MAPPING DATA FLOW GRAPHS ON VLSI PROCESSING ARRAYS

by

and G. M. Silberman ****

Technical Report #306

December 1983

* Mathematical Institute of the Hungarian Academy of Sciences, Budapest, Hungary.
** Dept. of Computer Science, Technion, Haifa, Israel.
*** Depts. of EE and Computer Science, Technion, Haifa, Israel.
**** Depts. of Computer Science and EE., Haifa, Israel.
ABSTRACT

This study examines the complexity of mapping data flow graphs onto square and hexagonal arrays of processors, implemented in VLSI technology. We specifically consider the problem of routing data from processors in a given (source) sequence to another (target) sequence. An analysis of the problem's complexity is given, and the implications of this result to the mapping obtained by our method are discussed. For example, our results show that reordering \( n \) values in the array will use, in the worst case, \( \Theta(n - \sqrt{n}) \) array rows (an upper bound of \( 2n \) is always achievable).
TABLE OF CONTENTS

1. INTRODUCTION

2. THE MAPPING PROBLEM: MOTIVATION AND DESCRIPTION
   2.1 DATA FLOW GRAPHS
   2.2 PROCESSING ARRAYS
   2.3 MAPPING GRAPHS ONTO ARRAYS
      2.3.1 A GENERAL DESCRIPTION
      2.3.2 FIRST PHASE
      2.3.3 SECOND PHASE
   2.4 PROBLEM STATEMENT

3. A MATHEMATICAL FORMULATION OF THE PROBLEM
   3.1 BASIC NOTATIONS
   3.2 AN EXAMPLE
   3.3 IMPLICATIONS TO THE MAPPING PROBLEM

4. MAIN RESULTS
   4.1 SUMMARY OF MAIN RESULTS
   4.2 PROOF OF THEOREM 1
   4.3 DISCUSSION OF THEOREM 2

5. CONCLUSIONS

REFERENCES
1. INTRODUCTION

Recent developments in VLSI technology have made it possible to build relatively large processing arrays in a single chip. The design of these regular structures takes considerably less time than custom circuits, making them economically attractive. Nevertheless, new algorithms must be developed which use these arrays, in order to take advantage of their potential for parallel processing.

In order to avoid the need for developing special purpose algorithms, we have adopted the data flow approach as proposed in [1]. In this manner, existing algorithms are translated into data flow graphs, which are then mapped onto regular processing arrays. The resulting structure exploits the concurrency inherent to the original problem at its lowest possible level, yielding an efficient implementation with no need to develop new algorithms (for a more detailed discussion of the data flow concept, see [2]).

This paper examines the complexity of mapping data flow graphs onto processing arrays. Specifically, we address those problems which arise from the non-planarity exhibited by these graphs, as a consequence of operand ordering and loops. The array topologies being considered are square and hexagonally connected structures.

In the above context, we study the routing of $n$ output values from a given row in the array, to be used as inputs to another such row. This latter row is determined by the constraints imposed by the given mapping process, as we shall see below. It is shown that under the given constraints, the mapping process requires at most $O(n)$ intermediate rows, in order to achieve the above routing. Furthermore, we show cases in which $O(n)$ rows are needed for the solution. This $O(n)$ bound can be easily obtained by applying any standard permutation network [3]. In view of our results, it follows that usage of such networks cannot be considerably improved upon, in the worst case.

In the following section we introduce our approach to the solution of the mapping problem, following [1]. Section 3 presents the mathematical formulation.
of the problem and introduces the notation used throughout. Section 4 presents the main results on the complexity of the mapping process. These results are then used in Section 5 to determine area properties for the array configuration obtained by applying our solution to the mapping problem.

2. THE MAPPING PROBLEM: MOTIVATION AND DESCRIPTION

2.1. DATA FLOW GRAPHS

We explore the problems involved in the mapping of data flow graphs onto an array of processors. These graphs represent programs in the context of data flow computers. In these computers - as opposed to control flow machines (i.e., using a "Program Counter") - execution of instructions may proceed as soon as all the corresponding input operands become available [4].

