DATA BROADCASTING IN LINEARLY SCHEDULED ARRAY PROCESSORS*

J. A. B. Fortes** and D. I. Moldovan
Dept. of Electrical Engineering-Systems
University of Southern California
Los Angeles, CA 90089

ABSTRACT

A major problem in executing algorithms in array processors is the implementation of broadcasts without unnecessary speed-up factor degradation. We discuss when and how broadcasts can be eliminated or reduced to easily implementable sequences of reduced local broadcasts. Algorithms are modelled as a structured set of indexed computations which operate on variables associated with a referencing or indexing function. The discussion is restricted to variables with linear indexing functions and to algorithms linearly scheduled for execution in array processors. Linear indexing functions are represented as affine matricial functions of the index set of the algorithm. The linear part of such representation is a coefficient matrix denoted the indexing matrix. Linear schedules are defined as linear time-space allocation functions mapping the computations of an algorithm into time and processors. We discuss necessary and sufficient conditions for the occurrence of broadcasts in a linearly scheduled algorithm. Necessary and sufficient conditions and constructive criteria are given for selecting linear schedules for which all broadcasts are eliminated or reduced to sequences of small local broadcasts.

1. Introduction

Several parallel computers have been designed and built around the concept of array architecture. Examples include the early ILLIAC IV computer [1], Massively Parallel Processor (MPP) [2], systolic arrays [9], the Wavefront Array Processor (WAP [10], the Configurable Highly Parallel (CHiP) computer [17], and many others. In array architectures every processor is directly connected to a small subset of the processors and the interconnection pattern is often the same for all processors. This restrictive interconnection structure may not be able to support fast communication of data from one to many processors. Such type of data communication is called a broadcast and is necessary when several processors need the same data in order to proceed with their scheduled computations. If broadcasts cannot be made, computations must be rescheduled, thus slowed down if correct results are to be obtained. For this and other reasons (e.g., parallelism exploitation, data routing, etc.), scheduling and allocation of computations to array processors is a difficult problem. Linear schedules (to be defined soon) can be and have been used successfully for many algorithms executed on array processors. Examples include the solution of PDE's on the ILLIAC IV [11], most algorithms executed in VLSI arrays, on the WAP, on the CHiP [3,9,10,13,14,17], and others (see, for example, [8]).

The problem of broadcasting variables in array processors has received considerable attention in the past. In some cases (most SIMD arrays) a central control unit has the capability to broadcast a single variable to all processors in the array (see, for example, [6]). However, the control unit becomes a serious bottleneck if many broadcasts have to be implemented at a given step of the algorithm. Skewed storage schemes and associative memories [6] have been proposed to reduce the need for broadcasts. In general, these schemes cannot eliminate the need for all broadcasts and are dependent on the algorithm being executed. Memory redundancy can also be used to avoid broadcasts. Examples include renaming and expansion of scalar variables used in optimizing compilers [6]. In [16] the interconnection network of the array processor is used to simulate a multi-stage network in which any broadcast can be implemented. The main disadvantage of this approach is that such simulation may be highly time consuming in algorithms requiring a moderate number of broadcasts. A more appealing solution has been proposed in [8]. In this reference, some algorithms are scheduled so that broadcasts overlap by pipelining variables through the array interconnection network. However, no general procedure was given for arbitrary algorithms. Our approach is based on the same idea. However, the techniques for broadcast presented in this paper can be applied to a large class of algorithms. Also, designers of VLSI architectures have long recognized the need for algorithm rescheduling in order to avoid broadcasts [12]. Any systematic procedure for designing VLSI algorithms must be able to obtain broadcast free or broadcast reduced schedules. Our results apply directly to the VLSI design techniques proposed in [3,4,13-15].

This paper discusses how and whether data broadcasts in an array processor with a given interconnection structure can either be eliminated or reduced by choosing an adequate linear schedule. In Section II we describe the models used to represent algorithms and array processors throughout the paper. Next, we define linear schedule as a time-space allocation of computations described by a linear function. Often, the need for broadcasts can be detected from the manner in which variables are referenced in the algorithm. Also, in Section II, we describe the variable referencing mechanism considered in this paper. Section III starts by stating sufficient and necessary conditions for the occurrence of broadcasts in the execution of an algorithm. Theorem 3.1 gives necessary and sufficient conditions for the existence of time schedules that avoid broadcasts. Theorem 3.2 gives the conditions that a space schedule

* This research was supported by NSF Grant ECS-8307258 and JSEP Contract No. F49620-81-C0070.

** J. A. B. Fortes is presently with the Department of Electrical Engineering, Purdue University

0194-7111/84/0000/0224$01.00©1984 IEEE

must satisfy to support the broadcast elimination provided by a given time schedule. Section IV considers the problem of implementing broadcasts in architectures with limited broadcasting capabilities. Theorem 4.1 gives sufficient conditions for the existence of time schedules that allow the implementation of a broadcast as a sequence of reduced broadcasts. Theorem 4.2 gives the conditions that a space schedule must satisfy in order to support a sequence of small broadcasts. Section V contains some conclusions and points out possible extensions to this research.

II. MODELLING ALGORITHMS, ARRAY PROCESSORS, LINEAR SCHEDULES, AND VARIABLE REFERENCES

We see an algorithm as a structured set of computations which operate on input variables and/or variables generated by other operations and produce a set of output variables. In order to represent an algorithm, we use a simplified version of the algorithm model introduced in [3]. The simplified concepts introduced next are good enough for the purposes of this paper and no previous knowledge of [3] is assumed.

