Math. Systems Theory 21, 85-98 (1988)

Mathematical Systems Theory



# Minimum-Diameter Cyclic Arrangements in Mapping Data-Flow Graphs onto VLSI Arrays

P. Erdös,[1] I. Koren,[2] S. Moran,[3] G. M. Silberman,[3] and S. Zaks[3]

[1] Mathematical Institute of the Hungarian Academy of Sciences, Budapest, Hungary

[2] Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, MA 01003, USA

[3] Department of Computer Science, Technion—Israel Institute of Technology, Haifa 32000, Israel

**Abstract.** Regular arrays of processing elements in VLSI have proved to be suitable for high-speed execution of many matrix operations. To execute an arbitrary computational algorithm on such processing arrays, it has been suggested mapping the given algorithm directly onto a regular array. The computational algorithm is represented by a data-flow graph whose nodes are to be mapped onto processors in the VLSI array.

This study examines the complexity of mapping data-flow graphs onto square and hexagonal arrays of processors. We specifically consider the problem of routing data from processors in a given (source) sequence to another (target) sequence.

We show that under certain conditions, the above problem is equivalent to the one of finding a minimum-diameter cyclic arrangement. The complexity of the latter problem is analyzed and upper and lower bounds on the number of intermediate rows of processors (between the source and target rows) are derived.

## 1. Introduction

Recent developments in VLSI technology have made it possible to build relatively large processing arrays in a single chip. Schemes for mapping certain classes of computational algorithms onto these arrays have been recently proposed [1]-[3], enabling the achievement of high performance through parallelism and pipelining. Most of these computational algorithms have inherent regularity (like vector and

matrix operations, and problems in signal and image processing), a property that simplifies the task of mapping.

There are, however, many computationally demanding problems which do not enjoy high regularity and, therefore, the task of mapping them onto processing arrays is substantially more complex. It has been shown in [4] that the mapping of algorithms onto processing arrays is equivalent, in its most general form, to the graph isomorphism problem (which is conjectured to be *NP*-complete). Consequently, we have to concentrate on efficient heuristics that find good mappings for most nonregular computational algorithms. A scheme which allows execution of arbitrary algorithms, and still takes advantage of their inherent parallelism, has been presented in [5]. This scheme represents an algorithm by a *data-flow graph* [6], and then maps it onto the processing array. A hexagonal array of *data-driven* processors, capable of hosting the result of this mapping has been shown in [7], whereas [8] presents the actual mapping onto the array of algorithms written in the data-flow language VAL [9].

This paper examines the complexity of mapping data-flow graphs onto square and hexagonally connected processing arrays. Specifically, we address those problems which arise from the nonplanarity exhibited by these graphs as a consequence of operand ordering. For square arrays, nonplanarity caused by iterative constructs, e.g., *do-loops*, cannot be addressed within the regular interconnections present among the processors, and some extra connections (e.g., buses) must be provided. In the case of a hexagonal array, these constructs can be handled *separately*, using the connections already present in the array, as shown in [8]. Therefore, in the following we restrict our discussion to acyclic data-flow graphs.

In the above context, we study the routing of $n$ output values (tokens) from a given row in the array, to be used as inputs to another such row. The permissible permutations of the $n$ values in the latter row are determined by the constraints imposed by the given mapping process, as we shall see below. This problem can also be considered as a special case of channel routing in VLSI circuits [10], [11]. In its general setting, this problem studies the situation where a routing, which connects $n$ pairs of terminals, one from each of two rows positioned opposite each other, must be performed under some constraints and optimizing some criteria.

