PARALLEL BLOCK JACOBI EIGENVALUE ALGORITHMS USING SYSTOLIC ARRAYS

David Scott and Robert C. Ward[1]

Department of Computer Sciences
University of Texas at Austin
Austin, Texas 78712

TR-85-01 January 1985

This work was supported in part by the Air Force Office of Scientific Research under interagency agreements DOE 40-1299-82 and DOD AFOSR-ISSA-84-00056 and the U.S. Department of Energy under contract DE-AC05-84OR21400.

[1]Mathematical Sciences Section, Engineering Physics and Mathematics Division, Oak Ridge National Laboratory, Oak Ridge, TN 37831

Table of Contents

List of Tables

Abstract

Systolic arrays have become established in principle, if not yet in practice, as a way of increasing computational speed for problems in linear algebra, such as computing eigenvalues, by exploiting low-level parallelism. Any systolic device that is actually implemented in silicon will necessarily have an upper limit on the size of the problems that can be solved. In this paper we consider the questions of whether such a systolic device for symmetric eigenvalue problems can be used to solve larger problems, and if so, whether such use would place any constraints on the design of the underlying systolic array. We will see that the answer to both questions is 'yes,' although the restrictions on the systolic array are mild. Our approach is to partition a large matrix into blocks that are small enough to be processed by the systolic device, and a Jacobi-like process is then carried out at the block level. If multiple copies of the systolic device are available, then this block Jacobi process can itself be implemented with a higher-level parallelism. To make any progress it is necessary to reorder the matrix between calls to the systolic arrays.

1. Introduction

The QR algorithm ([6], ch. 8) is the standard method for computing all the eigenvalues of a symmetric matrix. Unfortunately it is an essentially serial algorithm and does not adapt well to fast parallel computation. Most parallel eigenvalue algorithms are based on the older Jacobi algorithm in which individual offdiagonal elements are repeatedly annihilated using plane rotations. Convergence of the algorithm is usually measured by the sum of squares of the offdiagonal elements. This sum decreases at every step--the only question is whether it decreases all the way to zero.

In the original Jacobi algorithm, the largest offdiagonal element (in absolute value) is annihilated at each step. Such algorithms, in which the annihilation sequence depends on the size of the elements, will be refered to as a pivoting algorithms. To avoid the cost of searching for a pivot, modern computer implementations use the cyclic Jacobi algorithm in which the offdiagonal elements are repeatedly annihilated in a fixed order. The standard algorithm annihilates the off diagonal elements by rows. (That is a_{12}, a_{13}, ... ,a_{1n}, a_{23}, ..., $a_{n-1,n}$.) One pass through the annihilation sequence is called a **sweep**. Alternatively the cyclic algorithm is said to have **period** $m = n(n-1)/2$ since each offdiagonal is annihilated once every m steps. Neither algorithm uses explicit permutations; the rotations are computed and applied in place.

There are two choices for the angle of rotation used in the annihilations. In absolute value, one angle is always less than or equal to $\pi/4$ and the other is greater than or equal to $\pi/4$. These correspond to the two possible orderings of the eigenvalues of the 2x2 subproblem. The result of using the larger angle rotation is equivalent to first using the smaller angle and then permuting the two diagonal elements (and the rest of the rows and columns) involved. Convergence of the original algorithm follows immediately from the fact that the sum of squares of the offdiagonal elements decreases by at least a factor of (n(n-1) - 2)/(n(n-1)) at each step, independent of the choice of angle. To insure convergence of the cyclic algorithm, the small angle of rotation must be used. This guarantees that no implicit permutations are done and assures that large offdiagonal elements will eventually be annihilated. To show the necessity of the restriction, suppose the standard cyclic algorithm is applied to the following matrix, except that large angles are chosen for the annihilations. Since each annihilated element is zero, the annihilation reduces to a permutation. These permutations move the 3 ahead of the sweep so that it is never annihilated.

$$\begin{matrix} 0 & 0 & 3 \\ 0 & 1 & 0 \\ 3 & 0 & 2 \end{matrix}$$

When the Jacobi method converges, the convergence rate is asymptotically quadratic. (That is, asymptotically the sum of squares of the offdiagonal elements is squared after each period.) This is because the annihilation of a small offdiagonal element makes only a second order change in the rest of the matrix. A discussion of the Jacobi algorithm can be found in [6], ch. 9.

