Processor-Efficient Sparse Matrix-Vector Multiplication[1]

Lenwood S. Heath[2] Sriram V. Pemmaraju[3] Calvin J. Ribbens[2]

July 2, 2003

Abstract

The matrix-vector multiplication operation is the kernel of most numerical algorithms. Typically, matrix-vector multiplication algorithms exploit the sparsity of a matrix, either to reduce the time taken or the memory used. In the case of parallel implementations of numerical algorithms, the sparsity of a matrix can also be used to reduce the number of processing elements (PEs) required. For example, in Leiserson's systolic array algorithm for matrix-vector multiplication the nonzeros in the matrix are partitioned into bands and each band is fed into a different PE. Similarly, in Melhem's algorithm, the nonzeros are partitioned into stripes and each stripe is fed into a distinct PE. However, in many practical applications, such as the numerical solution of partial differential equations, a matrix is much sparser than is indicated by its bandwidth or by its stripewidth.

A new measure of sparsity of a matrix called *staircase width* is introduced. A *staircase* in a matrix $M = (m_{i,j})_{n \times n}$ is a set of matrix positions $S = \{(i,j) \mid m_{i,j} \neq 0\}$ such that no position in S is strictly to the northeast of another position in S. The *staircase cover* of a matrix is a partition of the positions of nonzero entries in the matrix into staircases and the *staircase width* of a matrix is the size of a smallest staircase cover of the matrix. The staircase width of a matrix is never larger than the bandwidth or the stripewidth of a matrix. Moreover, it is significantly smaller than either for some commonly occurring types of matrices such as arrowhead matrices. Experiments with sparse matrices derived from a variety of engineering problems suggest that, in practice, the staircase width of a matrix is about half the stripe width of the matrix. An efficient algorithm to compute a minimum width staircase cover of a matrix is presented. Staircase covers are used as the basis of a new parallel algorithm to compute a matrix-vector product on a linear systolic array of PEs. This algorithm uses as many PEs as the staircase width of the matrix and therefore saves on the number of PEs relative to algorithms of Leiserson and Melhem. It is shown that despite using fewer PEs, the algorithm uses no more time steps than Leiserson's or Melhem's algorithm.

[1]The authors gratefully acknowledge the support of National Science Foundation Grant CCR-9009953. This research was done while the second author was with the Department of Computer Science, Virginia Polytechnic Institute and State University.

[2]Department of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, VA 24061-0106, {heath, ribbens}@cs.vt.edu

[3]Department of Computer Science, The University of Iowa, Iowa City, IA 52242-1419, sriram@cs.uiowa.edu

Keywords: bandwidth; matrix-vector multiplication; partial orders; sparse matrices; staircase width; stripe width; systolic arrays

1 Introduction

Sparse matrix-vector multiplication is an important computational kernel in most scientific computations. Im and Yelick [1, 2] mention signal and image processing, document retrieval, and data mining as some of the applications in which efficient sparse matrix-vector multiplication is critical. Matrix-vector multiplication is often used in iterative solvers for linear systems, in explicit methods for ordinary differential equations, and in eigenvalue computations [1, 3]. In general, the performance of sparse matrix operations is much lower than their dense matrix counterparts. This is mainly because the memory access patterns are irregular and there is an overhead for manipulating the sparse matrix representation. Sparsity of a matrix is a problem for parallel implementations as well and implementations that work well for dense matrices often expend a large fraction of their computational resources performing unnecessary operations that involve zeros.

Given that the matrices used in most practical applications are sparse, attempts to exploit the sparsity of matrices to speed up matrix computations continue. Exploitation of sparsity typically involves symbolic preprocessing of a matrix based on its sparsity structure [4]. The preprocessing may yield a decomposition of the sparse matrix that can then be processed with a specialized algorithm. Sometimes the decomposition leads to a number of smaller dense matrix subproblems, which can then be solved with a system for dense linear algebra, such as BLAS [5] or LAPACK [5, 6]. Some of the important measures or techniques associated with sparsity are matrix bandwidth [7, 8]; matrix stripewidth [9, 3]; block matrices [10, 11]; the jagged diagonal data structure [12]; frontal and multifrontal methods [13, 14]; and panel clustering [15, 16].

Melhem [9, 3] introduced a systolic array network for performing matrix-vector multiplications and solving triangular systems of equations. In Melhem's scheme, the processing elements (PEs) in the systolic array avoid many unnecessary operations, while the number of PEs utilized is small. The basis of Melhem's scheme is a decomposition of the nonzero elements of the matrix into stripes. Each stripe is fed into a different PE, and therefore Melhem's scheme utilizes only as many PEs as the number of stripes required to cover the matrix. Melhem [3] mentions that, in matrices that arise in finite element analysis, the bandwidth of an $n \times n$ matrix is typically $\Theta(n^{1/2})$ (for 2-D problems) or $\Theta(n^{2/3})$ (for 3-D problems), while the number of stripes required to cover the matrix is bounded above by a constant (independent of n). Therefore, as compared with the scheme described by Leiserson [17], which uses as many PEs as the bandwidth of a matrix, Melhem's scheme uses far fewer PEs in application domains such as finite element analysis.

In this paper, we introduce a new way of imposing structure on sparse matrices, called *staircase covers*. The notion of staircase covers generalizes the notion of stripe covers, used by Melhem. We then present a new matrix-vector multiplication algorithm on systolic arrays that uses as many PEs as the number of staircases required to cover the matrix. We show that the minimum number of staircases required to cover a matrix is never greater than the minimum number of stripes required. Furthermore, in matrices such as arrowhead matrices that arise in various engineering applications, the number of staircases required to cover the matrix is bounded above by a constant, whereas the number of stripes required is $\Theta(n)$ for an $n \times n$ matrix. Section 4 describes experiments in which the sizes of minimum staircase covers and stripe covers are computed for a number of sparse matrices that arise in engineering applications. Experiments indicate that the average number of stripes required to cover a matrix is about two times the number of staircases required. Finally, we show that this saving in the number of PEs required by our scheme is achieved *without* using any additional time steps as compared to Melhem's algorithm.

As in Leiserson's and Melhem's algorithms, the amount of hardware required by our algorithm depends on the sparsity structure of the matrix and this is only known at run-time. There are two reasons why this is not a significant concern. First, reconfigurable VLSI technology [18], especially in the form of field-programmable gate arrays (FPGAs), is capable of being *reconfigured dynamically* to provide the resources necessary to implement a systolic array having any number of virtual processors [19, 20, 21, 22]. FPGAs can even be connected with buses in parallel, so there are few limits on how much hardware can be made available [23, 24]. Second, our scheme is sufficiently flexible that a smaller number of

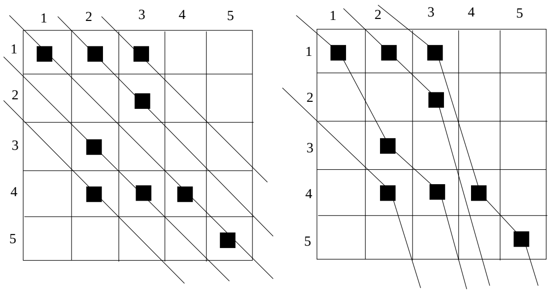

Figure 1: A matrix with bandwidth 5, stripewidth 4, and staircase width 2

PEs can be adapted through partitioning the set of staircases in a staircase cover, feeding each subset into the PEs, and accumulating the results.

2 Preliminaries

Throughout the paper we use $M = (m_{i,j})_{n\times n}$ to denote an $n \times n$ matrix with entries $m_{i,j}$ and use $x = (x_j)_{n\times 1}$ to denote an n-vector with entries x_j. Let $J_n = \{1, 2, \ldots, n\}$ be the set of row (or column) indices of M. Each ordered pair $(i, j) \in J_n^2$ is a *position index* of M. The position index (i, j) is said to *cover* the corresponding entry $m_{i,j}$ of M. Let $\Gamma \subseteq J_n^2$ be the set of position indices that cover nonzero entries of M; that is, $\Gamma = \{(i, j) \in J_n^2 \mid m_{i,j} \neq 0\}$. *Diagonal* k of M is the set of position indices $D_k = \{(i, j) \in J_n^2 \mid j - i = k\}$. The *band* of M is the smallest contiguous set of diagonals $\{D_p, D_{p+1}, \ldots, D_q\}$ that covers Γ; the *bandwidth* $\mathrm{BAND}(M)$ of M is $q - p + 1$, the cardinality of its band. Figure 1 illustrates a pattern of nonzero entries (black squares) in a 5×5 matrix. The matrix has bandwidth 5, and the leftmost drawing shows its nonzero entries covered by 5 bands. Leiserson [17] proposes a systolic array for multiplying a matrix M of bandwidth $b = \mathrm{BAND}(M)$ and a vector x using a linear array of b PEs, requiring $2n + b$ clock beats to accomplish the multiplication.

In many practical applications, such as finite element analysis, M is much sparser than is indicated by its bandwidth. Melhem [3] proposed a systolic network called MAT/VEC to take advantage of such sparsity. To describe MAT/VEC and our scheme in the same framework, we introduce some terminology related to partial orders (see Trotter [25]). A *partial order* is a binary relation that is reflexive, antisymmetric, and transitive. A *partially ordered set* (poset) $P = (S, \leq)$ is a set S with a partial order $\leq$ defined on it. For a pair of elements, $s, s' \in S$, we say $s < s'$ if $s \leq s'$ and $s \neq s'$. A set of elements $\{s_1, s_2, \ldots, s_k\}$ from S such that $s_i \leq s_{i+1}$, where $1 \leq i \leq k-1$, is called a *chain* in P. The *length* of a chain is simply the number of elements in it. An *antichain* in a poset $P = (S, \leq)$ is a subset $S' \subseteq S$ such that no two elements in S' are related by $\leq$. Suppose that $S' \subseteq S$ and $s \in S'$. Then s is a maximal element of S' if, for every $s' \in S'$, either $s' \leq s$ or s and s' are unrelated by $\leq$. Similarly, s is a minimal element of S' if, for every $s' \in S'$, either $s \leq s'$ or s and s' are unrelated by $\leq$.

