Wrap-around Partitioning for Block Bidiagonal Linear Systems

Markus Hegland, CSL, RSISE

M.R.Osborne, CMA, SMS

Abstract

A stable vector algorithm for the solution of block bidiagonal linear systems is obtained by a permutation of the unknowns called wrap-around partitioning combined with standard QR factorisation. Wrap-around partitioning uses blocking and selects the unknowns in the blocks in turns. After a suitable orthogonal elimination step one ends up with a reduced system which is again block bidiagonal and so wrap-around partitioning can be applied again. Using a simple model for vectorisation overhead it is shown that small block sizes give best performance. The minimal block size 2, which corresponds to cyclic reduction, is suboptimal due to memory bank conflicts.

1 Introduction

Wrap-around partitioning is a reordering of the unknowns in a system of linear equations. Such a reordering can highlight sets of unknowns which can be eliminated independently of one another, and hence increases the potential for vectorized or parallel solution strategies. Hegland considered wrap-around partitioning in [2] where he describes the nonrecursive application to tridiagonal systems of linear equations. The technique has application to narrow banded systems, and it is considered here for block bidiagonal systems where it has significant advantages not shared by the other cases. In the following let n denote the number of blocks, m the block size and hence nm is the dimension of the block bidiagonal matrix. Typical values used in the testing of the implementation developed here were $500 < n \leq 10,000$, $m = 3, 4$. Consider for example the case $n = 12$. Then the block bidiagonal matrix has the form shown in Figure 1. Note that a submatrix element (here labelled D_1) is included in the $(1, n)$ position. Problems leading to matrices of this kind occur naturally in the solution of systems of ordinary differential

	1	2	3	4	5	6	7	8	9	10	11	12
1	A_1											D_1
2	B_1	A_2										
3		B_2	A_3									
4			B_3	A_4								
5				B_4	A_5							
6					B_5	A_6						
7						B_6	A_7					
8							B_7	A_8				
9								B_8	A_9			
10									B_9	A_{10}		
11										B_{10}	A_{11}	
12											B_{11}	A_{12}

Figure 1: Block bidiagonal matrix.

equations, the application to boundary value problems being the best known. This case would correspond to the recurrence

$$B_{i-1}\mathbf{x}_{i-1} + A_i\mathbf{x}_i = \mathbf{f}_i,\ i = 2, 3, \cdots, n,$$

together with boundary conditions expressed by the equation

$$A_1\mathbf{x}_1 + D_1\mathbf{x}_n = \mathbf{f}_1.$$

However, it would seem that most narrow banded matrices of regular enough structure can be put in this form. The two following examples show possible transformations of tridiagonal matrices. The first corresponds to a reordering of the rows in the manner indicated in Figure 2. An alternative is to consider the three term recurrence relation

$$b_i x_{i-1} + (a_i^1 + a_i^2)x_i + c_i x_{i+1} = f_i.$$

This is equivalent to the 2×2 first order system with twice the number of unknowns

$$\begin{bmatrix} b_i & a_i^1 \\ 0 & -1 \end{bmatrix} \mathbf{y}_{i-1} + \begin{bmatrix} a_i^2 & c_i \\ 1 & 0 \end{bmatrix} \mathbf{y}_i = \begin{bmatrix} f_i \\ 0 \end{bmatrix},$$

where $\mathbf{y}_{i-1}^T = [x_{i-1}, x_i]$. It can be written more generally by introducing the change of variable $\mathbf{z}_i = T_i\mathbf{y}_i$. This gives

$$\begin{bmatrix} b_i & a_i^1 \\ 0 & -1 \end{bmatrix} T_{i-1}^{-1}\mathbf{z}_{i-1} + \begin{bmatrix} a_i^2 & c_i \\ 1 & 0 \end{bmatrix} T_i^{-1}\mathbf{z}_i = \begin{bmatrix} f_i \\ 0 \end{bmatrix}.$$

