BOUNDED BROADCAST IN SYSTOLIC ARRAYS

YOAV YAACOBY
Rafael, Dept. 84
P.O. Box 2250
Haifa, 31021, Israel

PETER CAPPELLO
Department of Computer Science
University of California
Santa Barbara, CA 93106

September 20, 2006

Abstract

Much work has been done on the problem of synthesizing a processor array from a system of recurrence equations. Some researchers limit communication to nearest neighbors in the array; others use broadcast. In many cases, neither of the above approaches result in an optimal execution time.

In this paper a technique called bounded broadcast is explored whereby an element of a processor array can broadcast to a bounded number of other processors. This technique is applied to the problems of transitive closure and all-pairs shortest distance, resulting in time complexities that are smaller than those reported previously. In general, the technique can be used to design bounded broadcast systolic arrays for algorithms whose implementation can benefit from broadcasting.

Keywords:

all-pairs shortest distance, broadcast, data dependence, parallel computation, recurrence equation, systolic array, transitive closure, VLSI architecture.

1 Introduction

A system of uniform recurrence equations, as defined by Karp, Miller, and Winograd [23, 22], maps especially well onto a systolic/wavefront array. Many

researchers have either linearly mapped systems of uniform recurrence equations into spacetime, or translated them to systolic/wavefront arrays [36, 6, 37, 8, 40, 35, 15, 12, 43, 38, 42, 11, 26, 49]. Another approach in parallel processing is to allow broadcasting. This approach has been considered by several authors (e.g., see [53, 5, 1, 34]). In broadcasting, a single processor can broadcast data to all processors simultaneously. A variation of this technique is to broadcast only in columns and rows, such that every processor can broadcast data to all other processors in the same row or column of a processor mesh [25, 50]. When broadcasting, signal propagation time depends on the number of processors in the multiprocessor system.

In this paper we suggest a compromise between global broadcast and nearest neighbor communication. This technique, called *bounded broadcast*, allows a processor to send data to all processors on the same row/column, which are within some bounded distance from it. This technique enhances the timing results of many algorithms which can use broadcast. We apply this technique to the transitive closure and all-pairs shortest distance problems [55, 14]. These problems have been considered by many researchers (e.g., see [18],[54, ch. 5], [43, page 289],[27, 47, 46, 48, 31, 20, 29, 28]). At present the best systolic array execution time reported for an $N \times N$ input matrix is $5N - 4$ [28], where the unit of time is the time it takes to make one computational step plus the time to transmit the result from one processor to its neighbor. By using a bounded broadcast, this time can be reduced to between N and $4N$ (ignoring low-order terms), depending on the design. Bounded broadcast can be used in other problems such as the general Algebraic Path Problem (APP) [30, 58, 16, 4, 10, 32, 41] and many matrix computations. (An execution time of about $4N$ is achieved by Delosme [10]. He uses a non-classical algorithm, avoiding broadcast altogether, and a larger number of processors.)

A handful of architectures that implement reconfigurable buses already have been presented [3, 52, 34, 33]. These architectures are well suited to implement the bounded broadcast described here, after the algorithm has been mapped to space and time *using the procedure described in this paper*. An earlier version of this work can be found in [56]. A similar approach recently has been developed independently by Risset and Robert [45, 44].

2 Using Bounded Broadcast to Reduce Latency

2.1 Definitions

Example 1

The system of recurrence equations (SRE) below finds the transitive closure, or all-pairs shortest distance, for a given input matrix $A_{N\times N}$. This algorithm is the implementation of the Warshall-Floyd algorithm suggested by Kung et al. [28]. We have changed the names used there as follows: $x \rightarrow a_1$, $r \rightarrow a_2$ and

$c \rightarrow a_3$. Also, the input and output parts of the SRE are not given here.

$$
\begin{array}{ll}
1 \le k \le N, & \\
\quad 1 \le j \le N, & \\
\qquad 1 \le i \le N, & a_1(i,j,k) = a_1(i \bmod N + 1, j \bmod N + 1, k-1) \oplus a_2(i,j,k) \otimes a_3(i,j,k) \quad (1) \\
1 \le k \le N, & \\
\quad 1 \le j \le N, & a_2(1,j,k) = a_1(2, j \bmod N + 1, k-1) \quad (2) \\
1 \le k \le N, & \\
\quad 1 \le j \le N, & \\
\qquad 2 \le i \le N, & a_2(i,j,k) = a_2(i-1,j,k) \quad (3) \\
1 \le k \le N, & \\
\quad 1 \le i \le N, & a_3(i,1,k) = a_1(i \bmod N + 1, 2, k-1) \quad (4) \\
1 \le k \le N, & \\
\quad 2 \le j \le N, & \\
\qquad 1 \le i \le N, & a_3(i,j,k) = a_3(i,j-1,k) \quad (5)
\end{array}
$$

The operation $\otimes$ models binary AND in the transitive closure case, and $+$ in the all-pairs shortest distance case. The operation $\oplus$ models binary OR in the transitive closure case, and the MINIMUM operation in the all-pairs shortest distance case[1].