Define the *strictly southeast poset* $(\Gamma, \leq_{sse})$ by $(i_1, j_1) \leq_{sse} (i_2, j_2)$ if either $(i_1, j_1) = (i_2, j_2)$ or $i_1 < i_2$ and $j_1 < j_2$. A *stripe* $\Upsilon \subseteq \Gamma$ in M is a nonempty chain in the poset $(\Gamma, \leq_{sse})$; stripes are generalizations of diagonals. A *stripe cover* of M is a set of stripes that covers the set of nonzero entries Γ. The *stripewidth* $\mathrm{STRIPE}(M)$ is the minimum cardinality of any stripe cover of M. For example, the matrix in Figure 1 has stripewidth 4, and the middle drawing shows its nonzeros covered by 4 stripes. The nonzero entries $(1,3)$, $(2,3)$, $(3,2)$, and $(4,2)$ have to belong to distinct stripes and hence 4 is the fewest number of stripes required. The optimal stripe cover shown in Figure 1 is:

$$\{\{(1,3),(4,4),(5,5)\},\{(1,2),(2,3)\},\{(1,1),(3,2),(4,3)\},\{(4,2)\}\}.$$

It is easy to see that a band of M is also a stripe cover of M and hence the stripewidth of M is no greater than its bandwidth.

Define the *southeast poset* $(\Gamma, \leq_{se})$ by $(i_1, j_1) \leq_{se} (i_2, j_2)$ if and only if $i_1 \leq i_2$ and $j_1 \leq j_2$. A *staircase* $\Psi \subseteq \Gamma$ in M is a nonempty chain in the poset $(\Gamma, \leq_{se})$. A *staircase cover* of M is a set of staircases that covers Γ. The *staircase width* $\mathrm{STAIR}(M)$ is the minimum cardinality of any staircase cover of M. For example, the matrix in Figure 1 has staircase width 2, and the rightmost drawing shows its nonzeros covered by 2 staircases. The nonzero entries $(3,2)$ and $(2,3)$ have to belong to different staircases and therefore 2 is the fewest number of staircases required. The optimal staircase cover shown in Figure 1 is:

$$\{\{(1,1),(1,2),(1,3),(2,3),(4,3),(4,4),(5,5)\},\{(3,2),(4,2)\}\}.$$

A stripe is also a staircase and hence the staircase width of a matrix is no greater than its stripewidth. To multiply the matrix in Figure 1 with any 5-vector, Leiserson's systolic array requires 5 PEs, Melhem's MAT/VEC requires 4 PEs, while our scheme requires 2 PEs.

This paper's contributions can be summarized as follows:

1. In Section 3, we show how to efficiently compute an optimal staircase cover of a matrix. Our algorithm takes a list of the position indices of m nonzero elements in an $n \times n$ matrix as input and has time complexity $O(m \log \log n)$.

2. In Sections 5 and 6, we present a new systolic array algorithm for matrix-vector multiplication called the *Generalized MAT/VEC*. As MAT/VEC does with stripes, Generalized MAT/VEC feeds the nonzero entries covered by each staircase to a corresponding PE. Thus Generalized MAT/VEC can perform the multiplication with as many PEs as STAIR(M).

3. Since the staircase width of a matrix is never greater than, and in many applications significantly smaller than, the stripewidth of the matrix, Generalized MAT/VEC saves on PEs relative to MAT/VEC. In Section 4, we describe experiments in which we have computed the bandwidth, stripewidth, and staircase width of sparse matrices that arise from a variety of applications, for example, fluid mechanics, financial portfolio optimization, structural engineering, and linear programming. For these matrices, the average value of STRIPE(M) is more than 2 times the average value of STAIR(M).

4. In Section 7, we show that, even though Generalized MAT/VEC saves on PEs as compared to MAT/VEC, it does not require additional time steps. Using Melhem's notion of *global cycles*, we show that Generalized MAT/VEC and MAT/VEC perform the same computations in each global cycle and thus take exactly the same number of global cycles to perform a matrix-vector multiplication (Theorem 8.) Thus Generalized MAT/VEC achieves a savings in hardware without requiring additional time steps.

3 Computing a Staircase Cover

In Section 5, we describe Generalized MAT/VEC and show that its correctness depends on a staircase cover of matrix M in which the staircases are totally ordered in a sense that will be made precise in the following. In this section, we sketch an algorithm that computes an optimal staircase cover of M that is totally ordered as required by Generalized MAT/VEC.

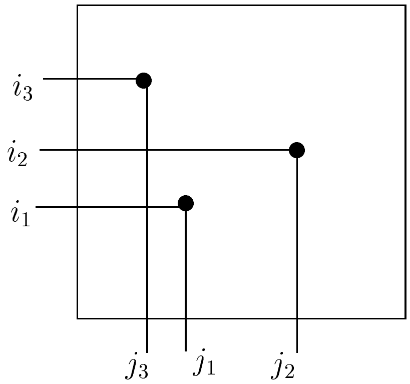

Figure 2: Illustrating the posets $\leq_{ne}$ and $\leq_{se}$. Here $(i_1, j_1) \leq_{ne} (i_2, j_2)$ and $(i_1, j_1) \leq_{sne} (i_2, j_2)$. Also, $(i_3, j_3) \leq_{se} (i_2, j_2)$ and $(i_3, j_3) \leq_{sse} (i_2, j_2)$.

First we present an alternate characterization of the staircase width of a matrix. Then we define a partial order $\leq_\gamma$ on the set of all staircases of M.

Define the *northeast poset* $(\Gamma, \leq_{ne})$ by $(i_1, j_1) \leq_{ne} (i_2, j_2)$ if and only if $i_1 \geq i_2$ and $j_1 \leq j_2$. Define the *strictly northeast poset* $(\Gamma, \leq_{sne})$ by $(i_1, j_1) \leq_{sne} (i_2, j_2)$ if and only if either $(i_1, j_1) = (i_2, j_2)$ or $i_1 > i_2$ and $j_1 < j_2$. Figure 3 illustrates the various posets we have defined. An *antistaircase* in M is a chain in the poset $(\Gamma, \leq_{sne})$. Using the following well-known theorem of Dilworth, we obtain an alternate characterization of the staircase width of M.

Theorem 1 (Dilworth [26]) *Let $P = (S, \leq)$ be a finite poset. The minimum number of chains required to cover all the elements of S equals the size of the largest antichain in P.*

The following corollary to Dilworth's theorem provides a characterization of the staircase width of a matrix.

Lemma 2 *The size of the largest antistaircase in Γ equals* $\mathrm{STAIR}(M)$.

Proof: We first claim that a subset of Γ is an antistaircase in M if and only if it is an antichain in the poset $(\Gamma, \leq_{se})$. To see this, consider an arbitrary antistaircase in M, $\{(i_1, j_1), (i_2, j_2), \ldots, (i_k, j_k)\}$. By definition, this is a chain in $\leq_{sne}$ and therefore, without loss of generality we have

$$i_1 > i_2 > \cdots > i_k \qquad \text{and} \qquad j_1 < j_2 < \ldots j_k.$$

From this it follows that for any pair of position indices (i_p, j_p) and (i_q, j_q), $(i_p, j_p) \not\leq_{se} (i_q, j_q)$ and $(i_q, j_q) \not\leq_{se} (i_p, j_p)$. Therefore, $\{(i_1, j_1), (i_2, j_2), \ldots, (i_k, j_k)\}$ is an antichain in the poset $(\Gamma, \leq_{se})$. The argument that proves the converse is similar.

Thus, a largest antistaircase in Γ is a largest antichain in the poset $(\Gamma, \leq_{se})$. STAIR(M) is the minimum number of chains in the poset $(\Gamma, \leq_{se})$ required to cover Γ. By Theorem 1 the lemma follows. $\square$

For the matrix shown in Figure 1 there are several largest antistaircases; $\{(3, 2), (2, 3)\}$ and $\{(3, 2), (1, 3)\}$ are two examples. From the lemma, it follows that the staircase width of the matrix is 2.

We now define binary relations $\leq_\gamma$ and $<_\gamma$ on the set of all staircases in M. For a pair of staircases Ψ_1 and Ψ_2 in M, $\Psi_2 <_\gamma \Psi_1$ if both $\Psi_1 \cap \Psi_2 = \emptyset$ and, for every position index $(i_2, j_2) \in \Psi_2$, there is a position index $(i_1, j_1) \in \Psi_1$ such that $(i_2, j_2) <_{sne} (i_1, j_1)$. Define $\Psi_2 \leq_\gamma \Psi_1$, if either $\Psi_2 <_\gamma \Psi_1$ or $\Psi_2 = \Psi_1$. As we show in the following lemma, $\leq_\gamma$ defines a partial order on the set of all staircases in M.

Lemma 3 *$\leq_\gamma$ is a partial order on the set of all staircases of M.*

Proof: By definition, $\leq_\gamma$ is reflexive. It suffices to show that $<_\gamma$ is antisymmetric and transitive. To show that $<_\gamma$ is antisymmetric, consider staircases Ψ_1, Ψ_2 such that $\Psi_2 <_\gamma \Psi_1$. Suppose $(i_2, j_2) \in \Psi_2$. By definition of $<_\gamma$, there exists $(i_1, j_1) \in \Psi_1$ such that $(i_2, j_2) <_{sne} (i_1, j_1)$. Now suppose $\Psi_1 <_\gamma \Psi_2$. Then, there exists $(i_3, j_3) \in \Psi_2$ such that $(i_1, j_1) <_{sne} (i_3, j_3)$. By transitivity of $<_{sne}$, we have $(i_2, j_2) <_{sne} (i_3, j_3)$. But, this is a contradiction to (i_2, j_2) and (i_3, j_3) belonging to the same staircase.

