MODULAR MATRIX MULTIPLICATION ON
A LINEAR ARRAY

I. V. Ramakrishnan
Department of Computer Science
University of Maryland
College Park, MD 20742

P. J. Varman
Department of Electrical Engineering
Rice University
Houston, TX 77001

November 1983
MODULAR MATRIX MULTIPLICATION ON A LINEAR ARRAY

I. V. Ramakrishnan
Department of Computer Science
University of Maryland
College Park, MD 20742

P. J. Varman
Department of Electrical Engineering
Rice University
Houston, TX 77001

ABSTRACT

A matrix-multiplication algorithm on a linear array using an optimal number of processing elements is proposed. The local storage required by the processing elements and the I/O bandwidth required to drive the array are both constants that are independent of the sizes of the matrices being multiplied. The algorithm is therefore modular, that is, arbitrarily large matrices can be multiplied on a large array built by cascading small arrays. The array is well-suited for VLSI implementation.
1. Introduction

Specialized array processors have been proposed as a means of handling compute-bound problems in a cost-effective and efficient manner [4,5,6]. These array processors are typically made up of simple, identical processing elements (which we will refer to as cells from now on) that operate in synchrony. Several array structures have been proposed that include linear arrays, rectangular arrays and hexagonal arrays. Simplicity and regularity of linear, rectangular and hexagonal array processors render them suitable for VLSI implementation. High performance is achieved by extensive use of pipelining and multiprocessing. In a typical application, such arrays would be attached as peripheral devices to a host computer which inserts input values into them and extracts output values from them.

In practice, linear arrays are more attractive than rectangular or hexagonal arrays for several reasons. Among them are the following: Linear arrays have bounded I/O requirements [6]. In a wafer containing faulty cells, a large percentage of non-faulty cells can be efficiently reconfigured into a linear array with constant wire length between adjacent cells in the linear array [7]. Synchronization between cells in a linear array can be achieved by a simple global clock whose rate is independent of the size of the array [2].

Linear-array algorithms for dense matrix multiplication have appeared in [1,3,8]. These algorithms require $O(n)$ cells and $O(n^2)$ time steps to multiply two $n\times n$ matrices. However, these algorithms require that each cell in the linear array must have $O(n)$ words of local storage. Hence, the maximum storage in the cells imposes an upper limit on the size of the matrices that can be multiplied. Consequently, these matrix multiplication algorithms are not modularly expandable, that is, matrices larger than $n\times n$

\[\forall n \in \mathbb{N} \exists C \in \mathbb{R}^+ \text{ such that } O(n) \leq Cn\]
cannot be multiplied by cascading several such linear arrays into one large array. To do this, the local storage in each of the cells would have to be increased.

In this paper we present a novel linear-array algorithm for multiplying two $n \times n$ dense matrices wherein the local storage required by each cell in the linear array is a constant that is independent of the sizes of the matrices being multiplied. Therefore the algorithm is modular, that is, arbitrarily large matrices can be multiplied by extending the linear array. The algorithm requires $O(n^2)$ cells and the multiplication is done in $O(n^2)$ steps. We will also show that $O(n^2)$ cells used by the algorithm is asymptotically optimal. The time required to perform the multiplication ($O(n^2)$) is also asymptotically optimal as at least $n^2$ time steps are required to insert the elements into and retrieve the results from the array through a constant number of I/O ports.

The rest of this paper is organized as follows. In Section 2, we describe the cell and the linear array model that we will be using to describe the algorithm. In Section 3, we present the algorithm to multiply two $n \times n$ matrices and illustrate it by an example. In Section 4, a proof of the algorithm is provided and in Section 5 we show that $O(n^2)$ cells used by the algorithm is optimal.

2. Cell and Linear Array Model

We begin with a description of the cell model. Each cell (see Figure 2.1) is capable of performing a matrix multiplication step (i.e., a multiplication and an addition) in every clock cycle.
$I^\Phi$ and $I^\Psi$ are the two control input ports and $O^\Phi$ and $O^\Psi$ are the corresponding control output ports. In every cycle, the control signal at $I^\Psi$ is transmitted unchanged to $O^\Psi$ and the control signal at $I^\Phi$ is transmitted to $O^\Phi$ through a buffer $BUF_2$ that delays it by one cycle. At every clock cycle, $I^\Phi$ has one of the following three control signals: $\Phi_1, \Phi_2$ and "don't-care". (A two-bit wide $I^\Phi$ is therefore adequate.) Similarly, at every clock cycle, $I^\Psi$ has one of the following three control signals: $\Psi_1, \Psi_2$ and
IA, IB and IC are the input data ports for the elements of the matrices A, B and C respectively where \( C = A \times B \). The input data value at port IA is accompanied by a tag bit. We will denote the input data value at port IA as active if the tag bit is "on", else we will refer to it as being inactive. In every clock cycle the DEC unit (read as "decoding unit") strips the tag from the input value at IA. T denotes the tag bit and D the data.