There is a natural parallelism in the Jacobi algorithm. Each plane rotation affects two rows and two columns. The effect of the rotation on each element of these rows and columns can be computed in parallel. Furthermore, several annihilations can be done in parallel provided that the annihilation order is chosen carefully. If {i,j,k,l} are distinct integers then a_{ij} and a_{kl} can be eliminated in parallel. If n is the dimension of the matrix, then as many as n/2 offdiagonal elements can be eliminated simultaneously. Sameh [8] gave a parallel cyclic algorithm which orders the set of n(n-1)/2 offdiagonal elements so that each successive n/2 elements can be annihilated simultaneously. With this ordering an entire elimination sweep can be done in n-1 steps. Sameh's ordering is not easy to implement in VLSI since it involves either annihilating elements which are far from the main diagonal or swapping rows (and columns) which are far apart. Parallel implementation of a pivoting algorithm is also possible except that the pivots must be found sequentially.

2. Systolic Algorithms for Eigenvalue Problems

Brent and Luk [1] modified Sameh's sequence so that only elements adjacent to the main diagonal are annihilated, and only nearby rows and columns are swapped. Another algorithm suitable for VLSI implementation was given by Luk [5]. In this section we give a description of the two algorithms. In both versions there are special processors (which appear on the diagonal) which actually compute and apply Jacobi rotations to annihilate offdiagonal elements next to the diagonal. The rest of the processors just propagate the effect of these rotations to the rest of the matrix. Since the matrix is symmetric, only the upper triangle is used.

In algorithm 1 [1] each diagonal processor is assigned to a particular pair of adjacent rows. The first processor has rows 1 and 2, the second processor has rows 3 and 4, etc. At each step, each diagonal processor annihilates its immediate offdiagonal element (i.e., a_{12}, a_{34}, ...), and then the rows are permuted to align new elements for annihilation. Row 1 remains fixed and the other rows are cyclically permuted in the order

$$(3, 5, 7, ..., n-1, n, n-2, n-4, ..., 2).$$

As in the standard Jacobi algorithm it is necessary to restrict the annihilating rotations to small angles (less than $\pi/4$ in absolute value) to ensure convergence.

In algorithm 2 [5] no explicit permutation of the rows is needed. Each diagonal processor affects three adjacent rows. Processor 1 has rows 1, 2, and 3, processor 2 has rows 3, 4, and 5, and so on. On odd steps each processor annihilates the offdiagonal element between its first two rows, and on the even steps the processors annihilate the off diagonal element between its last two rows. If n is even, then the last processor is active only on the odd steps. The need for explicit permutations is avoided by requiring large angles (greater than $\pi/4$ in absolute value) for each rotation. Indeed, convergence of Luk's algorithm reduces to convergence of an algorithm which uses small angle rotations and an unusual annihilation sequence. The annihilation sequence for $n = 8$ is given below.

step	annihilated elements
1	(1 2) (3 4) (5 6) (7 8)
2	(1 4) (3 6) (5 8)
3	(2 4) (1 6) (3 8) (5 7)
4	(2 6) (1 8) (3 7)
5	(4 6) (2 8) (1 7) (3 5)
6	(4 8) (2 7) (1 5)
7	(6 8) (4 7) (2 5) (1 3)
8	(6 7) (4 5) (2 3)

Monitoring convergence by computing the sum of squares of the offdiagonal elements is expensive in a systolic environment, and so Luk recommends that the algorithms be run for a "safe" number of steps to ensure convergence. A constraint on this number of steps will be introduced later.