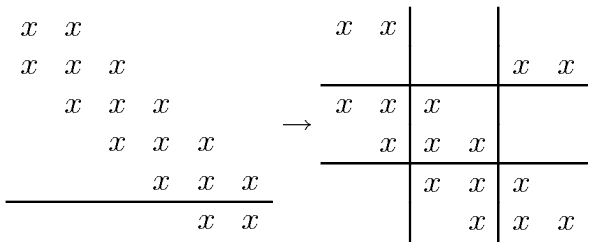

Figure 2: Partition of tridiagonal matrix

Here T_i can be chosen rather arbitrarily, and one possibility is that both the T_i and possible left transforming matrices S_i be selected to improve the condition number of the resulting block bidiagonal matrix. One example occurs when the difference equation corresponding to discretization of a second order differential equation undergoes the transformation

$$T_{i-1}\mathbf{y}_{i-1} = \begin{bmatrix} x_i \\ \frac{(x_i - x_{i-1})}{h} \end{bmatrix}.$$

This typically reduces the condition number of the linear system from $O(h^{-2})$ to $O(h^{-1})$ where h is the discretization interval. This reduction is analogous to the reduction of the second order differential equation to a first order system.

We note that for fairly general classes of band matrix problems converted to block bidiagonal form, structure inherited from the original form of the problem permits economies to be made in the first pass of wrap-around partitioning. However, it turns out that this structure cannot be preserved stably after this pass if techniques such as orthogonal factorization or row pivoting are used. The difficulty comes about whenever there is an increase in local bandwidth. Subsequent passes then require the full block bidiagonal solver. This is considered in [3].

The organisation of the paper is as follows. In the next Section, details of the wrap-around partitioning transformation, and of the associated index calculations are given. The determination of optimum transformation parameters under the standard model of vector computation is considered in Section 3., and certain implementation details are discussed in Section 4. Some conclusions are drawn in the final Section. It will be convenient to associate the same numerical value with the mathematical quantity p_i and the computational object (array element) `p(i)`.

	1	5	9	2	6	10	3	7	11	4	8	12
1	A_1											D_1
5		A_5								B_4		
9			A_9								B_8	
2	B_1			A_2								
6		B_5			A_6							
10			B_9			A_{10}						
3				B_2			A_3					
7					B_6			A_7				
11						B_{10}			A_{11}			
4							B_3			A_4		
8								B_7			A_8	
12									B_{11}			A_{12}

Figure 3: The wrap-around transformation.

2 Wrap-around partitioning

A step of wrap-around partitioning transforms the linear system into a system consisting of q (super)blocks of p (submatrix) elements each. Here p, q are called the partitioning factors). The transformation is illustrated in Figure 3 for the 12×12 block bidiagonal matrix in Figure 1 for the case of $q = 4$ super blocks of $p = 3$ blocks each. Next the elimination of the $q - 1$ subdiagonal $p \times p$ block submatrices is carried out. Here elimination by orthogonal transformations using Householder matrices is considered.

It is important to note that each subdiagonal block row in the current pivotal block column can be eliminated independently of the other block rows. Thus, in the numbering of Figure 3, the row pairs $(1, 2), (5, 6), (9, 10)$ have no elements in common. This observation provides the basis for vectorizing the computation over p for each $i : 1 \leq i \leq q - 1$, or for parallelising it by using the independence of the row pairs if the basic subblock size m is large enough (a case not considered here).

The factorization introduces additional fill, and this makes it necessary that the subblock elimination is computed locally on a stencil (or frontal matrix, or active submatrix) that has the form

$$\begin{bmatrix} A_i & C_i & D_i \\ B_i & A_{i+1} & D_{i+1} \end{bmatrix}.$$

Note that eliminatiion applied to Figure 1 works with a frontal matrix of the same structure. It follows that the number of floating point operations