The effect of rotations on the channel routing problem is studied in [12], where algorithms are shown which determine the rotations for minimizing the cost measures of *density*, *crossing number*, and *length of wires*. (In [13], the problem of minimizing density is solved using a heuristic approach, but allowing for the reordering of terminals within each row.) Viewing our problem in these terms, we are concerned with minimizing the length of the longest net, since this is directly related to the number of intermediate rows, which carry out the routing of the given $n$ values. The exact bounds for this measure are more difficult to estimate than those for the measures in [12] and [13]. It is shown that this problem is equivalent to finding cyclic arrangements with minimum diameter in a sense to be defined. It is further shown that under the given constraints, the mapping process requires at most $O(n)$ (more precisely, $n-\sqrt{n}+O(1)$) intermediate rows,

in order to achieve the necessary routing. This $O(n)$ bound can be easily obtained by a straightforward array implementation of any standard permutation network [14]. In view of our results, it follows that usage of such networks cannot be considerably improved upon, *in the worst case.* (This bound is seldom reached, as seen in [8].)

In the following section we introduce our approach to the solution of the mapping problem, following [5]. Section 3 presents the mathematical formulation of the problem and introduces the notation used throughout. Section 4 presents the main result on the complexity of the mapping process. These results are then used in Section 5 to determine bounds on the length of paths in the array configuration 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 acyclic data-flow graphs onto an array of processors. (Cycles caused by iterative constructs are handled separately.) 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 [6].

In data-flow graphs (see, for example, Fig. 1), each node corresponds to an operator (e.g., plus, minus, boolean *and*, etc.), and operands move as "tokens"

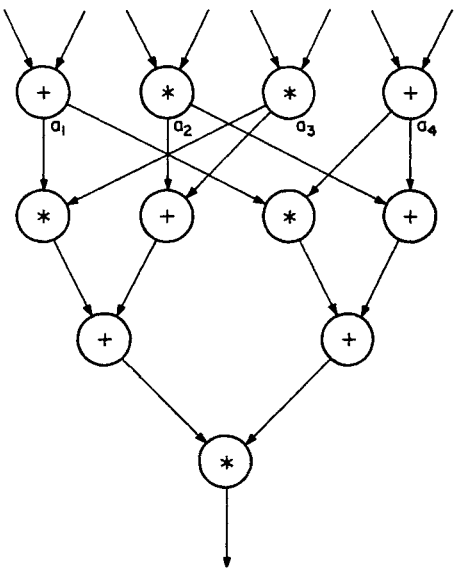


**Fig. 1.** A data-flow graph.

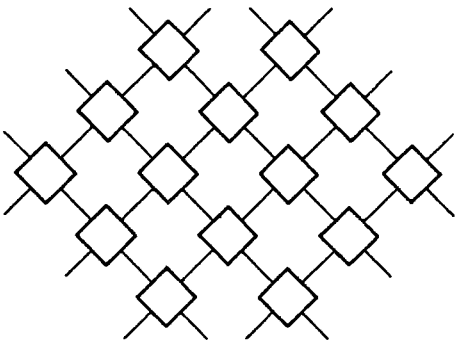

Fig. 2. A square array of processors.

along the directed arcs connecting these nodes. On the other hand, the processors described in [7] are capable of executing several such basic operations, and thus actual mapping of the data-flow graph onto the array may be preceded by a *compression* step. This step serves to tailor the graph to be mapped to the processors' capabilities (for details refer to [8]).

Notice (in Fig. 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 [6].

### 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. This suggests the square array of processors shown in Fig. 2. A somewhat more sophisticated setup is a hexagonal array of processors, such as the one used in [5] and shown in Fig. 3.

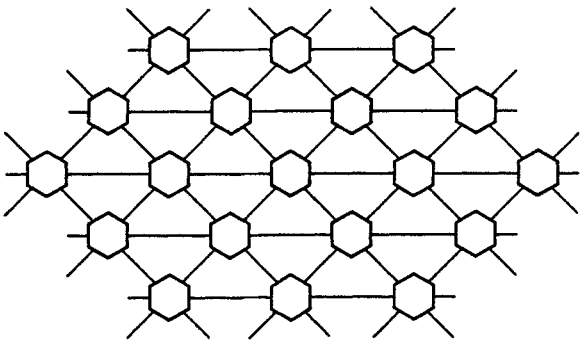

Fig. 3. A hexagonal array of processors.