Currently, neither algorithm can be implemented in VLSI, since not enough floating point processors can be put on one chip. This means that no timing data can be obtained to compare the two algorithms. Furthermore, different algorithms that use these systolic arrays to solve subproblems can be compared only on the basis of the number of calls to the the systolic array.

3. Block Jacobi Algorithms

Any VLSI implementation of the above algorithms will inevitably have an upper bound on the size of the problem that can be solved directly. Call this upper bound k. In the remainder of this paper we discuss block Jacobi algorithms which solve problems of size $n > k$ by using the systolic arrays to solve repeatedly problems of size k. Without the systolic array, these algorithms would not be cost effective since the effort involved in exactly solving a subproblem would be wasted when the created zeroes are filled in at the next step.

At each step of the algorithm, k indices (between 1 and n) are chosen to form the block that is sent to the systolic array. The generalization of the original Jacobi algorithm would be to chose the indices to maximize the sum of squares of the offdiagonal elements to be annihilated. For $k > 2$, this is a hard combinatorial problem. To prove convergence, it is sufficient that the largest offdiagonal element of the matrix be included in the block at each step. This requires at least one pass through the entire matrix. There remains the question of whether it is worth searching a long time to find a good block. In general, this is the question of how hard an outer iteration should work to reduce the number of inner iterations. Since the relative cost of searching the matrix and solving a subproblem on a systolic array is unknown at present, this question cannot be answered definitively. If k is small compared to n, then the cost of solving the subproblem and propagating the result to the rest of the matrix is likely to be small compared to the cost of searching the entire matrix. This is the same observation that led people to abandon the original Jacobi algorithm in favor of the cyclic Jacobi algorithm.

A cyclic block Jacobi algorithm is specified by a finite annihilation sequence. Each member of the sequence is a set of k integers between 1 and n which specify which diagonal elements will be part of the annihilation block at the current step. The finite sequence of annihilations is repeated until convergence. Convergence of cyclic block Jacobi algorithms depends on both the annihilation sequence and the solution technique used to annihilate the block. In particular, there are k! possible orderings of the eigenvalues of the block and each ordering leads to a different implicit relabelling of the diagonal elements. The same problem affects the scalar cyclic Jacobi algorithm. A sufficient solution for the scalar algorithm is to use only small angle rotations. Although we give no formal proof, the following proposition contains sufficient conditions for a cyclic block Jacobi algorithm to converge.

Proposition.

A block cyclic Jacobi algorithm will converge provided:

1. Every pair of indices appears together in at least one of the annihilation sets.

2. The subproblem solution technique does NOT reorder nearly diagonal matrices.

An informal justification of this proposition is as follows. As long as large offdiagonal elements are being annihilated, progress is being made. Asymptotically, the subproblems being solved become nearly diagonal. Then by condition 2 there are no implicit permutations of the matrix. This means that any large offdiagonal element will not be moved. Then by condition 1, both of its indices will be in the annihilation set at some step during the cycle, and the element will be annihilated. This shows that stagnation cannot occur.

Thus the QR algorithm (without sorting) or the standard Jacobi algorithm are acceptable subproblem solution techniques since neither of them reorders a nearly diagonal matrix. Unfortunately, neither of the systolic algorithms satisfy this constraint without modification to their termination criteria. The following lemmas contain the required restrictions. For algorithm 1 a step is always n/2 annihilations while for algorithm 2 it is either n/2 or n/2 -1 annihilations.

Lemma 1: Algorithm 1 does not reorder a diagonal matrix provided that the number of steps is a multiple of n-1.

Proof: This follows immediately from the fact that the rows are cyclically permuted and that the permutation is an n-1 cycle.

Lemma 2: Algorithm 2 does not reorder a diagonal matrix provided that the number of steps is a multiple of 2n.