The recurrence equations above are used to illustrate some of the following definitions, which are related to an SRE.

Index set: The set of points where an array is computed or used.

Domain of computation: The set of points C_i where an array a_i is computed (e.g., $C_1 = \{(i,j,k) | 1 \le i \le N,\ 1 \le j \le N,\ 1 \le k \le N\}$ in Eq. (1)).

Dependence map: A function δ_{ij} from the domain of computation of array a_j (or part of it) to the index set of a_i, on which the computation of a_j depends (e.g., $\delta_{12}(\ i \quad j \quad k\ )^T = (\ i \quad j \quad k\ )^T + (\ 1 \quad 1 \quad -1\ )^T$ in Eq. (2) for $j \neq N$).

Uniform dependence: A dependence map of the form: $\delta_{ij}(p) = p + d_{ij}$ where $d_{ij} \in \mathcal{Z}^n$, and n is the dimension of the array variables. The vector d_{ij} is referred to as the translation part of the dependence map.

A system of uniform recurrence equations (SURE): An SRE where the dependence maps are uniform, and every array is computed in one recurrence equation for its entire domain of computation. (The example given above thus is not an SURE.)

A system of quasi-uniform recurrence equations: An SRE that is uniform except for boundary points (for a more precise definition, see [57]). E.g., the SRE given in Ex. 1 is quasi-uniform.

[1] In general, these are the operators of a closed semiring [2].

Latency with respect to function f: The time T_f separating the arrival of the first input from the departure of the last output, for one computation of f.

Given a system of recurrence equations (SRE), one of the design outputs is a vector $\pi \in \mathcal{Z}^n$ and a set of constants $\{c_i \in \mathcal{Z}\}$, one for each array, such that for all problem sizes, if a variable $a_j(x_2)$ depends on a variable $a_i(x_1)$ (not necessarily directly), then $\pi^T x_1 + c_i < \pi^T x_2 + c_j$. This condition ensures that there exists a valid execution ordering. Array variable $a_i(x)$ is computed at time[2] $\pi^T x + c_i$.

To simplify our presentation, we use a less general formulation: We assume that:

- $c_i = 0$;
- there is only one π.

The vector π in this case is referred to as a *linear schedule vector*, and the above mapping is referred to as a *linear schedule*.

2.2 Using bounded broadcast to reduce latency

The latency (sometimes referred to as delay time) comprises three components: 1) input, 2) execution, and 3) output. Sometimes it is possible to overlap the execution with the input/output. The execution time should be the time at which the last computation is done minus the time that the first computation is done. In the case of a linear schedule, it is $\max_{p \in C}(\pi^T p) - \min_{p \in C}(\pi^T p)$, where π is the linear schedule vector, and $C = \bigcup_i C_i$ (i.e., it is the union of the domains of computation of the arrays in the SRE).

Much research has been done on the problem of minimizing the latency for an integral π (e.g., [21, 43, 12, 17, 39, 9, 11]). Shang and Fortes [51] mention the possibility of a rational π, but do not explain the physical meaning of such a schedule. Here we give physical meaning to such a schedule. The idea is to distinguish between two kinds of dependences:

1. dependence of one computation step on the result of a previous computation step;
2. dependence of a computation step on a 'propagating variable' (a variable that is transmitted unchanged from one processor to the next).

An array of only propagating variables is called a *propagating array*. Some researchers assume that the time taken to propagate a variable from a processor

[2]By 'time' we really mean an ordering of the steps.

to its neighbor, is the same as the execution time for one computation step. Other researchers assume that values can be broadcast such that all processors get the propagated value at the same time.

Both of these assumptions lead to a sub-optimal latency. In the first case, propagation of the variable is delayed until the computation step is finished; in the second case, the computation step is delayed until the propagated variable is received by all processors.

In this paper, we assume that a variable can be propagated through a bounded number of neighbor connections. The idea is to broadcast a propagating variable through as many neighbors as possible such that the time to do so *balances* with the time to execute one computation step, and communicate the results to the nearest neighbor.

In this paper, K denotes the number of neighbor processors that a variable can propagate through, in the same time that it takes to 1) execute one computation step, and 2) communicate the results to a neighbor processor. K grows as the time complexity of the computation step grows. (The time complexity of a computation step is measured by the depth of the circuit that implements it.) K also depends on the method by which the variable is propagated. The two principal methods are to:

1. use a bus that goes through K processors (with possible repeaters);
2. insert a latch stage in every processor.

