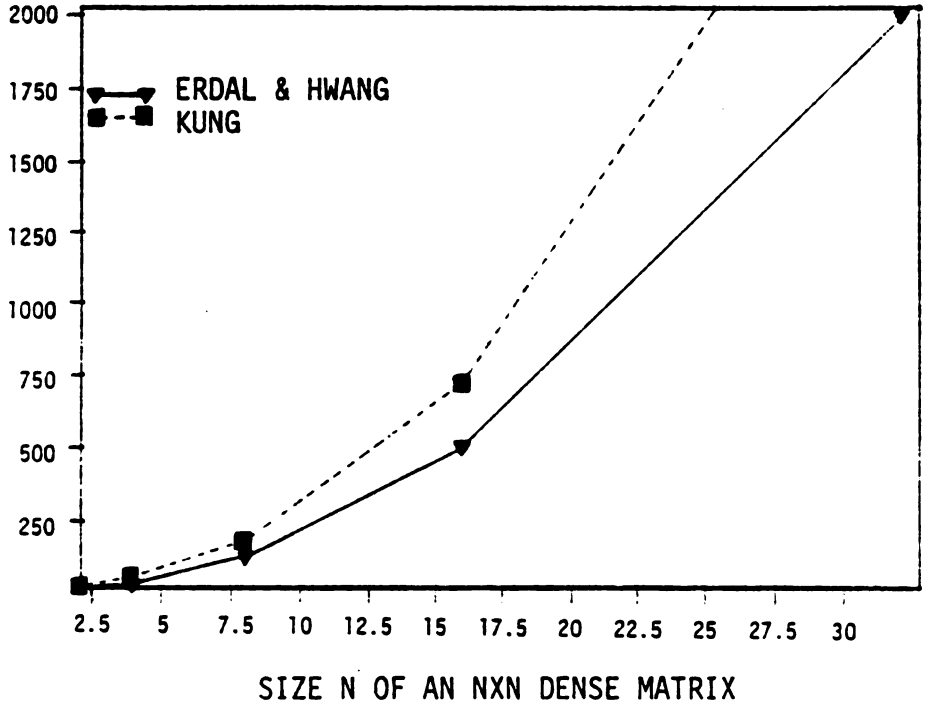


Figure 3-10a: Number of required processing elements vs. N.

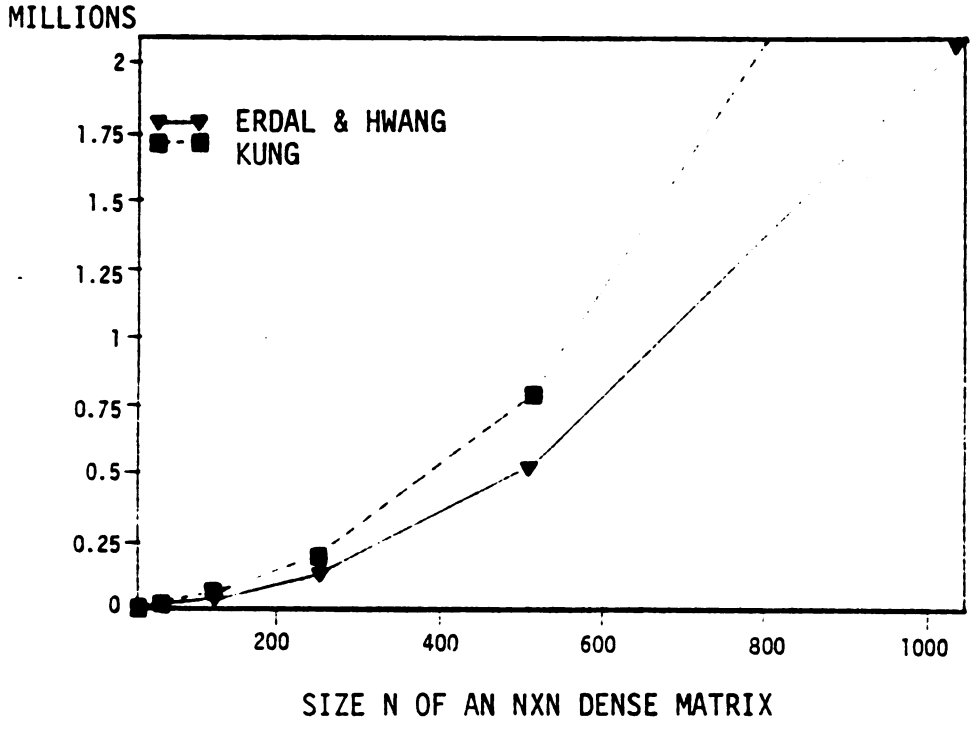


Figure 3-10b: Number of required processing elements vs. N.

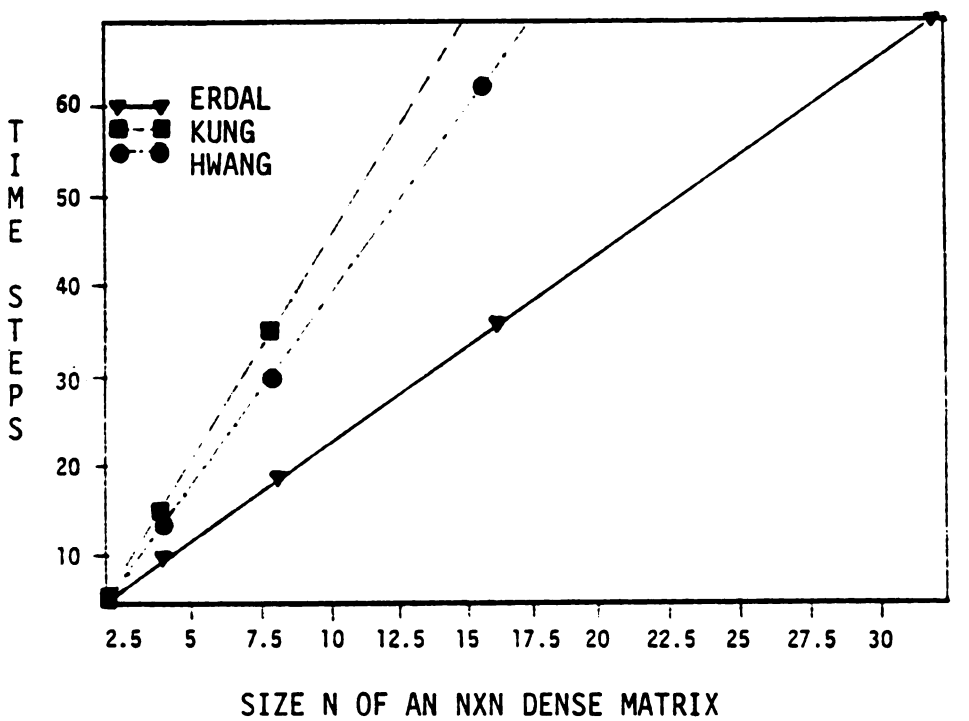


Figure 3-11a: Total computation time of an NxN dense matrix.