Proof: Two steps of algorithm 2 cyclically permute the elements of the indices in the order

$$(1, 3, 5, \ldots, n\text{-}1, n, n\text{-}2, \ldots, 4, 2)$$

Since this is an n-cycle, 2n steps restores the matrix to its original order.

With these restrictions on the systolic arrays, condition 2 is satisfied. There remains the question of choosing an annihilation sequence to satisfy condition 1. In addition it may be desirable that each pair of indices appear together equally often. This problem has been studied extensively by statisticians under the title of "balanced incomplete block experimental designs" (see for example, [7] or [2]). It is important to stress that there is no simple algorithm for generating designs for given n and k. Also, there is a significant difference between our context and experimental design. Rather than independent replications of an experiment, our annihilations are done sequentially, and the order is important. If two indices are together in two consecutive steps then the second step will be inefficient since one of the elements to be annihilated will already be zero. Therefore an arbitrary balanced incomplete block design may not make a good annihilation sequence. This point will be examined in more detail in section 5.

4. Parallel Block Jacobi Algorithms

As with the scalar Jacobi algorithm, the block Jacobi algorithm can also be implemented in parallel. As many as $\lceil n/k \rceil$ blocks can be annihilated in parallel. Henceforth, we assume that n is a multiple of k. At worst this means that the matrix must be padded with k-1 zero rows and columns. Both pivoting and cyclic algorithms are possible, and pivoting algorithms become more cost effective since **all** the annihilation blocks can be determined with one stage of pivoting. Two pivoting algorithms will be considered. Neither is optimal (that is, neither obtains the maximum possible decrease in the sum of squares of the offdiagonal elements) but both require only one pass through the matrix.

The first algorithm is a kind of partial pivoting. For each row $i = 1$ to n, find j such that

$$|a_{ij}| = \max \{|a_{im}| \text{ for } m \geq i\}$$

and permute the jth row and column to the (i+1)th. That is, the largest element in the remaining part of each row is permuted to be next to the diagonal. Then the diagonal blocks of size k are solved simultaneously. As described, this algorithm may converge very slowly or not at all. The problem is that the elements which were permuted in the kth, (2k)th, ... rows are **not** part of the blocks to be annihilated. In particular, for $k = 2$ and $n = 4$ the largest element in the second row is not annihilated in the first pass, and unless it is permuted to third row by a subsequent permutation in the first row, it will never be annihilated at all. For most matrices such a permutation will happen eventually, but there exist matrices for which the algorithm doesn't converge.

In order to guarantee convergence, it is necessary to prevent large elements from hiding in the last row of each block. In particular, if the (k,k+1)th element is larger than the (1,2) element in the current block, then it is sufficient to swap the 1st and (k+1)th rows (and columns). Other choices are possible, but this puts the (k,k+1)th into the block to be annihilated at the cost of removing only the (1,2) element. None of the other large elements chosen for annihilation are disturbed. With this additional patch it is clear that the algorithm converges (regardless of any reordering generated by the subproblem solutions) since the largest offdiagonal element in the entire matrix is annihilated at every step. This pivoting algorithm still costs $O(n^2)$ comparisons per step and may not be that efficient since only k-1 elements are known to be large while a total of $k(k-1)/2$ elements are annihilated.

The other pivoting algorithm is a block algorithm in which $k^2/4$ elements are chosen to be (collectively) large. Consider the matrix as a block matrix with blocks of size $k/2$. Find the largest (in norm) offdiagonal block in the first two (block) rows and permute it to the (1,2) element. Repeat this for each pair of (block) rows until all the annihilation blocks are determined. This algorithm is inefficient in that (after the first step) nearly half of the elements in each annihilation block will already be zero. On the other hand, many more elements are brought into the annihilation block by design rather than by chance. The searching costs for these two pivoting algorithms are about the same. Their overall efficiency will be examined in the next section.