To show that $<_\gamma$ is transitive, consider staircases Ψ_1, Ψ_2, and Ψ_3 such that $\Psi_3 <_\gamma \Psi_2$ and $\Psi_2 <_\gamma \Psi_1$. Suppose $(i_3, j_3) \in \Psi_3$. By definition of $<_\gamma$, there exists $(i_2, j_2) \in \Psi_2$ such that $(i_3, j_3) <_{sne} (i_2, j_2)$. Also by definition, there exists $(i_1, j_1) \in \Psi_1$ such that $(i_2, j_2) <_{sne} (i_1, j_1)$. Transitivity of $<_{sne}$ implies that $(i_3, j_3) <_{sne} (i_1, j_1)$. Since (i_3, j_3) was arbitrary, we conclude that for every $(i_3, j_3) \in \Psi_3$ there exists $(i_1, j_1) \in \Psi_1$ such that $(i_3, j_3) <_{sne} (i_1, j_1)$. Hence, $\Psi_3 <_\gamma \Psi_1$, as desired. $\square$

For the purposes of Generalized MAT/VEC, we are interested in a staircase cover of M that is totally ordered by $\leq_\gamma$. The proof of the following theorem includes an algorithm for

$S_0 \ := \ \Gamma;$
$\ell \ := \ 1;$
while $(S_{\ell-1} \neq \emptyset)$ **do**
  let Ψ_ℓ be the set of maximal elements in $(S_{\ell-1}, \leq_{sne})$;
  $S_\ell \ := \ S_{\ell-1} - \Psi_\ell;$
  $\ell \ := \ \ell + 1;$
endwhile;
return $\Psi_1, \Psi_2, \ldots, \Psi_{\ell-1}$

Figure 3: Staircase Cover Algorithm (SCA)

constructing such a staircase cover of M.

Theorem 4 *Every matrix M has a staircase cover consisting of* STAIR(M) *staircases that are totally ordered by $\leq_\gamma$.*

Proof: The proof uses an algorithm that we call the **Staircase Cover Algorithm (SCA)** (see Figure 3). SCA takes a matrix M as input and returns a set of nonempty sets of position indices:

$$\Pi = \{\Psi_1, \Psi_2, \ldots, \Psi_k\}.$$

We show that Π constitutes an optimal staircase cover of M and that $\Psi_\ell <_\gamma \Psi_{\ell-1}$ for all ℓ, $2 \leq \ell \leq k$. These properties are established in the following three steps.

1. **Π is a staircase cover of M.** By definition of maximal elements, if $(i_1, j_1), (i_2, j_2) \in \Psi_\ell$ are distinct position indices, then neither $(i_1, j_1) \leq_{sne} (i_2, j_2)$ nor $(i_2, j_2) \leq_{sne} (i_1, j_1)$. Hence either $(i_1, j_1) \leq_{se} (i_2, j_2)$ or $(i_2, j_2) \leq_{se} (i_1, j_1)$. As every pair of elements of Ψ_ℓ are related by $\leq_{se}$, we have that Ψ_ℓ is a chain in $(\Gamma, \leq_{se})$. Thus Ψ_ℓ is a staircase in M. After ℓ times through the while-do loop in SCA,

$$\Gamma = \Psi_1 \cup \Psi_2 \cup \cdots \cup \Psi_\ell \cup S_\ell.$$

   When SCA terminates, $S_k = \emptyset$. Hence, $\{\Psi_1, \Psi_2, \ldots, \Psi_k\}$ is a staircase cover of M.

2. **Π is an optimal staircase cover.** For an arbitrary element $(i_k, j_k) \in \Psi_k$, there exists at least one element $(i_{k-1}, j_{k-1}) \in \Psi_{k-1}$ such that $(i_k, j_k) <_{sne} (i_{k-1}, j_{k-1})$. Otherwise, (i_k, j_k) would have been included in Ψ_{k-1}. Extending this argument inductively, it is easy to see that there exists a sequence of position indices

$$(i_k, j_k) <_{sne} (i_{k-1}, j_{k-1}) <_{sne} \cdots <_{sne} (i_1, j_1)$$

such that $(i_\ell, j_\ell) \in \Psi_\ell \subseteq \Gamma$ for all ℓ, $1 \leq \ell \leq k$. This implies that there is an antistaircase of size k in Γ. Applying Lemma 2, we conclude that $\{\Psi_1, \Psi_2, \ldots, \Psi_k\}$ is an optimal staircase cover of M.

3. $\leq_\gamma$ **is a total order on** Π**.** Consider an arbitrary pair of staircases Ψ_r and Ψ_{r+1} in Π, where $1 \leq r \leq k-1$. As mentioned in the proof of property 2, for any position index $(i,j) \in \Psi_{r+1}$, there exists a position index $(i',j') \in \Psi_r$ such that $(i,j) <_{sne} (i',j')$. Hence, $\Psi_{r+1} <_\gamma \Psi_r$.

□

The application of SCA to the nonzeros of Figure 1 yields these two staircases —

$$\begin{aligned} \Psi_1 &= \{(1,1),(1,2),(1,3),(2,3),(4,3),(4,4),(5,5)\} \\ \Psi_2 &= \{(3,2),(4,2)\} \end{aligned}$$

— also illustrated in the rightmost picture in Figure 1. Note that Ψ_1 consists of the maximal elements of $(\Gamma, \leq_{sne})$, as for each $(i,j) \in \Psi_1$, there is no $(i',j') \in \Gamma$ that is strictly north and east of it. Note that there are other optimal staircase covers of the matrix in Figure 1 that are not totally ordered by $\leq_\gamma$. One example of such a staircase cover is

$$\{\{(1,1),(1,2),(1,3),(2,3)\},\{(3,2),(4,2),(4,3),(4,4),(5,5)\}\}.$$

The notion of a staircase in a matrix is equivalent to the notion of a queue layout of a graph [27, 28, 29, 30, 31]. In particular, the queue layout of a graph corresponds to a staircase cover of the adjacency matrix of the graph and finding the minimum number of queues for a particular layout of a graph is equivalent to computing the staircase width of the corresponding adjacency matrix. [31] presents an algorithm for determining the fewest queues required for a graph layout that runs in $O(m \log\log n)$ time on a graph with n vertices and m edges. With minor modifications, this algorithm can also be used to implement SCA in $O(|\Gamma| \log\log n)$ time on an $n \times n$ matrix, where Γ is the set of positions of nonzero entries in the matrix. The assumption here is that Γ is the input to the algorithm.

In the context of our application of staircase covers to matrix-vector multiplication on systolic arrays, a *systolic algorithm* to compute a minimum width staircase cover of a matrix may be useful. This is beyond the scope of the current paper, but we suspect that techniques traditionally used for sorting on systolic arrays (see [32] for an example) will be useful in computing staircase covers as well.

4 Comparing Bandwidth, Stripewidth, and Staircase width

As mentioned in Section 2, for any matrix M, the following inequality holds:

$$\mathrm{BAND}(M) \geq \mathrm{STRIPE}(M) \geq \mathrm{STAIR}(M).$$

It is of course good news for Generalized MAT/VEC that $\mathrm{STRIPE}(M) \geq \mathrm{STAIR}(M)$ for all matrices M, but the important question is how much smaller is the staircase width as compared to stripe width for the kinds of sparse matrices that arise in real applications. In this section, we describe experiments that show that for sparse matrices derived for a variety of applications, we can expect $\mathrm{STRIPE}(M)$ to be about twice the size of $\mathrm{STAIR}(M)$. Furthermore, these experiments show there are matrices that arise in real applications for which the staircase width is dramatically smaller than the stripewidth.

For our experiments we downloaded 25 real, non-symmetric matrices from the University of Florida sparse matrix collection, which currently contains 926 sparse matrices that have arisen in a variety of real applications[1]. These matrices are all stored in the popular Harwell/Boeing format. To complete our experiments in a reasonable amount of time, we restricted our sample to matrices with 5000 or fewer rows and columns. The matrices in the University of Florida collection are organized into directories, each directory containing matrices submitted by a research group. Roughly speaking, this organization partitions matrices into directories by problem domain. Therefore, to prevent any bias and to get matrices from a wide variety of problem domains, we considered the matrix directories in alphabetical order and selected up to 2 matrices satisfying our criteria (real, non-symmetric, number of rows, columns ≤ 5000) from each directory. Within each matrix directory, we selected the first two matrices in alphabetical order that satisfy our criteria. (While our programs to compute bandwidth, stripewidth, and staircase width work for non-square matrices also, all the matrices selected in this manner,

[1] This collection can be found at `http://www.cise.ufl.edu/research/sparse/matrices/`.

turned out to be square.) Our programs, data files, and documentation are all available at `http://www.cs.uiowa.edu/~sriram/sparseMatrices/experiments.html/`.

The results of our experiments are shown in the table in Figure 4. The second column gives the directory and file name of the matrix, as found at

`ftp://ftp.cise.ufl.edu/pub/faculty/davis/matrices/.`

The average sparsity of a matrix is more than 96%, so these matrices are quite sparse. The last three columns show the staircase width (SCW), stripe width (SW), and bandwidth (BW) computed by the programs available at the URL given above.

It is easy to construct examples of $n \times n$ matrices whose bandwidth is $\Theta(n)$ and whose stripewidth is $O(1)$. For example, consider the $n \times n$ matrix M, with even n, such that

$$\Gamma = \{(1, 1), (2, 3), (3, 5), \ldots , (n/2, n - 1)\}.$$

Then, STRIPE(M) $=$ 1 but the bandwidth of M is $n/2$. More to the point, Melhem mentions that in matrices that arise in finite element analysis, the bandwidth of an $n \times n$ matrix is typically $\Theta(n^{1/2})$ (for 2-D problems) or $\Theta(n^{2/3})$ (for 3-D problems), while the number of stripes required to cover the matrix is bounded above by a constant (independent of n). These observations are supported by our experiments. In the table in Figure 4, the average stripewidth is 187.92, while the average bandwidth is 2047.46. For several matrices in the table, the difference between the stripewidth and bandwidth is even more dramatic. For example, the average bandwidth of the three matrices `garon1.rua` (Row 8), `add32.rua` (Row 14), and `swang1.rua` (Row 22) is 6843, more than 60 times the average stripewidth, which is 106.3. The matrices `garon1.rua` and `swang1.rua` arise from 2-D and 3-D finite element discretization of the Navier-Stokes equation, whereas `add32.rua` arises in digital circuit simulation. Figure 5 shows the patterns of nonzeros in these matrices, and these patterns explain why it requires many bands to cover the nonzeros but very few stripes.