Definition 2.1

An *algorithm* is a 5-tuple $A = (J^n, C, D, X, Y)$ where

J^n is the index set of A, $J^n \subset Z^n$ *
C is the set of computations (indexed by J^n)
D is the set of dependencies
X is the set of input variables
Y is the set of output variables

In this definition, by dependency we mean a vector difference between the index of a computation where a variable is used and the index of the computation where that variable is generated. The set of output variables is a subset of the union of the set of generated variables and the set of input variables. To every variable it is associated an indexing function, i.e., a function $\bar{F}: J^n \rightarrow Z^m$ such that if the variable has index $\bar{F}(\bar{j})$ then this variable is involved in the computation indexed by $\bar{j}$. In this paper we assume that all variables have linear indexing functions, i.e., functions that satisfy the following definition.

Definition 2.2

A variable indexing function $\bar{F}: J^n \rightarrow Z^m$ is a linear indexing function if and only if

$$\bar{F}(\bar{j}) = \bar{C}_0 + C\bar{j} \qquad (2.1)$$

where
$\bar{C}_0 \in Z^{(m\times 1)}$ is the index displacement
$C \in Z^{(m\times n)}$ is the indexing matrix.

Example 2.1

The variable $a(j_1 - j_2, j_2 + j_3 - 1, j_3 - j_1)$ has a linear indexing function where

$$C = \begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & 1 \\ -1 & 0 & 1 \end{bmatrix} \qquad \bar{C}_0 = \begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix}$$

End of example.

* The symbols Z, I, I^+ denote the sets of integers, nonnegative integers and positive integers, respectively. Given a set S, S^n denotes the nth cartesian power of S.

The next definition introduces a simple model for array processors.

Definition 2.3

An *array processor* is a tuple (L^m, P) where

(i) L^m is the index set of the array processor, $L^m \subset Z^m$
(ii) P is the matrix of interconnection primitives $P = [\bar{p}_1, \ldots, \bar{p}_r] \in Z^{m\times r}$, where $r \in I$ is the number of interconnection primitives

In this definition of array processor, every point $\bar{\ell}$ in L^m represents an element of the array. We assume that all processors are identical and that the interconnections are regular and uniform. The matrix of interconnection primitives P is such that, if $\bar{p} \in P$ then $\bar{\ell} \in L^m$ is connected to $\bar{\ell}' = \bar{\ell} + \bar{p}$ if $\bar{\ell}' \in L^m$ and $\bar{\ell}$ is connected to an input/output port if $\bar{\ell}' \notin L^m$.

Example 2.2

Consider the N×N array configuration as shown in Figure 2.1 (for N=4), which is used in WAP [10] and other array processors. The structure of these arrays can be described by (L^2, P) where

$$L^2 = \{(\ell_1, \ell_2): 0 \le \ell_1, \ell_2 \le N-1\}$$

and

$$P = \begin{bmatrix} 1 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 \end{bmatrix}$$

End of Example.

In this paper, a schedule is a time-space allocation function $T: J_A^n \rightarrow J_B^n$ where J_A^n is the n-dimensional index set of the original algorithm A and J_B^n is the index set of a transformed algorithm B that is input-output equivalent to A. In this paper we assume that the original algorithm is such that $J_A^n \subset I^n$. For more details on equivalence preserving algorithm transformations the reader is referred to [3,4]. In J_B^n, n−r coordinates are associated with time and r coordinates are associated with space. In this paper we assume that the first coordinate j_B^1 is associated with time (i.e., r=n−1) and that

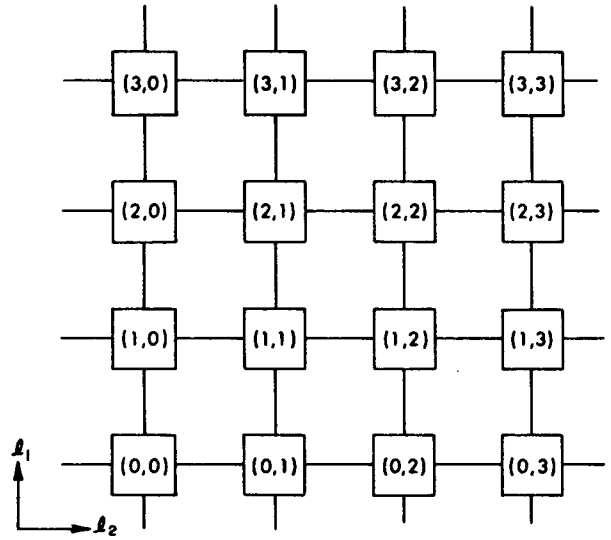

Figure 2.1. Interconnection structure of the array processor of example 2.2 (N=4)

the remaining coordinates map via the identity function into the index set of the array processor. If the array processor is small then the algorithm can be partitioned and the identity mapping can be used for every partition [15]. Then a schedule consists of a time schedule $\pi : J_A^n \rightarrow J_B^1$ and a space schedule $S : J_A^n \rightarrow J_B^{n-1}$. The dimensionality of the index set of the array processor equals the dimensionality of index set of the algorithm minus one.

Definition 2.4

A *linear schedule* $T : J_A^n \rightarrow J_B^n$ is such that T is a linear bijection function, i.e.,

$$T = \begin{bmatrix} \pi \\ S \end{bmatrix} \qquad \pi \in Z^{(1\times n)}, \quad S \in Z^{((n-1)\times n)}$$

and T is nonsingular.