The "dashed" lines are the control signals from the control unit to the adder, multiplier and the MOD unit (read as "modifying unit"). In every clock cycle, the MOD unit modifies the tag bit of the input value at IA depending on the control signal from the control unit. The modified tag bit from MOD is appended to the data at D in the ENC unit (read as "encoding unit").

BUF\(_1\) and BUF\(_2\) are two buffers whose sole purpose is to delay the input data at IB and the input control signal at I\(^\Phi\) respectively by one cycle.

We now describe the program executed by the cell in every cycle. At the beginning of a cycle, let \( a, b, c \) denote the data at ports IA, IB and IC respectively. Let \( t_8 \) denote the tag bit accompanying \( a \). Let \( c_1 \) and \( c_2 \) be the two input control signals at I\(^\Phi\) and I\(^Ψ\) respectively. The cell executes the following steps sequentially.

```
insert contents of BUF\(_1\) and BUF\(_2\) into output ports OB and O\(^Φ\) respectively;
if \( c_1 = \Phi_1 \) and \( c_2 = \Psi_1 \)
then begin
    set \( t_8 \) to "on" (i.e., activate \( a \));
    go to exit1;
end;
if \( c_1 = \Phi_2 \) and \( c_2 = \Psi_2 \)
then begin
    set \( t_8 \) to "off" (i.e., deactivate \( a \));
    go to exit1;
```
end;
if \( a \) is inactive then go to exit1;
if \( a \) is active then go to exit2;
exit1: insert \( c \) in output port \( OC \);
go to exit;
exit2: insert \( c + ab \) in output port \( OC \);
exit: insert output of ENC in output port \( OA \);
insert \( b \) into \( BUF_1 \); insert \( c_1 \) and \( c_2 \) into \( BUF_2 \) and \( OP \) respectively.

In every cycle, a cell either activates the data at \( IA \), or deactivates the data at \( IA \) or computes a matrix multiplication step provided the data at \( IA \) is active. The cell does not modify the tag bit when the control signals are "don't-care" control signals.

The linear array is comprised of cells indexed from 1 to \( m \) where \( m \) depends on the size of the matrices being multiplied. Figure 2.2 illustrates the linear array.

![Figure 2.2](image-url)

For any cell \( i \) in the linear array, its output ports \( O\Phi \), \( OA \) and \( OB \) are connected to the input ports \( I\Phi \), \( IA \) and \( IB \) respectively of cell \( i+1 \). Also, its output ports \( O\Psi \) and \( OC \) are connected to the input ports \( I\Psi \) and \( IC \) respectively of cell \( i-1 \).
External control signals are inserted at IΩ and IΨ of cell 1 and cell m respectively. The entries of matrix A are inserted at IA of cell 1. The tags accompanying each of these entries are set to “off” (i.e., the entries of matrix A are inactive when they enter the array). The entries of matrix B are inserted at IB of cell 1 and those of matrix C are inserted at IC of cell m.

3. Modular Matrix Multiplication Algorithm

We introduce the following notation to describe the algorithm. Let \( a_{ij}, b_{ij}, c_{ij} \) denote the \( ij \)th entry in matrices A, B and C respectively. Elements \( a_{ij} \) and \( a_{pq} \) in matrix A are said to be in the same diagonal if \( i+j=p+q \). The \( k \)th diagonal denotes the diagonal containing \( a_{ij} \) where \( i+j-1=k \).

The entries of matrix A are inserted in the following order: entries in the 1st diagonal, followed by entries in the 2nd diagonal, ..., followed by entries in the \((2n-1)\)th diagonal. Within any diagonal, the entries are inserted in increasing order of their column indices.

The entries of matrices B and C are inserted in the following order: entries in row 1, entries in row 2, ..., entries in row n. Within any row of matrix B the entries are inserted in decreasing order of their column indices. Within any row of matrix C the entries are inserted in increasing order of their column indices.

Recall that control signals pass through a cell without any change. A control signal at \( IΩ \) of a cell is transmitted unchanged to \( OΩ \) of the same cell at the end of two cycles and a control signal at \( IΨ \) of a cell is transmitted unchanged to \( OΨ \) at the end of one cycle. At each clock cycle a new control signal (either \( Φ_1, Φ_2, \) or “don’t-care”) is inserted at \( IΩ \) of cell 1. In the sequence of control signals inserted at \( IΩ \), let \( Φ_j \) (\( Φ_j^* \)) denote the \( j \)th \( Φ_1 \) (\( Φ_2 \)) signal (we assume that the indexing begins from 1). Similarly, in the
sequence of control signals inserted at \( \Psi \), let \( \Psi_i^1 \) (\( \Psi_i^2 \)) denote the \( i^{th} \) \( \Psi_1 \) (\( \Psi_2 \)) signal.

The number \( m \) of cells required by the algorithm is dependent on whether \( n \) is odd or even. Define \( r \) as follows: If \( n \) is odd, let \( r = \frac{n-1}{2} \) and if \( n \) is even let \( r = \frac{n}{2} \) (we assume \( n \geq 2 \)). Let \( t_0 \) denote the time at which \( \Psi_i^1 \) is inserted in the array.