The difference in the average values of stripewidth and staircase width is not as dramatic as the difference between the stripewidth and bandwidth. Nevertheless, the average stripewidth is about twice the average staircase width. The matrix `arc130.rua` is an example in which the stripewidth is dramatically larger than the staircase width. Figure 6 shows the pattern of nonzeros in this matrix and from the picture it is clear that the nonzeros have

	File name	Dimensions	Nonzeros	Sparsity	SCW	SW	BW
1	Bai/bfwa398.rua	398 × 398	3678	97.68	20	32	609
2	Bai/bfwa62.rua	62 × 62	450	88.29	15	28	99
3	Brethour/coater1.rua	1348 × 1348	19457	98.93	57	137	869
4	DRIVCAV/cavity01.rua	317 × 317	7327	92.71	33	105	205
5	DRIVCAV/cavity02.rua	317 × 317	7327	92.71	33	108	205
6	FIDAP/ex1.rua	216 × 216	4352	90.67	34	79	129
7	FIDAP/ex10.rua	2410 × 2410	54840	99.06	33	53	227
8	Garon/garon1.rua	3175 × 3175	88927	99.12	60	127	6343
9	Grund/b1_ss.rua	7 × 7	15	69.39	3	4	10
10	Grund/b2_ss.rua	1089 × 1089	4228	99.64	260	387	2017
11	HB/arc130.rua	130 × 130	1282	92.41	23	206	251
12	HB/bp_0.rua	822 × 822	3276	99.52	41	267	1337
13	Hamm/add20.rua	2395 × 2395	17319	99.70	86	232	4627
14	Hamm/add32.rua	4960 × 4960	23884	99.90	32	84	8059
15	Mallya/lhr01.rua	1477 × 1477	18592	99.15	72	223	2537
16	Mallya/lhr02.rua	2954 × 2954	37206	99.58	72	223	2537
17	Shyy/shyy41.rua	4720 × 4720	20042	99.91	4	6	81
18	Simon/raefsky1.rua	3242 × 3242	294276	97.20	111	320	871
19	Simon/raefsky2.rua	3242 × 3242	294276	97.20	111	320	871
20	TOKAMAK/utm1700b.rua	1700 × 1700	21509	99.26	24	43	512
21	TOKAMAK/utm300.rua	300 × 300	3155	96.50	23	41	141
22	Wang/swang1.rua	3169 × 3169	20841	99.79	58	108	6127
23	Wang/swang2.rua	3169 × 3169	20841	99.79	58	108	6127
24	Zitney/extr1.rua	2837 × 2837	11407	99.86	913	1136	5318
25	Zitney/radfr1.rua	1048 × 1048	13299	98.79	15	33	1075
	Means			96.27	87.64	187.92	2047.36
	Medians			99.06	34	108	871

Figure 4: The staircase width (SCW), stripe width (SW), and the bandwidth (BW) of 25 real, non-symmetric matrices is shown here. These matrices were downloaded from the University of Florida sparse matrix collection.

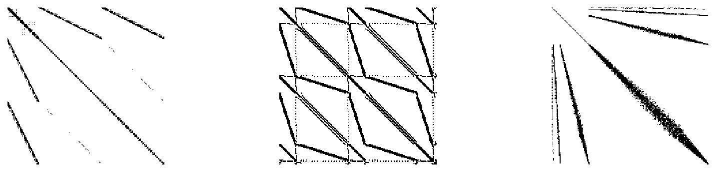

Figure 5: The pattern of nonzeros in the matrices `garon1.rua` (3175×3175), `add32.rua` (4960×4960), and `swang1.rua` (3169×3169).

an arrowhead pattern. Matrices with this nonzero pattern occur frequently in the numerical solution to partial differential equations and there are papers that focus on efficient matrix operations for arrowhead matrices [33].

5 Generalized MAT/VEC

The problem at hand is to multiply an $n \times n$ matrix $M = (m_{i,j})_{n\times n}$ and an n-vector $x = (x_j)_{n\times 1}$ to produce as result, an n-vector $y = (y_i)_{n\times 1}$. Let $\{\Psi_1, \Psi_2, \ldots, \Psi_k\}$ be the staircase cover of M constructed by SCA (described in Section 3). From the proof of Theorem 4, it follows that $\Psi_\ell <_\gamma \Psi_{\ell-1}$ for all ℓ, $2 \leq \ell \leq k$. As Generalized MAT/VEC is described, it will become clear that for our algorithm to work the staircases have to be totally ordered by $<_\gamma$. Generalized MAT/VEC uses k PEs, each PE having six input ports, labeled I_1, I_2, I_3, I_4, I_5 and I_6, and two output ports, labeled O_1 and O_2. A typical PE in Generalized MAT/VEC is shown in Figure 7(a). The k PEs are labeled 1 through k and

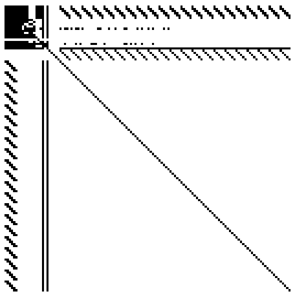

Figure 6: The pattern of nonzeros in the matrix `arc130.rua` (130×130), revealing an arrowhead pattern.

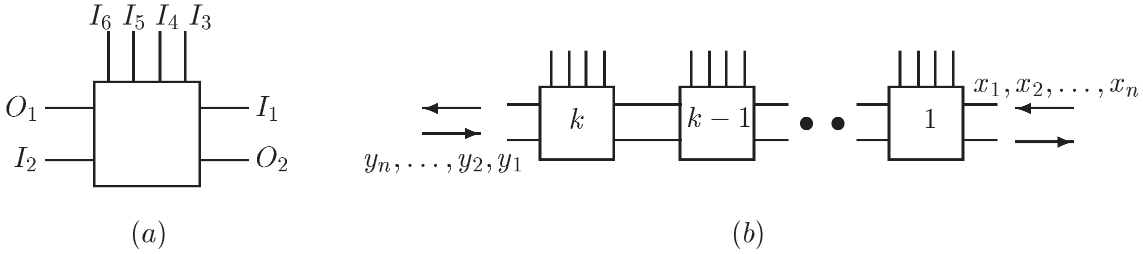

Figure 7: Sparse Matrix-Vector Multiplication Systolic Network.

are connected in a linear array as shown in Figure 7(b). The components of the x-vector are fed from the right of Generalized MAT/VEC through input port I_1 of the PE labeled 1, in the order $x_1, x_2, \ldots, x_n$. The components of the y-vector, initialized to 0, are fed from the left of Generalized MAT/VEC, through input port I_2 of the PE labeled k, in the order $y_1, y_2, \ldots, y_n$. The product $M \cdot x$ is accumulated in y, that is, when y_i exits PE 1, it contains the value $\sum_{j=1}^{n} m_{i,j} x_j$. The elements of each staircase Ψ_ℓ, where, $1 \le \ell \le k$, are fed into the PE labeled ℓ in increasing order of $\le_{se}$. In particular, if $(i, j) \in \Psi_\ell$, then $m_{i,j}$ is fed into input port I_3, i is fed into input port I_5, and j is fed into input port I_4 of the PE labeled ℓ. Thus, along with an entry $m_{i,j}$ of M, the row index i and the column index j are also fed to the PE. Since a staircase is fed into a PE in increasing order of $\le_{se}$, the sequence of values

Step 1: $R_m := [I_3]$; $R_j := [I_4]$; $R_i := [I_5]$;
Step 2: **While** $(C_x < R_j)$ **or** $(C_y < R_i)$ **do**
Step 2.1 **if** (**not** empty(I_1)) **and** $(C_x < R_j)$ **then**
$\{R_x := [I_1]; C_x := C_x + 1;$
if $(C_x < R_j)$ **then** $O_1 := R_x;\}$
Step 2.2 **if** (**not** empty(I_2)) **and** $(C_y < R_i)$ **then**
$\{R_y := [I_2]; C_y := C_y + 1;$
if $(C_y < R_i)$ **then** $O_2 := R_y;\}$
Step 3: $R_y := R_y + R_m \cdot R_x$
Step 4: **if** $([I_6] = h)$ **then** $\{O_1 := R_x\}$
else if $([I_6] = v)$ **then** $\{O_2 := R_y\}$
else $\{O_1 := R_x; O_2 := R_y\}$

Figure 8: Program for each PE

fed into port I_4 (the j values) and the sequence of values fed into port I_5 (the i values) are both non-decreasing. Once PE ℓ has $m_{i,j}$ on its input port I_3, it then waits for x_j and y_i to appear on input ports I_1 and I_2 respectively and when they do appear, it updates y_i according to $y_i := y_i + m_{i,j} \cdot x_j$. The value fed into a PE via input port I_6 depends on the relationship between the current input on I_3 and the next input on I_3. If the next input on I_3 is a matrix element that is in the same row as the current input, then I_6 has the value h (horizontal). If the next input on I_3 is a matrix element that is in the same column as the current input, then I_6 has the value v (vertical). Otherwise I_6 has the value d (diagonal).