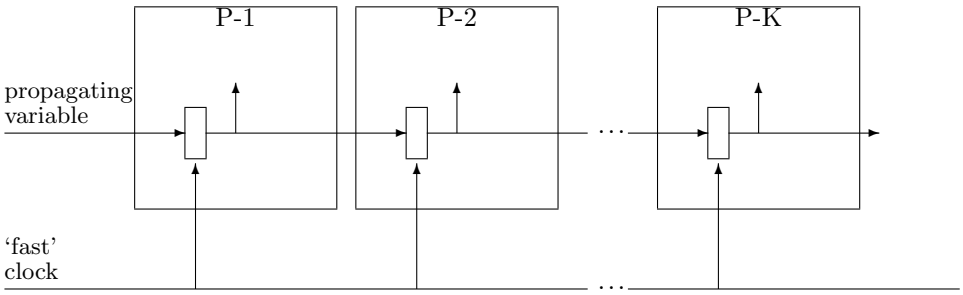

Figure 1: Propagating variable through a latch stage in each processor.

Again, if a variable is propagated to either fewer than K processors or more than K processors, then the overall latency is *not* minimized; in the former case, a propagating variable waits unnecessarily for a computation to complete; in the latter, a computation waits unnecessarily for a propagating variable to

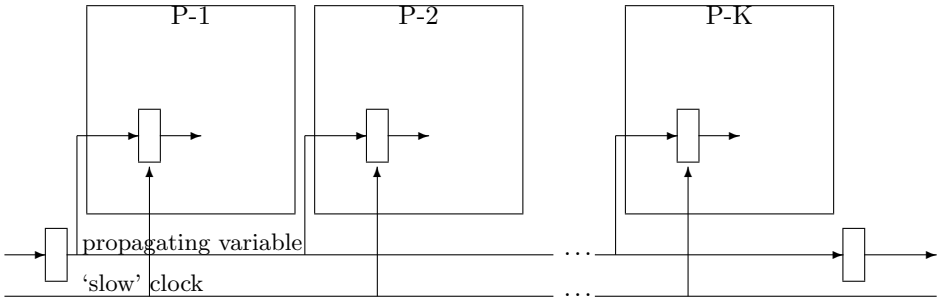

Figure 2: Propagating variable through a bus-like structure.

complete propagation. By realizing this K-broadcast, the inserted latches therefore *decrease* overall latency. The two variable propagation methods are shown schematically in Fig. 2 and Fig. 1, respectively. The value of K clearly decreases if latches are inserted. K also depends on the technology. Another factor which may affect the value of K is the time for clock distribution (if this exceeds the time for one computation step) [13, 7]. We ignore this factor, since clock distribution usually can be reduced by appropriate design [24, 19]. For present MOS technologies, and the simplest computation (1 – 3 gates), $K \gg 1$. Similarly it also may be that input and output steps take less time than the computation step. In this case, the input and/or output steps may be synchronized by a faster clock.

The discussion above indicates that we can choose π with fractional entries. In what follows, we assume that all entries of π are multiples of $1/K$. All the results generalize to any rational π. A minimization of the latency comprises a solution to an integer linear program. If π is multiple of $1/K$, then we can get an integer linear program by multiplying π by K to get an intermediate vector that is integer. One is still faced with 2 problems: 1) finding the constraints on π in the case of *bounded broadcast*; 2) finding a solution to the integer linear program, which is an NP-complete problem. Our distinction between propagating variables and other variables enables us to find these constraints. Following [43, page 129], we define the *iteration vector* to be a vector $u \in \mathcal{Z}^n$ that satisfies:

1. u is perpendicular to space.
2. The greatest common divisor of the components of u is 1.
3. The first non-zero component of u is greater than zero.

From the above definition, the length of u is the minimum distance between two index points in that direction.

Consider first the case where variables are propagated using latches (see Fig. 1). The choice of the vector π, in this case, is subject to the following constraints:

1. $|\pi^T u| \geq 1$.

   This constraint guarantees that two consecutive computations done in the same processor are done no less than one time step apart. (In case the computation step is pipelined in each processor, this restriction is changed: Instead of 1, the right hand side of the inequality becomes $1/L$, where L is the number of pipeline stages.)

2. For every dependence map of a propagating array (i.e., a dependence map for an array that depends on a propagating array), the corresponding translation part d satisfies $\pi^T d \leq -1/K$.

   This constraint guarantees that sufficient time exists for a propagating variable to be communicated to the location where it is used.

3. For every dependence map of a non-propagating array (i.e., a dependence map for an array that depends on a non-propagating array), the corresponding translation part d satisfies $\pi^T d \leq -1$.

   This constraint guarantees that the computation is complete before its result is required.

One jointly determines the spatial embedding and the linear schedule. The choice of rational entries in π implies that there is a faster basic clock in the array (e.g., if the minimum entry is $1/K$ as assumed here, this clock would be K times faster than the clock would be with a minimum entry of 1). A computation at index point p is executed at time step $\pi^T p$; the period of each time step is $1/K$. Latches thus must be present in each processor, as shown in Fig. 1.

If the computation is very simple, it may be impractical to pass a propagating variable through a latch of a processor before transmitting it to the next processor. In this case, the basic clock remains the same, and the propagating variable is transmitted via a bus to K processors before being latched, as shown in Fig. 2. The time step for a computation at index point p in this case is $\lceil \pi^T p \rceil$. In order to ensure valid timing in this case, we add one more constraint:

4. For every dependence map of a propagating array on a non-propagating array, the corresponding translation part d satisfies $\pi^T d \leq -2$.