### 2.3. *Mapping Data-Flow Graphs onto Arrays*

*2.3.1. A General Description.* The problems we study deal with the process of mapping a given acyclic 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. We consider here an array of unbounded size, whereas in reality the number of processors which fit in a single chip is limited by technology. To bridge this gap, the design proposed in [7] provides the means for building a large array from a number of chips, each containing a subarray of between 50 and 100 processors.

For example, the mapping of the data-flow graph in Fig. 1 onto a square array is shown in Fig. 4. We notice from the above example that the mapping process is complicated by the nonplanarity 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 [5]. 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

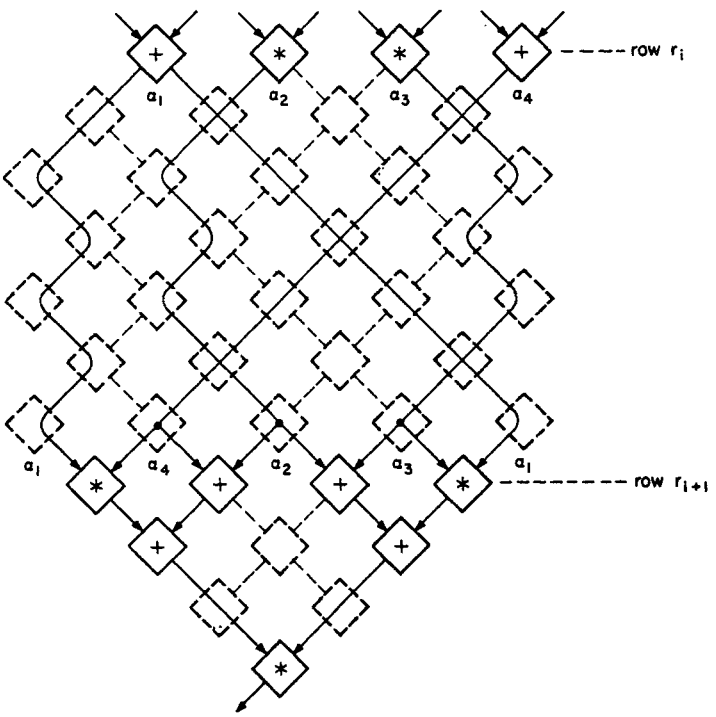


**Fig. 4.** Mapping a data-flow graph onto a square array.

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 form higher-numbered levels.

In the mapping process all the operators at level $i$ will be mapped into 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 $r_i$ (if one of these processors is at row $r_{i-1}$ or lower, its output is first routed to row $r_i$). When two such processors (at row $r_i$) are not adjacent, our approach—following [5]—is to route their values (outputs) to a pair of adjacent processors at row $r_{i+1}-1$.

Let $P_1, \ldots, P_n$ be the sequence of processors at row $r_i$, and let $Q_1, \ldots, Q_m$ be the sequence of processors at row $r_{i+1}-1$ which supply the inputs to row $r_{i+1}$ (Fig. 5). Then it follows from the above that if $P_i$ and $P_j$ are to supply the inputs to a certain processor at row $r_{i+1}$, there must be a pair of adjacent processors, $Q_k$ and $Q_{k+1}$ which receive their output values (see Fig. 5). Routing of values from $P_1, \ldots, P_n$ so as to achieve an ordering $Q_1, \ldots, Q_m$ 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 intermediate rows (between $r_i$ and $r_{i+1}-1$), as a function of the number of values to be routed, in a special case.