The above definition could be generalized to include schedules described by unions of integer affine functions; however, in this paper we restrict our discussion only to linear schedules. If a linear schedule is used for executing an algorithm in an array processor then we say that the array is *linearly scheduled.*

III. ON THE ELIMINATION OF BROADCASTS

During the execution of an algorithm a variable needs to be broadcasted if and only if the following conditions are both satisfied:

(1) at least two computations use the variable and
(2) such computations are scheduled for execution at the same instant of time

Notice that the first condition is independent of the ordering of computations while the second depends on the time schedule. We start by discussing how the first condition can be checked. Clearly, a variable with indexing function $\bar{F}$ is used by computations indexed by $\bar{j}'$ and $\bar{j}''$ if and only if

$$\bar{F}(\bar{j}') = \bar{F}(\bar{j}'') \tag{3.1}$$

i.e.

$$\bar{F}(\bar{\Gamma}) = 0 \tag{3.2}$$

where $\bar{\Gamma} = \bar{j}' - \bar{j}''$. From (2.1), condition (3.2) can be rewritten as

$$C\bar{\Gamma} = 0 \tag{3.3}$$

From elementary facts of linear algebra we can conclude the following:

Lemma 3.1

A variable with indexing matrix C is used in more than one computation if and only if rank(C) < n, where n is the dimensionality of the algorithm index set.

In general, we can rewrite C (assuming rank(C)=k, $1 \leq k \leq n$) as

$$C' = \begin{bmatrix} C_1 \\ C_2 \end{bmatrix} = R_r\, C$$

where

R_r – is a row rearranging matrix, i.e., an (m×m) matrix such that its (i,j)th entry is a "1" if row j in C becomes row i in C′ and is a "0" otherwise.

C_1 – is a (k×n) matrix such that its columns can always be rearranged to obtain a matrix

$$C_1' = [B \;\; N] = C_1 R_c$$

where

B – a nonsingular (k×k) matrix
N – a (k×(n−k)) matrix (for n−k > 0)
R_c is a column rearranging matrix, i.e., an (n×n) matrix such that its (i,j)th entry is a "1" if column i in C_1 becomes column j in C_1' and is a "0" otherwise.

C_2 – is a ((m−k)×n) matrix (for m−k > 0)

Hence, we have

$$C = R_r^T \begin{bmatrix} [B \;\; N] \;\; R_c^T \\ C_2 \end{bmatrix} \tag{3.3a}$$

and we will assume hereon that for any C we know R_r, B, N, R_c and C_2. Now consider the following equation

$$C_1 \bar{\Gamma} = 0 \tag{3.4}$$

Clearly, (3.3) and (3.4) have the same set of solutions. Also, (3.4) can be rewritten as

$$C_1 R_c R_c^T \bar{\Gamma} = 0 \tag{3.5}$$

or, letting $R_c^T \bar{\Gamma} = \bar{\Gamma}_N'$ and using the definition of C'_1

$$C'_1 \bar{\Gamma}' = 0 \tag{3.6}$$

i.e.,

$$[B \;\; N] \;\; [\bar{\Gamma}_B' \;\; \bar{\Gamma}_N']^T = 0 \tag{3.7}$$

The solution to (3.7) satisfies the relation

$$\bar{\Gamma}_B' = -B^{-1} \;\; N \;\; \bar{\Gamma}_N'$$

or, equivalently

$$\bar{\Gamma}' = \begin{bmatrix} -B^{-1}N \\ I \end{bmatrix} \;\; \bar{\Gamma}_N' \tag{3.8}$$

Finally, from (3.5) and (3.8) we obtain the following relation that all solutions of (3.3) must satisfy

$$\bar{\Gamma} = R_c\, \bar{\Gamma}_B' = \begin{bmatrix} -B^{-1}N \\ I \end{bmatrix} \;\; \bar{\Gamma}_N' \tag{3.9}$$

At this point we can discuss how the second condition for the necessity of a broadcast can be tested. Recalling the definition of linear time schedule, broadcasting is required for a linear schedule π_0 if and only if there are at least two distinct computations with indices $\bar{j}'$, $\bar{j}''$, such that (3.3) is satisfied and $\pi_0 \bar{j}' = \pi_0 \bar{j}''$, i.e.,

$$\pi_0 \bar{\Gamma} = \pi_0 (\bar{j}' - \bar{j}'') = 0 \tag{3.10}$$

The solutions to (3.3) are given by (3.9). Replacing $\bar{\Gamma}$ in (3.10) by the right-hand side of (3.9) yields

$$\pi_0 R_c \begin{bmatrix} -B^{-1}N \\ I \end{bmatrix} \;\; \bar{\Gamma}_N' = 0 \tag{3.11}$$

The next lemma relates the existence of nontrivial solutions of (3.11) to the need for broadcasting a variable. The proof follows easily from the considerations made so far and it is omitted.

Lemma 2

Consider an algorithm with a set of variables with a (m×n) indexing matrix C as in (3.3a) and $1 \leq \text{rank}(C) = k < n$. Broadcasting is required during the execution of the algorithm if and only if there is at least one nontrivial solution $\bar{\Gamma}_N'$ to the equation

$$\pi_0 R_c \begin{bmatrix} -B^{-1}N \\ I \end{bmatrix} \bar{\Gamma}_N' = 0 \qquad (3.12)$$

where $\bar{\Gamma}_N'$ is a $((n-k)\times 1)$ vector and π_0 is the linear time schedule used to execute the algorithm.

The next theorem shows that there is a large class of indexing functions for which data broadcasts can never be avoided no matter what linear time schedule is used (besides strictly sequential execution). For indexing functions outside such class, a simple test is provided to decide if a given linear time schedule requires data broadcasts.