In data flow graphs (see for example Figure 1), each node corresponds to an operator (e.g., plus, minus, boolean and, etc.), and operands move as "tokens" along the directed arcs connecting these nodes.

Notice (in Figure 1) that several operations may proceed concurrently, as soon as their operands arrive. There is no need to synchronize execution of the different operations, or otherwise determine their order. Therefore, data flow graphs enable concurrency of activities at the lowest possible level, resulting in "fine grain parallelism" [4].

2.2. PROCESSING ARRAYS

In considering array topologies which are appropriate for embedding data flow graphs, we begin by observing that we may restrict these graphs to nodes having at most two inputs and two outputs (in the case of two outputs, we may further require them to be identical). This immediately suggests the square array of processors shown in Figure 2. A somewhat more sophisticated setup is an hexagonal array of processors, such as the one used in [1] and shown in Figure 3.
A data flow graph
Figure 1
A square array of processors  
Figure 2

An hexagonal array of processors  
Figure 3
2.3. MAPPING GRAPHS ONTO ARRAYS

2.3.1. A GENERAL DESCRIPTION

The problems we study deal with the process of mapping a given data flow graph onto a given array of processors. This process is carried out by assigning certain individual processors in the array to nodes in the data flow graph, in a one-to-one fashion, and assigning edge-disjoint paths in the array to arcs in the graph.

For example, the mapping of the data flow graph in Figure 1 onto a square array is shown in Figure 4.

We notice from the above example that the mapping process is complicated by the non-planarity of the data flow graph. In this example, six rows are used only for the rearrangement of the data operands $a_1, a_2, a_3,$ and $a_4$, while the actual processing is performed by the processors on the other rows.

In this study we analyze the mapping process as proposed in [1]. This process may be viewed as composed of two phases, as detailed below.

2.3.2. FIRST PHASE

We begin by assigning levels to the nodes (operators) of the data flow graph. The operators accepting only external input operands are said to be at level 0. We say that an operator is at level $i$, if one of its inputs comes from an operator at level $i-1$, and none come from higher-numbered levels.

In the mapping process all the operators at level $i$ will be mapped onto some row $r_i$ of the processor array, where $r_i < r_{i+1}$ for all $i$. The values of the $r_i$-s will be successively determined, for increasing $i$, in the second phase (note that $r_0=1$).

2.3.3. SECOND PHASE

Each processor at row $r_{i+1}$ receives its inputs (one or two) from at most two processors at row $r_i$. When two such processors (at row $r_i$) are not adjacent, our approach - following [1] - is to route their values (outputs) to a pair of adjacent
Mapping a data flow graph onto a square array

Figure 4
processors at row $r_{i+1} - 1$.

Let $P_1, ..., P_n$ be the sequence of processors at row $r_i$, and let $Q_1, ..., Q_m$ be the sequence of processors at row $r_{i+1} - 1$ which supply the inputs to row $r_{i+1}$. Then it follows from the above that if $P_i$ and $P_j$ are to supply the inputs to a certain processor at row $r_{i+1}$, then there must be a pair of adjacent processors, $Q_k$ and $Q_{k+1}$ which receive their output values (see Figure 5). Routing of values from $P_1, ..., P_n$ so as to achieve an ordering $Q_1, ..., Q_m$ satisfying the above property is performed in this phase.

2.4. PROBLEM STATEMENT

In the sequel we study the complexity of the second phase of the mapping process, and obtain upper and lower bounds for the required number of rows as a sequence of row $r_i$.
function of the number of external inputs to the data flow graph in some special cases.