Each PE in the network executes the program shown in Figure 8 repeatedly until $M \cdot x$ has been computed. Each PE has five internal registers, R_m, R_i, R_j, R_x, and R_y and two counters C_x and C_y. The counters C_x and C_y are initialized to 0 and represent the index of the most recently received components of the x-vector and y-vector respectively. Consider a PE labeled ℓ for some ℓ, $1 \leq \ell \leq k$. After receiving $m_{i,j}$, i, and j into registers R_m, R_i, and R_j respectively (Step 1), the PE waits for the appropriate components of the x-vector (x_j) and the y-vector (y_i) to appear on input ports I_1 and I_2 respectively (Step 2). More precisely, the PE first checks input port I_1 (Step 2.1) and then checks input port I_2 (Step 2.2) for input. Step 2.1 can be explained as follows. If the PE has not already received the correct component of the x-vector, that is, if $C_x < R_j$, and if there is some input, on input port I_1, then the input is copied into register R_x and the corresponding counter C_x is incremented. If C_x is still smaller than R_j, then the PE has no use for the component of the x-vector just received and therefore this component is moved to the output port O_1. Step 2.2 is similar to Step 2.1, except that a component of the y-vector is received as input. Step 2 terminates when $C_x = R_j$ and $C_y = R_i$. Thus, for any PE, C_x (respectively, C_y) keep track of the number of components of x (respectively, y) that have reached the PE. Hence, when Step 2 is completed the correct components of the x-vector and y-vector have been received. On receiving x_j and y_i, the PE updates y_i according to $y_i := y_i + m_{i,j} \cdot x_j$ (Step 3). In Step 4, the PE decides whether to retain either x_j or y_i for future use. In particular, if the next element in the staircase is in the same row as the current element, that is, $[I_6] = h$, then y_i is retained and x_j is passed on to the left neighbor (PE $\ell+1$). If the next element in the staircase is in the same column as the current element, that is, $[I_6] = v$, then x_j is retained and y_i is passed on to the right neighbor (PE $\ell - 1$). If neither is true ($[I_6] = d$), then both x_j and y_i are passed on to the left and right neighbor respectively.

The main difference between the program executed by each PE in Generalized MAT/VEC and the program executed by each PE in MAT/VEC is in Step 4. In MAT/VEC, each stripe in a stripe cover is fed into a distinct PE and after y_i is updated (as in Step 3), both x_j and y_i are passed on to the neighboring PEs, while in Generalized MAT/VEC whether x_j or y_i (or both) is (are) passed on to the neighboring PE(s) depends on whether the next matrix element to be fed into the PE is in the same row or column as the current one. Note that Generalized MAT/VEC is a generalization of MAT/VEC in the sense that a stripe may be fed into a PE and if so I_6 will always have the value d.

Each connection between a pair of adjacent PEs can be thought of as a queue with capacity n. The reader may be concerned about this requirement; specifically, that the queue capacity grows linearly in the array size. Section 8 addresses this concern by explaining how a staircase cover can be constructed that accommodates a constant upper bound on queue capacities. For example, the connection between port O_1 of PE ℓ and port I_1 of PE $\ell + 1$ is a queue into which components of the x-vector are enqueued by PE ℓ and dequeued by PE $\ell + 1$. The net result is that each PE has access to four queues: two queues (at input ports I_1 and I_2) that it can dequeue elements from and two queues (at the output ports O_1 and

O_2) that it can enqueue into. For the purpose of our analysis we give these queues names. The input queue for PE ℓ that contains components of the x-vector (respectively, y-vector) will be called $IQ_x(\ell)$ (respectively, $IQ_y(\ell)$). Accordingly, the output queues will be called $OQ_x(\ell)$ and $OQ_y(\ell)$. Note that $IQ_x(\ell+1)$ and $OQ_x(\ell)$ are identical and likewise $IQ_y(\ell)$ and $OQ_y(\ell+1)$ are identical for all ℓ, $1 \leq \ell \leq k-1$. For notational convenience, we will use $IQ_x(k+1)$ as an alternate name for $OQ_x(k)$ and $IQ_y(0)$ as an alternate name for $OQ_y(1)$.

Since Generalized MAT/VEC is systolic, the PEs execute their programs at different speeds depending upon the availability of data. To analyze the number of time steps required by Generalized MAT/VEC, we focus on the slowest PE. To be precise, we introduce a hypothetical global synchronization scheme as suggested by Melhem. Replace Step 3 in Figure 8 by the following:

Step 3: Wait for a synchronization signal SYNC;

$$R_y \;:=\; R_y + R_m \cdot R_x;$$

The purpose of the synchronization signal SYNC is to force the execution of Generalized MAT/VEC into two alternating phases: a *communications phase* and a *processing phase*. During the communications phase data flows through the network until each PE is either waiting for data (in Step 2), or waiting for SYNC (in Step 3). We assume that all the PEs are connected to a hypothetical controller that issues the signal SYNC after detecting the termination of the communications phase. At the instant SYNC is received, all PEs that are waiting in Step 3 perform the computation $R_y \;:=\; R_y + R_m \cdot R_x$, while the other PEs remain idle. This is the processing phase. A communications phase followed by a processing phase is called a *global cycle* of the network. We measure the time taken by Generalized MAT/VEC to compute $M \cdot x$ by the number of global cycles required. We denote by CP_t and PP_t, the communications phase and the processing phase, respectively, in global cycle t. Note that the above synchronization scheme is hypothetical and simply serves as a way of defining a measure on the amount of time required for Generalized MAT/VEC to compute the matrix-vector product. Furthermore, since this is the same measure of time used by Melhem [3], the comparison between the two schemes is made on the same basis.

Figures 9, 10, and 11 illustrate how Generalized MAT/VEC performs a matrix-vector multiplication. Figure 9 illustrates a 5×5 matrix whose staircase width is 3. The nonzero

		1	1	
2	2	2	1	
	3		1	
	3	2		1

Figure 9: A matrix with staircase width 3.

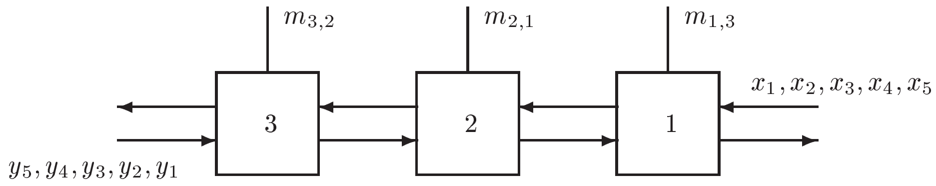

Figure 10: Initial status of Generalized MAT/VEC.

elements covered by Ψ_1 are labeled 1, those covered by Ψ_2 are labeled 2, and those covered by Ψ_3 are labeled 3. Note that $\Psi_3 <_\gamma \Psi_2 <_\gamma \Psi_1$. Figure 10 shows the status of Generalized MAT/VEC initially and Figure 11 shows the 6 global cycles that Generalized MAT/VEC goes through in order to perform the matrix-vector multiplication. For each global cycle, the status of Generalized MAT/VEC after the communications phase and the computations performed during the corresponding processing phase are shown. Notice that during global cycle 1, components x_1 and x_2 are both placed in queue $IQ_x(2)$. It is easy to see that the stripewidth of the matrix shown in Figure 9 is 6. Thus MAT/VEC uses 6 PEs and the reader is invited to verify that MAT/VEC performs the same computations as Generalized MAT/VEC, in each processing phase.

6 Correctness of Generalized MAT/VEC

The proof of correctness of Generalized MAT/VEC depends fundamentally on two properties of our scheme that are established below (Lemma 5 and Lemma 6). The first property is that, for all i, y_i does exit Generalized MAT/VEC. In other words, Generalized MAT/VEC

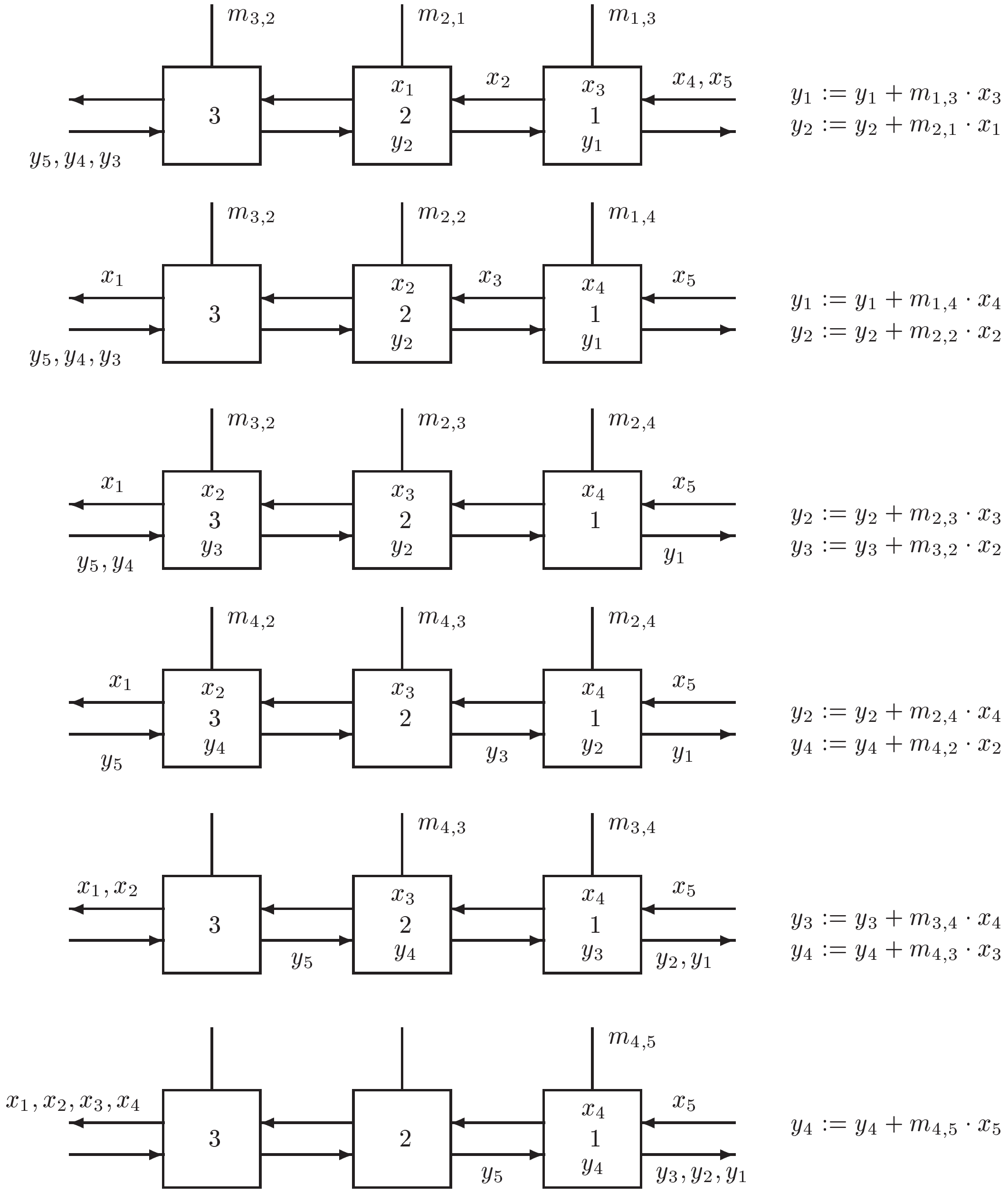

Figure 11: The computations performed by Generalized MAT/VEC.