Theorem 3.1

For an algorithm with an arbitrarily large index set and a set of variables with an $(m\times n)$ indexing matrix C as in (3.3a),

(i) there exist linear time schedules for which no broadcasts are necessary if and only if $n-1 \leq \text{rank}(C) \leq n$

(ii) if $\text{rank}(C) = n-1$ then a time schedule π_0 requires broadcasting if and only if

$$\pi_0 R_c \begin{bmatrix} -B^{-1}N \\ I \end{bmatrix} = 0 \qquad (3.13)$$

Proof

We start by proving (ii). Clearly, if (3.13) is satisfied, (3.12) has non trivial solutions and, from lemma 2, broadcasts are required. Conversely, if some variable needs to be broadcasted, there are nontrivial solutions to (3.12). Noticing that $\pi_0 R_c \begin{bmatrix} -B^{-1}N \\ I \end{bmatrix}$ is a scalar (for $\text{rank}(C) = n-1$) it follows that (3.13) must also be satisfied. To prove (i), from lemma 1 and from (ii), if $n-1 \leq \text{rank}(C) \leq n$ there exist schedules for which broadcasts are not required. To prove the "only if" part, let

$$\bar{Z}^T = \pi_0 R_c \begin{bmatrix} -B^{-1}N \\ I \end{bmatrix}$$

and rewrite (3.12) as $\bar{Z}^T \bar{\Gamma}_N' = 0$. Noticing that $\bar{Z}$ is a $((n-k)\times 1)$ vector it follows that there are always nontrivial solutions to (3.12) whenever $\text{rank}(C) = k \leq n-1$. Hence, broadcasts will always be necessary, independently of the time schedule π_0 used. Q.E.D.

Example 3.1

Let $n=4$ and consider the variables $a(j_1-j_2, j_2-j_1, j_3, 2*j_4-j_3)$. We have

$$C = \begin{bmatrix} 1 & -1 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 2 \end{bmatrix} =$$

$$= \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \left[\begin{array}{ccc|c} 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 \\ 0 & -1 & 2 & 0 \\ \hline 1 & -1 & 0 & 0 \end{array} \right] \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} = R_r^T \begin{bmatrix} [B & N] R_c^T \\ & C_2 \end{bmatrix}$$

$\text{rank}(C) = 3 = n-1$

$$B^{-1} = \frac{1}{2} \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 1 & 1 \end{bmatrix} \qquad R_c \begin{bmatrix} -B^{-1}N \\ I \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

If $\pi = [t_1\, t_2\, t_3\, t_4]$ and $t_1 + t_2 = 0$ broadcasting is required. To illustrate this fact, let $\pi_1 = [0\ 1\ 1\ 1]$, $\pi_2 = [-1\ 1\ 0\ 0]$ and consider the variable a $(-1, 1, 1, 1)$ used at $\bar{j}' = (2\ 3\ 1\ 1)^t$ and $\bar{j}'' = (4\ 5\ 1\ 1)$. No broadcasting is required for π_1 because it schedules $\bar{j}'$ and $\bar{j}''$ for distinct instants of time (i.e., $\pi_1(\bar{j}') = 5 \neq \pi_1(\bar{j}'') = 7$). Broadcasting is required for π_2 because $\pi_2(\bar{j}') = \pi_2(\bar{j}'') = +1$.

Example 3.2

Let $n=4$ and consider the variables $a(j_1, j_2, j_3, j_4)$, $b(j_1-j_2+j_3,\ 2*j_2-j_1+1,\ j_3, 2*j_4-j_3)$ and $c(j_1-j_2, j_2-j_1+j_4,\ j_2+j_3+j_4,\ 2*j_4-j_3)$. The rank of the indexing function of each variable is n. Hence, for any linear time schedule, no broadcasting is required.

Example 3.3

Let $n=4$ and consider the variables $a(j_1)$, $b(j_1-j_2, j_2-j_1)$, $c(j_1-j_2, j_2-j_1,\ 2*j_3-j_4,\ j_4-2*j_3)$. These variables have indexing functions with rank less than $n-1$. For any linear time schedule broadcasts will be required.

The meaning of part (ii) of theorem 3.1 can be explained as follows: if $\text{rank}(C)=n-1$ then there is a schedule π_0 such that a single copy of each variable is needed for the execution of the algorithm; in other words, the time schedule allows the propagation of each variable to the processors which execute the computations using that variable. Clearly, this is possible only if the space schedule S is such that the available processor interconnections support such data communication. For this reason, we proceed to discuss what conditions must be satisfied by S in order to avoid broadcasts. Let $\bar{j}'$ and $\bar{j}''$ denote two points where computations use the same variable. The interval of time between such computations is

$$\tau = \left| \pi(\bar{j}' - \bar{j}'') \right| = \left| \pi\bar{\Gamma} \right| \qquad (3.14)$$

Hence, S must be such that

$$S\bar{\Gamma} = P\bar{W} \qquad (3.15)$$

where $\bar{W}$ is an $(r\times 1)$ vector with non-negative entries W_i, $i = 1,2,\ldots,r$ and $0 < \sum_{i=1}^{r} W_i \leq \tau$ and r is the number of columns of P.

Equation (3.15) means that the propagation of the variable from the processor executing the first computation to the processor executing the second computation can be made in an interval of time less than or equal to τ thru a path resulting from composing the interconnection primitives of the array processor. We assume that a single computation and propagation of a result over a single interconnection primitive takes one unit of time. From (3.9), theorem 3.1 and (3.14) it follows that the minimum (scalar) value of τ is

$$\tau_{\min} = \left| \pi L R_c \begin{bmatrix} -B^{-1}N \\ I \end{bmatrix} \right| \qquad (3.16)$$

where L is the smallest positive integer that makes $L R_c \begin{bmatrix} -B^{-1}N \\ I \end{bmatrix}$ an integer vector.

The next theorem gives the necessary conditions that a space schedule S must satisfy so that, for a given

π, broadcasts are not required and the array interconnections support the necessary data communication.

Theorem 3.2

Consider an array processor (L^{n-1},P), $P \in Z^{((n-1)\times r)}$ and an algorithm with index set J^n and a set of variables with indexing matrix C as in (3.3a) and rank(C) = n−1. Let π denote a time schedule such that $\tau_{min} \neq 0$, where τ_{min} is as in (3.16). Let S denote a space schedule such that $T = \begin{bmatrix}\pi \\ S\end{bmatrix}$ is nonsingular and

$$S R_c \begin{bmatrix}-B^{-1}N \\ I\end{bmatrix} = P\bar{W} \qquad (3.17)$$

$$\text{where } \bar{W} \in I^{(r\times 1)} \text{ and } 0 < \sum_{i=1}^{r} W_i \leq \tau_{min} \qquad (3.18)$$

The execution of the algorithm on the array linearly scheduled by T does not require broadcasts.