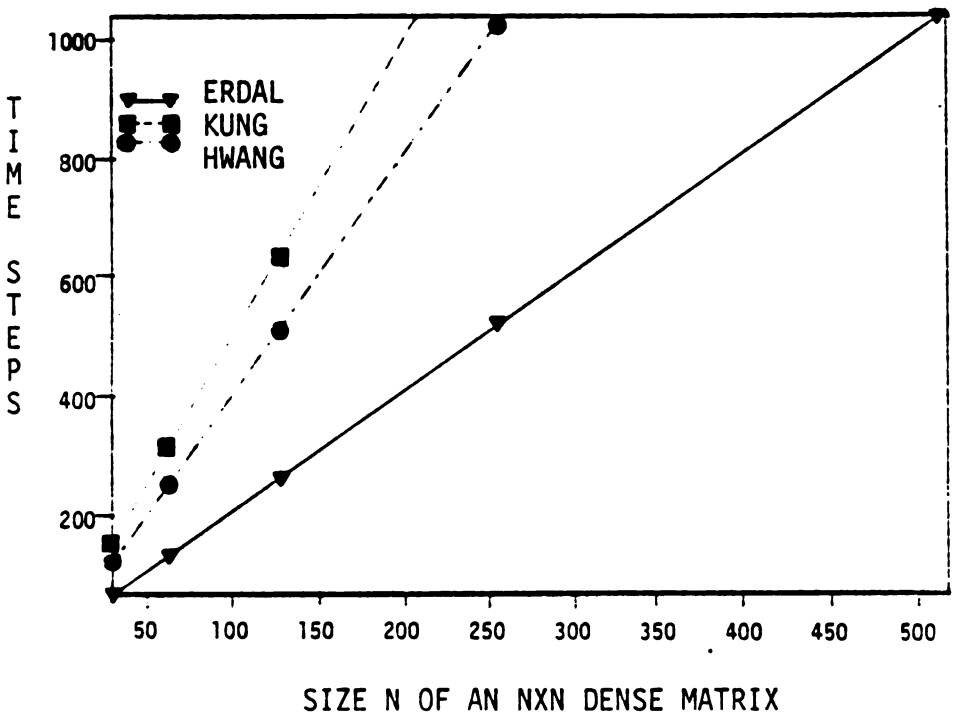


Figure 3-11b: Total computation time of an NxN dense matrix.

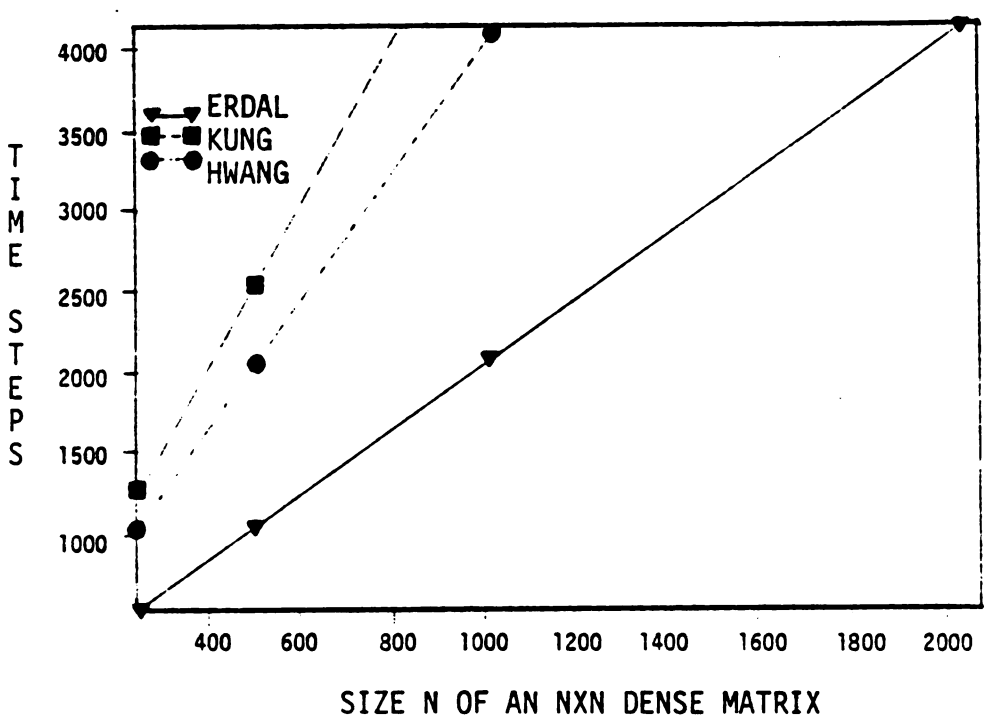


Figure 3-11c: Total computation time of an NxN dense matrix.

E, is regarded as the ratio of the actual speedup to the maximum possible speedup using p processing elements, as shown below [10,11]:

$$S = t_s / t_p \geq 1$$

$$E = S / p \leq 1$$

where t_s and t_p are serial and parallel computation times, respectively. Figures 3.12a and 3.12b illustrate the speedup of each algorithm for the range of n = 2 to n = 2048. Percent efficiency of these algorithms for the range of n = 2 to n = 1024 is depicted in Figure 3.13a and 3.13b. As shown in Figures 3.12 and 3.13, higher speedup and higher efficiency is obtained by the algorithm introduced in this paper. For example, for n = 32, approximately, (9 / 4) and (9 / 5) times more speedup, and (10 / 3) and (9 / 5) times more efficiency are obtained over the Kung's and Hwang's algorithm, respectively. Note that efficiency stays almost constant for n 512 for all algorithms; however, much higher speedups can still be obtained for larger values of n over the other two algorithms. One of the reasons that the tree structure performs better than the other structures is that the longest data path that one element has to travel is O(log n) instead of O(n). Another important reason for higher performance is that the tree structure makes use of all of its PE's all the time, whereas the other structures use only half or less than half of their total PE's for the actual calculation, and the rest of the PE's are used for simply passing data, most of the time. More detailed performance analysis of these algorithms can be accomplished by including other performance criteria such as, utilization and redundancy as defined in [10,11].

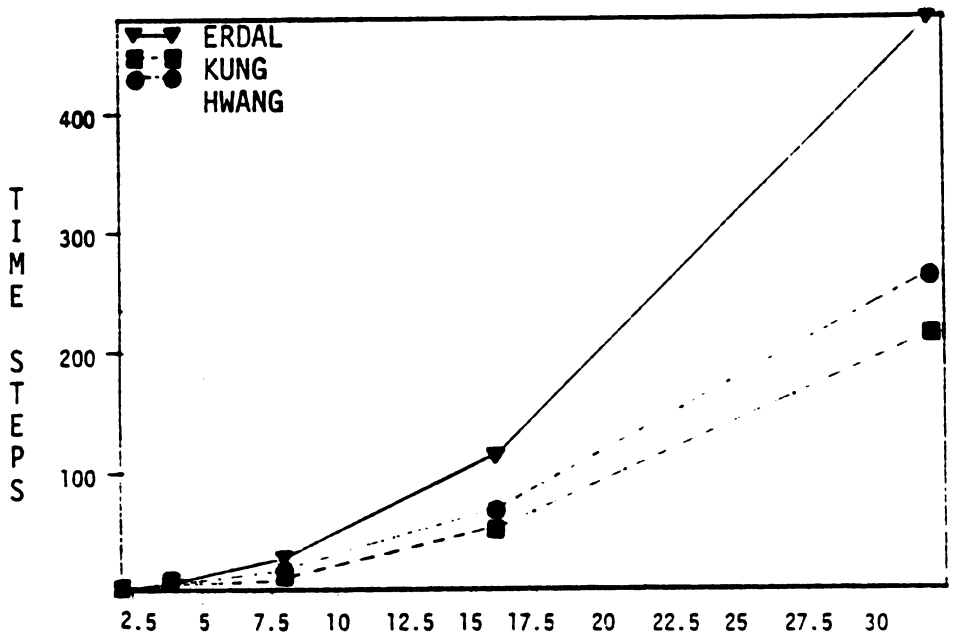


Figure 3-12a: Speedup vs. N.