Algorithm (for odd \( n \))

The number \( m \) of cells required by the algorithm for odd \( n \) is \( (n-1)(r+1)+n^2+2 \) and the algorithm is comprised of the following steps.

1. Insert \( a_{ij} \) into IA of cell 1 at time \( t_0+2+(n-1)(r)+n(i+j)+2+2j \);
2. Insert \( b_{ij} \) into IB of cell 1 at time \( t_0+1+3(r+1)(i-1)+(j-2) \);
3. Insert 0 into IC of cell 1 at time \( t_0+2+3n(i-1)+2(j-1) \);
4. Insert \( d_j \) into I\( d_j \) of cell 1 at time \( t_0+2+3(r+1)(j-1) \);
5. Insert \( d_j \) into I\( d_j \) of cell 1 at time \( t_0+2+3(r+1)(j-1) \);
6. Insert \( \Psi_i^1 \) into I\( \Psi \) of cell 1 at time \( t_0+3n(i-1) \);
7. Insert \( \Psi_i^1 \) into I\( \Psi \) of cell 1 at time \( t_0+2n+2+3n(i-1) \);
8. For all cycles between \( t_0-(n-1)(r+1)-n^2-1 \) and \( t_0+5n^2-2n+1 \) do the following:
   a. if no entry of matrix \( A \) is being inserted into IA of cell 1 then insert 0;
   b. if no valid control signals are being inserted into I\( \Phi \) and I\( \Psi \) of cell 1 and cell m respectively then insert "don't-care" control signals.

The number of cells required by the algorithm for even \( n \) is \( 3n(r+1) \). The algorithm is similar to the algorithm for odd \( n \) except steps 1 and 8. For even \( n \), \( a_{ij} \) is inserted at time \( t_0+1+(n-r)(n-1)+(i+j-2)(n-1)+(j-1) \) in step 1 and step 8 is carried out between cycles \( t_0-3n(r+1)-1 \) and \( t_0+1+(n-1)(3n+r+5) \).
Example: We illustrate the algorithm by multiplying two $3 \times 3$ matrices. $n=3$ and so the number of cells required is 15, that is, $m=15$. Let $t_0=14$.

Tables 1, 2 and 3 show the times at which the elements in matrices $A$, $B$ and $C$ respectively are inserted into the array.

\begin{center}
\begin{tabular}{ccc}
1 & 2 & 3 \\
1 & 20 & 24 & 28 \\
2 & 23 & 27 & 31 \\
3 & 26 & 30 & 34 \\
\end{tabular} \hspace{1cm}
\begin{tabular}{ccc}
1 & 2 & 3 \\
1 & 15 & 14 & 13 \\
2 & 21 & 20 & 19 \\
3 & 27 & 26 & 25 \\
\end{tabular} \hspace{1cm}
\begin{tabular}{ccc}
1 & 2 & 3 \\
1 & 16 & 18 & 20 \\
2 & 25 & 27 & 29 \\
3 & 34 & 36 & 38 \\
\end{tabular}
\end{center}

Table 1 \hspace{1cm} Table 2 \hspace{1cm} Table 3

In Tables 1, 2 and 3 the entry in the $i^{th}$ row and $j^{th}$ column is the time at which the $(ij)^{th}$ element in matrices $A$, $B$ and $C$ respectively is inserted into the array.

Entries in Table 4 below indicate the times at which the control signals are inserted. The entry 24 in the 3$^{rd}$ column of the 2$^{nd}$ row is the time at which $\phi_2^3$ is inserted into the port $I\Phi$ of cell 1.
Tables 5 and 6 give the times and the cells where $\Psi_i$ meets $\Phi_i$ and $\Psi_i$ meets $\Phi_i$ respectively.

<table>
<thead>
<tr>
<th>$\Phi_i$</th>
<th>$\Phi_i^2$</th>
<th>$\Phi_i^3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Psi_i^1$</td>
<td>$\langle a_{11}, 5, 24 \rangle$</td>
<td>$\langle a_{12}, 3, 26 \rangle$</td>
</tr>
<tr>
<td>$\Psi_i^2$</td>
<td>$\langle a_{21}, 8, 30 \rangle$</td>
<td>$\langle a_{22}, 6, 32 \rangle$</td>
</tr>
<tr>
<td>$\Psi_i^3$</td>
<td>$\langle a_{31}, 11, 36 \rangle$</td>
<td>$\langle a_{32}, 9, 38 \rangle$</td>
</tr>
</tbody>
</table>

Table 5

<table>
<thead>
<tr>
<th>$\Phi_i^2$</th>
<th>$\Phi_i^3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Psi_i^1$</td>
<td>$\langle a_{11}, 9, 28 \rangle$</td>
</tr>
<tr>
<td>$\Psi_i^2$</td>
<td>$\langle a_{21}, 12, 34 \rangle$</td>
</tr>
<tr>
<td>$\Psi_i^3$</td>
<td>$\langle a_{31}, 15, 40 \rangle$</td>
</tr>
</tbody>
</table>

Table 6
The entry in the $i^{th}$ row and $j^{th}$ column of Table 5 is a 3-tuple $<a_{ij}, x, y>$ where $x$ is the cell where $\Phi_j$ and $\Psi_j$ meet and $y$ is the time at which they meet at $x$. At the same time $a_{ij}$ also appears at the port $IA$ of $x$. Consequently $a_{ij}$ is activated in $x$ at time $y$.