Parallel cyclic algorithms are also possible. An annihilation sequence is now a sequence of partitions of the n indices into sets of size k. Statisticians call such a sequence a "resolvable incomplete block design," where "resolvable" refers to the ability to group the sets into partitions. Many incomplete block designs have been tabulated for small values of n and k ([4], for example). Unfortunately, resolvability of a design is not an important characteristic, and therefore most tabulations are not resolved and many designs cannot be resolved. Also, not all tabulated designs are balanced (meaning that some pairs of indices appear together more often than others). Finally, it will be shown in the next section that the order in which the partitions are used can have a significant effect on the efficiency of the algorithm. Therefore, finding an appropriate sequence of partitions to be used for particular values of n and k can be quite difficult.

If balance is not considered crucial, then it is possible to inflate a partition sequence for a particular pair (n,k) into a partition sequence for (mn,mk) for any m. This is done by considering the matrix of size mn to be a block matrix of size n with blocks of size m. The systolic array can then handle a kxk block matrix. The blocks are considered indivisible and are permuted according to the partition sequence for (n,k). For example, algorithm 1 yields a partition sequence for $k = 2$ and any even n. It can then be used to generate a partition sequence for any even k. For example, the inflated partition sequence for (8,4) obtained from algorithm 1 is as follows.

$$\{1, 2, 3, 4 \mid 5, 6, 7, 8\}$$

$$\{1, 2, 7, 8 \mid 3, 4, 5, 6\}$$

$$\{1, 2, 5, 6 \mid 7, 8, 3, 4\}$$

This partition sequence is not balanced. The indices 1 and 2 are together all three times, while 1 and 3 are together only once. A better balanced partition sequence (computed from scratch) is

$$\{1, 2, 3, 7 \mid 4, 5, 6, 8\}$$

$$\{1, 2, 4, 5 \mid 3, 6, 7, 8\}$$

$$\{1, 3, 5, 6 \mid 2, 4, 7, 8\}$$

$$\{1, 4, 6, 7 \mid 2, 3, 5, 8\}$$

$$\{1, 3, 4, 8 \mid 2, 5, 6, 7\}$$

$$\{1, 5, 7, 8 \mid 2, 3, 4, 6\}$$

$$\{1, 2, 6, 8 \mid 3, 4, 5, 7\}$$

since every pair appears together three times. On the other hand, from a different point of view the second sequence is much worse than the first. Henrici, [3] defined quasi-Jacobi algorithms to be annihilations sequences in which some offdiagonal elements were annihilated several times before others were annihilated at all. Block Jacobi algorithms are quasi-Jacobi algorithms since every offdiagonal in the annihilated block is annihilated many times before any of the rest of them are annihilated. Henrici defined the **quasi-period** of a quasi-Jacobi algorithm to be the maximum time between annihilations of any particular element. Henrici showed that convergence of quasi-Jacobi algorithms was asymptotically quadratic in the length of the quasi-period. For block parallel Jacobi algorithms the quasi-period is the maximum number of steps between annihilations of any particular offdiagonal element. The first sequence has a quasi-period of three steps while the second one has a period of five steps (since 1 and 8 appear together in the last three partitions). The question of whether a short period or balance is more important in an annihilation sequence will be answered (computationally) in the next section.

5. Numerical Results

This section will give the results of several numerical experiments. A step of all of the algorithms is defined as one call to the set of systolic arrays. All results give the average number of steps needed by different algorithms to solve randomly generated example problems. Twenty replications were done in all experiments. The algorithms used are labelled as follows:

R	Random partitions. Elements to be annihilated are determined randomly. This algorithm is not cyclic and satisfies condition 1 only in a probabilistic sense.
Dx	A priori designs. The particular design(s) used will be described for each experiment.
B1	Algorithm 1, inflated to match the given value of k.
PS	The scalar pivoting algorithm described earlier.
PB	The block pivoting algorithm describe earlier.

