A LINEAR ARRAY MATRIX MULTIPLICATION ALGORITHM*

I.V.Ramakrishnan
D.S.Fussell
A.Silberschatz

TR-210 September 1982

Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712

ABSTRACT

A systolic algorithm for multiplying two n x n matrices on a linear array of O(n) processors is described that operates in $O(n^2)$ time. The algorithm uses simple processing elements requiring no control units or addressable memory. Previous systolic algorithms for matrix multiplication on a linear array have had the same processor and time complexity. However the processors used were complex requiring control units and addressable memory.

---

*This research was supported in part by the National Science Foundation under Grants MCS-81Ø4Ø17 and ONR Contract NØØØ14-8Ø-k-Ø987

1. INTRODUCTION

In [5], Kung has proposed systolic arrays as a simple and effective means of employing VLSI technology to handle compute-bound problems. The basic goal is to achieve more computations per time unit from an existing system through the addition of an array of simple and identical processing elements. No major changes to the existing system architecture should be required; thus the existing memory bandwidth should remain constant and the device should be interfaced to an existing system bus.

The constraints imposed on the structure of systolic arrays by the above goals make linear array processors the most useful class of systolic structures. A number of important computational algorithms fit very naturally onto a linear processor array, and many such algorithms have been designed [1, 2, 4, 6, 7]. Others fit more naturally onto two-dimensional array structures, which unfortunately are poorly suited for the environment in which they are likely to be used. Matrix multiplication is a problem of the latter type; it can be implemented very naturally on a square mesh of processing elements, or, for band matrices, on a hexagonal array [4]. In [3], a linear systolic array which performs matrix multiplication was described. The architecture of this device differed in fundamental ways from that of other systolic devices in that each processing element contained an addressable memory and a reasonably sophisticated control unit which could respond to control instructions passed through the array. This design resulted in rather large, complex processing elements which resembled in many ways miniature versions of traditional instruction-set processors.

In this paper we present a linear systolic algorithm for multiplying two

n x n matrices which employs much simpler processing elements. No dynamic control or addressable memory is required, so the complexity of the control unit is vastly reduced, and the RAM can be replaced by a simpler and more compact shift register. Our algorithm requires O(n) processing elements and requires $O(n^2)$ time steps to perform the computation. The rest of the paper is organized as follows. In section 2 we describe the linear array model. In section 3 we describe the algorithm to multiply two n x n matrices, in section 4 we give a proof of its correctness, and in section 5 we generalize the algorithm to handle non-square matrices.

2.LINEAR ARRAY MODEL

The linear array consists of 3n-2 identical processors numbered from 1 to 3n-2 as shown in figure 2.1. The entire array is driven by a single phase global clock.

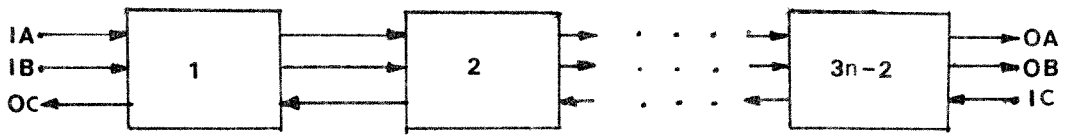

Figure 2.1: Linear Array

For the sake of clarity we will be using three matrices, A, B and C throughout this paper and we wish to compute C=A x B. The $(i,j)^{th}$ entry in A, B and C will be referred to as a_{ij}, b_{ij} and c_{ij} respectively. The $(i,j)^{th}$ entry in A, B and C will be referred to as a_{ij}, b_{ij} and c_{ij} respectively. The elements of A, B and C are pumped into the array through the input ports IA, IB and IC respectively. The elements of A and B and the

result elements of C flow out of the array through the output ports OA, OB and OC respectively. A processor in the array is shown in greater detail in figure 2.2.