We consider the special case where (a) the number of processors in row $r_{i+1}$ is equal to the number of processors in row $r_i$, and (b) the output of each processor in row $r_i$ is the input operand to exactly two processors in row $r_{i+1}$ (see Figure 4). Consequently, there are $n+1$ processors on row $r_{i+1} - 1$, (i.e., $m = n + 1$), and there is a permutation $\pi$ of $\{1,2,...,n\}$ such that

$$Q_1, Q_2, \ldots, Q_n, Q_{n+1} = P_{\pi(1)}, P_{\pi(2)}, \ldots, P_{\pi(n)}, P_{\pi(1)}.$$ 

Notice that the permutation $\pi$ can be replaced by any of its cyclic permutations, and still fulfill the requirements of the mapping process, e.g., the arrangements

$$P_{\pi(2)} P_{\pi(3)} \ldots P_{\pi(n)} P_{\pi(1)} P_{\pi(2)}$$

$$P_{\pi(3)} P_{\pi(4)} \ldots P_{\pi(1)} P_{\pi(2)} P_{\pi(3)}$$

$$P_{\pi(n)} P_{\pi(1)} \ldots P_{\pi(n-2)} P_{\pi(n-1)} P_{\pi(n)}$$

can replace the original one in the mapping process.

Each such arrangement determines the ordering of the operators in row $r_{i+1}$ and, together with the order of the processors on level $r_i$, we then determine the value $r_{i+1}$, so as to minimize some complexity criteria.

3. A MATHEMATICAL FORMULATION OF THE PROBLEM

3.1. BASIC NOTATIONS

Let $\sigma = (a_1, a_2, \ldots, a_n)$ be a permutation of the elements $1,2,\ldots,n$. Define

$$\sigma^R = (a_n, \ldots, a_1)$$

the reverse permutation of $\sigma$.

$$D(\sigma) = \max \{|a_i-i| \mid i=1,2,\ldots,n\}$$

is the diameter of $\sigma$. We also define
\[ CYC(\sigma) = \{ \pi \mid \pi \text{ is a cyclic permutation of } \sigma \} \]

\[ F(\sigma) = \min \{ D(\pi) \mid \pi \in CYC(\sigma) \} \]

\[ G(\sigma) = \min \{ D(\pi) \mid \pi \in CYC(\sigma) \cup CYC(\sigma^R) \} \]

\[ F(n) = \max \{ F(\sigma) \mid \sigma \in S_n \} \]

\[ G(n) = \max \{ G(\sigma) \mid \sigma \in S_n \} \]

\( (S_n \text{ denotes the set of all the } n! \text{ permutations of the elements } 1,2,...,n. \)

It follows immediately that

\[ F(\sigma) \geq G(\sigma) \]

for every \( \sigma \in S_n \), and therefore

\[ F(n) \geq G(n) \]

for every \( n \).

The proof of the following lemma is straightforward (but a little tedious), and is left to the reader.

**Lemma:** For every \( n \), the following inequalities hold:

\[ F(n+1) \leq F(n) + 1, \]

\[ G(n+1) \leq G(n) + 1. \]

### 3.2. AN EXAMPLE

To demonstrate the above, let \( n=5 \), and

\[ \sigma = (1,2,5,3,4). \]

For short we shall write

\[ \sigma = 1 \underline{2534}. \]

Then

\[ \sigma^R = 4 \underline{3521}. \]

and

\[ D(\sigma) = \max \{|1-1|,|2-2|,|5-3|,|3-4|,|4-5|\} = |5-3| = 2. \]

The meaning of \( D(\sigma) \) is depicted in Figure 6.
The function $D(a)$
Figure 6

In the first row we draw $n$ nodes and label them with the numbers 1 through $n$, and in the second row we draw $n$ nodes and label them with the numbers $a_1$ through $a_n$. We then connect the nodes having the same label. Note that $|a_i - i|$ is the distance of the element $a_i$ from his original location. In our example $|a_3 - 5| = |5 - 3| = 2$, and this reflects the fact that $a_3 = 5$ is the element farthest from his origin.