The first experiment investigated the sensitivity of the algorithms to matrices generated by different procedures. All the matrices had $n = 16$ and $k = 4$. Six sets of matrices were generated. The six sets were:

U100	Each element in each matrix was randomly chosen from the uniform distribution (on $[0,1)$].
E100	Each element in each matrix was randomly chosen from the exponential distribution (with parameter $\lambda = 1$).
U10	random sparse matrices--each element in each matrix was zero with probability 0.9 and otherwise was chosen from the uniform distribution (on $[0,1]$).
E10	Random sparse matrices--each element in each matrix was chosen as in U10 except that nonzero elements were chosen from the exponential distribution (with parameter ($\lambda = 1$).
SU	Structured sparse matrices--a_{ij} was zero unless $\lvert i-j \rvert$ was zero, one or five. The nonzeros were chosen uniformly.
SE	Structured sparse matrices--a_{ij} was zero unless $\lvert i-j \rvert$ was zero, one or five. The nonzeros were chosen from the exponential distribution.

The five algorithms that were applied to each were R, D, B1, PS, and PB. Algorithm D used the following annihilation sequence (found by hand), which is perfect since each pair of indices appears together exactly once.

$$\{1, 2, 3, 4 \mid 5, 6, 7, 8 \mid 9, 10, 11, 12 \mid 13, 14, 15, 16\}$$

$$\{1, 5, 9, 14 \mid 3, 6, 11, 13 \mid 4, 8, 10, 16 \mid 2, 7, 12, 15\}$$

$$\{1, 6, 10, 15 \mid 4, 7, 9, 13 \mid 2, 8, 11, 14 \mid 3, 5, 12, 16\}$$

$$\{1, 8, 12, 13 \mid 3, 7, 10, 14 \mid 4, 5, 11, 15 \mid 2, 6, 9, 16\}$$

$$\{1, 7, 11, 16 \mid 3, 8, 9, 15 \mid 4, 6, 12, 14 \mid 2, 5, 10, 13\}$$

The results are given in table 5.1.

There was almost no difference in performance between choosing numbers from

Table 5-1: Average Number of Steps for Different Matrices

Algorithm	U100	E100	U10	E10	SU	SE
R	57.20	58.10	47.70	47.70	53.55	54.15
D	21.35	21.55	18.50	18.35	20.25	20.30
B1	31.55	30.20	26.15	26.40	29.00	29.00
PS	24.50	24.65	13.60	13.65	23.75	22.40
PB	28.85	29.40	21.55	21.00	27.25	26.00

the uniform and exponential distributions. Furthermore perfomance on the structured sparse matrices was almost identical to the performance on the dense matrices. The only significant difference was the performance of the pivoting algorithms on the random sparse matrices. This was caused by the fact that the random sparse matrices were (usually) the direct sum of smaller matrices. This meant that many of the initial zeros persisted throughout the solution process and could be always avoided by the pivoting algorithms. On the other hand, the structured sparse matrices are irreducible, the initial zeros were quickly filled in, and thereafter behaved like the dense matrices. Since reducible matrices are unlikely to occur in practice and the rest of the matrices behaved very similarly, the rest of the experiments used U100 type matrices.

The next experiment investigated the relative behavior of different annihilation sequences. All twenty matrices had n = 12 and k = 4. Algorithms R, B1, and three designs (labelled D1, D2, and D3) were used. All three designs start with the partition

$$\{1,2,3,4 \mid 5,6,7,8 \mid 9,10,11,12\}$$

and all of them were cyclic, in that each new partition was obtained from the previous one by permuting the indices in a fixed order. All of them used eleven cycles in which index 1 remained fixed. D1 used the permutation

$$(2,5,9,6,10,11,7,8,3,12,4).$$

D2 used the permutation

$$(2,5,6,7,9,3,8,10,11,4,12).$$

D3 used the permutation

$$(2,4,6,8,10,12,11,9,7,5,3).$$

The D3 sequence is the sequence obtained from the ordering used by algorithm 1 (for n = 12 and k = 2) but then annihilating blocks of size four instead of size two.