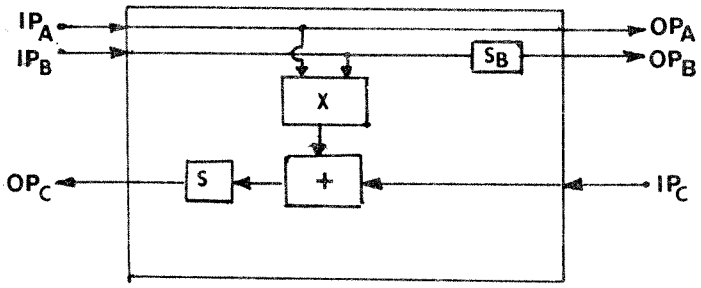

Figure 2.2: A Processor

The processor consists of three input ports IP_A, IP_B, IP_C, three output ports OP_A, OP_B and OP_C, and two shift registers S_B and S_C of lengths 1 and n-2 respectively. Thus area required by the array is proportional to $O(n^2)$, as is the case with the algorithm in [3]. The input port IP_A is directly connected to the output port OP_A. The input port IP_B is connected to the input of the shift register S_B. The inputs from the input ports IP_A and IP_B are fed into the multiplier unit denoted by '[x]' in the figure above. The output from the multiplier and the input from the input port IP_C are fed into the adder unit denoted by '[+]' in the figure above. The output from the adder unit is fed into the input of the shift register S_C and the outputs of S_B and S_C are connected to the output ports OP_B and OP_C. OP_A, OP_B of processor i is connected to IP_A, IP_B respectively of processor i+1 and OP_C is connected to IP_C of processor i-1.

In each cycle (which is the same as a clock phase of the global clock) the processor computes a result value which is the value at IP_C in that cycle added to the product of the values at IP_A and IP_B, .i.e., if a, b and

c are the values at IP_A, IP_B and IP_C respectively in cycle t then the result computed by the processor in cycle t is c+a*b. The processor then transmits a, b and c+a*b to OP_A, OP_B and OP_C respectively at the end of cycle t, t+1 and t+n-2 respectively and so a, b and c+a*b are available at OP_A, OP_B and OP_C in cycles t+1, t+2 and t+n-1 respectively. The delays are implemented by the two shift registers.

The processors in the array do not have any decison-making ability. Hence distinct elements of A, B, and C must not simultaneously reach the input ports IP_A, IP_B, and IP_C respectively of any processor.

3. THE ALGORITHM

Herein we describe the algorithm to compute C=A x B on a linear array of 3n-2 processors. A, B and C are n x n matrices. The result entries in C are computed by the following recurrences:

$$c_{ij}^{(k+1)}=c_{ij}^{(k)}+a_{ik} * b_{kj} \quad ..(1)$$

$$c_{ij}^{(1)}=\emptyset \quad ..(2)$$

All the entries in C are initialized to $\emptyset$ before being pumped into the array. We will refer to the initial value of c_{ij} as $c_{ij}^{(1)}$ and the final value as $c_{ij}^{(n+1)}$.

Let t_s be the time at which $c_{11}^{(1)}$ is fed into IC. For at least 3n-2 cycles prior to t_s pump $\emptyset$ into IA. This ensures that the input port IP_A of every processor in the array is initialized to $\emptyset$. Next perform the following steps:

1. Initialize $c_{ij}^{(1)}$ to $\emptyset$ and pump it into IC at time

$t_s+(i+j-2)*n+(i-1)$

2. Pump a_{ij} into IA at time $t_s+(2n-3)*(n-1)+(j-1)*n+(i-1)$

3. Pump b_{ij} into IB at time $t_s+(2n-5)*(n-1)+(n-j)+(i-1)*(n+1)$

4. For all times less than $t_s+(2n-3)*(n-1)$ (which is the time when a_{11} is pumped into IA) and for all times greater than $t_s+(n-1)*(3n-2)$ (which is the time at which a_{nn} is pumped into IA) pump $\emptyset$ into IA.

This completes the algorithm. The final value of c_{11} is the first result that comes out of OC at time $t_s+(3n-2)*(n-1)$ and the final value of every c_{ij} appears in the output port OC at time $t_s+(3n-2)*(n-1)+(i+j-2)*n+(i-1)$.

If the size of the two matrices is less than n x n then we can add dummy entries so that the matrices are converted to size n x n. If the size is greater than n x n then we can decompose the matrices to size n x n and compute each of the decomposed matrices separately.

Let us illustrate how the algorithm functions through a simple example. Let A, B and C be three 3 x 3 matrices. In figure 3.1 we illustrate 10 successive time steps in the computation of C=A x B using the above algorithm. For each time step we show the elements which arrive at the input ports of the processors. The element within curly brackets in a processor is the value evaluated by that processor in that time step. Figure 3.1 begins at time step 6 since in the previous time steps $\emptyset$ is fed in at IA.