Proof

Theorem 3.1 proves that if $\tau_{min} \neq 0$ then no broadcasts are required. The discussion preceding this theorem shows that if (3.17) holds then the necessary data communication can be done using the interconnection primitives of the array. It remains to show that such communication takes less time than the interval of time between any two consecutive usage of the same variable. This interval of time is at least τ_{min} and if (3.18) holds for τ_{min} then it will also hold for longer intervals. From the assumption that one interconnection primitive can be used in one unit of time if follows that data communication can be done. Q.E.D.

Example 3.4

Let n=3, assume that P is as in example 2.2, and consider the variables a $(j_1 + 2*j_2 + j_3,\ 3*j_1 + j_2 + 2*j_3,\ 2*j_1 - j_2 + j_3)$. We have

$$C = \left[\begin{array}{cc|c} 1 & 2 & 1 \\ 3 & 1 & 2 \\ \hline 2 & -1 & 1 \end{array}\right] = \left[\begin{array}{c|c} B & N \\ \hline C_2 \end{array}\right], \quad \text{rank}(C) = 2 = n-1$$

$$B^{-1} = \frac{1}{5}\begin{bmatrix}-1 & 2 \\ 3 & -1\end{bmatrix}, \quad \pi L \begin{bmatrix}-B^{-1}N \\ I\end{bmatrix} = \frac{\pi L}{5}\begin{bmatrix}-3 \\ -1 \\ 5\end{bmatrix}$$

Let $L = 5$, $\pi = [1\ 0\ 0]$. Then $S[-3\ -1\ 5]^T = P\bar{W}$ for $S = \begin{bmatrix}0 & -1 & 0 \\ 1 & 2 & -1\end{bmatrix}$, $\bar{W} = [1\ 0\ 0\ 0\ 0]^T$. Also $T = \begin{bmatrix}\pi \\ S\end{bmatrix}$ is nonsingular. Notice that other solutions for S may exist. For illustration purposes consider a(6, 12, 6) used at $\bar{j}' = (3\ 1\ 1)^T$ and $\bar{j}'' = (0\ 0\ 6)^T$. Communication between processor $S\bar{j}' = (-1\ 6)^T$ and $S\bar{j}'' = (0\ 6)^T$ can be done in less than $\pi\bar{j}' - \pi\bar{j}'' = \tau_{min} = 3$ units of time by using primitive $(1\ 0)^T$.

4. On Reducing Large Broadcasts to Small Broadcasts

In an array architecture, every processor can broadcast a variable to all processors directly connected to it. In this section we discuss how linear schedules can explore this limited broadcasting capability to implement large broadcasts. We show that if there is an unique solution to a linear program involving the index set of the algorithm and there is a certain relation between the rank of the matrix of interconnection primitives, the rank of the indexing matrix of the variable to be broadcasted and the dimensionality of the array then there is a class of linear schedules for which broadcasting is possible.

Before deriving this result in a formal manner, we proceed to discuss informally what conditions must be checked for. First, the linear time schedule must be such that the first use of any given variable during the execution of the algorithm must occur for a single computation. In other words, in this section we preclude the possibility of inputing several copies of the same variable. This would correspond to a change in the rank of the indexing matrix in the sense that additional indexing information would be required in order to distinguish such copies. It also means that the original variable can only be replicated locally by each processor at a rate that depends on the interconnections available. Then, the second condition that we must check is that the rate of increase in the number of computations using the same variable is not larger than the rate at which the array is able to replicate that variable. Finally the third and last condition is that the space schedule must be such that communication can take place using the interconnection primitives of the array in the interval of time separating consecutive usages of the variable.

In order to discuss the first condition mentioned above, we recall (from (3.10) and (3.11)) that if the same variable is used at the same time by computations with indices $\bar{j}^0$ and $\bar{j}*$ then

$$\pi\bar{\Gamma} = \pi(\bar{j}^0 - \bar{j}*) = \pi R_c \begin{bmatrix}-B^{-1}N \\ I\end{bmatrix} \bar{\Gamma}'_N = 0 \qquad (4.1)$$

where

$$\bar{\Gamma}'_N = R_c^T(\bar{j}^0_N - \bar{j}^*_N) = \bar{j}^{0\prime}_N - \bar{j}^{*\prime}_N$$

We want to select π such that there exists a point $\bar{j}^0$ that is the index of the unique computation where the variable is used for the first time during the execution of the algorithm. This is equivalent to saying that, for a given algorithm with index set J^n, for a linear time schedule π and for an indexing matrix C, we have

Condition 4.1

there exists an unique point $\bar{j}^0 \in J^n$ such that
$\pi\bar{j}^0 = \min\{\pi\bar{j} : \bar{j} \in J^n,\ C(\bar{j}) = C(\bar{j}^0)\}$

or, equivalently,

$$\pi V \bar{j}^{0\prime}_N = \min\{\pi V \bar{j}^{0\prime}_N : V\bar{j} \in J^n\} \quad \text{where} \quad V = R_c \begin{bmatrix}-B^{-1}N \\ I\end{bmatrix}$$

Now, we discuss the second condition, i.e., the number of computations using a variable does not grow faster than the number of copies of that variable generated by local broadcasts. We show (in lemmas 4.1 and 4.2) that this condition is satisfied if the following condition holds: for any $\bar{j} \in J^n$, let $\bar{j} = \pi V \bar{j}'_N$ and let $\bar{X}_{max}$ and $\bar{X}_{min}$ denote the $((n-k)\times 1)$ vectors whose entries are the maximum and minimum values, respectively, of the corresponding components in $\bar{j}'_N$ for all $\bar{j} \in J^n$; a point $\bar{j}^0 \in J^n$ satisfies

Condition 4.2

If every entry of $\bar{j}^{0\prime}_N$ has the same value of the corresponding entry of either $\bar{X}_{max}$ or $\bar{X}_{min}$.