   This constraint guarantees that there is enough time for a propagating variable to first be computed, and then broadcast to K processors.

   For example, suppose we choose π such that $\pi^T p = J + (1/K)$ for some integer J, and index point p in which a propagated variable is computed. The computation at p is done at time step $J+1$. Suppose this propagating

variable is used at computation index points $q_1, q_2, \ldots, q_K$, each of which satisfy $\pi^T q_i = J + 1 + (i/K)$. All the computations at these index points are done at time step $J + 2$. The constraint above ensures that there is enough time to compute the variable, and also propagate its value to all the above index points.

3 Applying bounded broadcast to the Warshall-Floyd algorithm

The following example illustrates the advantage of using bounded broadcast.

Example 1A

The SRE below, given input matrix $A_{N\times N}$, computes its transitive closure (or all-pairs shortest distance), A^+, which is the SRE's output. This SRE is an implementation of the Warshall-Floyd algorithm, derived from Ex. 1 by 1) eliminating global connections as explained in [28], 2) adding input statements (for A) and output statements (for A^+), and 3) eliminating zero translation vectors (in the dependence maps) by substitution. In this SRE, g represents the value 0 for the all-pairs shortest distance problem, and 1 for the transitive closure problem[3].

$$
\begin{array}{lll}
2 \le j \le N, & & \\
\quad 2 \le i \le N, & a_1(i,j,0) = A_{i-1,j-1} & (1) \\
2 \le j \le N, & & \\
\quad 1 \le i \le N, & a_2(i,j,0) = A_{i,j-1} & (2) \\
1 \le j \le N, & & \\
\quad 2 \le i \le N, & a_3(i,j,0) = A_{i-1,j} & (3) \\
2 \le k \le N, & a_1(N,N,k) = A_{k-1,k-1} = g & (4) \\
 & a_1(N,N,1) = A_{N,N} = g & (5) \\
1 \le k \le N, & & \\
\quad 2 \le i \le N-1, & a_1(i,N,k) = a_3(i+1,N,k-1) \oplus a_2(i-1,N,k) \otimes a_3(i,N-1,k) & (6) \\
 & a_1(1,N,k) = a_3(2,N,k-1) & (7) \\
\quad 2 \le j \le N-1, & a_1(N,j,k) = a_2(N,j+1,k-1) \oplus a_2(N-1,j,k) \otimes a_3(N,j-1,k) & (8) \\
 & a_1(N,1,k) = a_2(N,2,k-1) & (9) \\
\quad 2 \le j \le N-1, & & \\
\quad\quad 2 \le i \le N-1, & a_1(i,j,k) = a_1(i+1,j+1,k-1) \oplus a_2(i-1,j,k) \otimes a_3(i,j-1,k) & (10) \\
\quad 2 \le i \le N-1, & a_1(i,1,k) = a_1(i+1,2,k-1) & (11) \\
\quad 2 \le j \le N-1, & a_1(1,j,k) = a_1(2,j+1,k-1) & (12) \\
 & a_1(1,1,k) = a_1(2,2,k-1) & (13) \\
\quad 1 \le i \le N-1, & a_3(i,1,k) = a_1(i+1,2,k-1) & (14) \\
\quad 1 \le j \le N-1, & a_2(1,j,k) = a_1(2,j+1,k-1) & (15)
\end{array}
$$

[3] In general it is the multiplicative identity of the closed semiring.

$$a_3(N,1,k) = a_2(N,2,k-1) \tag{16}$$
$$a_2(1,N,k) = a_3(2,N,k-1) \tag{17}$$
$2 \le j \le N$, $1 \le i \le N$: $$a_3(i,j,k) = a_3(i,j-1,k) \tag{18}$$
$1 \le j \le N$, $2 \le i \le N$: $$a_2(i,j,k) = a_2(i-1,j,k) \tag{19}$$
$1 \le j \le N-1$, $1 \le i \le N-1$: $$A^+_{ij} = a_1(i+1,j+1,N) \tag{20}$$
$1 \le j \le N-1$: $$A^+_{Nj} = a_2(N,j+1,N) \tag{21}$$
$1 \le i \le N-1$: $$A^+_{iN} = a_3(i+1,N,N) \tag{22}$$
$$A^+_{NN} = A_{NN} \tag{23}$$

The array a_1 holds the transitive closure (or shortest distance) as it is being computed; a_2 and a_3 are propagating arrays in the i and j directions, respectively.

The distinct translation parts of the dependence maps in this SRE are:

$$\begin{pmatrix} -1 \\ 0 \\ 0 \end{pmatrix}; \quad \begin{pmatrix} 0 \\ -1 \\ 0 \end{pmatrix}; \quad \begin{pmatrix} 1 \\ 0 \\ -1 \end{pmatrix}; \quad \begin{pmatrix} 0 \\ 1 \\ -1 \end{pmatrix}; \quad \begin{pmatrix} 1 \\ 1 \\ -1 \end{pmatrix}.$$

The first four translation parts above correspond to propagating arrays (i.e., their dependence maps are for arrays that depend on a propagating array); the last translation part corresponds to a non-propagating array (i.e., a translation part of a dependence map for an array that depends on a_1, a non-propagating array). The above observation is used in picking a valid schedule vector π, that satisfies the constraints mentioned in § 2.2.