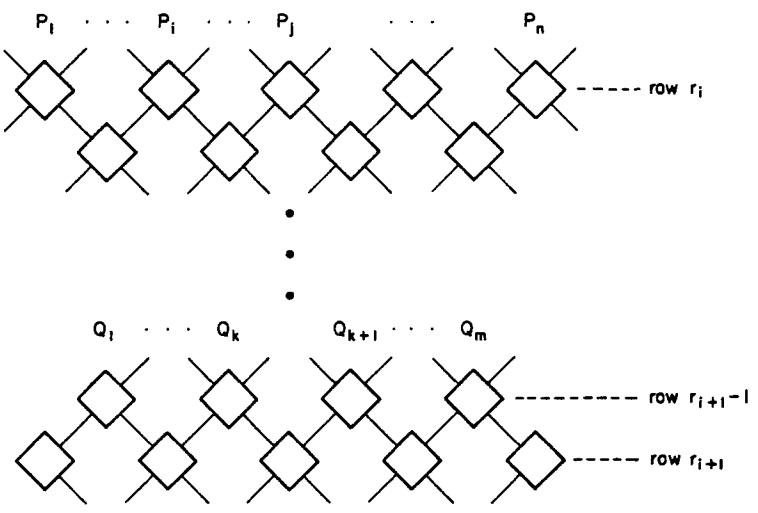


**Fig. 5.** Routing in the second phase.

In the general case, there is no expression relating $m$ and $n$, the number of processors in the sequence $Q_1, Q_2, \ldots, Q_m$ and $P_1, \ldots, P_n$, respectively. Moreover, the output of $P_i$ may have to be routed to an arbitrary number of processors in the sequence $Q_1, Q_2, \ldots, Q_m$. Consequently, for the general case, only trivial upper and lower bounds on the required number of intermediate rows can be derived.

We study below the mathematically tractable 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 $P_j$ $(1 \leq j \leq n)$ in row $r_i$ is the input operand to exactly two adjacent processors $P'_k$ and $P'_{k+1}$ $(1 \leq k < n)$ in row $r_{i+1}$ (see Fig. 4).

Even though assumption (b) simplifies the problem, we consider it to be of practical interest. First, we feel that routing a single value, which feeds two adjacent processors, has a potential for relieving routing bottlenecks, particularly in those graphs which exhibit complicated loop structures. Furthermore, if we look at the dynamic routing of operands from a VLSI technology point of view, it is always advantageous to have less transistors switching at the same time. This reduces the circuit's power consumption and heat dissipation, two critical factors in the design of VLSI circuits.

Thus, in this case there are $n+1$ processors on row $r_{i+1}-1$ (i.e., $m = n+1$), and there is a permutation $\pi$ of $\{1, 2, \ldots, n\}$ such that the output from $P_{\pi(1)}$ is routed to both $Q_1$ and $Q_{n+1}$, and the output from each $P_{\pi(j)}$, $1 < j \leq n$, is routed to $Q_j$.

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

$$\begin{array}{ccccccc}
P_{\pi(2)} & P_{\pi(3)} & \cdots & P_{\pi(n)} & P_{\pi(1)} & P_{\pi(2)} \\
P_{\pi(3)} & P_{\pi(4)} & \cdots & P_{\pi(1)} & P_{\pi(2)} & P_{\pi(3)} \\
 & & \vdots & & & \\
P_{\pi(n)} & P_{\pi(1)} & \cdots & P_{\pi(n-2)} & P_{\pi(n-1)} & P_{\pi(n)}
\end{array}$$

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 $r_i$, can determine the value $r_{i+1}$, so as to minimize the number of intermediate rows.

## 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 the *reverse* permutation of $\sigma$, $\sigma^R = (a_n, \ldots, a_1)$, and the *diameter* of $\sigma$, $D(\sigma) = \max\{|a_i - i| \mid i = 1, 2, \ldots, n\}$.

We also define

$$\mathrm{CYC}(\sigma) = \{\pi | \pi \text{ is a cyclic permutation of } \sigma\},$$

$$F(\sigma) = \min\{D(\pi) | \pi \in \mathrm{CYC}(\sigma)\},$$

$$G(\sigma) = \min\{D(\pi) | \pi \in \mathrm{CYC}(\sigma) \cup \mathrm{CYC}(\sigma^{\mathrm{R}})\},$$

$$F(n) = \max\{F(\sigma) | \sigma \in S_n\},$$

$$G(n) = \max\{G(\sigma) | \sigma \in S_n\},$$

where $S_n$ denotes the set of all the $n!$ permutations of the elements $1, 2, \ldots, n$.