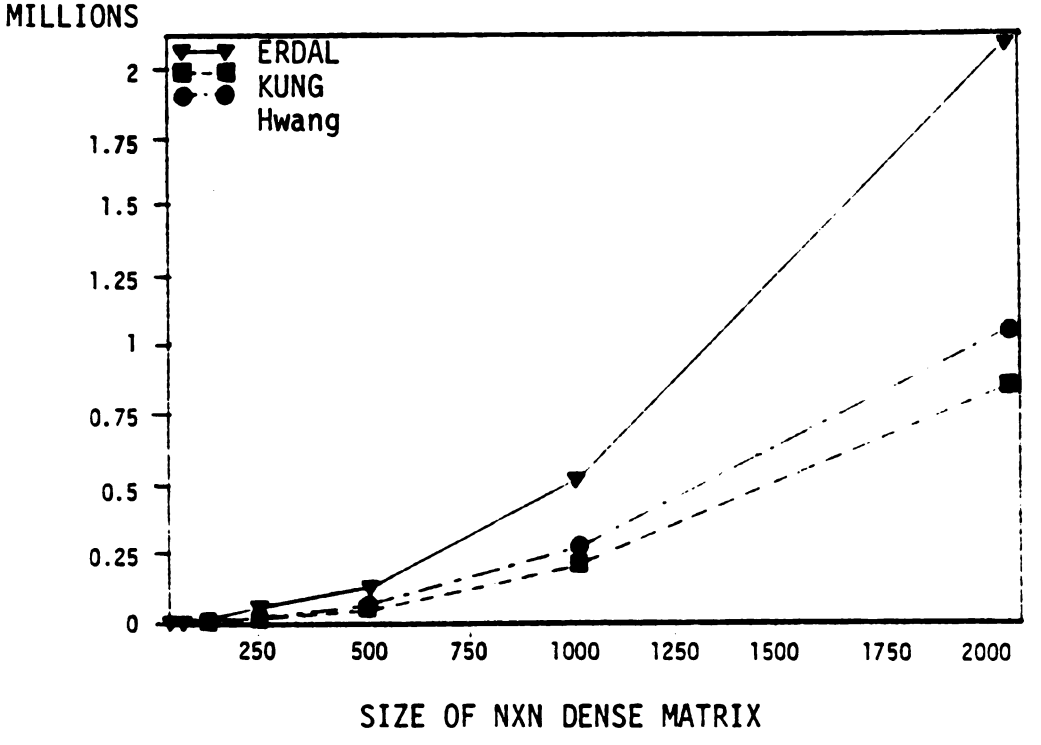


Figure 3-12b: Speedup vs. N.

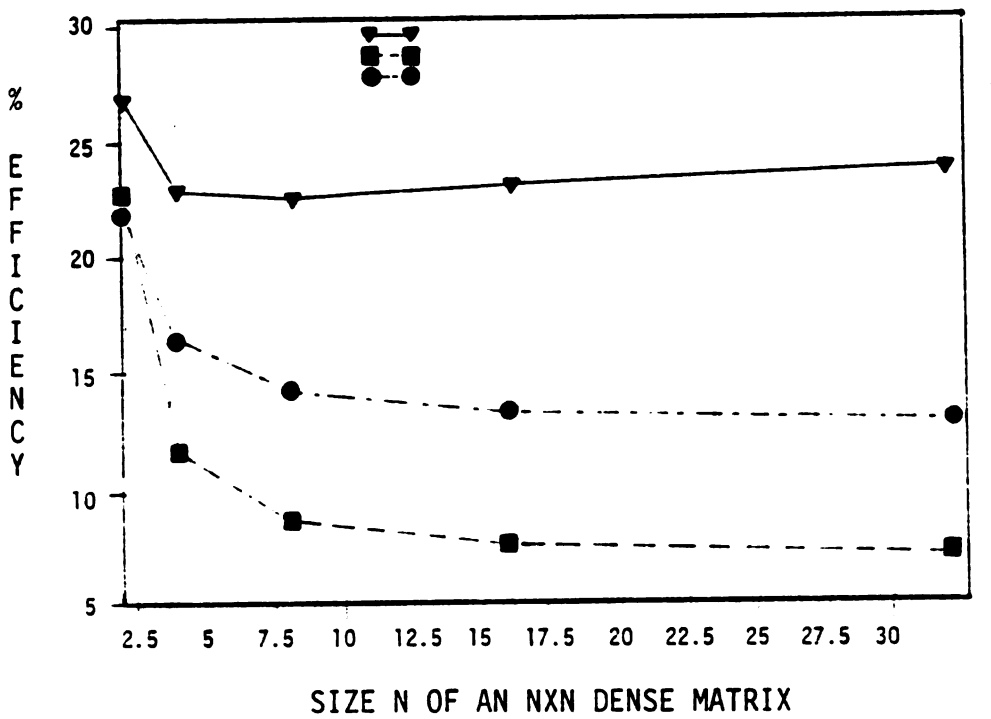


Figure 3-13a: Percent efficiency vs. N.

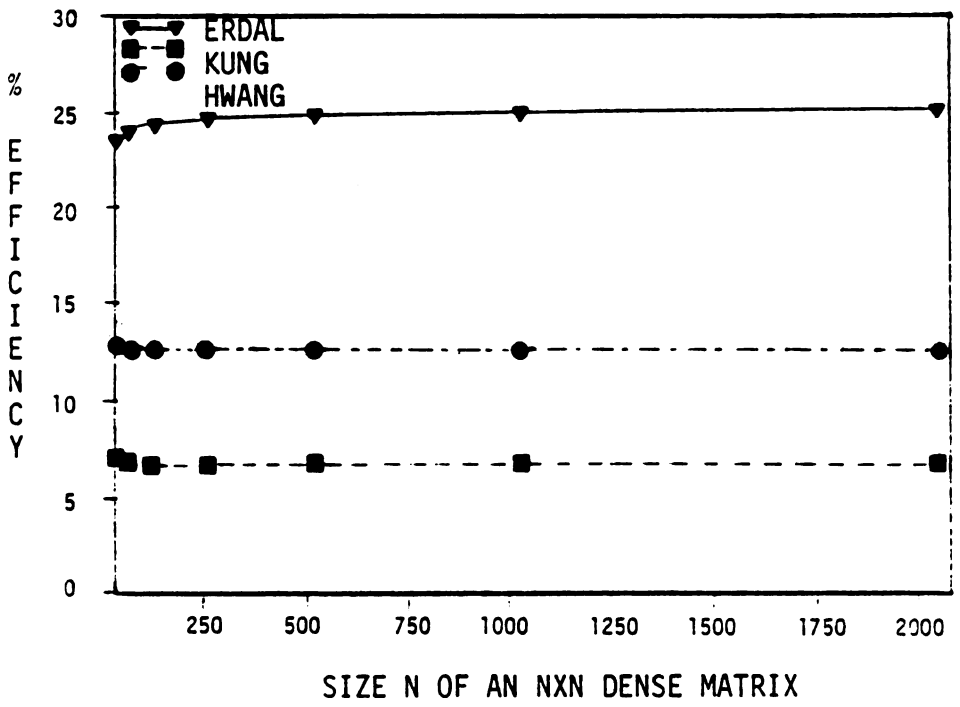


Figure 3-13b: Percent efficiency vs. N.

Instead, this research focuses on other important issues such as, system architecture and system applications as discussed in the following chapter.