Similarly the entry in the $i^{th}$ row and $j^{th}$ column of Table 6 gives the time and cell wherein $a_{ij}$ gets deactivated.

We will trace the computation of $c_{12}$ as an illustration. The trace is depicted in Table 7.

<table>
<thead>
<tr>
<th>$t$</th>
<th>index</th>
<th>$IA$</th>
<th>$IB$</th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td>15</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>14</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>13</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>12</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>11</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>10</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>9</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>8</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>7</td>
<td>$a_{11}$</td>
<td>$b_{12}$</td>
</tr>
<tr>
<td>27</td>
<td>6</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>5</td>
<td>$a_{12}$</td>
<td>$b_{22}$</td>
</tr>
<tr>
<td>29</td>
<td>4</td>
<td>$a_{31}$</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>3</td>
<td>$a_{13}$</td>
<td>$b_{32}$</td>
</tr>
<tr>
<td>31</td>
<td>2</td>
<td>$a_{32}$</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Table 7

Consider any $i^{th}$ row in Table 7. The $1^{st}$ column in the $i^{th}$ row is the time at which $c_{12}$ appears at the input port $IC$ of the cell whose index appears in the $2^{nd}$ column. The entries in the $3^{rd}$ and $4^{th}$ columns are the elements at the cell's $IA$ and $IB$ ports at that time. For instance, the $9^{th}$ row indicates that $c_{12}$ appears at the input port $IC$ of cell 7 at
time 26 and $a_{11}$ and $b_{12}$ are the elements at the ports IA and IB respectively of cell 7 at time 26.

The “starred” entries in the 3rd column are used to indicate that the corresponding entries are active. For instance, $a_{11}$ is active when it appears at the input port IA of cell 7 at time 26. On the other hand $a_{31}$ is inactive when it appears at the port IA of cell 4 at time 29.

From Table 7 it can be seen that $c_{12}$ gets updated only in cells 7, 5 and 3. In any other cell it is not updated as either it encounters a 0 or an inactive element of matrix A at the cell's IA port.

4. Proof of Correctness

We now establish the correctness of the algorithm. We will only prove this for odd $n$ as the proof for even $n$ is similar. Let $c_{ij}$ denote the element 0 inserted at IC of cell $m$ at time $t_0+2+3n(i-1)+2(j-1)$ in step 3 of the algorithm.

We will say that the elements of the three matrices and the control signals meet at a cell whenever they appear at the cell's input ports in the same cycle. (For instance, $a_{ii}$ and $b_{ij}$ meet at cell h if $a_{ii}$ and $b_{ij}$ appear at the input ports IA and IB respectively of cell h in the same cycle.)

Each cell in the linear array has five I/O ports (three for inserting and extracting elements of matrices A, B and C and two for inserting and extracting $\Phi_1$, $\Phi_2$ and $\Psi_1$, $\Psi_2$ control signals). In the following Lemma we show that these I/O ports are never “overloaded” by showing that distinct elements can never appear simultaneously at the same input port of any cell in the linear array.

Lemma 4.1: Distinct elements of matrices A, B and C do not simultaneously reach the
input ports IA, IB and IC of any cell in the linear array. Distinct \( \Phi_1, \Phi_2 \) control signals do not simultaneously reach the input port IA of any cell, and distinct \( \Psi_1, \Psi_2 \) control signals also do not simultaneously reach the input port IA of any cell.

**Proof:** We will show that distinct elements of matrix A do not simultaneously reach the input port IA of any cell and the proof will be similar for elements of matrix B and matrix C as well as for the control signals \( \Phi_1, \Phi_2, \Psi_1 \) and \( \Psi_2 \).

Let \( a_{ij} \) and \( a_{pq} \) be two distinct elements that appear simultaneously at the input port of cell s. The time taken by \( a_{ij} \) to reach the input port of s is 
\[
[t_0+2+(n-1)(n-r)+n(i+j-2)+(j-1)] + \{s\}.
\]
The expression within \([ \]\) is the time at which \( a_{ij} \) is inserted into the array and the expression within \( \{ \} \) is the time taken by \( a_{ij} \) to reach s after it is inserted. Similarly, the time taken by \( a_{pq} \) to reach s is 
\[
t_0+2+(n-1)(n-r)+n(p+q-2)+(q-1) + \{s\}.
\]
Equating these two times and simplifying we obtain 
\[
(i-p+j-q)\frac{j-q}{n} = \frac{j-q}{n}.
\]
Now the left-hand-side is an integer and the right-hand-side is a fraction since \( 0 < j-q < (n-1) \). So for equality to hold \( j=q \) and \( i=p \). So \( a_{ij} \) and \( a_{pq} \) are not distinct as assumed -- a contradiction.