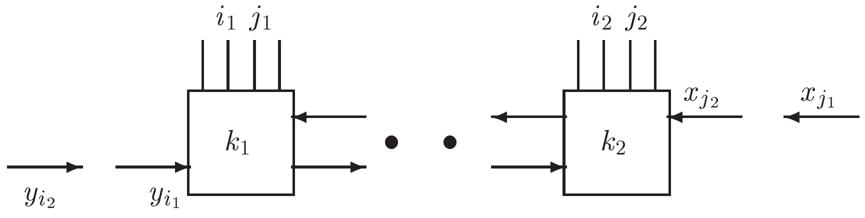

Figure 12: Generalized MAT/VEC in a deadlock state

does not deadlock. In order to prove this we first need to formalize the notion of a deadlock. Generalized MAT/VEC is said to be in a *deadlock* if for some PE k_1, $[I_4] = j_1$, $[I_5] = i_1$, and for some PE k_2, $[I_4] = j_2$ and $[I_5] = i_2$ such that $k_1 > k_2$, $i_1 \leq i_2$, and $j_1 \geq j_2$. The implication of the above condition is that PE k_1 is waiting for x_{j_1} which is blocked by PE k_2, which in turn is waiting for y_{i_2}, which is blocked by PE k_1. A deadlock state is illustrated in Figure 12. The fact that staircases are totally ordered according to $\leq_\gamma$ is crucial in ensuring that Generalized MAT/VEC does not deadlock.

Lemma 5 *Generalized MAT/VEC does not reach a deadlock.*

Proof: To show this, we assume that Generalized MAT/VEC reaches a deadlock state and obtain a contradiction. Because $\Psi_{k_1} <_\gamma \Psi_{k_2}$, it follows from the definition of $<_\gamma$ that $(i_1, j_1) <_{sne} (i_3, j_3)$ for some $(i_3, j_3) \in \Psi_{k_2}$. Since, $i_1 \leq i_2$ and $j_1 \geq j_2$, $(i_2, j_2) \leq_{sne} (i_1, j_1)$. By, transitivity of $\leq_{sne}$ we have $(i_2, j_2) <_{sne} (i_3, j_3)$, which is a contradiction because (i_2, j_2) and (i_3, j_3) belong to the same staircase. $\square$

We now know that all y_i's exit the systolic array; we do not know that each y_i has the correct value. To establish this we show a second property of our algorithm, which is that, when $m_{i,j}$ reaches a PE, x_j and y_i have not traveled past the PE and therefore will participate in the computation $y_i := y_i + m_{i,j} \cdot x_j$ eventually. This implies that when y_i exits the system, it contains the value $\sum_{j=1}^{n} m_{i,j} \cdot x_j$.

Lemma 6 *For any PE ℓ, $1 \leq \ell \leq k$, just before the execution of Step 2 in the program shown in Figure 8, the following holds:*

$$C_x \leq R_j \quad \textit{and} \quad C_y \leq R_i.$$

Proof: The proof is by induction on the number of times Step 2 has already been executed by PE ℓ.

Base Case: Suppose that Step 2 has been executed 0 times by PE ℓ. Due to initialization, $C_x = 0$ and $C_y = 0$. After Step 1 has been executed, R_i and R_j acquire values in J_n. Thus, after Step 1 has been executed and just before Step 2 is executed the first time, it is the case that $C_x < R_j$ and $C_y < R_i$.

Induction Case: Assume that for some integer $p \geq 0$, the lemma holds after Step 2 has been executed p times and just before it is executed for the $(p+1)^{th}$ time. We now show if PE ℓ executes Step 2 for the $(p+2)^{nd}$ time, then just before this execution of Step 2, the lemma 2 holds. Two cases are possible depending on whether the condition in Step 2 evaluates to true or false when Step 2 is executed for the $(p+1)^{th}$ time.

Case 1: Suppose that the condition $(C_x < R_j)$ **or** $(C_y < R_i)$ evaluates to true when Step 2 is executed the $(p+1)^{th}$ time. In the body of the loop (Steps 2.1 and 2.2) C_x is incremented by at most one, if $C_x < R_j$ and C_y is incremented by at most one, if $C_y < R_i$. In any case, just before Step 2 is executed for the $(p+2)^{nd}$ time, we have $C_x \leq R_j$ and $C_y \leq R_i$.

Case 2: Suppose that the condition $(C_x < R_j)$ **or** $(C_y < R_i)$ evaluates to false when Step 2 is executed the $(p+1)^{th}$ time. By the induction hypothesis, we have $C_x = R_j$ and $C_y = R_i$. The steps executed immediately after Step2 are Steps 3 and 4 and these steps do not change the value of C_x and C_y. When the PE finishes the execution of Step 4, it begins again at Step 1. If there is more input on I_3, then in Step 1, R_i and R_j get new values, but these new values are greater than or equal to the corresponding old values. This is because the elements fed into PE ℓ belong to Ψ_ℓ and are fed in increasing order according to $\leq_{se}$. Hence, just before Step 2 is executed for the $(p+2)^{nd}$ time $C_x \leq R_j$ and $C_y \leq R_i$.

By induction, the lemma follows. $\square$

It is easy to see that for any PE ℓ, $1 \leq \ell \leq k$, C_x is the number of components of the vector x that have reached the PE. Similarly, it is easy to see that C_y is the number of components of the vector y that have reached the PE. Thus Lemma 6 implies that when

$m_{i,j}$ reached PE ℓ, the components x_j and y_i have not already traveled past the PE. Since, Lemma 5 shows that Generalized MAT/VEC does not deadlock, eventually the components x_j and y_i will reach PE and the computation $y_i := y_i + m_{i,j} \cdot x_j$ is performed. This implies the correctness of Generalized MAT/VEC, as stated in the following theorem.

Theorem 7 *For each i, $1 \leq i \leq n$, y_i exits at output port O_2 of the rightmost PE in the array with value $\sum_{j=1}^{n} m_{i,j} \cdot x_j$.*

7 Performance of Generalized MAT/VEC

In this section, we analyze the running time of Generalized MAT/VEC relative to the running time of MAT/VEC. As mentioned before, to compute $M \cdot x$, Generalized MAT/VEC requires STAIR(M) PEs, while MAT/VEC requires STRIPE(M) PEs. For any M we have STAIR(M) $\leq$ STRIPE(M) and the experiments in Section 4 indicate that Generalized MAT/VEC often has a significantly smaller hardware requirement. In the main result in this section, we show that, despite having a smaller hardware requirement, Generalized MAT/VEC requires the same time (number of global cycles) as does MAT/VEC. We use the notion of a global cycle as our unit of time for ease of analysis and for appropriateness of comparison. At the end of this section we mention that even if we had used another, possibly more realistic measure of time, we would have concluded that Generalized MAT/VEC is at least as fast as MAT/VEC.

7.1 Counting Global Cycles

Define inductively a sequence of position index sets $CF_1, CF_2, CF_3, \ldots$ as follows:

$$\Gamma_0 = \Gamma,$$

and, for all $i \geq 1$,

$$\begin{aligned} CF_i &= \text{set of minimal elements of } (\Gamma_{i-1}, \leq_{se}), \\ \Gamma_i &= \Gamma_{i-1} - CF_i,. \end{aligned}$$

Each CF_i is called a *computational front.* For the matrix in Figure 9,

$$\begin{aligned} CF_1 &= \{(1,3),(2,1)\} & \qquad CF_4 &= \{(4,2),(2,4)\} \\ CF_2 &= \{(1,4),(2,2)\} & CF_5 &= \{(3,4),(4,3)\} \\ CF_3 &= \{(2,3),(3,2)\} & CF_6 &= \{(4,5)\}. \end{aligned}$$

Note that a computational front is a chain in the poset $(\Gamma, \leq_{sne})$. Melhem [3] shows that elements in each computational front participate in a computation in each processing phase. In other words, Melhem shows that MAT/VEC performs the computation $y_i := y_i + m_{i,j} \cdot x_j$ in processing phase PP_t if and only if $(i,j) \in CF_t$. Theorem 8 shows that the same is true for the computations performed by Generalized MAT/VEC. The implication, of course, is that MAT/VEC and Generalized MAT/VEC perform the same computations in each global cycle.

Theorem 8 *For all t, $(i,j) \in CF_t$ if and only if the computation $y_i := y_i + m_{i,j} \cdot x_j$ is performed in processing phase PP_t by Generalized MAT/VEC.*

Proof: For the convenience of argument, let $CF_0 = \emptyset$ and let PP_0 be a dummy processing phase in which no computations are performed. The proof of the theorem is by induction on t.

Base Case: The claim is trivially true when $t = 0$ because of the definitions of CF_0 and PP_0.