The sets $CYC(a)$ and $CYC(a^R)$ are shown in Figure 7. For each permutation $\pi$ we show its diameter $D(\pi)$ where all the elements $i$ satisfying $|a_i - i| = D(\pi)$ are shown enlarged.

<table>
<thead>
<tr>
<th>CYC(a)</th>
<th>CYC(a^R)</th>
<th>$\pi$</th>
<th>$D(\pi)$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 2 5 3 4</td>
<td>1 2 5 3 4</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>4 1 2 5 3</td>
<td>4 1 2 5 3</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>3 4 1 2 5</td>
<td>3 4 1 2 5</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>5 3 4 1 2</td>
<td>5 3 4 1 2</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>2 5 3 4 1</td>
<td>2 5 3 4 1</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>4 3 5 1 2</td>
<td>4 3 5 1 2</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>1 4 3 5 2</td>
<td>1 4 3 5 2</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>2 1 4 3 5</td>
<td>2 1 4 3 5</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>5 2 1 4 3</td>
<td>5 2 1 4 3</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>3 5 2 1 4</td>
<td>3 5 2 1 4</td>
<td>3</td>
</tr>
</tbody>
</table>

The sets $CYC(a)$ and $CYC(a^R)$
Figure 7
We see that $F(\sigma) = 2$, and this means that if we consider the permutation $\sigma = 1 2 5 3 4$ and its cyclic shifts, then we can find at least one permutation with diameter 2, and none with a lower diameter. Here this smallest diameter is achieved by the permutations 1 2 5 3 4 and 3 4 1 2 5.

$G(\sigma) = 1$

and this reflects the fact that among the ten permutations in $CYC(\sigma) \cup CYC(\sigma^R)$ the minimal diameter is 1. This is achieved by the permutation 2 1 4 3 5.

$F(n)$ and $G(n)$ are harder to demonstrate, since we have to consider all $5! = 120$ permutations of five elements, compute the functions $F$ and $G$ for each permutation, and take the largest one found. The reader may verify that $F(5) = 3$ (take, for example, the permutation 3 4 1 5 2), and that $G(5) = 2$ (take, for example, the permutation 5 2 4 3 1).

The values of $F(n)$ and $G(n)$ for $n = 1, 2, \ldots, 10$ are shown in Figure 8.

<table>
<thead>
<tr>
<th>$n$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F(n)$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>$G(n)$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

$F(n)$ and $G(n)$ for $n = 1, 2, \ldots, 10$

Figure 8

In fact, it will follow that for all $n$

$$F(n) - G(n) = 0, 1 \text{ or } 2$$

and we conjecture that

$$F(n) > G(n) \text{ for } n > 3.$$
processors \( Q_1, Q_2, \ldots, Q_n \) on row \( r_{i+1} - 1 \) correspond to the permutation \( \sigma = a_1 a_2 \ldots a_n \).

If we concentrate on those special cases when \( Q_k \) and \( Q_{k+1} \) supply the inputs to the \( k \)-th processor on row \( r_{i+1} \), then every circular shift of the \( Q_i \)'s is also a valid configuration. In our terms, each of the permutations in \( CYC(\sigma) \cup CYC(\sigma^R) \) will satisfy the desired requirements. Hence, the functions \( F(\sigma) \) and \( F(n) \) are of interest. If we consider the case when all the operators are commutative, then every circular shift of the \( Q_i \)’s or their reversals is also a valid configuration. In our terms, each of the permutations in \( CYC(\sigma^R) \) will satisfy the desired requirements. Thus, the functions \( G(\sigma) \) and \( G(n) \) are of interest, in this case.

4. MAIN RESULTS

4.1. SUMMARY OF MAIN RESULTS

Our main results are that both \( F(n) \) and \( G(n) \) are very close to \( n - \sqrt{n} \). In fact, we have an exact form for \( F(n) \) and an almost exact form for \( G(n) \). More specifically, we have the following two theorems.