Lemma 4.1

Let $\bar{j}^0 \in J^n$ satisfy conditions 4.1 and 4.2 for some π. There exist at most $k^* = \binom{k+q-1}{k-1}$, $q \in I$, points $\bar{j}^1,\ldots,\bar{j}^{k^*} \in J^n$ such that

$$\pi(\bar{j}^i) = \pi(\bar{j}^0)+q \qquad i = 1,\ldots,k^* \tag{4.2}$$

Proof

Because π is a linear transformation we can rewrite (4.2) as

$$\pi(\bar{j}^i-\bar{j}^0) = \pi\bar{\Gamma} = q$$

or, from (4.1)

$$\pi R_c \begin{bmatrix} -B^{-1}N \\ I \end{bmatrix} \bar{\Gamma}'_N = \pi V \bar{\Gamma}'_N = q \tag{4.3}$$

Let $\pi V = [t_{n-k} \cdots t_n]$. Then (4.3) can be written as

$$Z^T \bar{\Gamma}_N' = t_{n-k} \frac{S_{n-k}}{|t_{n-k}|} + \ldots + t_n \frac{S_n}{|t_n|} = q \tag{4.4}$$

for all vectors $\bar{S}$ such that

(a) $\sum_{i=n-k}^{n} S_i = q$

(b) $S_i \in Z \quad i=n-k,\ldots,n$

(c) $\bar{j} = (\bar{j}^0+\bar{\Gamma}) \in J^n$ where $\bar{\Gamma}$ is such that

$$\bar{\Gamma}'_N = \left[\frac{S_{n-k}}{|t_{n-k}|} \cdots \frac{S_n}{|t_n|}\right]$$

All parcels of (4.4) are positive for otherwise, $\bar{j}^0$ would not satisfy conditions 4.1 and 4.2. Hence, there are at most as many vectors satisfying (a), (b) and (c) as there are vectors satisfying (a) and (b), i.e., vectors for which

$$\sum_{i=n-k}^{n} S_i = q \qquad S_i \in I$$

and there are exactly $k^* = \binom{k+q-1}{k-1}$ such vectors.

Q.E.D.

The next lemma characterizes the broadcasting capabilities of array processors as a function of the rank of the matrix of interconnection primitives and the length of the interconnection paths used to implement broadcasts. We consider array processors of arbitrarily large size due to the fact that a small array can always be extended in time by using the algorithm partitioning techniques described in [3,5,15].

Lemma 4.2

Consider an arbitrarily large array processor with matrix of interconnection primitives P. If rank(P) = k then there exist at least $\binom{k+q-1}{k-1}$ paths of length q built by composing k linearly independent interconnection primitives and connecting an arbitrary processor to exactly $\binom{k+q-1}{k-1}$ processors.

Proof

A path resulting from composing linearly independent interconnection primitives is unique up to commutativity (in the sense that the starting and ending processors are the same). Hence, a path of length q can be represented by a vector $\bar{v}$ such that

$$\bar{v} = \sum_{i=1}^{k} S_i \bar{p}_i \quad \text{where } \bar{p}_i \in P,\ S_i \in I \quad \text{and} \quad \sum_{i=1}^{k} S_i = q$$

Then, there are exactly $\binom{k+q-1}{k-1}$ processors connected to the starting processor through at least as many paths (there can be more paths due to commutativity, as mentioned before). Q.E.D.

The next theorem and its corollaries show that for a large class of algorithms there exist linear schedules for which broadcasts of non-scalar variables can be implemented as sequences of small local broadcasts.

Theorem 4.1

Consider an algorithm with index set J^n and a set of input variables for which the indexing matrix C is as in (3.3a). Consider also an array processor of dimension (n−1) and a matrix of interconnection primitives $P \in Z^{((n-1)\times k)}$. There exist linear time schedules π_0 for which all broadcasts can be implemented if

(i) $n-\text{rank}(P) \le \text{rank}(C) \le n$ and

(ii) π_0 is such that conditions 4.1 and 4.2 are satisfied.

Proof

If rank(C) = n then broadcasts are not required (from lemma 4.3.1). From lemma 4.2 a processor can communicate with $\binom{\text{rank}(P)+q-1}{\text{rank}(P)-1}$ in q interconnection steps. From lemma 4.1, if (ii) holds then only broadcasts to at most $\binom{n-\text{rank}(C)+q-1}{n-\text{rank}(C)-1}$ processors need to be implemented in q units of time when the time schedule π_0 is used. Hence, if $\text{rank}(P) \ge n-\text{rank}(C)$, i.e., $n-\text{rank}(P) \le \text{rank}(C)$, then all broadcasts can be implemented. Q.E.D.

Corollary 4.1.1

If $\bar{0} \in J^n$, $n-\text{rank}(P) \le \text{rank}(C) < n$ and $\pi_0 R_c \begin{bmatrix} -B^{-1}N \\ I \end{bmatrix}$ has all its entries positive then all broadcasts can be implemented.

Proof

Part (ii) of theorem 4.1 is satisfied for $\bar{j}^0_N = \bar{0}$. ($\bar{0}$ denotes a zero vector.) Q.E.D.

Corollary 4.1.2

If the conditions of corollary 4.1.1 hold and the array processor is fully connected then the broadcasts required by non-scalar variables can be implemented

Proof

From corollary 4.1.1 if we note that in a fully connected array, rank(P) = n−1 and the indexing matrix of a non-scalar variable has rank larger than zero. Q.E.D.