Recall that a cell performs a matrix multiplication step only if the element at its IA port is active. Hence, for any \( c_{ij} \) to be correctly updated it must meet an active \( a_{is} \) (\( \forall s \mid 1 \leq s \leq n \)). We next identify the cells in which \( a_{is} \) is active.

**Lemma 4.2:** Let \( p=n(i-1)+(r+1)(n-s)+1 \) and \( q=n(i-1)+(r+1)(n-s)+n+2 \). If \( a_{is} \) is active in a cell \( y \) then \( p \leq y \leq q \).

**Proof:** \( a_{is} \) is activated whenever it meets a \( \Phi \) and \( \Psi \) control signal simultaneously. Let \( h \) be the cell index where \( a_{is} \) meets \( \Phi \) and \( \Psi \) simultaneously. Let \( t(a_{is}), t(\Phi) \) and...
t(\psi^s_i) denote the times at which a_i, \Phi^i and \psi^s_i respectively are inserted into the array. Let h(a_i), h(\Phi^i) and h(\psi^s_i) denote the time taken by a_i, \Phi^i and \psi^s_i respectively to reach h after being inserted into the array. Now a_i, \Phi^i and \psi^s_i meet at h. Hence 
\[ t(a_i)+h(a_i)=t(\Phi^i)+h(\Phi^i)=t(\psi^s_i)+h(\psi^s_i). \]
From the algorithm we obtain the following:

1. \[ t(a_i)=t_0+2+(n-1)(n-r)+n(i+s-2)+(s-1) \text{ and } h(a_i)=h-1. \]
2. \[ t(\Phi^i)=t_0+2+3(r+1)(f-1) \text{ and } h(\Phi^i)=2(h-1) \] (The multiplication factor 2 appears in h(\Phi^i) as \Phi^i control signals travel at a velocity of \( \frac{1}{2} \) a cell per clock cycle).
3. \[ t(\psi^s_i)=t_0+3(n-1) \text{ and } h(\psi^s_i)=m-h \] (h is subtracted from m in h(\psi^s_i) as \psi^s_i control signals travel from cell m to cell 1).

Now \[ t(\Phi^i)+h(\Phi^i)=t(\psi^s_i)+h(\psi^s_i) \] and so from (2) and (3) we can obtain \[ h=n(g-1)+(r+1)(n-f)+1. \] Also \[ t(a_i)+h(a_i)=t(\Phi^i)+h(\Phi^i) \] and so from (1) and (2) we can obtain \[ (n-1)(n-r)+n(i-1)+(n+1)(s-1)=3(r+1)(f-1)+h-1 \] which on substituting \[ h=n(g-1)+(r+1)(n-f)+1 \] simplifies to \[ n(s-f-g+i)=f-s. \] Since \( 0 \leq |f-s| \leq n-1 \), so for equality to hold \( f=s \) and \( g=i \). So \( \Phi^i = \Phi^i \) and \( \psi^s_i = \psi^s_i \) and \( h=p \). So \( a_i \) only meets \( \Phi^i \) and \( \psi^s_i \). It meets them at cell p. Hence \( a_i \) is activated in cell p.

We can similarly show that \( a_i \) only meets \( \Phi^s_j \) and \( \psi^j_i \) and it meets them at cell q and hence \( a_i \) is deactivated in cell q. Consequently, \( a_i \) is active only in a cell \( y \) where \( p \leq y \leq q. \)

Having identified the cells in which \( a_i \) is active, we will now establish that \( c_{ij} \) always meets an active \( a_i \) and \( b_{ij} \) (\( \forall s |1 \leq s \leq n \)) in the same cell.

**Lemma 4.3** Let \( p=n(i-1)+(r+1)(n-s)+1 \) and \( x=p+j \). Then, for any \( i,j,s \) (\( 1 \leq i,j,s \leq n \)),
1. $a_{i\hbox{\scriptsize{a}}}, b_{i\hbox{\scriptsize{j}}}$ and $c_{i\hbox{\scriptsize{j}}}$ will only meet at cell $x$, and

2. $a_{i\hbox{\scriptsize{a}}}$ is active then.

Proof: Let $a_{i\hbox{\scriptsize{a}}}, b_{i\hbox{\scriptsize{j}}}$ and $c_{i\hbox{\scriptsize{j}}}$ meet at cell $h$. Let $t(a_{i\hbox{\scriptsize{a}}}), t(b_{i\hbox{\scriptsize{j}}})$ and $t(c_{i\hbox{\scriptsize{j}}})$ denote the time at which $a_{i\hbox{\scriptsize{a}}}, b_{i\hbox{\scriptsize{j}}}$ and $c_{i\hbox{\scriptsize{j}}}$ respectively are inserted into the array. Let $h(a_{i\hbox{\scriptsize{a}}}), h(b_{i\hbox{\scriptsize{j}}})$ and $h(c_{i\hbox{\scriptsize{j}}})$ denote the time taken by $a_{i\hbox{\scriptsize{a}}}, b_{i\hbox{\scriptsize{j}}}$ and $c_{i\hbox{\scriptsize{j}}}$ respectively to reach cell $h$ after being inserted into the array. Equating $t(a_{i\hbox{\scriptsize{a}}})+h(a_{i\hbox{\scriptsize{a}}})$ to $t(b_{i\hbox{\scriptsize{j}}})+h(b_{i\hbox{\scriptsize{j}}})$ we can obtain $h=x=n(i-1)+(r+1)(n-s)+1+j$.