We consider 5 design cases for the above SRE. The first, presented in [28], is referred to as Design $\mathcal{K}$. The other 4 designs, denoted $\mathcal{A}$ – $\mathcal{D}$, use bounded broadcast. In what follows, we define the unit of time as the time to execute one computation step and communicate the results to a neighbor processor. We denote by K the number of processors to which a variable can be propagated in one time unit. Designs $\mathcal{A}$ and $\mathcal{B}$ use latches in the propagation path, as shown in Fig. 1; designs $\mathcal{C}$ and $\mathcal{D}$ use a bus-like structure, as shown in Fig. 2.

Table 1 contains the design parameters, and the times. In designs $\mathcal{A}$ and $\mathcal{C}$, input [output] is done before [after] the computation, taking N/M time steps. M = (unit of time)/(the time for a processor to communicate a variable to its neighbor). In order to further reduce the period (down to the execution time), one can incorporate special hardware so that input and output overlap execution.

In designs $\mathcal{K}$, $\mathcal{A}$, and $\mathcal{B}$, the computation at index point p is executed in time step $\pi^T p$. In designs $\mathcal{C}$ and $\mathcal{D}$, the computation at index point p is executed in time step $\lceil \pi^T p \rceil$. All these designs use N^2 processors. All of them are valid, because they satisfy the constraints for π. The latency of design $\mathcal{K}$ is claimed in [28] to be optimal. It is indeed optimal when π is restricted to be integer.

Table 1: Comparison of five designs for the Warshall-Floyd algorithm.

	Design $\mathcal{K}$	Design $\mathcal{A}$	Design $\mathcal{B}$	Design $\mathcal{C}$	Design $\mathcal{D}$
π^T	$(1,1,3)$	$(\frac{1}{K},\frac{1}{K},1+\frac{2}{K})$	$(1,\frac{1}{K},2+\frac{1}{K})$	$(\frac{1}{K},\frac{1}{K},2+\frac{2}{K})$	$(1,\frac{1}{K},3+\frac{1}{K})$
Iteration vector u^T	$(1,0,0)$	$(0,0,1)$	$(1,0,0)$	$(0,0,1)$	$(1,0,0)$
Execution time T	$5N-4$	$(N-1)(\frac{4}{K}+1)$ $+\frac{1}{K}$	$(N-1)(\frac{2}{K}+3)$ $+\frac{1}{K}$	$\lceil N(\frac{4}{K}+2)\rceil-2$	$\lceil N(\frac{2}{K}+4)\rceil-4$
Latency	T	$T+2N/M$	T	$T+2N/M$	T
Period	N	$T+N/M^b$	N	$T+N/M^b$	N
Asymptotic execution time[a]	$5N$	N	$3N$	$2N$	$4N$
Asymptotic latency[a]	$5N$	$N+2N/M$	$3N$	$2N+2N/M$	$4N$
Asymptotic period[a]	N	$N+N/M^b$	N	$2N+N/M^b$	N
Execution time for $N=100$, $K=10, M=1$	$4.96N$	$1.387N$	$3.169N$	$2.38N$	$4.16N$
Latency for $N=100$, $K=10, M=1$	$4.96N$	$3.387N$	$3.169N$	$4.38N$	$4.16N$
Period for $N=100$, $K=10, M=1$	N	$2.387N$	N	$3.38N$	N

[a] For $K \to N$, and ignoring additive constants.
[b] The last term can be eliminated by adding special hardware for input/output.

However, as can be seen from Table 1, it is not optimal when this restriction is removed.

If $K \geq 4$ when latching the propagating variable in every processor, then designs $\mathcal{A}$ and $\mathcal{B}$ are better than designs $\mathcal{C}$ and $\mathcal{D}$. $K = 3$ is a boundary case. If $K < 3$, then designs $\mathcal{C}$ and $\mathcal{D}$ are better than $\mathcal{A}$ and $\mathcal{B}$. Designs $\mathcal{A}$ and $\mathcal{B}$ thus are suited to a complex computation (such as in the all-pairs shortest distance problem); designs $\mathcal{C}$ and $\mathcal{D}$ are better suited to a simpler computation (such as the transitive closure problem).

Designs $\mathcal{A}$ and $\mathcal{C}$ are preferable in case either:

- matrix A does not need to be input, and matrix A^+ does not need to be output;
- input/output can be done fast in comparison with execution (i.e., $M > 1$) and the latency is more important than the period;
- hardware is used to overlap input/output with execution.

Otherwise designs $\mathcal{B}$ and $\mathcal{D}$ are preferable.