Consider the case when there is a linear time schedule satisfying the conditions of theorem 4.1. In the

next theorem we state the conditions that a linear space schedule must satisfy so that the interconnections of the array effectively support the required local broadcasts. The reasoning behind this theorem is similar to the one used for theorem 4.3.2. The difference is in equation (3.15) where the vector $\overline{W}$ is replaced by a matrix W. The entries of each column of W must satisfy the same conditions as the entries of $\overline{W}$. The proof of theorem 4.3.2 can be easily extended to prove the next theorem and we omit such extension here.

Theorem 4.2

Consider the algorithm and the array of theorem 4.1 and assume that conditions (i) and (ii) of the same theorem are satisfied for a given time schedule π_0. Let S denote a linear space schedule such that $T = \begin{bmatrix}\pi_0 \\ S\end{bmatrix}$ is non-singular and

$$SR_c\begin{bmatrix}-B^{-1}N \\ I\end{bmatrix} = PW$$

where $W \in I^{(r\times(n-k))}$ and $0 < \sum_{i=1}^{r} W_{ij} \leq (\tau_{min})_j$, $j = 1,\ldots,n-k$ where $(\tau_{min})_j$ is the jth entry of $\pi_0 LR_c\begin{bmatrix}-B^{-1}N \\ I\end{bmatrix}$. All broadcasts required by the execution of the algorithm using schedule $T = \begin{bmatrix}\pi_0 \\ S\end{bmatrix}$ can be implemented as a sequence of local broadcasts using the interconnection primitives of the array.

Example 4.1

Consider the algorithm described by the following program.

```
FOR J1=0 TO 5
FOR J2=0 TO 5
FOR J3=0 TO 5
G(J1,J2,J3)=G(J1,J2-1,J3)*A(J1-J2,2*J1-2*J2,-J1+J2)
END J3
END J2
END J1
```

Suppose that we wish to execute this algorithm in a (6×6) array like the WAP described in example 2.2 (for which rank(P) = 2). The indexing matrix of the input variables A(J1−J2,2*J1−2*J2,−J1+J2) is

$$C = \begin{bmatrix}1 & -1 & 0 \\ 2 & -2 & 0 \\ -1 & 1 & 0\end{bmatrix}$$

and rank(C) = 1 = n−rank(P) = 3−2. Also,

$$B = [1] \quad N = [-1 \quad 0] \quad \begin{bmatrix}-B^{-1}N \\ I\end{bmatrix} = \begin{bmatrix}1 & 0 \\ 1 & 0 \\ 0 & 1\end{bmatrix}$$

Choosing $\pi^0 = [0\ 1\ 1]$ and $S = \begin{bmatrix}1 & 0 & 0 \\ 0 & 0 & 1\end{bmatrix}$ it follows from corollary 4.1.1 and theorem 4.2 that all broadcasts can be implemented. To illustrate this fact consider the variable A(−1,−2,1) used at time $\pi(J1\ J2\ J3) = 5$ for the computations with index points (0 1 4), (1 2 3), (2 3 2), (3 4 1) and (3 5 0) in processors (0 4), (1 3), (2 2), (3 1), and (3 0). This variable was communicated using the interconnection primitives (0 1) and (1 0) from processors (0 3), (1 2), (2 1), and (3 0) where it was used at time t=4 for the computations with index points (0 1 3), (1 2 2), (2 3 1), and (3 4 0). These processors, in turn, received a copy of the variable from processors (0 2), (1 1) and (2 0) where it was used at time t=3 for the computations indexed by (0 1 2), (1 2 1) and (2 3 0). Similarly, these had received the same variable from processors (0 1) and (1 0) where it was used at time t=2 for computations indexed by (0 1 1) and (1 2 0). These, in turn, received A(−1,−2,−1) from processor (0 0) where it was used for the first time at t=1. This succession of reduced broadcasts is illustrated in figure 4.1. In this figure we also show the "broadcasting wavefronts" that are not to be confused with the computational wavefronts that are parallel to the direction of the coordinate ℓ_1 (in this example).

From figure 4.1 and our description of the successive broadcasts, some processors can receive the same variable from two distinct processors. In an actual implementation some local control rule would determine which processor sends the variable. Assume that each processor has a bit denoted X that has the value 1 if data comes from its southern neighbor and has the value 0 if data comes from its left neighbor. This allows us to rewrite the original program in a form that clearly shows how each variable is propagated.

```
FOR J1=0 TO 5
  FOR J2=0 TO 5
    FOR J3=0 TO 5
      IF X=1 THEN
        A(L1-L2,2*L1-2*L2,-L1+L2)|L1=J1,L2=J2,L3=J3 =
          A(L1-L2,2*L1-2*L2,-L1+L2)|L1=J1-1,L2=J-1,L3=J3
      ELSE
        A(L1-L2,2*L1-2*L2,-L1+L2)|L1=J1,L2=J2,L3=J3 =
          A(L1-L2,2*L1-2*L2,-L1+L2)|L1=J1,L2=J2,L3=J3-1
      G(J1,J2,J3)=G(J1,J2-1,J3)*A(J1-J2,2*J1-2*J2,-J1+J2)
    END J3
  END J2
END J1
```

End of example.

V. CONCLUSIONS AND FURTHER RESEARCH

We showed how to linearly schedule array processors so that broadcasts can be eliminated or reduced to sequences of local broadcasts. We considered the case of linear schedules mapping n-dimensional algorithms into the dimension of time and into (n−1)-dimensional arrays. In the general case, linear schedules map n-dimensional algorithms into (n−x)-dimensional arrays, where $1 \leq x < n$. Although these schedules are less understood, we believe that our results can be extended to such cases. For example part (i) of theorem 3.1 would state that there are schedules for which no broadcasts are required if and only if $n-x \leq \text{rank}(C) \leq n$. The philosophy behind our technique is to order computations (perhaps losing potential computational speed) so that a given variable can be shared or locally replicated by every processor using it. However, the mathematical formulation of such techniques allows us to look for schedules with minimal speed degradation. Linear schedules must satisfy other conditions besides those discussed here. In particular, they must not violate the algorithm data dependences. These conditions have

been studied in [3, 14]. Also, it is often the case that a same variable is referenced more than once in an algorithm with distinct indexing matrices. One possible solution is to consider a distinct variable for each indexing function by replicating the original variable. Further research could concentrate on more efficient solutions to these cases.