Now $a_{i\hbox{\scriptsize{a}}}, b_{i\hbox{\scriptsize{j}}}$ and $c_{i\hbox{\scriptsize{j}}}$ will pass through every cell indexed from 1 to $m$. We will first show that they pass through $h$ by showing that $1 \leq h \leq m$. The minimum value of $h$ is 2 which is obtained when $i=j=1$ and $s=n$. Clearly, $2 \geq 1$ and hence $h \geq 1$. The maximum value of $h$ is $n^2+(n-1)(r+1)+1$ which is obtained when $i=j=n$ and $s=1$. Clearly $n^2+(n-1)(r+1)+1 \leq m$ and hence $h \leq m$.

1. Hence $a_{i\hbox{\scriptsize{a}}}, b_{i\hbox{\scriptsize{j}}}$ and $c_{i\hbox{\scriptsize{j}}}$ meet at cell $x$. Lastly, cell $x$ is the only cell where they will meet as $c_{i\hbox{\scriptsize{j}}}$ travels in a direction opposite to that of $a_{i\hbox{\scriptsize{a}}}$ and $b_{i\hbox{\scriptsize{j}}}$.

2. That $a_{i\hbox{\scriptsize{a}}}$ is active follows immediately from Lemma 4.2.

From Lemma 4.3 we can assert that $c_{i\hbox{\scriptsize{j}}} \geq \sum_{s=1}^{s=n} a_{i\hbox{\scriptsize{s}}} b_{i\hbox{\scriptsize{s}}}$. To assert that $c_{i\hbox{\scriptsize{j}}} = \sum_{s=1}^{s=n} a_{i\hbox{\scriptsize{s}}} b_{i\hbox{\scriptsize{s}}}$ we must ensure that if $c_{i\hbox{\scriptsize{j}}}$ does not meet an active $a_{i\hbox{\scriptsize{a}}}$ in a cell then either it encounters an inactive element of matrix $A$ or the element 0 at the cell's IA port.

Lemma 4.4: If an active $a_{i\hbox{\scriptsize{a}}}$ meets $c_{i\hbox{\scriptsize{u}}}$ then $u=i$.

Proof: Let $p=n(i-1)+(r+1)(n-s)+1$ and $q=n(i-1)+(r+1)(n-s)+n+2$. By Lemma 4.2 $a_{i\hbox{\scriptsize{a}}}$ is active in any cell $y$ such that $p \leq y \leq q$. \hfill \Box
Let $t(a_u)$, $t(c_{uv})$ respectively denote the times at which $a_u$ and $c_{uv}$ are inserted into the array. Let $p(a_u)$ and $q(a_u)$ denote the times taken by $a_u$ to reach cell $p$ and cell $q$ respectively. Let $y(c_{uv})$ denote the time taken by $c_{uv}$ to reach cell $y$ after being inserted into the array. $c_{uv}$ meets an active $a_u$ and hence $t(a_u) + p(a_u) \leq t(c_{uv}) + y(c_{uv}) \leq t(a_u) + q(a_u)$.

Let $y = p + \Delta$. As $q - p = n + 1$ and $p \leq y \leq q$, so $0 \leq \Delta \leq n + 1$. Now $t(c_{uv}) = t_0 + 2 + 3n(u - 1) + 2(v - 1)$ and $y(c_{uv}) = m - p - \Delta$. Since $t(a_u) + p(a_u) \leq t(c_{uv}) + y(c_{uv})$ we can obtain:

$$\Delta \leq 3n(u - i) + 2v ......(a)$$

Also as $t(c_{uv}) + y(c_{uv}) \leq t(a_u) + q(a_u)$ we obtain:

$$\Delta \geq 3n(u - i) + 2v - n - 1 ......(b)$$

$\Delta \geq 0$ and so $3n(u - i) \geq -2v$. For $u < i$ this inequality does not hold as the minimum value of $-2v$ is $-2n$ and the maximum value of $3n(u - i)$ is $-3n$ when $u - i = -1$.

So $u \geq i ......(c)$

$\Delta \leq n + 1$ and so $3n(u - i) + 2v - n - 1 \leq n + 1$ which reduces to $3n(u - i) \leq 2(n + 1 - v)$. For $u > i$ this inequality does not hold as the maximum value of $2(n + 1 - v)$ is $2n$ when $v = 1$.

The minimum value of $3n(u - i)$ is $3n$ when $u > i$ and $u - i = 1$.

So $u \leq i ......(d)$

From (c) and (d), $u = i$. □

**Lemma 4.5**: In any cell $y$ in the linear array and for any $i, j$ ($1 \leq i, j \leq n$) $c_{ij}$ always encounters an element of matrix $A$ or a 0 at the IA port of cell $y$.