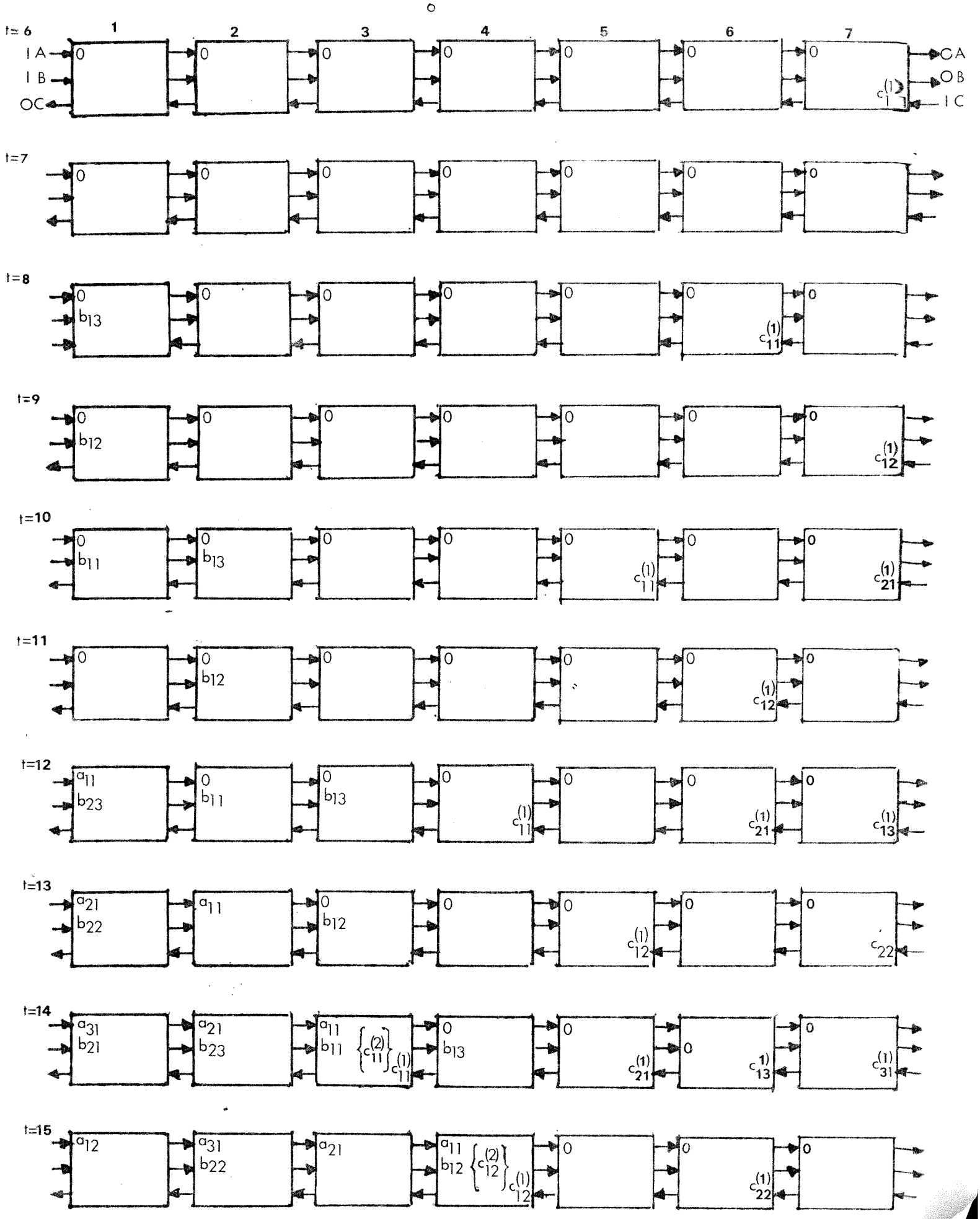

Figure 3.1

4. PROOF OF CORRECTNESS

In this section we demonstrate that our proposed algorithm functions correctly.

Proposition 4-1: For any i and j if a_{ij} is fed into the input port IA at time t then the element at the input port IP_A of processor k at time t+(k-1) is a_{ij}.