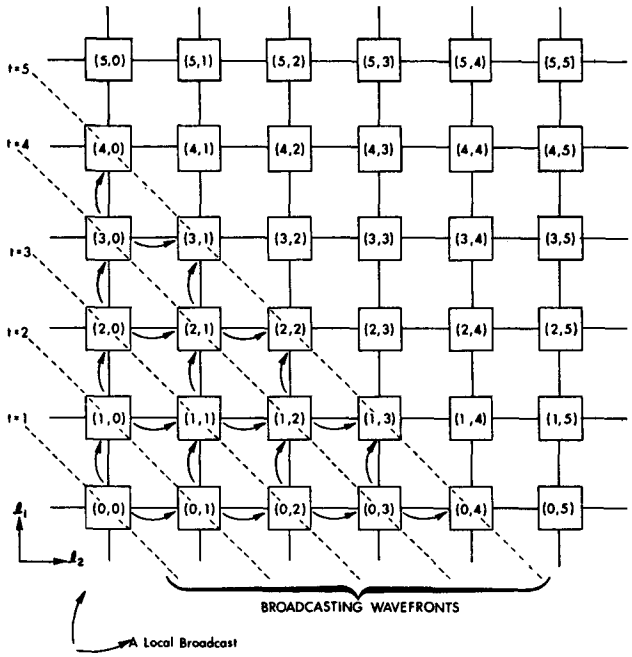

Figure 4.1. A succession of local broadcasts that implement the large broadcasts discussed in example 4.1

Also, the results of this paper show a direct relation between the rank of the indexing matrix of a variable and the need for broadcasts. By changing the rank of the indexing matrix we can reduce such broadcasts. Hence, our results can be used as a basis for renaming and expansion algorithms for optimizing compilers. Finally, some parallel computers have broadcasting capabilities and they can be used to achieve fast execution of an algorithm. Our results can be easily modified so that broadcasts are used instead of avoided (by selecting π_0 so that it satisfies (3.12)).

Acknowledgments: To Nancy Lein and Carol Gordon for the careful typing of the manuscript.

REFERENCES

[1] Barnes et al., "The ILLIAC-VI computer," *IEEE Trans. on Computers,* C-17, Aug. 1968, pp. 746-757.

[2] K. E. Batcher, "Design of a massively parallel processor," *IEEE Trans. on Computers,* C-29, pp. 836-840.

[3] J. A. B. Fortes, "Algorithm transformations for parallel processing and VLSI architecture design," Ph.D. Thesis, Dept. of EE-Systems, University of Southern California, December 1983.

[4] J. A. B. Fortes, D. I. Moldovan, "Parallelism detection and algorithm transformation techniques useful for VLSI designs," Tech. Report. PPP83-1, Dept. of EE-Systems, Univ. of Southern Calif., 1983.

[5] K. Hwang, Y-H Chen, "Partitioned matrix algorithms for VLSI arithmetic systems," *IEEE Trans. on Computers,* C-31, No. 12, December 1982, pp. 1215-1224.

[6] D. J. Kuck, "The structure of computers and computations," Vol. 1, John Wiley & Sons, Inc., 1978.

[7] D. J. Kuck, et al., "Dependence graphs and compiler optimizations," in *Proc. 8th ACM Symp. Principles of Programming Languages,* Jan. 1981, pp. 207-218.

[8] R. H. Kuhn, "Optimization and interconnection complexity for parallel processors, single stage networks and decision trees," Ph.D. Thesis, Dept. of Computer Science, Report 80-1009, Univ. of Illinois, Urbana-Champaign, Ill., Feb. 1980.

[9] H. T. Kung, "Let's design algorithms for VLSI systems," in *Proc. of Caltech Conf. on VLSI,* California Institute of Technology, Jan, 1979.

[10] S. Y. Kung, "Wavefront array processor: language, architecture and applications," *IEEE Trans. on Computers,* Vol. C-31, No. 11, Nov. 1982, pp. 1054-1066.

[11] L. Lamport, "The parallel execution of DO-loops," *Comm. of ACM,* Vol. 17, No. 2, February 1976, pp. 83-93.

[12] C. E. Leiserson, J. B. Saxe, "Optimizing synchronous systems," in *Proc. 22nd Annual Symposium on Foundations of Computer Science, IEEE Computer Society,* Oct. 1981, pp. 23-34.

[13] D. I. Moldovan, "On the analysis and synthesis of VLSI algorithms," *IEEE Trans. on Computers,* Vol. C-31, No. 11, Nov. 1982.

[14] D. I. Moldovan, "On the design of algorithms for VLSI systolic arrays," *Proc. of IEEE,* Vol. 71, No. 1, pp. 113-120, Jan. 1983.

[15] D. I. Moldovan, J. A. B. Fortes, "Partitioning algorithms for fixed size VLSI architectures," Tech. Report PPP83-5, Dept. of EE-Systems, University of Southern California, 1983.

[16] D. Nassimi, S. Sahni, "Data Broadcasting in SIMD computers," *IEEE Trans. on Comp.,* Vol. C-30, No. 2, Feb. 1982.

[17] L. Snyder, "Introduction to the configurable highly parallel computer," *Computer,* Vol. 15, No. 1, Jan. 1982, pp. 47-64.