CHAPTER IV

APPLICATIONS OF THE TREE STRUCTURES

Many digital signal and image processing applications employ computational tools such as convolution and discrete Fourier transform, which can be formulated as matrix-matrix or matrix-vector multiplications. For example, convolution is used to compute auto and cross-correlation functions, to design and implement finite impulse response (FIR) and infinite impulse response (IIR) digital filters, to solve difference equations, and to compute power spectra [12]. For this reason, it is possible to achieve high performance digital signal and image processing devices by making use of the tree structures introduced in Chapter III for matrix multiplications.

In this chapter, based on the tree structured approach introduced in Chapter III, designs of special-purpose devices that can be used in digital signal and image processing, to perform convolution and discrete Fourier transform are proposed and discussed. In Sections 4.1 and 4.2, algorithms for 1-dimensional (1-D) and 2-dimensional (2-D) convolution and discrete Fourier transform (DFT) operations are introduced, respectively. Tree structure for 1-D computations is also shown in these sections. In Section 4.3 processor level, chip level, and system level architectures are proposed to perform convolution and DFT operations to be used on digital signal and image processing areas.

4.1 (

wher

y(n)

zero

in F

x(n)

sequ

step

tim

is

The

sh

wh

op

mu

TH

se

4.1 Convolution

One dimensional noncyclic convolution can be defined as

$$y(n) = \sum_{k=0}^{N-1} h(n - k)\, x(k)$$

$$\text{for } n = 0,1,\ldots,2N - 1$$

where h(n) and x(n) are input sequences of length N. Output sequence, y(n), has length (2N - 1), and all these sequences are defined to be zero outside these lengths. The matrix formulation for N = 4 is shown in Figure 4.1.

Assume that $N = 2^k$, k = 1,2,..., and input sequences h(n) and x(n) both have length N. Then, it is possible to compute the output sequence y(n) with (2N - 1) processing elements in (2N + log N) time steps. Figure 4.2 shows the 1-D convolution tree with input and output timing tables. Note that it is assumed that the input sequence h(n) is preloaded either sequentially or parallel into the R_B registers. Then, at each subsequent time step, elements of the sequence x(n) are shifted to the right, one element at a time, through the shift registers, where it is assumed that the shift registers are initially zero. The operands in registers R_B and the operands in the shift registers are multiplied at each time step following the every shift operation. This 1-D convolution tree can be extended to perform 2-D convolution.

Two dimensional cyclic convolution, $y(n_1, n_2)$, of the two input sequences $h(n_1, n_2)$ and $x(n_1, n_2)$ can be defined as

$$y(n_1, n_2) = \sum_{K_1=0}^{N_1-1} \sum_{K_2=0}^{N_2-1} h(n_1 - k_1, n_2 - k_2)\, x(k_1, k_2)$$

$$\begin{bmatrix} h_0 & 0 & 0 & 0 \\ h_1 & h_0 & 0 & 0 \\ h_2 & h_1 & h_0 & 0 \\ h_3 & h_2 & h_1 & h_0 \\ 0 & h_3 & h_2 & h_1 \\ 0 & 0 & h_3 & h_2 \\ 0 & 0 & 0 & h_3 \end{bmatrix} \times \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \end{bmatrix}$$

(a)

$$\begin{aligned} y_0 &= h_0x_0 \\ y_1 &= h_0x_1 + h_1x_0 \\ y_2 &= h_0x_2 + h_1x_1 + h_2x_0 \\ y_3 &= h_0x_3 + h_1x_2 + h_2x_1 + h_3x_0 \\ y_4 &= h_1x_3 + h_2x_2 + h_3x_1 \\ y_5 &= h_2x_3 + h_3x_2 \\ y_6 &= h_3x_3 \end{aligned}$$

(b)

Figure 4-1: Noncyclic convolution for N = 4.

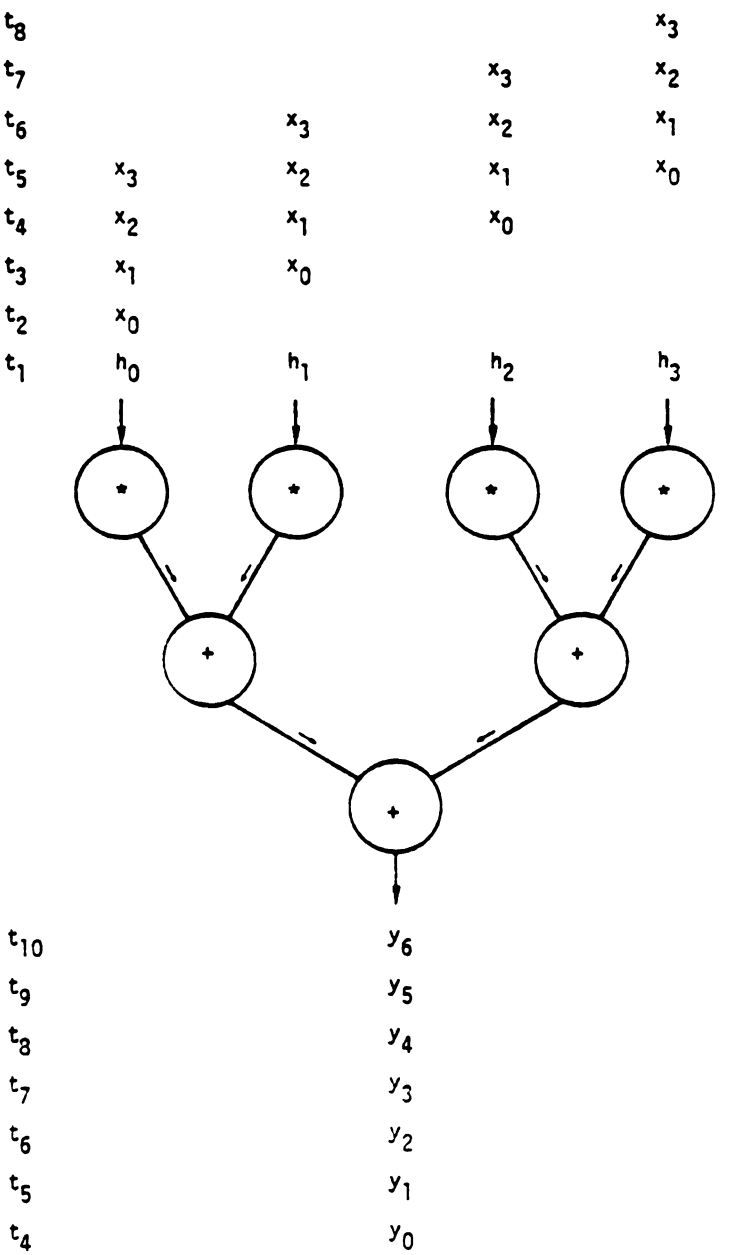


Figure 4-2a: I-D Convolution tree.

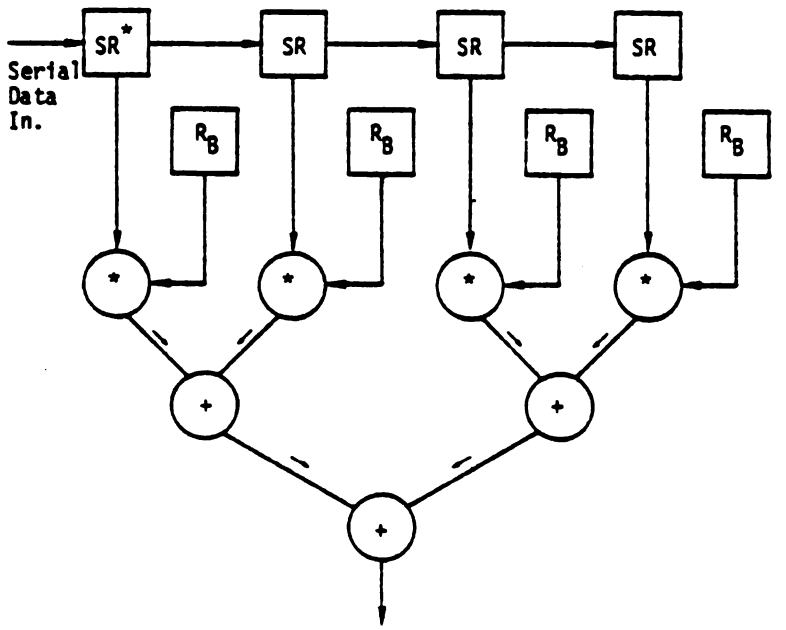


Figure 4-2b: Realization of I-D convolution.