We have assumed thus far that the processors are not pipelined. If the computation is complex, then we may employ internal pipelining. This is advantageous when the first restriction, $|\pi^T u| \geq 1$, is the bottleneck. For example, consider design $\mathcal{B}$. Suppose we have L pipelining stages in each processor, and $L \leq K$. Then the following linear schedule vector can be chosen: $\pi^T = (\frac{1}{L}, \frac{1}{K}, 1 + \frac{1}{L} + \frac{1}{K})$. This vector satisfies all restrictions. The time step now is larger, since latches are added for pipelining. Let τ denote one time unit. Let $h(L) = (\tau \text{ using } L \text{ stages})/(\tau \text{ using 1 stage})$. The times for the pipelined version of design $\mathcal{B}$ are shown in Table 2. In this case the period is less than N. The 'penalty' here is that hardware is added to pipeline each processor.

4 Conclusions

We presented bounded broadcasting, an architectural feature that can improve the performance of systolic arrays. We suggested a distinction between propagating variables and other variables. Using this distinction, we identified several conditions on the linear schedule vector for a system of recurrence equations, which are sufficient to implement the SRE with bounded broadcast.

We then illustrated bounded broadcast on the problem of transitive closure/all-pairs shortest distance. The asymptotic latency, period, and execution times of the design of Kung et al. [28] (which are optimal for designs that do not use bounded broadcast) are $5N$, N, and $5N$, respectively. These same measures are $3N, 2N, N$ for design $\mathcal{A}$; $3N, N, 3N$ for design $\mathcal{B}$; $4N, 3N, 2N$ for design $\mathcal{C}$; and $4N, N, 4N$ for design $\mathcal{D}$. (These times can be reduced further by pipelining the computation step, and by providing hardware that overlaps input/output with execution.)

Table 2: A pipelined version of design $\mathcal{B}$ for the Warshall-Floyd algorithm.

	Pipelined version of design $\mathcal{B}$
π^T	$(\frac{1}{L}, \frac{1}{K}, 1 + \frac{1}{K} + \frac{1}{L})$
Iteration vector u^T	$(1, 0, 0)$
Execution time T	$h(L)((N-1)(\frac{2}{L} + \frac{2}{K} + 1) + \frac{1}{K})$
Latency	T
Period	$h(L)N/L$
Asymptotic execution time[a]	$h(L)N(1 + \frac{2}{L})$
Asymptotic latency[a]	$h(L)N(1 + \frac{2}{L})$
Asymptotic period[a]	$h(L)N/L$
Execution time for $N = 100, L = 2, h(2) = 1.1$ $K = 10, M = 1$	$2.4N$
Latency for $N = 100, L = 2, h(2) = 1.1$ $K = 10, M = 1$	$2.4N$
Period for $N = 100, L = 2, h(2) = 1.1$ $K = 10, M = 1$	$0.55N$

[a] For $K \to N$, and ignoring additive constants.

The MP/C [3], the CHiP computer [52], and the meshes with reconfigurable buses described by Miller et al. [34] all are well suited to implement bounded broadcast. Unlike an SIMD broadcast step, the time to perform a bounded broadcast does not grow as the array grows. Systolic computing systems should make use of bounded broadcast for the following reasons:

- Essentially all systolic algorithms have propagated variables, hence can benefit from bounded broadcast.
- All other things being equal, a systolic computing system that implements an algorithm using bounded broadcast will perform significantly better than one that does not.

Acknowledgements

This work is supported by the National Science Foundation under grant MIP89-20598, and the Lawrence Livermore National Laboratories.

References

[1] Alok Aggarwal. Optimal bounds for finding maximum on an array of processors with ka global buses. *IEEE Trans. on Computers*, 35(1):62–64, January 1986.

[2] Alfred V. Aho, John E. Hopcroft, and Jeffrey. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley Publishing Co, Reading, Mass, 1974.

[3] Bruce W. Arden and Ran Ginosar. MP/C: a multiprocessor/computer architecture. *IEEE Trans. on Comput.*, 31(5):455–473, May 1982.

[4] A. Benaini and Yves Robert. Space-time-minimal systolic arrays for gaussian elimination and the algebraic path problem. *Parallel Computing*, 15:211–225, 1990.

[5] Shahid H. Bokhari. Finding maximum on an array processor with a global bus. *IEEE Trans. on Computers*, 33(2):133–139, February 1984.

[6] Peter R. Cappello. *VLSI Architectures for Digital Signal Processing*. PhD thesis, Princeton University, Princeton, NJ, Oct 1982.

[7] Peter R. Cappello, Andrea LaPaugh, and Kenneth Steiglitz. Optimal choice of intermediate latching to maximize throughput in VLSI circuits. *IEEE Trans. on Acoustic, Speech, and Signal Processing*, ASSP-32(1):28–33, Feb 1984.

[8] Peter R. Cappello and Kenneth Steiglitz. Unifying VLSI array design with linear transformations of space-time. In Franco P. Preparata, editor, *Advances in Computing Research*, volume 2: VLSI theory, pages 23–65. JAI Press, Inc., Greenwich, CT, 1984.