Proof: Follows immediately from the fact that at the end of a cycle a processor pumps the element at its input port IP_A directly to its output port OP_A without any intervening shift register.

□

Proposition 4-2: For any i and j if b_{ij} is fed into the input port IB at time t then the element at the input port IP_B of processor k at time t+(k-1)*2 is b_{ij}.

Proof: Follows immediately from the fact that at the end of a cycle a processor pumps the element at its input port IP_B directly to its output port OP_B through shift register S_B of length 1.

□

Proposition 4-3: For any i and j if c_{ij} is fed into the input port IC at time t then the element at the input port IP_C of processor k at time (3n-2-k)*(n-1) is c_{ij}.

Proof: Follows immediately from the fact that at the end of a cycle a processor pumps the result value computed in the cycle to its output port OP_C through shift register S_C of length n-2.

□

The lemmas and theorems in the remainder of this section are based on

the algorithm described in section 3.

Lemma 4-1: Distinct elements of A, B and C do not simultaneously reach the input ports IP_A, IP_B and IP_C respectively of any processor.

Proof: Let a_{ij} and a_{st} be two distinct elements in A that simputaneously reach the input ports IP_C of processor k. Let t_{ij} and t_{st} be the times at which a_{ij} and a_{st} respectively are fed into IA. From proposition 4-1 and step 2 of the algorithm it follows that (j-t) =(s-i)/n. Now $|s-i|<n$ and (j-q) is an integer. Hence for the equality to hold we must have i=s and j=t. Hence a_{ij} is not identical to a_{st} as assumed.

Using similar arguments we can show that elements of B and C do not simultaneously reach the input ports IP_B and IP_C respectively of any processor.

□

Lemma 4-2: For any i and j, the initial value $c_{ij}^{(1)}$ of c_{ij} remains unchanged as c_{ij} travels from IC until it reaches the input port IP_C of the processor numbered n+(i+j)-2.

Proof: Let t_c be the time when c_{ij} reaches the input port IP_C of a processor numbered k where $n+(i+j)-1 \leqslant k \leqslant 3n-2$.

Now $t_c=[t_s+(i+j-2)*n+(i-1)]+\{(3n-2-k)*(n-1)\}$

From the algorithm the term within the square brackets in the above expression is the time at which c_{ij} is pumped into IC and from proposition 4-3 the term within the curly brackets is the total time it takes c_{ij} to reach the input port IP_C of k after being fed into IC.

Let x be the element at the input port IP_A of k at t_c. Let t_x be the time at which x is fed into IA. From proposition 4-1 the total time it takes x to reach the input port of IP_A after it is fed into IA is k-1 and so $t_c=t_x+k-1$. If x is Ø then the lemma is satisfied. By the algorithm, if

$t_x < t_s+(2n-3)*(n-1)$ then x is $\emptyset$. Hence we must show that $t_c-(k-1) < t_s+(2n-3)*(n-1)$ and this reduces to showing $[n*(n+(i+j)-1)-k*n]+\{i-n\} < 1$ The term within the square and curly brackets in the above expression are both $\leqslant \emptyset$ and hence the inequality holds. Hence x is $\emptyset$ and the lemma follows.

□

Lemma 4-3: For any i and j, the final value $c_{ij}^{(n+1)}$ of c_{ij} remains unchanged as c_{ij} travels from the output port of processor numbered i+j-1 until it reaches OC.

Proof: Let t_c be the time c_{ij} reaches the input port IP_C of a processor numbered k where $1 \leqslant k \leqslant i+j-2$.

Now $t_c=[t_s+(i+j-2)*n+(i-1)]+\{(3n-2-k)*(n-1)\}$

As in the proof of lemma 4-2 the term within the square brackets is the time at which c_{ij} is pumped into IC and the term within the curly brackets is the total time taken by c_{ij} to reach the input port IP_C after being fed into IC.

Let x be the element at the input port IP_A of k in t_c. Let t_x be the time at which x is fed into IA. The total time it takes x to reach the input port of IP_A is k-1 and so $t_c=t_x+k-1$. If x is $\emptyset$ then the lemma is satisfied. By the algorithm if $t_x > t_s+(3n-2)*(n-1)$ then x is $\emptyset$. Hence we must show that $t_c-(k-1) > t_s+(3n-2)*(n-1)$ and this reduces to showing that $[n*(i+j-2)-k*n]+i > \emptyset$. The term within the square brackets is $> \emptyset$ and i $> \emptyset$ and hence the inequality holds. Hence x is $\emptyset$ and so the lemma follows.