Induction Case: Let $t \geq 1$ and assume that, for all $t' \leq t-1$, we have $(i',j') \in CF_{t'}$ if and only if Generalized MAT/VEC performs the computation $y_{i'} := y_{i'} + m_{i',j'} \cdot x_{j'}$ in $PP_{t'}$. Using this induction hypothesis, we show that $(i,j) \in CF_t$ if and only if Generalized MAT/VEC performs the computation $y_i := y_i + m_{i,j} \cdot x_j$ in PP_t. Let

$$\Gamma_\ell = \Gamma - (CF_1 \cup CF_2 \cup \ldots \cup CF_\ell),$$

for all $\ell \geq 0$. Let us focus on the status of Generalized MAT/VEC just after PP_{t-1}. Let $(i,j) \in CF_t \cap \Psi_p$ for some p, $1 \leq p \leq k$. By the induction hypothesis, $m_{i,j}$ has not yet been used in a computation. By definition of CF_t, (i,j) is a minimal element with respect to $\leq_{se}$ in Γ_{t-1}. Hence, $m_{i,j}$ is the I_3-input to the PE labeled p in global cycle t. We will now show that during CP_t, the components x_j and y_i reach PE p and thus the PE computes $y_i := y_i + m_{i,j} \cdot x_j$ in PP_t.

By Lemma 6, we know that y_i is in queue $IQ_y(p_L)$, where $p_L \geq p$, and x_j is in queue $IQ_x(p_R)$, where $p_R \leq p$. We first show that for all PEs to the left of PE p, $[I_5] > i$, implying that y_i is not blocked by any PE to the left of PE p and reaches PE p in CP_t. The proof is by contradiction. Suppose that there is a PE labeled $p' > p$ (that is, to the left of PE p) for which $[I_5] \leq i$. Specifically, assume that for PE $p' > p$,

$$[I_5] = i', \qquad [I_4] = j', \qquad i' \leq i.$$

Then, $j' > j$, because otherwise if $j' \leq j$, then $(i', j') \leq_{se} (i, j)$ and (i, j) is not a minimal element of Γ_{t-1} with respect to $\leq_{se}$. Thus we have

$$i' \leq i \qquad \text{and} \qquad j' > j. \tag{1}$$

Since $p' > p$, we know that $\Psi_{p'} \leq_{\gamma} \Psi_p$. Hence, by the definition of $\leq_{\gamma}$, there is a position index $(i'', j'') \in \Psi_p$ such that $(i', j') <_{sne} (i'', j'')$. In other words,

$$i' > i'' \qquad \text{and} \qquad j' < j''. \tag{2}$$

But, (1) and (2) together imply that $(i, j) <_{sne} (i'', j'')$. This is a contradiction because (i, j) and (i'', j'') both belong to the same staircase Ψ_p. Hence, for every PE to the left of PE m, $[I_5] > i$. This allows y_i to reach PE p in CP_t. A similar argument suffices to show that for all PEs to the right of PE p, $[I_4] > j$, thus allowing x_j to reach PE p in CP_t. Hence, $y_i := y_i + m_{i,j} \cdot x_j$ is computed in PP_t.

To complete the proof we need to show that if $(i, j) \notin CF_t$ then $y_i := y_i + m_{i,j} \cdot x_j$ is not computed in PP_t. Suppose $(i, j) \in CF_{t'}$ for some $t' < t$; by the induction hypothesis $m_{i,j}$ has already participated in a computation and will not participate in any further computations. So assume that $(i, j) \in CF_{t'}$ for some $t' > t$. This implies that $(i, j) \in \Gamma_{t-1}$. Now let us be more specific and assume that $(i, j) \in \Gamma_{t-1} \cap \Psi_p$ for some p, $1 \leq p \leq k$. There are two cases depending on whether (i, j) is a minimal position index in $\Gamma_{t-1} \cap \Psi_p$ with respect to $\leq_{se}$.

Case 1: If (i, j) is not the minimal position index in $\Gamma_{t-1} \cap \Psi_p$, then $m_{i,j}$ is not the I_3-input to PE p in global cycle t and hence the computation $y_i := y_i + m_{i,j} \cdot x_j$ is not performed in PP_t.

Case 2: Suppose that (i, j) is the minimal position index in $\Gamma_{t-1} \cap \Psi_p$. Since $(i, j) \notin CF_t$, there is a position index $(i', j') \in \Gamma_{t-1}$ such that $(i', j') \leq_{se} (i, j)$. Now suppose that $(i', j') \in \Psi_{p'}$ for some $p' \neq p$. If $p' > p$, then PE p' is to the left of PE p. By Lemma 6, component $y_{i'}$ is to the left of PE p' and since $i \geq i'$, component y_i is also to the left of PE p'. Thus y_i is blocked by PE p' and cannot reach PE p in CP_t. Similarly, it can be shown that if $p' < p$, then x_j is blocked by some PE to the right of PE p and cannot reach PE p in CP_t. This implies that as claimed, $y_i := y_i + m_{i,j} \cdot x_j$ is not computed in PP_t.

By induction, the theorem follows. □

Melhem provides a lower bound (see Proposition 3.1 in [3]) and an upper bound (see Proposition 3.2 in [3]) on the number of global cycles that MAT/VEC requires to complete $M \cdot x$. Neither of the bounds is tight, and he leaves the precise characterization of the number of global cycles required by MAT/VEC an open question. The following theorem provides the precise number of global cycles required by Generalized MAT/VEC (and MAT/VEC) to compute $M \cdot x$, as a function of how the nonzeros are distributed in matrix M.

Theorem 9 *Let Ψ be a largest staircase in M. Then, the number of global cycles required by* Generalized MAT/VEC *or* MAT/VEC *to compute $M \cdot x$ is $|\Psi|$.*

Proof: Let $CF_1, CF_2, \ldots, CF_T$ be the non-empty computational fronts of M. From the definition of computational fronts, it is easy to see that there is a sequence

$$(i_1, j_1) \leq_{se} (i_2, j_2) \leq_{se} \cdots \leq_{se} (i_T, j_T)$$

such that $(i_p, j_p) \in CF_p$, for $1 \leq p \leq T$. The above sequence is an antichain in $(\Gamma, \leq_{sne})$. This implies that $CF_1, CF_2, \ldots, CF_T$ is a smallest set of chains in $(\Gamma, \leq_{sne})$ covering Γ. Therefore, by Theorem 1, T is equal to the size of a largest antichain in $(\Gamma, \leq_{sne})$. An antichain in $(\Gamma, \leq_{sne})$ is a staircase, all of whose elements belong to Γ. The theorem follows. □

In the matrix shown in Figure 9, the size of the largest staircase is 6. This is equal to the number of computational fronts in the matrix and is therefore equal to the number of global cycles required by Generalized MAT/VEC and MAT/VEC.

7.2 Counting Clock Pulses

The reader may be concerned that using a global cycle as a unit of time ignores the amount of communication among PEs that has to go on between consecutive processing phases. An alternative approach is to count the number of *clock pulses* required to complete the task, assuming that at each clock pulse, each PE performs a read-compute-write cycle. More precisely, at each clock pulse, each PE reads the input ports that it is expecting input on, performs a computation if it has the requisite data, and then writes data that it no longer needs on output ports. It is possible to show that for any matrix M the number of clock pulses required by Generalized MAT/VEC is no greater than the number of clock pulses required by MAT/VEC. The proof of this claim is more tedious than the proof of the corresponding claim for global cycles and is not included here. We merely present an example to provide insights into the situation. In Figure 13 a 3×3 matrix is shown with a staircase cover of width 2 on the left and a stripe cover of width 4 on the right. The blank slots correspond to zeroes; entry i belongs to the ith staircase or stripe. Figure 14 shows

2	2	1
		1
	2	2

4	3	1
		2
	4	3

Figure 13: A 3×3 matrix. On the left a staircase cover with width 2 is shown and on the right a stripe cover with width 4 is shown.

the matrix-vector multiplication performed on this matrix by Generalized MAT/VEC and organized by clock pulses. Since the matrix is covered by 2 staircases, we have 2 PEs. Each row in the picture corresponds to the array of 2 PEs at a particular clock pulse. Each cell corresponds to a PE. The entry at the top of each cell denotes the position indices of the next matrix element, the entry at the left denotes the index of the next element of the y-vector, and the entry at the right denotes the index of the next element of the x-vector. An empty input port is denoted by $*$. Shaded cells represent PEs and clock pulses where actual computation (as opposed to mere communication) took place. For example, in clock pulse 4, PE 1 had data $m_{1,3}$, y_1, and x_3 and performed the computation $y_1 \leftarrow y_1 + m_{1,3} \cdot x_3$. At that clock pulse, PE 2 is unable to perform any computation because it is missing y_3. It takes 7 clock pulses for all the y_i's to exit the array. Figure 15 shows the same matrix-

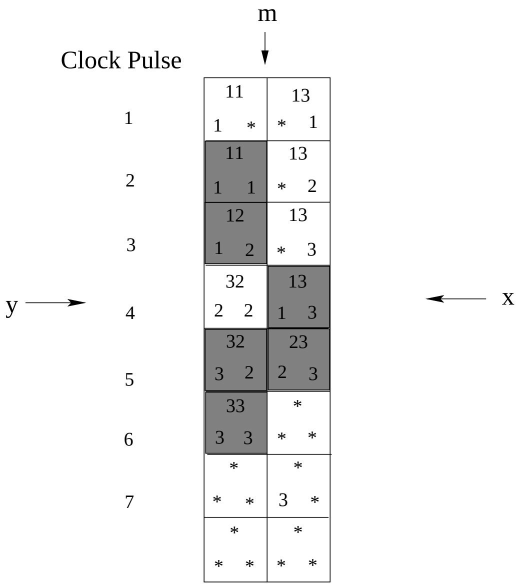

Figure 14: Computing a matrix-vector product on Generalized MAT/VEC with 2 PEs.

vector multiplication performed by MAT/VEC. Four PEs are used because 4 stripes were required to cover the matrix (see Figure 13). Despite the additional PEs, MAT/VEC takes additional clock pulses (11 in all) to complete the computation. It can be shown by induction that at any clock pulse, the computations completed by MAT/VEC is a subset of the computations completed by Generalized MAT/VEC. The intuition is that because of a larger array of PEs, elements have further to travel to reach the appropriate PE. This increases communication time and decreases parallelism.