* (SR = Shift Register)

$$\text{for } n_1 = 0,1,\ldots,N_1 - 1$$

$$n_2 = 0,1,\ldots,N_2 - 1$$

with the length of $N_1 \times N_2$. For 2-D convolution, N 1-D convolution trees are used with additional adder PE's to merge the results of each 1-D convolution tree. The input sequence $h(n_1, n_2)$ is preloaded to the R_B registers, either sequentially or parallel (N elements at a time). Then, the input sequence $x(n_1, n_2)$ is shifted to the right one element at a time through the shift registers. After each shifting operation, the elements in the shift registers are multiplied by the elements in the R_B registers, and the result is shifted down to merge all the partial products. It is assumed that initially all the shift registers are cleared. At certain times it is necessary to feed zero values down to the multiplier PE's without losing the contents of the corresponding shift registers. For this we may use (log N) bit decoders to determine which shift registers will provide zero value to the multiplier PE's to which they are connected. Input to the decoder can be a simple (log N)-bit counter, which initializes itself every N counts. Two dimensional convolution, with the length N x N, can be computed in $(N^2 + 3 + \log N)$ time steps by using $(2N^2 - 1)$ PE's and N^2 shift registers, where N^2 of the total PE's are multiplier PE's, and $(n^2 - 1)$ of them are adder PE's. It is assumed that the input sequence $h(n_1, n_2)$ is preloaded, which takes $(N^2 + 1)$ time steps if it is loaded sequentially, first by shifting right through the shift registers, then loading the content of every shift register to the registers R_B's in one time step. The first output appears after (3 + log N) time steps. Then there will be a new output at each time step.

4.2 Discrete Fourier Transform (DFT)

A 1-dimensional DFT [13] can be defined as

$$y(n) = \sum_{k=0}^{N-1} x(k)W^{nk}$$

$$\text{for } n = 0,1,\ldots,N-1$$

where $W = \exp(-2\pi j/N)$, (j is imaginary number).

Let X(n) be the input sequence for N = 4, where it is assumed that x(n) is represented by real numbers. Then, the DFT of the vector X(n), Y(n), can be calculated as shown in Figure 4.3. Note that this is a simple matrix-vector multiplication. One way of computing this 1-D DFT is shown in Figure 4.4a. First, all the elements of the vector X(n) are loaded, in parallel, into the R_B registers in one time step. Then the matrix W^{nk}, assuming it has already been loaded to a memory, can be pipelined down through the tree N-elements at a time. Note that the binary trees shown in Figure 4.4 are only for the real part or the imaginary part of the overall output. So, for the total implementation of DFT, we need twice as many PE's than as shown in these figures. With this tree structure it takes (1 + N + log N) time steps to compute N-point DFT's using 2(2N - 1) PE's. All the memory units (MU) can be accessed by one counter, which counts zero through N, sequentially. Another possibility is to use 2N(2N - 1) PE's as shown in Figure 4.4b. In this case N-point DFT can be computed in (log N + 1), assuming the matrix W^{nk} is preloaded to the R_B registers.

A two dimensional DFT of size N x N can be defined as

$$y(n_1, n_2) = \sum_{k_1=0}^{N-1} \sum_{k_2=0}^{N-1} x(k_1, k_2)W^{n_2k_2}W^{n_1k_1}$$

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & w^1 & w^2 & w^3 \\ 1 & w^2 & w^4 & w^6 \\ 1 & w^3 & w^6 & w^9 \end{bmatrix} \times \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

(a)

$$y_0 = x_0 + x_1 + x_2 + x_3$$
$$y_1 = x_0 + x_1 w + x_2 w^2 + x_3 w^3$$
$$y_2 = x_0 + x_1 w^2 + x_2 w^4 + x_3 w^6$$
$$y_3 = x_0 + x_1 w^3 + x_2 w^6 + x_3 w^9$$

(b)

Figure 4-3: DFT for N = 4.

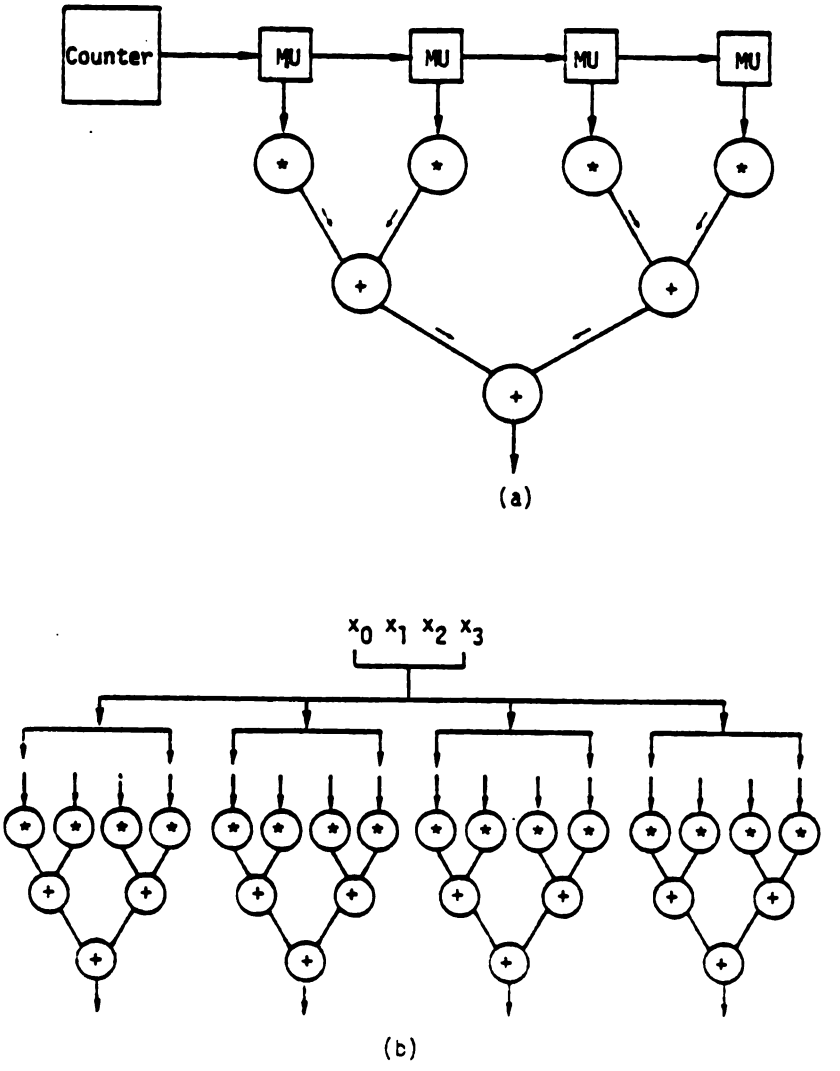


Figure 4-4: One dimensional DFT trees. (Real or imaginary part of the overall I-D DFT design.)

for $n_1, n_2 = 0,1,\ldots,N-1$

where $w = \exp(-2\pi j/N)$.