Designs D1 and D2 are balanced in that each offdiagonal element is annihilated three times in each cycle of length eleven. Design D3 is unbalanced: in each cycle some offdiagonal elements are only annihilated once while others are annihilated seven times. B1 is also unbalanced: in each cycle of length five some offdiagonal elements are annihilated five times and others are annihilated only once. B1 has a period of five, D1 has a period of six, D2 has a period of eight, and D3 has a period of eleven. Table 5.2 gives the average number of steps and average number of periods for each of the sequences applied to the twenty random matrices.

Table 5-2: Average Number of Steps and Periods

Algorithm	steps	periods
R	37.25	
D1	22.40	3.73
D2	28.35	3.54
D3	36.20	3.29
B1	19.55	3.91

The results clearly indicate that period length is more important than balance in assessing the effectiveness of a scheme. In particular, the random scheme performs very poorly since it has only a probabilistic period which is rather large (about 2log(n) times the minimum possible period).

The final experiment examined the effects of different block sizes on the algorithms. All matrices had n = 32 and both k = 4 and k = 16 were used. Algorithms D1, D2, B1, PS, and PB were used. For (32,16) B1 is the only partition sequence which has the minimal period of three and so no other design sequence was used in this case. The design sequences for (32,4) were constructed as follows. Each subset (of size four) of each partition in D1 was generated randomly subject to the constraint of annihilating as many new off diagonal elements as possible. Many sequences were generated and the shortest found had fourteen partitions (and hence period 14). D1 is not balanced--some pairs appear twice and others appear only once but only one pair (8,20) appears three times. The partitions in D1 are given below.

Partition Sequence for D1

{1,5,6,7|2,9,17,19|3,4,12,18|8,10,15,20|11,14,21,23|13,16,22,32|24,25,29,30|26,27,28,31}
{1,4,17,25|2,3,7,10|5,8,11,31|6,9,13,24|12,14,19,20|15,18,22,28|16,23,27,30|21,26,29,32}
{1,3,10,17|2,4,6,28|5,12,15,21|7,11,22,30|8,19,24,26|9,14,16,18|13,23,29,31|20,25,27,32}
{1,16,21,31|2,14,15,30|3,5,24,32|4,20,22,23|6,9,11,27|7,13,17,26|8,12,25,28|10,18,19,29}
{1,19,22,27|2,5,13,25|3,11,15,16|4,8,21,24|6,18,30,31|7,9,20,29|10,12,23,26|14,17,28,32}
{1,8,18,23|2,7,15,31|3,20,28,29|4,5,11,26|6,14,22,25|9,10,30,32|12,16,17,24|13,19,21,27}
{1,13,20,30|2,18,24,27|3,5,22,23|4,19,31,32|6,10,21,28|7,8,14,16|9,15,25,26|11,12,17,29}
{1,2,12,32|3,21,25,31|4,14,27,29|5,8,17,30|6,16,20,26|7,13,15,18|9,19,23,28|10,11,22,24}
{1,9,24,31|2,21,22,29|3,8,13,27|4,7,10,25|5,16,19,28|6,15,23,32|11,17,18,20|12,14,26,30}
{1,15,26,29|2,11,25,32|3,6,19,30|4,16,18,21|5,10,13,14|7,23,24,28|8,9,12,22|17,20,27,31}
{1,11,13,28|2,16,23,26|3,4,9,14|5,12,27,29|6,15,17,24|7,19,21,32|8,20,22,30|10,18,25,31}
{1,4,10,11|2,14,20,24|3,7,26,27|5,9,18,32|6,8,16,29|12,13,22,31|15,19,23,25|17,21,28,30}
{1,14,22,26|2,8,28,31|3,18,19,24|4,13,15,30|5,9,20,21|6,7,11,12|10,16,27,32|17,23,25,29}
{1,8,20,32|2,6,10,21|3,12,16,25|4,9,13,23|5,7,24,29|11,14,19,31|15,17,22,27|18,26,28,30}

The design for D2 is a balanced cyclic design with cycle length 31. The index 1 remains fixed and each of the remaining indices are circularly permuted using the cycle

(2,6,20,3,29,7,23,24,30,27,10,11,8,15,28,12,31,4,32,16,5,9,13,17,18,21,19,25,14,26,22).