It follows immediately that

$$(\forall \sigma \in S_n) \qquad [F(\sigma) \geq G(\sigma)]$$

and therefore

$$(\forall n) \qquad [F(n) \geq G(n)].$$

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

**Lemma 1.** *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 brevity we write $\sigma = 1\,2\,5\,3\,4$. Then $\sigma^{\mathrm{R}} = 4\,3\,5\,2\,1$ 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 Fig. 6.

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

The sets $\mathrm{CYC}(\sigma)$ and $\mathrm{CYC}(\sigma^{\mathrm{R}})$ are shown in Fig. 7. For each permutation $\pi$ we show its diameter $D(\pi)$ where all the elements $i$ satisfying $|a_i - i| = D(\pi)$ are shown in boldface. 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, we can find at least one

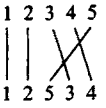


**Fig. 6.** The function $D(\sigma)$.

| | $\pi$ | $D(\pi)$ |
|---|---|---|
| CYC($\sigma$) | 1 2 **5** 3 4 | 2 |
| | **4** 1 2 5 3 | 3 |
| | **3 4 1 2** 5 | 2 |
| | **5** 3 4 1 2 | 4 |
| | 2 5 3 4 **1** | 4 |
| CYC($\sigma^R$) | 4 3 5 **1** 2 | 4 |
| | 1 4 3 5 **2** | 3 |
| | **2 1 4 3** 5 | 1 |
| | **5** 2 1 4 3 | 4 |
| | 3 **5** 2 **1** 4 | 3 |

**Fig. 7.** The sets CYC($\sigma$) and CYC($\sigma^R$).

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. In addition, $G(\sigma) = 1$, 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 Fig. 8.

In fact, it will follow from this work and from [15] that, for all $n$,

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

and we conjecture that

$$F(n) > G(n) \qquad \text{for} \quad n > 3.$$

### 3.3. *Implications to the Mapping Problem*

We consider again the mapping process, as described in Section 2. The processors $P_1, P_2, \ldots, P_n$ on row $r_i$ correspond to the numbers $1, 2, \ldots, n$, respectively. The processors $Q_1, Q_2, \ldots, Q_n$ on row $r_{i+1} - 1$ correspond to the permutation $\sigma = a_1 a_2 \cdots a_n$. (Notice that the implied arrangements is $\sigma = a_1 a_2 \cdots a_n a_1$, since $Q_1$ and $Q_{n+1}$ provide only one input each to row $r_{i+1}$.) If we concentrate on those

| $n$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| $F(n)$ | 0 | 0 | 1 | 2 | 3 | 3 | 4 | 5 | 6 | 7 |
| $G(n)$ | 0 | 0 | 1 | 1 | 2 | 2 | 3 | 4 | 4 | 5 |

**Fig. 8.** $F(n)$ and $G(n)$ for $n = 1, 2, \ldots, 10$.

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 $\mathrm{CYC}(\sigma)$ 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. (Actually, the design proposed in [7] makes every operation look as if it was commutative, since it allows the specification of which input register will receive which operand, through the "programming" of each processor in the array.) In our terms, each of the permutations in $\mathrm{CYC}(\sigma^R)$ will also satisfy the desired requirements. Thus, the functions $G(\sigma)$ and $G(n)$ are of interest too, in this case.

Before proceeding to prove the main result of this paper, we note that if we are given a permutation $\sigma$, routing can be done with $2 \times D(\sigma)$ intermediate rows. The idea is that in two rows we can attain a permutation $\sigma'$, such that $D(\sigma') = D(\sigma) - 1$ (the details of the algorithm are left to the reader).

## 4. Main Result

Our main result is that $F(n)$ is $n - \sqrt{n} + O(1)$. More specifically, we have the following theorem.