DFT of the input matrix $X(k_1, k_2)$, $Y(n_1, n_2)$, can be redefined as a matrix-vector multiplication, because $w^{n_1 k_1}$ and $w^{n_2 k_2}$ can be precalculated for all the values of n_1, n_2, k_1, and k_2, and can be loaded into memory before the DFT operation starts. As a result, we obtain a $N^2 \times N^2$ matrix. If we visualize the input matrix $X(k_1, k_2)$ as a vector of length N^2, then the multiplication of the matrix $W(z_1, z_2)$ and the vector $X(g)$, where $g = 0,1,\ldots,(N_2 - 1)$ is the output vector $y(g)$. First, the input matrix is loaded into the R_B registers of the multiplier PE's in N time steps, N elements at a time. Then the matrix $W(z_1, z_2)$ is pipelined through the tree (see Figure 4.5) one column at a time to compute the DFT.

4.3 The Design of a Special-Purpose Device for 1-D Convolution and DFT

The overall design can be divided into three design levels; processor level, chip level, and systems level design.

* Processor Level Design--As introduced in previous chapters, tree structures employ only two different types of PE's. Namely, multiplier PE's (MPE) and adder PE's (APE). An MPE consists of an n-bit, signed, 2's-complement, fixed-point multiplier and two registers, R_B and R_C (see Figure 4.6a). R_B is an n-bit input register that is used to store one of the operands for the multiplication, whereas R_C is an 2n-bit output buffer register that is used to store the output of the multiplication. APE has only one register, R_C, and is used for the same purpose to store the output of an addition as shown in

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & w & 1 & w \\ 1 & 1 & w & w \\ 1 & w & w & w^2 \end{bmatrix} \times \begin{bmatrix} x_{00} \\ x_{01} \\ x_{10} \\ x_{11} \end{bmatrix} \qquad \begin{bmatrix} y_{00} \\ y_{01} \\ y_{10} \\ y_{11} \end{bmatrix}$$

(a)

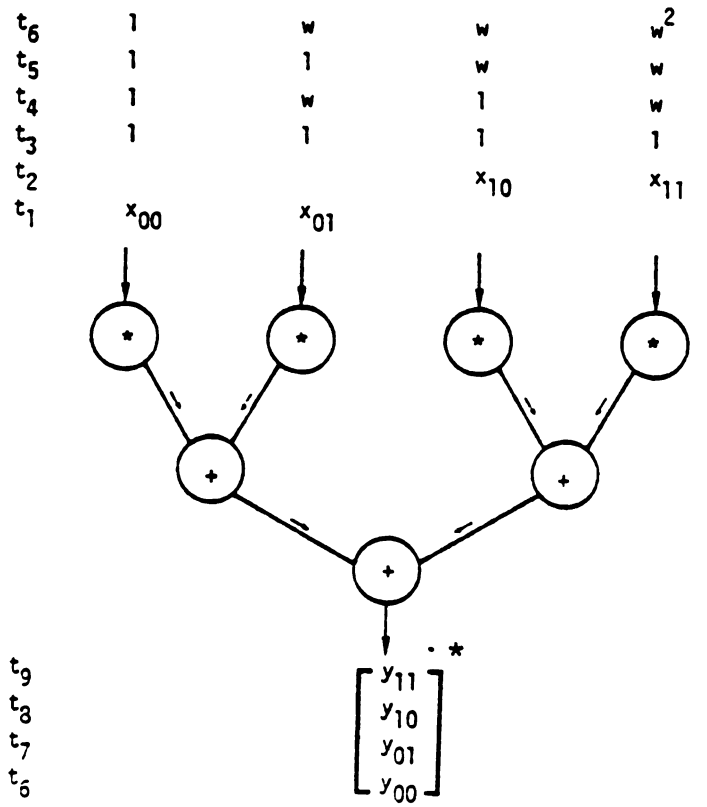


Figure 4-5: Two dimensional DFT tree for N = 2.

* (Real or imaginary part of the complex output y(n).)

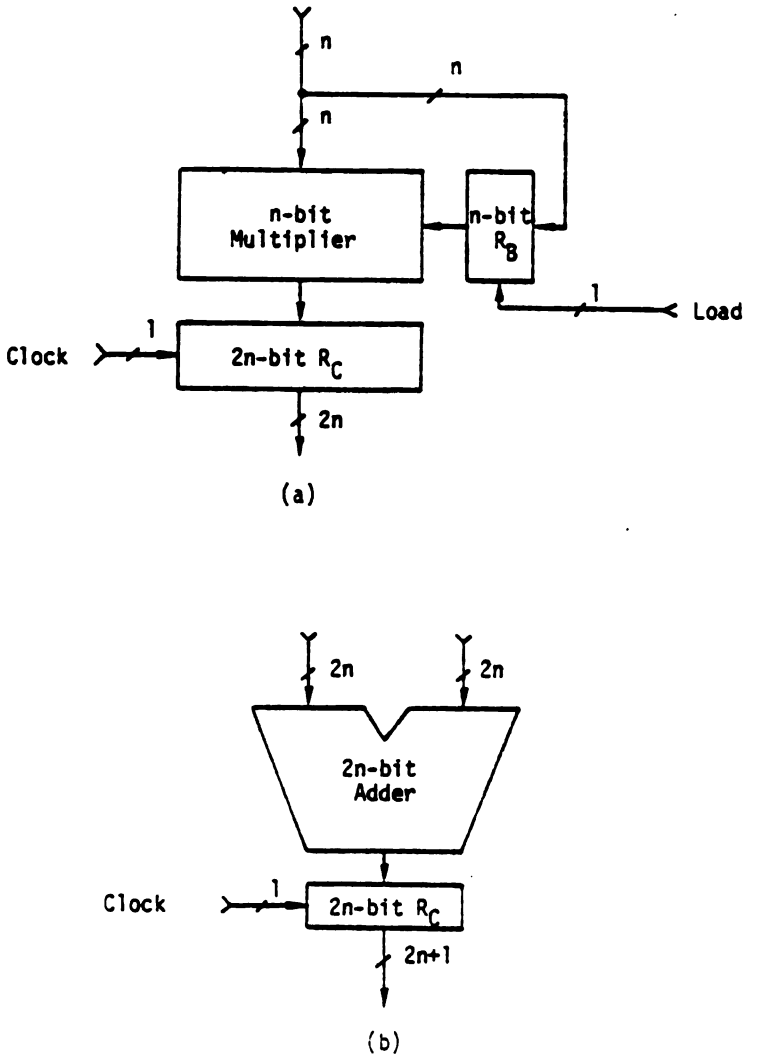


Figure 4-6: Block diagram of MPE and APE.

Figure 4.6b. MPE can be implemented, for example, by using the Booth [14] algorithm or the Baugh [15] array multiplier algorithm. The Baugh [15] algorithm is implemented with only full adders, and hence provides simple and regular layout. Carry lookahead adders [16,17] can be used to implement the APE for fast addition time.

* Chip Level Design--The word length and the integration technology are the two main factors that basically determine the number of devices that can be put into a chip. (A device can be defined as a transistor, gate, processing element, and so on.) One way of implementing the tree structures is to assume one PE per chip, and use off the shelf IC's for the overall design. Another approach would be to put more than one PE's into the chip. If we assume a word length of 8-bits, then, with VLSI technology it is possible to pack seven PE's into a single chip (see Figure 4.7). We assume that the multipliers are implemented by using the Baugh [15] algorithm as described in [16] and the adders be implemented by carry lookahead adders as described in [17]. Tri-state buffers are used to connect each chip to its neighboring chip to implement shifting operations for large matrices. For larger word lengths, we either reduce the number of PE's to be planted on a single chip, or increase the number of pins as long as the technology permits us to do so. Note that the chip shown in Figure 4.7 can be housed in a conventional 64-pin package.