□

Lemma 4-4: For any i and j and for any k, $2 \leqslant k \leqslant n+1$, the time at which $a_{i(k-1)}$ and $b_{(k-1)j}$ reach the input ports IP_A and IP_B of the processor numbered $n+(i+j)-k$ is the same as the time at which c_{ij} reaches the input port IP_C of the same processor.

Proof: Let t_a, t_b and t_c denote the time at which $a_{i(k-1)}$, $b_{(k-1)j}$, and c_{ij} reach the input ports IP_A, IP_B and IP_C respectively of $n+(i+j)-k$. These times are given by the following three expressions:

1. $t_c=[t_s+(i+j-2)*n+(i-1)]+\{(3n-2-(n+i+j)+k)*(n-1)\}$

2. $t_a=[t_s+(2n-3)*(n-1)+(k-1-1)*n+i-1]+\{n+(i+j)-k-1\}$

3. $t_b=[t_s+(2n-5)*(n-1)+(n-j)+(k-2)*(n+1)]+\{(n+(i+j)-k-1)*2\}$

The term within the square and curly brackets have the same significance as in the proofs of lemma 4-2 and lemma 4-3. Each of these expressions can be reduced to $t_s+2n^2-6n+1+2i+j+k*n-k$. Hence $t_a=t_b=t_c$.

□

Lemma 4-5: For any k, $2 \leqslant k \leqslant n+1$, the value of c_{ij} when it leaves the processor numbered $n+(i+1)-k$ is $\sum_{m=1}^{k-1} a_{im}*b_{mj}$.

Proof: We prove this lemma by induction on k.

Basis: Let k=1. By lemma 4-4, a_{i1}, b_{1j} and c_{ij} appear at the input ports IP_A, IP_B, and IP_C respectively of the processor numbered $n+(i+1)-2$ at the same time. By lemma 4-2 the value of c_{ij} at the input port IR_C of this processor is $\emptyset$ and hence this processor computes $a_{i1}*b_{1j}$ and so the lemma holds for the base step.

Induction step: Assume lemma holds for k. Now by lemma 4-4 a_{ik}, b_{kj} and c_{ij} appear at the same time at the input ports IP_A, IP_B and IP_C respectively of processor numbered $n+(i+1)-(k+1)$. The value of c_{ij} at the input port IP_C of this processor is $\sum_{m=1}^{k-1} a_{im}*b_{mj}$ and this processor

computes $a_{ik}*b_{kj}$ and adds it to the the value of c_{ij} at its input port IP_C. Hence the lemma holds for k+1 and the proof is complete.

□

Theorem 4-1: For any i and j, $c_{ij}= \sum_{k=1}^{n} a_{ik}*b_{kj}$ when it leaves OC.

Proof: Immediate from lemma 4-5 and lemma 4-3.

□

Theorem 4-2: The time complexity of this algorithm is $O(n^2)$.

Proof: The difference between the time when c_{11} is pumped into IC and the time when c_{nn} appears at OC is $(3n-2)*(n-1)+(2n-2)*n+(n-1)$. Initialization takes (3n-2) cycles and hence the theorem follows immediately.

□

5. GENERALIZATION

We generalize the algorithm to handle non-square matrices where A is a p x q matrix and B is a q x r matrix and $p \geq r$. We use a linear array of p+q+r-2 processors numbered from 1 to p+q+r-2. The lengths of shift registers S_B and S_C in a processor are 1 and d-2 respectively where $d \geq p$. Let t_a, t_b and t_s be the times at which a_{11}, b_{11} and c_{11} respectively are pumped into IA, IB and IC respectively where $t_a=t_s+(d-1)*(p+r-2)-(q-1)$ and $t_b=t_s+(d-1)*(p+r-2)-(q-1)-(q+r-2)$. The algorithm in section 4 is modified as follows:

1. Pump $c_{ij}^{(1)}$ into IC at time $t_s+(i+j-2)*d+(i-1)$

2. Pump a_{ij} into IA at time $t_a+(j-1)*d+(i-1)$

3. Pump b_{ij} into IB at time $t_b+(r-j)+(i-1)*(p+1)$

4. For all times less than t_a and for all times greater than $t_a+(q-1)*d+(p-1)$ (which is the time at which a_{pq} is pumped into IA) pump $\emptyset$ into IA.