[9] Marina C. Chen. A design methodology for synthesizing parallel algorithms and architectures. *J. of Parallel and Distributed Computing*, pages 461–491, Dec. 1986.

[10] Jean-Marc Delosme. A parallel algorithm for the algebraic path problem. In M. Cosnard et al., editor, *Parallel and Distributed Algorithms*, pages 67–78. North-Holland, 1989.

[11] Jean-Marc Delosme and Ilse Ipsen. Efficient systolic arrays for the solution of toeplitz systems: An illustration of a methodology for the construction of systolic architectures in vlsi. In W. Moore et al., editor, *Systolic Arrays*, pages 37–46. Adam Hilger, 1987.

[12] Jean-Marc Delosme and Ilse C. F. Ipsen. An illustration of a methodology for the construction of efficient systolic architectures in VLSI. In *Proc. 2nd Int. Symp. on VLSI Technology, Systems and Applications*, pages 268–273, Taipei, 1985.

[13] A. L. Fisher and H.-T. Kung. Synchronizing large VLSI processor arrays. *IEEE Trans. Computers*, C-34(8):734–740, August 1985.

[14] R. W. Floyd. Algorithm 97: Shortest path. *Commun. ACM*, 5, June 1962.

[15] José A. B. Fortes and Dan I. Moldovan. Parallelism detection and algorithm transformation techniques useful for VLSI architecture design. *J. Parallel Distrib. Comput*, 2:277–301, Aug. 1985.

[16] M. Gondran, M. Minoux, and S. Vajda. *Graphs and Algorithms*. Wiley, New York, 1984.

[17] C. Guerra and R. Melham. Synthesizing non-uniform systolic designs. In *Proc. Int. Conf. Parallel Processing*, pages 765–772, St. Charles, IL, August 1986.

[18] Leonidas J. Guibas, H.-T. Kung, and Clark D. Thompson. Direct VLSI implementation of combinatorial algorithms. In *Proc. Caltech Conf. on VLSI*, pages 509–525, 1979.

[19] M. Hatamian and G. L. Cash. Parallel bit-level pipelined VLSI designs for high-speed signal processing. *Proc. of the IEEE*, 75(9):1192–1202, Sep. 1987.

[20] C-H Huang and Christian Lengauer. An incremental, mechanical development of systolic solutions to the algebraic path problem. Technical Report 28, University of Texas, Dept. Computer Science, Austin, Dec. 1986.

[21] Guo jie Li and Benjamin W. Wah. The design of optimal systolic algorithms. *IEEE Trans. on Computers*, C-34(1):66–77, 1985.

[22] Richard M. Karp, Richard E. Miller, and Shmuel Winograd. Properties of a model for parallel computations: Determinacy, termination, queueing. *SIAM J. Appl. Math*, 14:1390–1411, 1966.

[23] Richard M. Karp, Richard E. Miller, and Shmuel Winograd. The organization of computations for uniform recurrence equations. *J. ACM*, 14:563–590, 1967.

[24] Steven D. Kugelmass and Kenneth Steiglitz. A probabilistic model for clock skew. In Keith Bromley, Sun-Yuan Kung, and Earl Swartzlander, editors, *Proc. Int. Conf. on Systolic Arrays*, pages 545–554. IEEE Computer Society, May 1988.

[25] V.K. Prasanna Kumar and C. S. Raghavendra. Array processor with multiple broadcasting. *J. of Parallel and Distributed Computing*, pages 173–190, 1987.

[26] Sun-Yuan Kung. *VLSI Array Processors*. Prentice-Hall, Englewood Cliffs, NJ, 1988.

[27] Sun-Yuan Kung and S. C. Lo. A spiral systolic architecture/algorithm for transitive closure problems. In *Proc. IEEE Int. Conf. Comput. Design*, 1985.

[28] Sun-Yuan Kung, Sheng-Chun Lo, and Paul S. Lewis. Optimal systolic design for the transitive closure and the shortest path problems. *IEEE Trans. on Computers*, 36(5):603–614, May 1987.

[29] G. Lakhani and R. Dorairaj. A VLSI implementation of all-pair shortest path problem. In Sartaj K. Sahni, editor, *Proc. Int. Conf. Parallel Processing*, pages 207–209, St. Charles, IL, August 1987.

[30] D. H. Lehman. Algebraic structures for transitive closure. *Theoret. Comput. Sci.*, 4:59–76, 1977.

[31] Paul S. Lewis and Sun-Yuan Kung. Dependence graph based design of systolic arrays for the algebraic path problem. In *Proc. 12th Ann. Asilomar Conf. Signals, Syst., Comput.*, pages 13–18, Nov 1986.

[32] Paul S. Lewis and Sun-Yuan Kung. An optimal systolic array for the algebraic path problem. *IEEE Trans. on Computers*, 40(1):100–105, January 1991.