* System Level Design--One of the important issues that needs to be resolved is the determination of the largest N for the overall system that will be implemented in a single pass through the system. We may need to partition the computations for matrices that are larger than the given system's size.

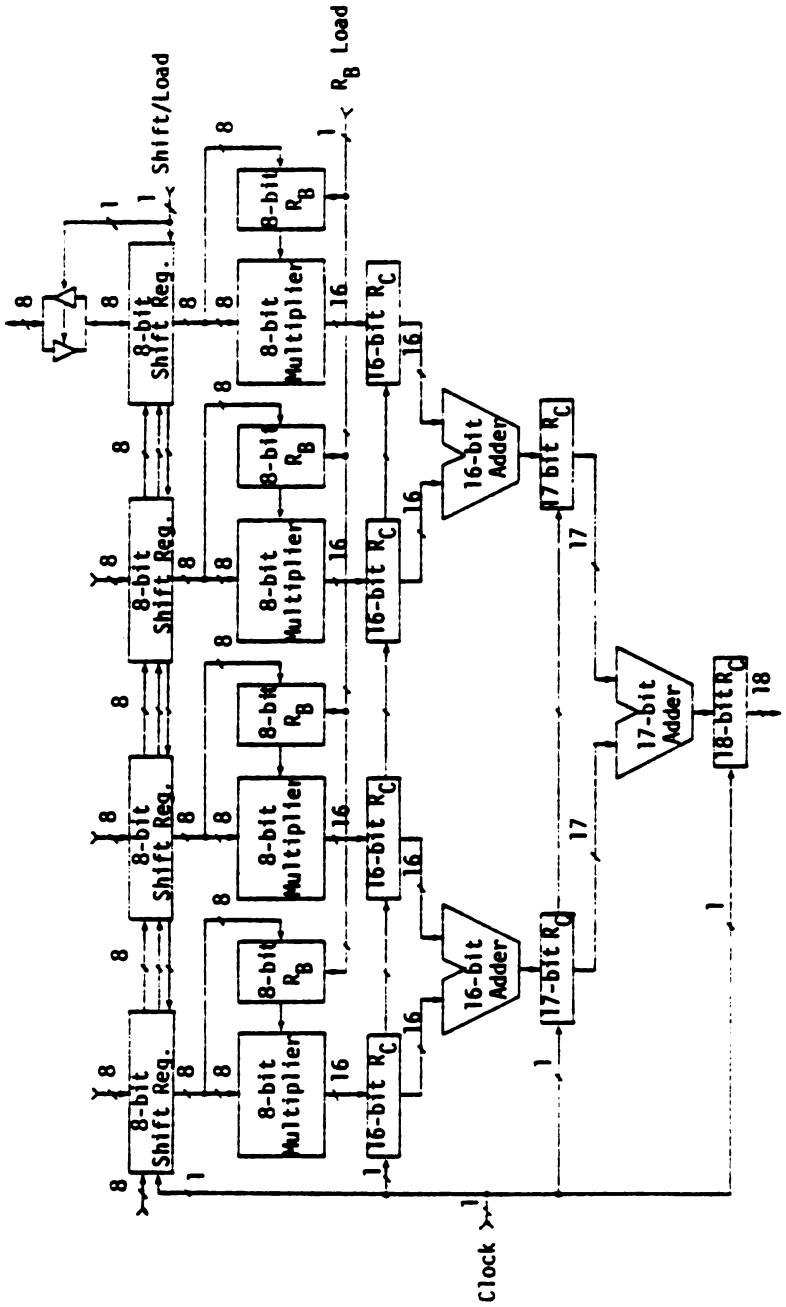


Figure 4-7: Block diagram of a possible single chip implementation of the tree structure ("Tree-IC").

Assume that N = 16. Then the overall design can be implemented by employing the "Tree-IC's, single chip adders, memory units, and a counter as shown in Figure 4.8. This system can be used for 1-D convolution, Y(n), where n = 0,1,...,(N - 1). If we let N = 8, and assume sequence x(n) is real, then, the same tree structure shown in Figure 4-8 can be used for 8-part DFT calculation. The overall system requires one 4-bit address counter, 16 memory units with 16 words each, 4 Tree-IC's, and three single chip adder PE's.

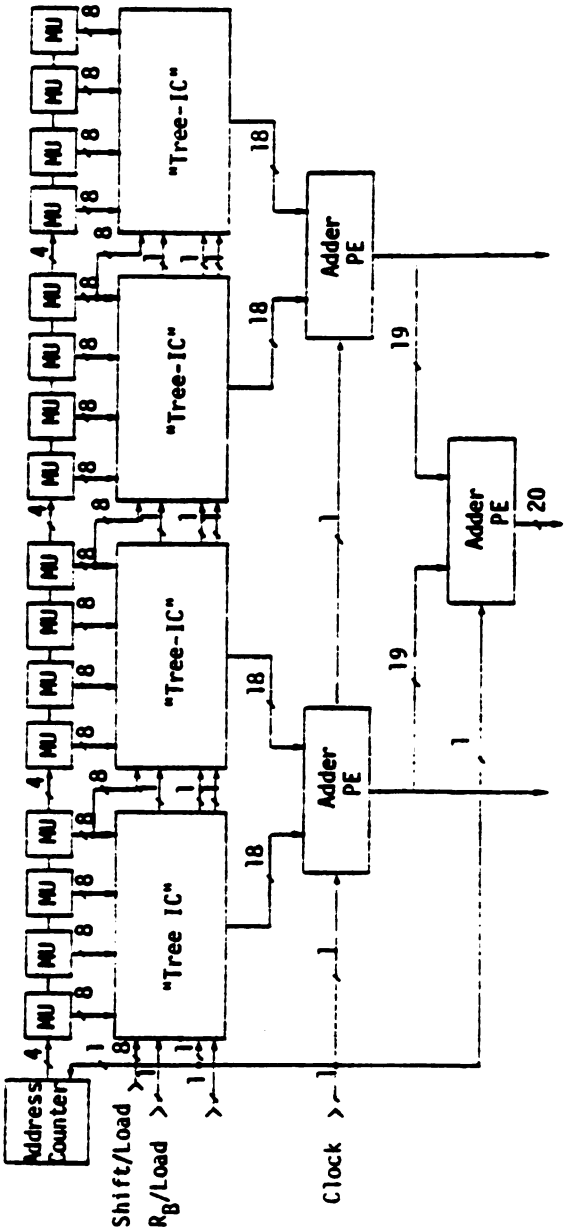


Figure 4-8: Tree structured I-D convolution system for N = 16, and I-D DFT system for N = 8.

CHAPTER V

CONCLUSION

5.1 Summary

The purpose of this research was to develop and investigate a high performance special-purpose device for matrix multiplication which works as a co-processor attached to a host computer. This device may be employed to perform discrete Fourier transform and convolution operations for digital signal and image processing applications. The design was constrained by the requirement that basic VLSI rules be followed; i.e., the overall co-processor has a simple, regular structure with short, nearest-neighbor communication paths.