Correctness proofs of this generalized algorithm are similar to those presented in section 4. In the following theorem we show that the choice of d is not arbitrary.

Theorem 5-1: The algorithm ensures that distinct elements of A, B and C do not simultaneously reach the input ports IP_A, IP_B and IP_C respectively of any processor if and only if $d>p$.

Proof:

(only if part): Suppose $d<p$. Let $d=p-m$ where $m\geq 1$. Consider the two distinct elements $a_{(p-m)1}$ and a_{12} in A. Let t_1 and t_2 be the times at which $a_{(p-m)1}$ and a_{12} respectively are pumped into IA. From step 2 of the generalized algorithm and proposition 4-1 we can show that $t_1=t_2$ and hence two distinct elements of A reach the input port IP_A reach the input port IP_A of processor 1 at the same time.

(if part): Let $d\geq p$. We will show that distinct elements of A do not simultaneously reach the input port IP_A of any processor. Let a_{ij} and a_{st} be two distinct elements of A that simultaneously reach the input port IP_A of processor k. From step 2 of the generalized algorithm and proposition 4-1 we can show that $(j-t)=(s-i)/d$. Now $|s-i|<p$ and $d\geq p$. Hence for the equality to hold we must have $i=s$ and $j=t$ and hence a_{ij} and a_{st} are not distinct elements of A.

We can prove similarly that distinct elements of B and C do not simultaneously reach the input ports IP_B and IP_C respectively of any processor.

□

In the generalized algorithm we have assumed $p \geqslant r$. If $p < r$ then we choose d such that $d \geqslant r$ and replace a_{ij}, b_{ij} and c_{ij} in the generalized algorithm by b_{ji}, a_{ji} and c_{ji} respectively.

6. CONCLUSIONS

We have given a systolic algorithm to multiply two n x n matrices on a linear array of processors having O(n) processor complexity, $O(n^2)$ time complexity and $O(n^2)$ space complexity. The processors used are simple requiring no control. We then generalized the algorithm to handle non-square matrices. A clever data arrangement used by the algorithm results in a linear array without complex control and addressing logic. More importantly, the linear array uses only three input and three output ports independent of the sizes of the matrices being multiplied. As a result, the I/O bandwidth is a constant for the array. Similarly the size of the matrices being multiplied using a single chip does not affect pin count considerations of the chip.

Several variations to this algorithm are possible. For instance the elements in A, B and C matrices can all move in the same direction. All such variations are described in [8].

REFERENCES

[1] Chen,T.C.,Lum,V.Y. and Tung,C.
The Rebound Sorter:An Efficient Sort Engine for Large Files.
In Proceedings of the 4th International Conference on Very Large Data Bases, pages 312-318. , 1978.

[2] Foster,M.J.,Kung,H.T.
The Design of Special Purpose VLSI Chips.
Computer 13(1):26-40, January, 1980.

[3] Frank,G.A.,Greenawalt,E.M.,Kulkarni,A.V.
A Systolic Processor for Signal Processing.
In NCC-82. , 1982.

[4] Kung,H.T.,Leiserson,C.E.
Systolic Arrays (for VLSI).
In Sparse Matrix Proc., pages 256-282. Society for Industrial and Applied Mathematics, 1978.

[5] Kung H.T.
Let's Design Algorithms for VLSI Systems.
Technical Report CMU-CS-79-151, Computer Science Dept.,Carnegie--Mellon Univ., 1979.

[6] Kung,H.T.,Lehman,P.L.
Systolic (VLSI) Arrays for Relational Database Operations.
In Proc. ACM-Sigmod 1980, pages 105-116. , 1980.

[7] Leiserson,C.E.
Systolic Priority Queues.
In Proc. Conf. Very Large Scale Integration:Architecture, Design,Fabrication, pages 199-214. California Institute of Technology, 1979.

[8] Ramakrishnan,I.V.
Characterization of Programs Correctly Executable on a Model of VLSI Array Processors (in preparation).
PhD thesis, Department of Computer Sciences, U.T.Austin, 1982.