**Proof**: Let $t(c_{ij})$ denote the time when $c_{ij}$ is inserted into the array and $y(c_{ij})$ denote the
time taken to reach cell \( y \) after insertion. Now \( t(c_{ij}) = t_0 + 2 + 3n(i-1) + 2(j-1) \) and \( y(c_{ij}) = m - y \). The element encountered by \( c_{ij} \) at the IA port of cell \( y \) must have been inserted into the array at cell \( 1 \) at time \( z = t(c_{ij}) + y(c_{ij}) - y + 1 \). Recall from step 8 of the algorithm that either the element 0 or an element of matrix \( A \) is inserted into the array between cycles \( t_0 - (n-1)(n+1) - n^2 - 1 \) and \( t_0 + 5n^2 - 2n + 1 \). If we show that \( t_0 - (n-1)(n+1) - n^2 - 1 \leq z \leq t_0 + 5n^2 - 2n + 1 \) then clearly the element inserted into the array at the IA port of cell \( 1 \) is either the element 0 or the element of matrix \( A \).

Now \( z = t(c_{ij}) + y(c_{ij}) - y + 1 = t_0 + 2 + 3n(i-1) + 2(j-1) + n^2 + 2 + (n-1)(r+1) - 2y + 1 \)

It can be easily seen from the expression above that \( z \) is minimum when \( i \) and \( j \) are minimum and \( y \) is maximum. \( i=1 \) and \( j=1 \) are the minimum values for \( i \) and \( j \) and \( y = m = (n-1)(r+1) + n^2 \) is the maximum value of \( y \). Similarly, \( z \) is maximum when \( i=n \), \( j=n \) and \( y=1 \). Let \( z_{\text{max}} \) and \( z_{\text{min}} \) denote the maximum and minimum \( z \) respectively. It can be easily shown that \( z_{\text{min}} \geq t_0 - (n-1)(r+1) - n^2 - 1 \) and \( z_{\text{max}} \leq t_0 + 5n^2 - 2n + 1 \).

We can now assert that \( c_{ij} \) is correctly computed when it exits the array.

**Theorem 4.1:** For any \( i,j \) (\( 1 \leq i,j \leq n \)), the value of \( c_{ij} \) is \( \sum_{s=1}^{i=n} a_{is} b_{sj} \) when it exits the array.

**Proof:** By Lemma 4.5 \( c_{ij} \) will either meet an element of matrix \( A \) or the element 0 at any cell.

1. By Lemma 4.3 it will meet \( a_{is} \) and \( b_{sj} \) in the same cell.

2. By Lemma 4.4 if it meets an element \( a_{uv} \) of matrix \( A \) and \( u \neq v \) then \( a_{uv} \) is inactive.
From (1) and (2) the Theorem follows.

5. Proof of Optimality

We will now establish that the number of cells used by the modular linear-array algorithm is asymptotically optimal. We establish this result under the following assumptions:

1. Any special-purpose machine (like a linear array) that multiplies matrices A and B must compute $a_{ik}b_{kj}$ (for all $i,j,k$ such that $1 \leq i,j,k \leq n$).
2. The special-purpose machine has a constant number of I/O ports.
3. The elements of the matrices A, B and C are inserted into the special-purpose machine only once through the input ports.

Under these assumptions we will establish that $\Omega(n^2)$ is a lower bound on the storage that is required by any special-purpose machine that multiplies two $n \times n$ matrices. We obtain this bound by formulating the computation of matrix multiplication as a game played with tokens on an undirected graph constructed as follows:

Let $G_k=(V_k, E_k)$, $k=1,\ldots,n$ where

$V_k=\{f_{ik}, h_{kj} \mid i=1,\ldots,n \text{ and } j=1,\ldots,n\}$ and

$E_k=\{<f_{ik}, h_{kj}> \mid i=1,\ldots,n \text{ and } j=1,\ldots,n\}$

The rules of the game are as follows:

1. A token is placed on $f_{ik}$ ($h_{kj}$) when $a_{ik}$ ($b_{kj}$) is inserted into the machine.
2. Updating $c_{ij}$ (by adding $a_{ik}b_{kj}$ to $c_{ij}$ for some $k$) results in removing the edge $<f_{ik}, h_{kj}>$ from $G_k$.

$f(n) = \Omega(n^2)$ if there exists a positive constant $c$ for which $f(n) \geq cn^2$
3. An edge is removable only if there are tokens at both end vertices.

4. A token from a vertex is removable only if all the edges incident on the vertex are removable. When a token from a vertex is removed then all the incident edges on the vertex are deleted. (The token will eventually leave the machine and will never reenter.)

We will assume that each token occupies unit storage ($O(1)$). We also assume that a partially updated $c_{ij}$ also occupies unit storage. (At any instant of time $c_{ij}$ is partially updated if there exists some $k$ ($1 \leq k \leq n$) such that $a_{ik}b_{kj}$ either has not been computed and/or added to $c_{ij}$ by that time instant.)

Let $x_k$ be the earliest time at which the first token in $G_k$ is removable and let $y_k$ be the earliest time at which all the tokens in $G_k$ are removable. Since only a constant number of tokens enter the machine at any time, by choosing $n$ sufficiently large, we can ensure that $\forall k$ ($1 \leq k \leq n$) $x_k < y_k$. $\forall k$ ($1 \leq k \leq n$), let $l_k = [x_k, y_k]$ denote the time interval between and including $x_k$ and $y_k$.