[33] T. Maeba, S. Tatsumi, and M. Sugaya. Algorithms for finding maximum and selecting median on a processor array with separable global buses. *Electronics and Communications in Japan, Part 3*, 73(6):39–47, 1990.

[34] Russ Miller, Viktor K. Prasanna, D. Reisis, and Quentin F. Stout. Meshes with reconfigurable buses. In *5th MIT Conf. on Advanced Research in VLSI*, pages 163–178, March 1988.

[35] Willard L. Miranker and Andrew Winkler. Spacetime representations of computational structures. *Computing*, 32:93–114, 1984.

[36] Dan I. Moldovan. On the analysis and synthesis of VLSI algorithms. *IEEE Trans. Comput.*, C-31:1121–1126, Nov. 1982.

[37] Dan I. Moldovan. On the design of algorithms for VLSI systolic arrays. *Proc. IEEE*, 71(1):113–120, Jan. 1983.

[38] Dan I. Moldovan and José A. B. Fortes. Partitioning and mapping algorithms into fixed systolic arrays. *IEEE Trans. on Computers*, C-35(1):1–12, Jan. 1986.

[39] Mathew T. O'Keefe and José A. B. Fortes. A comparative study of two systematic design methodologies for systolic arrays. *Proc. Int. Conf. Parallel Processing*, pages 672–675, Aug. 1986.

[40] Patrice Quinton. Automatic synthesis of systolic arrays from uniform recurrent equations. In *Proc. 11th Ann. Symp. on Computer Architecture*, pages 208–214, 1984.

[41] Sanjay V. Rajopadhye. An improved systolic algorithm for the algebraic path problem. *INTEGRATION: The VLSI Journal*, 1992. in press (a preliminary version appears in 'Algorithms and Parallel VLSI Architectures II, Bonas France, June 1991, Elsevier').

[42] I. V. Ramakrishnan, D. S. Fussell, and A. Silberschatz. Mapping homogeneous graphs on linear arrays. *IEEE Trans. Computers*, C-35(3):189–209, Mar. 1986.

[43] Sailesh K. Rao. *Regular Iterative Algorithms and Their Implementation on Processor Arrays*. PhD thesis, Stanford University, October 1985.

[44] Tanguy Risset. A method to synthesize modular systolic arrays with local broadcast facility. In *Proc. Int. Conf. on Application Specific Array Processors*, pages 415–428. IEEE Computer Society Press, Oakland, CA, August 1992.

[45] Tanguy Risset and Yves Robert. Uniform but non-local dags: a trade-off between pure systolic and SIMD solutions. In *Proc. Int. Conf. on Application Specific Array Processors*, pages 296–308. IEEE Computer Society Press, Barcelona, September 1991.

[46] Yves Robert and D. Trystram. Systolic solution of the algebraic path problem. In W. Moore et al., editor, *Systolic Arrays*, pages 171–180. Adam Hilger, 1987.

[47] G. Rote. A systolic array algorithm for the algebraic path problem (shortest paths; matrix inversion). *Computing*, 34(3):191–219, 1985.

[48] G. Rote. On the connection between hexagonal and unidirectional rectangular systolic arrays. In F. Makedon, K. Mehlhorn, T. Papatheodorou, and P. Spirakis, editors, *Lecture Notes in Computer Science 227*, pages 70–83. Springer-Verlag, 1986.

[49] Chris Scheiman and Peter R. Cappello. A processor-time minimal systolic array for transitive closure. *IEEE Trans. on Parallel and Distributed Systems*, 3(3):257–269, May 1992.

[50] Isaac D. Scherson and Y. Ma. Analysis and applications of the orthogonal access multiprocessor. *J. Parallel and Distributed Comput.*, 7(2):232–255, January 1989.

[51] Weijia Shang and José A. B. Fortes. Time optimal linear schedules for algorithms with uniform dependencies. In *Int. Conf. on Systolic Arrays*, pages 393–402, San Diego, May 1988.

[52] Larry Snyder. Introduction to the configurable highly parallel computer. *Computer*, 15(1):47–56, Jan 1982.

[53] Quentin F. Stout. Mesh connected computers with broadcasting. *IEEE Trans. on Computers*, pages 826–830, 1983.

[54] Jeffrey D. Ullman. *Computational Aspects of VLSI.* Computer Science Press, Inc, Rockville, MD 20850, 1984.

[55] S. Warshall. A theorem on boolean matrices. *J. ACM*, 9, Jan 1962.

[56] Yoav Yaacoby. *Computing Systems of Affine Recurrence Equations on Processor Arrays.* PhD thesis, University of California, Santa Barbara, Santa Barbara, CA, May 1988.

[57] Yoav Yaacoby and Peter R. Cappello. Converting affine recurrence equations to quasi-uniform recurrence equations. In John H. Reif, editor, *VLSI Algorithms and Architectures*, pages 319–328. Springer-Verlag, June 1988.

[58] U. Zimmermann. Linear and combinatorial optimization in ordered algebraic structure. *Annals of Discrete Mathematics*, 10, 1981.