Significant computer time was expended in finding this balanced permutation and it was disappointing to discover that it has period 22.

Table 5-3: The Effect of Block Size

algorithm	k = 4		k = 16	
	steps	periods	steps	periods
D1	64.25	4.59		
D2	84.000	3.82		
B1	80.30	5.35	11.50	3.83
PS	49.80		12.85	
PB	65.85		11.35	

6. Conclusions

The most important conclusion is that parallel block cyclic Jacobi algorithms can be effectively implemented using systolic arrays to solve the block subproblems. Furthermore if pivoting algorithms are used then any systolic array will do but for cyclic algorithms it is necessary that the systolic array does not reorder a nearly diagonal matrix. For the two types of systolic arrays considered here, Lemmas 1 and 2 give sufficient conditions on the termination criteria of the algorithms to insure this property.

Some conclusions about the relative effectiveness of different implementations can also be drawn. First, randomly chosen partitions are inferior in all cases and should never be used. This is not surprising since random sequences are only probabilistically balanced, and, more importantly, the expected period is rather large. Second, in judging block designs the most important criterion is minimizing the length of the period. Balance is a secondary criterion which can be used to choose between two designs with the same period. Third, the scalar pivoting scheme is better than the block pivoting scheme particularly as the size of the block increases. This is because the block pivoting scheme always includes in the new block some zeros from the previous block and the percentage of included zeros grows toward 50% as the size of the block increases. On the other hand the scalar pivoting algorithm never includes zeros from the previous block.

In general it is not easy to find an annihilation sequence with a short period for any particular pair (n,k). The major advantage of the pivoting algorithms and the B1 algorithm is that no a priori computation of an annihilation sequence is needed. If many problems of the same size are going to be solved then it might be worth looking for a reasonably balanced design with a short period. It is noted again that balanced incomplete block designs are not necessarily useful. The design must be resolvable and the order of the partitions (which is of no importance to statisticians) must be carefully chosen to yield a short period.

If only one problem of a particular size is to be solved then either the B1 algorithm or the PS algorithm should be used. The choice between B1 and PS would be based on the relative cost of pivoting and computing, which is not currently available.

[1] Brent, R. P. and F. T. Luk.
The solution of singular-value and symmetric eigenvalue problems on multiprocessor arrays.
Technical Report TR83-562, Cornell University, 1983.

[2] Cochran, W. G. and G. M. Cox.
Experimental Designs.
Wiley and Sons, 1957.

[3] Henrici, P.
On the Speed of Convergence of Cyclic and Quasicyclic Jacobi Methods for Computing Eigenvalues of Hermitian Matrices.
J. Soc. Ind. Appl. Math. :144-162, 1958.

[4] John, J. A., F. W. Wolock, and H. A. David.
NBS Applied Mathematics Series. Number 62: *Cyclic Designs.*
US Government Printing Office, 1972.

[5] Luk, F. T.
A triangular processor array for computing the singular value decomposition.
Technical Report TR84-625, Cornell University CS Dept., 1984".

[6] Parlett, B. N.
The Symmetric Eigenvalue Problem.
Prentice-Hall, 1980.

[7] Rhagavarao, D.
Construction and Combinatorial Problems in Design of Experiments.
John Wiley and Sons, 1971.

[8] Sameh, A. A.
On Jacobi and Jacobi Like Algorithms on a Parallel Computer.
Math. of Comp. 25:579-590, 1971.