**Theorem.** *For all* $n \geq 1$, $F(n) = n - \alpha(n)$, *where* $\alpha(n) = \min\{k | k^2 + k - 1 \geq n\}$.

The study of the function $G(n)$ appears to be much more involved. The techniques originally developed here were later generalized in [15] to derive less tight bounds for $G(n)$. More specifically, it is shown in [15] that, for all $n \geq 8$, $n - \beta(n) \leq G(n) \leq n - \gamma(n)$, where $\beta(n) = \min\{k | k^2 - k - 4 \geq n\}$ and $\gamma(n) = \min\{k | k^2 + k/2 \geq n\}$.

*Proof of Theorem.* 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 $a_1, a_2, \ldots, a_n$, defined as follows: $a_1 = 1$, $a_{k+1} = 2$, $a_{2k+1} = 3, \ldots, a_{ik+1} = i+1, \ldots, a_{(k-1)k+1} = k$, and $a_j$ is arbitrary otherwise. In Fig. 9 we show the permutation $\sigma$ for $k = 3, 4$. (A * means that any

| $k=3$ | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| $a_i$ | 1 | * | * | 2 | * | * | 3 | * | * |

| $k=4$ | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| $a_i$ | 1 | * | * | * | 2 | * | * | * | 3 | * | * | * | 4 | * | * | * |

**Fig. 9.** The permutation $\sigma$ for $k = 3, 4$.

"unused" 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 $i$, $i \le k$, and this means that $G(\sigma) \ge n - 2\sqrt{n}$, as desired.

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

*Upper Bound.* Let $n$ be given. We show that $F(n) \le n - \alpha(n)$, where $\alpha(n) = \min\{k | k^2 + k - 1 \ge n\}$. To show this, we prove that $k^2 + k \le n$ 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 show that for each $\sigma \in S_n$ the number of permutations $\pi$ in CYC($\sigma$) for which $D(\pi) \ge n - k$ is at most $k^2 + k$, and hence less than $n$; this implies that for each permutation $\sigma$ there exists 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 original location, and we say that $i$ is *far* in these permutations. Clearly, there are exactly $k$ permutations in CYC($\sigma$) for which 1 is far, namely, the permutations for which 1 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 we can easily verify that

$$m_2 = m_{n-1} = k - 1$$

$$\vdots$$

$$m_k = m_{n-k+1} = 1,$$

and

$$m_i = 0 \qquad \text{for} \quad k < i \le n - k.$$

Hence, $\sum m_i = 2(k + (k-1) + \cdots + 1) = k^2 + k$. 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 overcounted 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 $F(\sigma) \le n - k$, with equality being satisfied 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

$$\pi_1 = \{a_i, a_{i+1}, \ldots, a_n, a_1, \ldots, a_{i-1}\},$$

$$\pi_2 = \{a_{i+1}, a_{i+2}, \ldots, a_n, a_1, \ldots, a_i\}$$

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

Since none of the elements $a_{i+1}, \ldots, a_{i+k}$ 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 $a_i$, which means that $a_i \geq n-k+1$. Similarly, the unique far element in $\pi_2$ is the last element, which is also $a_i$, meaning that $a_i \leq k$. 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 $F(\sigma) \geq n-\alpha(n)$. The following program generates permutations $\sigma = a_1, \ldots, a_n$, where $n = k^2+k-1$, such that $F(\sigma) = n-k$ (the proof is left to the reader).

```
begin
  [distribution of large elements n-k+1 to 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 to 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.
```

The construction for the cases $k = 3, 4$ is shown in Fig. 10.

From Lemma 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 $F(n) = n-k-1$. Furthermore, since, for any such $n$, $\alpha(n) = k+1$, we have, for all $n$, $F(n) = n-\alpha(n)$. □

| $k=3$ | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| $a_i$ | 9 | * | 10 | * | 1 | 11 | * | 2 | * | 3 | * |

| $k=4$ | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| $a_i$ | 16 | * | 17 | * | * | 18 | * | * | 1 | 19 | * | * | 2 | * | * | 3 | * | 4 | * |

**Fig. 10.** Lower bound for $F(n)$.

## 5. Conclusions

In this section we examine the implications of our analysis on the path lengths and number of intermediate rows required for mapping a data-flow graph on a square or hexagonal processor array. These results apply only to the special case defined in Section 2.4, and do not represent a solution to the general data-flow mapping problem.