**Theorem 1:** For all \( n \geq 1 \)

\[
F(n) = n - \alpha(n),
\]

where

\[
\alpha(n) = \min\{ k \mid k^2 + k + 1 \geq n \}.
\]

**Theorem 2:** For all \( n \geq 8 \)

\[
n - \beta(n) \leq G(n) \leq n - \gamma(n),
\]

where

\[
\beta(n) = \min\{ k \mid k^2 - k - 4 \geq n \},
\]

\[
\gamma(n) = \min\{ k \mid k^2 - 2 \geq n \}.
\]

4.2. PROOF OF THEOREM 1

We now give a full proof of Theorem 1. In order to demonstrate our technique we first show the following weaker result:

\[
G(\sigma) \geq n - 2\sqrt{n}.
\]
To see this we take for simplicity \( n = k^2 \) for some \( k \). Let \( \sigma \) be the permutation \( a_1, a_2, ..., a_n \), defined as follows: \( a_1=1, a_{2k+1}=2, a_{2k+2}=3, ..., a_{(k-1)k+1}=k, \) and \( a_j \) is arbitrary otherwise. In Figure 9 we show the permutation \( \sigma \) for \( k = 3, 4 \) (a * means that any element can be put in that location; we use this notation throughout). It is clear that in every cyclic permutation of \( \sigma \) or \( \sigma^R \), one out of the last \( k \) entries will be occupied by some \( i, i \leq k \), and this means that \( G(\sigma) \geq n - 2\sqrt{n} \), as desired.

\[ \begin{array}{c|cccccccc}
 k=3 \\
i & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
a_i & 1 & * & * & 2 & * & * & 3 & * & *
\end{array} \]

\[ \begin{array}{c|cccccccccccccccc}
 k=4 \\
i & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 \\
a_i & 1 & * & * & * & 2 & * & * & * & 3 & * & * & * & 4 & * & * & *
\end{array} \]

The permutation \( \sigma \) for \( k = 3, 4 \)  
Figure 9

The proof of Theorem 1 is done separately for the upper and lower bounds, as follows.

Upper bound

Let \( n \) be given. We show that

\[ F(n) \leq n - \alpha(n), \]

where

\[ \alpha(n) = \min \{ k \mid k^2 + k - 1 \geq n \}. \]

This means that
\[ k^2 + k \geq n \implies F(n) < n - k. \]

We first consider the case when \( k^2 + k > n \), and then the case when \( k^2 + k = n \) (if such a \( k \) exists).

**Case 1:** \( k^2 + k > n \).

We shall show that for each \( \sigma \in S_n \) the number of permutations \( \pi \) in \( CYC(\sigma) \) for which \( D(\pi) \geq n - k \) is at most \( k^2 + k \), and hence less than \( n \); this will imply that for each permutation \( \sigma \) there exist at least one permutation in \( CYC(\sigma) \) with diameter smaller than \( n - k \), and hence that \( F(n) < n - k \).

Let \( i \) be in \( \{1, 2, \ldots, n\} \). We denote by \( m_i \) the number of permutations \( \pi \) in \( CYC(\sigma) \) in which \( i \) is at distance at least \( n - k \) from its natural location, and we say that \( i \) is far in these permutations. Clearly, there are exactly \( k \) permutations in \( CYC(\sigma) \) in which \( i \) is far – namely, the permutations in which \( i \) occupies one of the last \( k \) entries, that is \( a_i = 1 \) for some \( i \) in \( \{n - k + 1, n - k + 2, \ldots, n\} \). Similarly, there are exactly \( k \) permutations in which \( n \) is far. Hence,

\[ m_1 = m_n = k. \]

In the same way one can easily verify that

\[ m_2 = m_{n-1} = k - 1. \]

\[ m_3 = \ldots = m_{n-k+1}. \]

\[ m_k = m_{n-k+1} = 1. \]

and

\[ m_i = 0 \text{ for } k < i < n - k. \]

Hence,

\[ \sum m_i = 2(k + (k-1) + \ldots + 1) = k^2 + k. \]

This means that the number of far permutations (i.e., permutations in which at least one element is far) in \( CYC(\sigma) \) cannot exceed \( k^2 + k \), since in \( \sum m_i \) each far permutation is counted at least once. This completes the proof of case 1.
Note that the number of far permutations in $CYC(\sigma)$ is exactly $k^2 + k$ if and only if no permutation is over-counted in $\sum m_i$, i.e., each far permutation contains a unique far element.