Systolic arrays, introduced by Kung [2-4], as well as other VLSI computing structures, introduced by Hwang [5], were reviewed in Chapter II. Examples for matrix multiplication algorithms were shown, and the corresponding space-time complexity of each of these algorithms were indicated. Specifically, it was indicated that an nxn dense matrix multiplication can be computed in 5(n - 1) and (4n - 2) time steps by using $(3n^2 - 3n + 1)$ and n(2n - 1) processing elements by the Kung's [2-4] and the Hwang's [5] algorithms, respectively.

A new binary tree structure matrix multiplication algorithm was introduced in Chapter III. The algorithm capitalizes on the properties of the VLSI to obtain high throughput rates and high efficiency. The algorithm first forms the products of the matrix elements in the leaf

nodes, and then sums the partial results by merging them as they are pipelined toward the root node of the binary tree. In this way, massive parallelism can be achieved to introduce high degrees of pipelining and multiprocessing. Processing elements, which are the nodes of the binary tree, and the basic tree structures were also defined in this chapter. It was shown that, the binary tree structures can be implemented by only a few different types of simple processing elements, and they have simple, regular, and short communication geometry, which are considered as necessary attributes of an efficient VLSI implementation. This in turn yields cost-effective special-purpose devices.

Chapter III concluded with a discussion of the space-time complexity of this binary tree structure and the comparison of its speedup and efficiency with those of the structures reviewed in Chapter II. It was shown that, for any nxn dense matrix multiplication, the binary tree structures algorithm provides higher speedup and efficiency over the other algorithms. Specifically, it was shown that any nxn dense matrix multiplication can be performed in [(2n + log(n))] time steps by using n(2n - 1) processing elements. From this, we can conclude that the binary tree structures use fewer processing elements than Kung's [2-4] algorithm but use the same number of processing elements as compared to the Hwang's [5] algorithm. And the binary tree structure provides a solution faster for any nxn dense matrix multiplication than either of the algorithms mentioned above. Note that a processing element that is used in the binary tree structures is either a multiplier or an adder with some additional registers, whereas a processing element for the other structures contains both a multiplier and an adder, with some additional registers. Furthermore, each PE in the

tree structure has only one register as compared to three registers for the others. The reason for this is that each PE requires only three input/output ports in binary tree structure as compared to six input/output ports. All of these advantages yield a smaller chip area and fewer input/output connection lines per PE for the binary tree structures.

Some other important advantages of tree structures were also indicated in Chapter II. For example, in binary tree structures there is no need for data skewing for matrix multiplication, whereas it is necessary to skew the data before inputting for the other structures which introduces additional computation times. For matrix multiplication, once the matrix B is loaded to the R_B registers, matrix A is broadcasted to all column vector processing units at the same time one row at a time, which simplifies the data accessing problem. As the matrix A is pipelined through the binary tree, all the processing elements are kept busy and utilized to perform the actual calculations, instead of simply passing data from one PE to another as in the case of Kung's [2-4] and Hwang's [5] algorithms. The algorithm introduced in this thesis utilizes the inherent characteristics of the matrix multiplication, and optimally matches this algorithm to the binary tree structure.

In Chapter IV, special-purpose devices that can be used in digital signal and image processing to perform convolution and discrete Fourier transform were proposed and discussed. Specifically, one dimensional convolution operation and its binary tree structure chip level and system level architectures were illustrated. Applicability of the binary tree structure to many other digital signal and image processing

applications, which can be represented as matrix-matrix multiplication, is evident. The use of this binary tree structures can substantially reduce the amount of computations required for the solution of many problems in digital signal and image processing.

This binary tree structure is intended to be used in conjunction with a conventional computer. It can be used to implement a complete digital signal or image processing device, or can be used as part of another device. And it can easily be extended to any size because of its modularity.

Advances in the design and fabrication of VLSI circuits will soon make it feasible to construct special-purpose devices, such as the binary tree structures, which are excellent candidates for the VLSI technology, with their simple, regular, and short communication geometry. They can be used to obtain very high throughput rates for the problems which are not amenable to solutions even with the state-of-the-art conventional computers. The geometry, flexibility, and cost effectiveness computers. The geometry, flexibility, and cost effectiveness will make them very valuable tools for many important tasks.

5.2 Further Research

One area that should be investigated in the future concerns the application of binary tree structures introduced to other signal and image processing algorithms. Using these structures, it may be possible to construct complete digital signal and image processing devices with very high speeds. A second investigation that could be pursued is the development of single chips with multiple PE's which could result in a substantial reduction on the time-space complexity of the overall

system. And, finally, further investigation that could be taken is the development of fast and large interlieved memories for the tree structures and their scheduling problems between them, because the communication line between the host computer and the attached special-purpose device can present a bottleneck.

REFERENCES

1. Haynes, L.S., "Highly Parallel Computing," IEEE Computer, Vol. 15, No. 1, Jan. 1982, pp. 7-8.

2. Kung, H.T. and Leiserson, C.E., Systolic Arrays (for VLSI), Technical Report, Carnegie-Mellon University, Department of Computer Science, Dec. 1978.

3. Kung, H.T., "Let's Design Algorithms for VLSI Systems," Proc. of Caltech Conf. on VLSI, Jan. 1979, pp. 65-90.

4. Kung, H.T., "The Structure of Parallel Algorithms," In Advances in Computers, (Yovits, M.C., Editor), Academic Press, New York, 1979.

5. Hwang, K. and Cheng, Y.H., "VLSI Computing Structures for Solving Large Scale Linear Systems of Equations," Proc. of Intr'l Conf. on Parallel Processing, Aug. 1980, pp. 217-230.

6. Swartzlander, E.E. et al., "Inner Product Computers," IEEE Trans. on Computers, Vol. 27, No. 1, Jan. 1978, pp. 21-31.

7. Despain, A.M. and Patterson, D.A., "X-Tree: A Tree Structures Multiprocessor Computer Architecture," Proc. of Fifth Intr'l. Symp. on Computer Architecture, Apr. 1978, pp. 144-151.

8. Browning, S.A., The Tree Machine: A Highly Concurrent Computing Environment, Technical Report (Ph.D. Thesis), Computer Science, California Institute of Technology, Jan. 1980.

9. Budnik, P. and Kuck, D.J., "The Organiation and Use of Parallel Memories," IEEE Trans. on Computer, Dec. 1971, pp. 1566-1569.

10. Kuck, D.J., The Structure of Computers and Computations, John Wiley & Sons, 1978.

11. Lee, R.B-L., "Empirical Results on the Speed, Efficiency, Redundancy, and Quality of Parallel Computations," Proc. of Intr'l Conf. on Parallel Processing, 1980, pp. 91-100.

12. Agarwal, R.C. and Cooley, J.W., "New Algorithms for Digital Convolution," IEEE Trans. on Acoustics, Speech, and Signal Processing, Vol. 25, No. 5, Oct. 1977, pp. 392-410.

13. Cochran, W.T. et al., "What is the Fast Fourier Transform?" IEEE Trans. on Audio and Electroacoustics, Vol. 15, 1967, pp. 45-55.

14. Booth, A.D., "A Signed Binary Multiplication Algorithm," Quart. J. Mech. Appl. Math., Vol. 4, 1951, pp. 236-240.

15. Baugh, C.R. and Wooley, B.A., "A Two's Complement Parallel Array Multiplication Algorithm," IEEE Trans. on Computers, Vol. 22, Dec. 1973, pp. 1045-1047.

16. Hwang, K., Computer Arithmetic, Principles, Architectures and Design, John Wiley & Sons, 1979.

17. Erdal, A.C. and Fisher, P.D., "Relaxed Pinout Constraints Yield Improved Parallel Adders for VLSI-Based Systems," IEEE Intr'l. Conf. on Circuits and Computers, Sept. 1982, pp. 142-146.