From the results in the last section we derive bounds on the longest path which an operand (value) has to travel, from row $r_i$ to row $r_{i+1}$. In the square array, this length is at least $2 \times F(\sigma)$, or $2 \times G(\sigma)$ for the commutative case (two rows are required to exchange a pair of values, thus the factor of 2), where $\sigma$ is the permutation of operands to be used as inputs by row $r_{i+1}$ (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 $r_{i+1}$.

On the other hand, it is well known that permutation networks [14] 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. However, in special cases which might be encountered in practice, the additional computational effort incurred by allowing such cyclic shifts can be justified.

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. In practice, [8] shows that it is worthwhile developing heuristics to perform the routing of values in the array, as the number of rows dedicated to this task can be enormously reduced over the worst case.

Some interesting problems remain open:

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

[1] Fisher, A. L. *et al.*, Design of the PSC: A Programmable Systolic Chip, *Proc. Third Caltech Conf. on VLSI*, March 1983, pp. 287-302.

[2] Kung, S. Y., On Supercomputing with Systolic/Wavefront Array Processors, *Proc. IEEE*, Vol. 72, July 1984, pp. 867-884.

[3] Li, G., and Wah, B. W., The Design of Optimal Systolic Arrays, *IEEE Trans. Comput.*, Vol. C-34, January 1985, pp. 66-77.

[4] Bokhari, S. H., On the Mapping Problem, *IEEE Trans. Comput.*, Vol. C-30, March 1981, pp. 207-214.

[5] Koren, I., and Silberman, G. M., A Direct Mapping of Algorithms onto VLSI Processing Arrays Based on the Data Flow Approach, *Proc. 12th International Conf. on Parallel Processing*, August 1983, pp. 335-337.

[6] Special issue on Data Flow Systems, *IEEE Comput.*, Vol. 15, No. 2, February 1982.

[7] Koren, I., and Peled, I., The Concept and Implementation of Data-Driven Processor Arrays, *IEEE Comput.*, Vol. 20, No. 7, July 1987, pp. 102-103.

[8] Mendelson, B., and Silberman, G. M., Mapping Data Flow Programs on a VLSI Array of Processors, *Proc. 14th International Symp. on Computer Architecture*, Pittsburgh, PA, June 1987, pp. 72-80.

[9] Ackerman, W. B., and Dennis, J. B., VAL—A Value-Oriented Algorithmic Language; Preliminary Reference Manual, Technical Report MIT/LCS/TR-218, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA, 1974.

[10] Mizraian, A., Channel Routing in VLSI, *Proc. 16th Annual ACM Symp. on Theory of Computing*, 1984, pp. 101-107.

[11] Preparata, F. P., and Lipski, W., Optimal Three-Layer Channel Routing, *IEEE Trans. Comput.*, Vol. C-33, 1984, pp. 427-437.

[12] Atallah, M. J., and Hambrusch, S. E., Optimal Rotation Problems in Channel Routing, *IEEE Trans. Comput.*, Vol. C-35, September 1986, pp. 843-847.

[13] Savage, J. E., Heuristics in the SLAP Layout System, *Proc. International Conf. on Computer Design (ICCD)*, Port Chester, New York, October 1983, pp. 637-640.

[14] Knuth, D. E., *The Art of Computer Programming*, Vol. 3, Addison-Wesley, Reading, MA, 1973, Section 5.3.4.

[15] Erdös, P., Linial, N., and Moran, S., Extremal Problems on Permutations under Cyclic Equivalence, *Discrete Math.*, Vol. 64, No. 1, 1987, pp. 1-13.

*Received September 28, 1987, and in revised form February, 1988, and March 31, 1988, and in final form May 27, 1988.*