Case 2: $k^2 + k = n$.

The theorem holds clearly for the case where $k=1$ (and $n=2$), so assume $k>1$.

Let $\sigma$ be a permutation in $S_n$ and let $k$ be such that $k^2+k=n$. We have to show that

$$F(\sigma) < n-k.$$ 

From the above discussion $F(\sigma) \leq n-k$, with equality if and only if in each cyclic shift of $\sigma$ exactly one element is far. It remains to show that this last case (corresponding to $F(\sigma) = n-k$) is impossible.

Consider all the cyclic shifts of the permutation $\sigma$ and assume that indeed there is a unique far element in each of them. This far element can be either among the last $k$ elements (and hence is equal to one of the elements $1, 2, \ldots, k$), or among the first $k$ elements (and hence is equal to one of the elements $n-k+1, n-k+2, \ldots, n$). It follows that there must be two successive permutations

$$\pi_1 = \{a_1, a_2, \ldots, a_n, a_1, \ldots, a_{k-1}\}$$

$$\pi_2 = \{a_{k+1}, a_{k+2}, \ldots, a_n, a_1, \ldots, a_k\}$$

such that in $\pi_1$ the far element is among the first $k$ elements (i.e., belongs to the set $\{a_1, \ldots, a_{k-1}\}$), and in $\pi_2$ the far element is among the last $k$ elements (i.e., belongs to the set $\{a_{k+1}, \ldots, a_n\}$). In all the cases mentioned above an index $j$ denotes $(j-1) \mod n + 1$.

Since none of the elements $a_1, a_2, \ldots, a_{k-1}$ is far in $\pi_2$, none of them can be far in $\pi_1$; hence, the unique far element in $\pi_1$ is the first element $a_k$, which means that

$$a_k \geq n - k + 1.$$ 

Similarly, the unique far element in $\pi_2$ is the last element, which is also $a_k$, which means that

$$a_k \leq k.$$
Thus, we have

\[ k^2 + 1 = n - k + 1 \leq a_i \leq k, \]

a contradiction.

**Lower bound**

We show a construction for permutations \( \sigma \) satisfying

\[ F(\sigma) \geq n - a(n). \]

The following program generates permutations \( \sigma = a_1, \ldots, a_n \), where

\[ n = k^2 + k - 1, \text{ such that } F(\sigma) = n - k. \]

\begin{verbatim}
begin
  [distribution of large elements n-k+1 - n]
  i := 1;
  a_i := n - k + 1;
  for j := 2 to k do
  begin
    i := i + j;
    a_i := n - k + j;
  end;
  [distribution of small elements 1 - k]
  i := i - 1;
  a_i := 1;
  for j := 2 to k do
  begin
    i := i + k + 1 - j;
    a_i := j;
  end;
  [distribute the remaining elements arbitrarily]
end.
\end{verbatim}

The construction for the cases \( k = 3, 4 \) is shown in Figure 10.
From the lemma (section 3.1) it follows that for any \( n \) such that
\[
k^2 + k - 1 < n < (k + 1)^2 + (k + 1) - 1 = k^2 + 3k + 1
\]
we have
\[
F(n) = n - k - 1.
\]
Since for these \( n \)'s
\[
a(n) = k + 1
\]
we have that for all \( n \)
\[
F(n) = n - a(n).
\]

4.3. DISCUSSION OF THEOREM 2

The proof of Theorem 2 follows the same outline of that of Theorem 1 (but is more involved). Note that in the proof of Theorem 1 we first showed that the total number \( \phi \) of occurrences of far elements in \( \text{CYC}(\sigma) \) satisfies
\[
\phi = \sum m_i = k^2 + k.
\]
and then we showed that there must be at least one permutation in $\text{CYC}(\sigma)$ which contains at least two such occurrences. Since there are $n$ permutations in $\text{CYC}(\sigma)$, this implied that

$$k^2 + k = \sum m_i \geq n + 1,$$

which gave the upper bound for $F(n)$.