8 Future Directions

In this paper we have presented a natural generalization of the systolic array scheme proposed by Melhem [3] for matrix-vector multiplication. Our scheme, called Generalized MAT/VEC outperforms MAT/VEC in that it has, typically a significantly smaller hard-

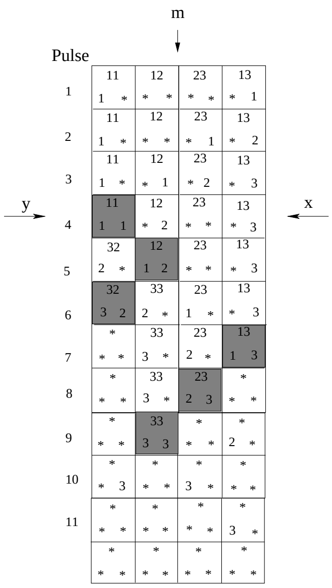

Figure 15: Computing a matrix-vector product on MAT/VEC with 4 PEs.

ware requirement, without paying in extra time.

The reader may be concerned about the fact that Generalized MAT/VEC assumes that PEs have access to unbounded queues. Since PEs are relatively simple pieces of hardware, this may seem unreasonable. For example, consider the "border matrix" $M = (m_{i,j})_{n\times n}$ matrix, in which $m_{i,j} \neq 0$ if and only if $i \in \{1, n\}$ or $j \in \{1, n\}$. When M is processed by generalized MAT/VEC, we eventually reach the state shown in Figure 16 in which each of the two queues between PEs has $(n - 2)$ elements in it. This illustrates worst case behavior

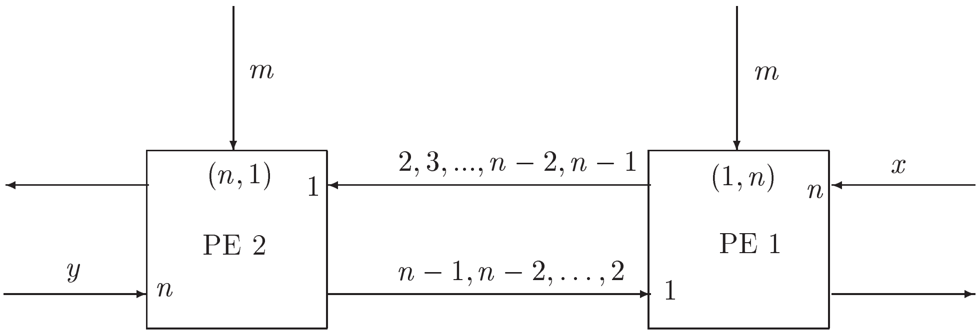

Figure 16: Each of the queue between the two PEs contains $(n - 2)$ elements.

and in general queues need to be as large as the largest "step" – the number of elements belonging to a single row (or column) – in a staircase. If we are given a bound, say a constant bound k, on queue size, then our strategy would be to partition the positions of nonzero entries in the matrix into *k-staircases*, which are staircases which contains at most k elements from a row or a column. This will ensure that all queues need to hold at most k elements at any time during the algorithm. Finding a minimum width covering of a matrix by k-staircases is as easy as finding a minimum width staircase cover. As k increases, larger queues are possible, but the width of a minimum k-staircase cover decreases. The extremes are familiar to us: when $k = 1$ we have an instance of Melhem's stripe covers and when $k = n$, we have an instance of the staircase covers as described in this paper. The precise nature of the trade-off between the value k and the width of k-staircases is not clear and might be an interesting future direction of research.

In Section 3 we presented SCA an algorithm for computing the staircase width of a matrix. We did not consider the issue of rearranging the rows or columns of the matrix to minimize staircase width. This problem, similar to the problem of minimizing bandwidth of a matrix, is NP-complete [31], but nothing is known about obtaining approximate solutions to this problem. This is another future direction of research that interests us.

Finally, as mentioned earlier, reconfigurable VLSI technology, especially FPGAs, provides the flexibility to implement a variable number of processing elements and queues of varying sizes in fast hardware. Mapping staircase-based algorithms to pipelined FPGA bus systems is an interesting challenge for the engineer.

Acknowledgement: We acknowledge the detailed comments of anonymous referees that has helped make this a better paper.

References

[1] Eun-Jin Im and Katherine Yelick. Optimizing sparse matrix kernels for data mining. Submitted to SIAM Internaional Conference on Data Mining, URL: `http://www.cs.berkeley.edu/~ejim/publication`.

[2] Eun-Jin Im and Katherine Yelick. Optimizing sparse matrix vector multiplication for register reuse. Submitted to International Journal of High Performance Computing

Applications (SCI), URL: `http://www.cs.berkeley.edu/~ejim/publication/`.

[3] Rami Melhem. Parallel solution of linear systems with striped sparse matrices. *Parallel Computing*, 6:165–184, 1988.

[4] Iain S. Duff. Sparse numerical linear algebra: direct methods and preconditioning. In *The state of the art in numerical analysis (York, 1996)*, pages 27–62. Oxford Univ. Press, New York, 1997.

[5] P. E. Bjørstad and T. Sørevik. Data-parallel BLAS as a basis for LAPACK on massively parallel computers. In *Linear algebra for large scale and real-time applications (Leuven, 1992)*, pages 13–20. Kluwer Acad. Publ., Dordrecht, 1993.

[6] Jack Dongarra. Performance of LAPACK. In *Large-scale matrix problems and the numerical solution of partial differential equations (Lancaster, 1992)*, pages 55–67. Oxford Univ. Press, New York, 1994.

[7] K. Y. Chen. Minimizing the bandwidth of sparse symmetric matrices. *Computing*, 11:103–110, 1973.

[8] J.-C. Luo. Algorithms for reducing the bandwidth and profile of a sparse matrix. *Comput. & Structures*, 44(3):535–548, 1992.

[9] Rami Melhem. Determination of stripe structures for finite element matrices. *SIAM J. Numer. Anal.*, 24(6):1419–1433, 1987.

[10] David J. Evans. Block matrix multiplication and LU factorisation systolic arrays. *Int. J. Comput. Math.*, 76(1):45–57, 2000.

[11] P. Rózsa. Block matrices and their applications. In *The role of mathematics in modern engineering (Melbourne, 1994)*, pages 85–103. Studentlitteratur, Lund, 1996.

[12] Y. Saad. ILUM: a multi-elimination ILU preconditioner for general sparse matrices. *SIAM J. Sci. Comput.*, 17(4):830–847, 1996.

[13] Timothy A. Davis and Iain S. Duff. A combined unifrontal/multifrontal method for unsymmetric sparse matrices. *ACM Trans. Math. Software*, 25(1):1–20, 1999.

[14] Joseph W. H. Liu. The multifrontal method for sparse matrix solution: theory and practice. *SIAM Rev.*, 34(1):82–109, 1992.

[15] W. Hackbusch. A sparse matrix arithmetic based on $\mathcal{H}$-matrices. I. Introduction to $\mathcal{H}$-matrices. *Computing*, 62(2):89–108, 1999.

[16] S. A. Sauter. Variable order panel clustering. *Computing*, 64:223–261, 2000.

[17] Charles E. Leiserson. *Area-Efficient VLSI Computation.* MIT Press, Cambridge, MA, 1982.

[18] A. DeHon. The density advantage of configurable computing. *Computer*, 33:41–50, 2000.

[19] K. Compton and S. Hauck. Reconfigurable computing: A survey of systems and software. *ACM Computing Surveys*, 34:171–210, 2002.

[20] M. Eisenring and M. Platzner. A framework for run-time reconfigurable systems. *Journal of Supercomputing*, 21:145–159, 2002.

[21] M. Platzner. Reconfigurable accelerators for combinatorial problems. *Computer*, 33:58–61, 2000.

[22] X. J. Zhang and K. W. Ng. A review of high-level synthesis for dynamically reconfigurable fpgas. *Microprocessors and Microsystems*, 24:199–211, 2000.

[23] K. Q. Li. Constant time Boolean matrix multiplication on a linear array with a reconfigurable pipelined bus system. *Journal of Supercomputing*, 11:391–403, 1997.

[24] K. Q. Li, Y. Pan, and S. Q. Zheng. Fast and processor efficient parallel matrix multiplication algorithms on a linear array with a reconfigurable pipelined bus system. *IEEE Transactions on Parallel and Distributed Systems*, 9:705–720, 1998.

[25] William T. Trotter. *Combinatorics and Partially Ordered Sets: Dimension Theory.* The Johns Hopkins University Press, Baltimore, 1992.

[26] R. P. Dilworth. A decomposition theorem for partially ordered sets. *Annals of Mathematics*, 51:161–166, 1950.

[27] Lenwood S. Heath, Frank T. Leighton, and Arnold L. Rosenberg. Comparing queues and stacks as mechanisms for laying out graphs. *SIAM Journal of Discrete Mathematics*, 5(3):398–412, 1992.

[28] Lenwood S. Heath and Sriram V. Pemmaraju. Stack and queue layouts of posets. *SIAM J. Discrete Math.*, 10(4):599–625, 1997.

[29] Lenwood S. Heath, Sriram V. Pemmaraju, and Ann N. Trenk. Stack and queue layouts of directed acyclic graphs. I. *SIAM J. Comput.*, 28(4):1510–1539, 1999.

[30] Lenwood S. Heath and Sriram V. Pemmaraju. Stack and queue layouts of directed acyclic graphs. II. *SIAM J. Comput.*, 28(5):1588–1626, 1999.

[31] Lenwood S. Heath and Arnold L. Rosenberg. Laying out graphs using queues. *SIAM Journal of Computing*, 21(5):927–958, 1992.

[32] H.W. Lang, M. Schimmler, H. Schmeck, and H. Schroder. Systolic sorting on a mesh-connected network. *IEEE Transactions on Computers*, 34(7):652–658, 1985.

[33] Suely Oliveira. A new parallel chasing algorithm for transforming arrowhead matrices to tridiagonal form. *Math. Comp.*, 67(221):221–235, 1998.