Lemma 5.1: At any time $t$ such that $x_k \leq t < y_k$, there are at least $n$ tokens in $G_k$.

Proof: Without any loss of generality, let the first (or one of the first if there are more than one) token(s) that can be removed from $G_k$ be the one on vertex $f_{mk}$. At $t_1 = x_k$, then, there must be tokens on all $h_{kj}$ ($1 \leq j \leq n$). We claim that no token on any $h_{kj}$ will be removable at any $t$ ($x_k \leq t < y_k$).

Assume this is not the case, and at $t < y_k$, let $h_{kj}$ be the first vertex (or one of the first vertices) from which a token is removable. This implies that there must be tokens on all vertices $f_{jk}$ that still have incident edges. This means that all the edges still remaining in $G_k$ are removable, and consequently all the remaining tokens in $G_k$ are...
removable at time $t$. But then $t = y_k$ -- a contradiction. Hence no token on any $h_{kj}$ is removable at any time $t$ ($x_k \leq t < y_k$). Each $h_{kj}$ has a token and hence the Lemma.

\[ \square \]

**Lemma 5.2:** Let $m < n$. For any $i$, if $t \geq y_i$ and $G_i$ has $m$ tokens then at least \( \frac{n^2}{2} \) edges must have been deleted from $G_i$.

**Proof:** There are $m$ tokens in $G_i$. Since $t \geq y_i$, the absence of a token on a vertex means that all the $n$ edges incident on the vertex have been deleted. (At $t = y_i$, all edges in $G_i$ are removable). The number of absent tokens $= 2n - m$ which is greater than $n$ as $m < n$. Now one edge is in common with at most two vertices. Thus the $2n - m$ absent tokens result in at least $\frac{n^2}{2}$ deleted edges. \[ \square \]

Let us impose an ordering on the sets $I_k$ such that $x_1 \leq x_2 \leq \ldots \leq x_n$ and let $\Gamma = \{ I_k \mid y_k \leq x_k \}$ and $\Lambda = \{ I_k \mid y_k > x_k \}$.

**Theorem 5.1:** Any matrix-multiplication machine requires $\Omega(n^2)$ storage.

**Proof:** Since $|\Gamma| + |\Lambda| = n$, either $|\Gamma| \geq \frac{n}{2}$ or $|\Lambda| \geq \frac{n}{2}$.

**Case 1:** $|\Lambda| \geq \frac{n}{2}$ (see Figure 5.1)
At $t=x_i$, all the intervals in $A$ satisfy Lemma 5.1. Hence at $t=x_i$, there are at least
$n(\frac{n}{2})$ tokens in the machine. So the storage required is $\Omega(n^2)$.

Case 2: $|\Gamma| \geq \frac{n}{2}$ (see Figure 5.2)

At $t=x_i$, either all $G_k$, such that $I_k \in A$, have $n$ tokens on them, or at least one of them
has less than $n$ tokens. If every $G_k$ has $n$ tokens then the storage required is again
$\Omega(n^2)$. If any one, say $G_1$, has less than $n$ tokens then by Lemma 5.2 $G_1$ must have
released at least $\frac{n^2}{2}$ edges. Now each released edge corresponds to a partially updated
c_{ij}. None of the $c_{ij}$'s could have left the machine as all of them are finally updated only
at $t \geq x_i$. Thus at any time $t$ ($y_k \leq t \leq x_i$) there are at least $\frac{n^2}{2}$ partially updated $c_{ij}$'s in
the machine. The case $y_k = x_i$ is covered by assumption 2 which precludes the possibil-
ity of all these \( c_{ij} \)'s being instantaneously updated and leaving the machine. So the storage required for the partially updated \( c_{ij} \)'s must be \( \Omega(n^2) \).

**Theorem 5.2:** \( O(n^2) \) cells used by the modular linear-array algorithm is optimal.

**Proof:** From Theorem 5.1 it follows that the modular linear-array algorithm requires \( \Omega(n^2) \) storage. Now each cell in the linear array has constant storage and hence the Theorem.

**Conclusion**

We have described a novel linear-array matrix multiplication algorithm that uses an asymptotically optimal number of cells. The cells used in the array are simple requiring a constant amount of local storage that is independent of the sizes of the matrices being multiplied. The cells can be built using off-the-shelf components. The array can be modularly expanded to accommodate arbitrary matrix sizes by adding more of these simple cells.

**References**


(October, 1982), pp. 1000-1009.


MODULAR MATRIX MULTIPLICATION ON A LINEAR ARRAY

A matrix-multiplication algorithm on a linear array using an optimal number of processing elements is proposed. The local storage required by the processing elements and the I/O bandwidth required to drive the array are both constants that are independent of the sizes of the matrices being multiplied. The algorithm is therefore modular, that is, arbitrarily large matrices can be multiplied on a large array built by cascading small arrays. The array is well-suited for VLSI implementation.