Here it can be shown that, if $n \geq 2k$, then

$$2(k^2 + k) = \sum m'_i \geq 2n + 2k + 4 \quad (**)$$

where $m'_i$ is the number of permutations in $\text{CYC}(\sigma) \cup \text{CYC}(\sigma^R)$ in which $i$ is far.

This implies the upper bound for $G(n)$. To show (**) we first observe that

$$m'_i = 2m_i,$$

and that for $n > 2$

$$|\text{CYC}(\sigma) \cup \text{CYC}(\sigma^R)| = 2n;$$

and then we prove that there are at least $2k+4$ permutations in $\text{CYC}(\sigma) \cup \text{CYC}(\sigma^R)$ which contain at least two far elements each.

As for the lower bound, we give here a program that generates permutations $\sigma = \alpha_1, \ldots, \alpha_n$, where $n = k^2 - k - 4$, such that $G(\sigma) = n - k$, for $n \geq 8$.

```
begin
  [distribution of large elements n-k+2 - n]
  $i := 1$;
  $\alpha_i := n$;
  for $j := 2$ to $k-1$ do
    begin
      $i := i + k + 1 - j$;
      $\alpha_i := n - j + 1$;
    end;

  [distribution of small elements 1 - k-1]
  $i := i$;
  $\alpha_i := 1$;
  for $j := 2$ to $k-1$ do
    begin
      $i :=(i + k + 1 - j)mod n + 1$;
      $\alpha_i := j$;
    end;

  [distribute the remaining elements arbitrarily]
end.
```

The construction for the cases $k = 4, 5$ is shown in Figure 11.
In the next Section we examine the implications of the results obtained on the path lengths and area requirements for mapping a data flow graph on a square or hexagonal processor array. These results apply to the special cases defined in Section 2.4.

5. CONCLUSIONS

From the results in the last section we obtain bounds on the longest path which a value has to travel from row \( r_i \) to row \( r_{i+1} \). In the square array, this length is at least \( 2 \cdot \tau P(\sigma) \), or \( 2 \cdot G(\sigma) \) for the commutative case (two rows are required to exchange a pair of values, thus the factor of 2), where \( \sigma \) is the permutation (refer to Section 3.1). This implies a lower bound of \( 2(n - \sqrt{n}) \) in the worst-case (where \( n \) is the number of values to be routed) on the path length, and thus on the number of rows required between \( r_i \) and \( r_{i+1} \).

It is well known that permutation networks [3] achieve any reordering of \( n \) values using \( n \) stages. Therefore, we see that in the worst case we save very little (in terms of rows) if we allow cyclic shifts of the target permutation. Moreover, if we take into account the presence of loops in the data flow graph, this restricts the usage of cyclic shifts. The reason for this comes from the requirement for
those elements acting as loop extrema to be at one end of the permutation (see [1]).

In the hexagonal array case, when we take into consideration the horizontal links, our results imply a lower bound of $n - \sqrt{n}$ on the longest path. Again, implementing a permutation network in this case can be done in $n$ rows (since interchanging two elements may be done in a single row). However, this implementation may result in longer paths.

Some interesting problems remain open:

- Finding an efficient routing algorithm for the hexagonal case (the permutation network mentioned above does not take full advantage of the horizontal links).
- Extending the results herein to a general permutation, i.e., including more than one cycle.
- Examining the behavior of a random permutation, i.e., finding the average complexity of the problem at hand.

REFERENCES