is the same for the original and transformed systems. Referring to Figure 3 it will be seen that this stencil requires the renaming of the wrapped around blocks. Here this requires $B_4 \rightarrow D_5, B_8 \rightarrow D_9$. This has been implemented by an explicit copy to avoid involving the user in unnecessary detail in order to set the top level. The new elements C_i are set to zero initially, and fill as a result of mixing with A_{i+1} in the current elimination step if either orthogonal factorization is used or a partial pivoting interchange occurs, while the D_i contain the fill which propagates forward in the q'th block column (indices 4,8,12 in Figure 4). This is a result both of the wrapping around indicated above, and the potentially nonzero element D_1. This shows that the 4 block arrays A_i, B_i, C_i, D_i are needed to display the factorization. The result of the elimination applied to the matrix in Figure 3 is shown in Figure 4. Here '-' indicates the eliminated blocks, '*' denotes the original entries which have not been eliminated while the integer entries show where and at what stage fill-in occurred. This Figure is meaningful also for an elimination scheme based on partial pivoting as it shows the maximum fill that can occur in this case.

The procedure that has been implemented in these elimination steps is orthogonal factorization. This is chosen because it is known to have better stability properties than Gaussian elimination with partial pivoting for important classes of block bidiagonal matrices [4]. In principle, Gaussian elimination with partial pivoting does produce less fill-in than QR factorisation. However, the location of the zeros of Gaussian elimination with partial pivoting within the nonzero structure defined by QR factorization is only known at run-time, and a vectorised implementation would have to allow for all possible fill-in. Thus an important component of the argument for the cost advantage of Gaussian elimination is lost in the vector case.
It is important to note that the fill patterns combine in the $p \times p$ block in the (q, q) position but do not interfere. *But the crucial observation is that this final block is again block bidiagonal in structure. This forms the basis for the recursive implementation because now wrap-around partitioning can be applied again.* Let the dimension of this (unprocessed) block at recursive depth i, $i = 1, 2, \cdots, k$, be r_i, and let the corresponding partitioning factors be $\{p_i, q_i\}$ where $p_i q_i = n_i \leq r_i$ as the factors are not required to be exact. Here wrap-around partitioning is applied to the first n_i elements in the unprocessed block at stage i.

For example, taking $n_1 = 10, p_1 = 5, q_1 = 2$ in the $n = 12$ problem gives a wrap-around transformation that applies only to the first 10 components. This is shown in Figure 5. Now the pattern of fill is shown in Figure 6. Again the (q, q) block is block bidiagonal. This shows that exact factors of the order of the block to be transformed at each recursive step are not required.

	1	5	9	2	6	10	3	7	11	4	8	12
1	$*$			1								$*$
5		$*$			1					$*$		
9			$*$			1					$*$	
2	$-$			$*$			2					1
6		$-$			$*$			2		1		
10			$-$			$*$			2		1	
3				$-$			$*$			3		2
7					$-$			$*$		2	3	
11						$-$			$*$		2	3
4							$-$			$*$		3
8								$-$		3	$*$	
12									$-$		3	$*$

Figure 4: Elimination pattern for wrapped around matrix.

The following conditions must be satisfied by the block orders and partitioning factors:

$$p_1 q_1 = n_1 \leq r_1 = n, \text{ and} \tag{1}$$

$$p_i q_i = n_i \leq r_i = p_{i-1} + (r_{i-1} - n_{i-1}),\ i = 2, \cdots, k. \tag{2}$$

The recursive implementation of a sequence of wrap-around transformations can be organised as follows. Define a vector O corresponding to an array `O(i)`, $i = 1, 2, \cdots, k$, with elements which are the base points for the sequence of transformations by

$$O_1 = 0,\ O_i = \sum_{j=1}^{i-1} p_j (q_j - 1),\ i = 2, \cdots, k.$$

In particular, r_i in (2) satisfies

$$r_i = n - O_i.$$

To minimize rewriting in the general case, an indirection array `A1` can be used to point to the pivotal subblocks. In Figure 5 this corresponds to

$$\texttt{A1} \leftarrow (1, 3, 5, 7, 9, 2, 4, 6, 8, 10, 11, 12).$$

This array can be set recursively:

	1	3	5	7	9	2	4	6	8	10	11	12
1	*											*
3		*				*						
5			*				*					
7				*				*				
9					*				*			
2	*					*						
4		*					*					
6			*					*				
8				*					*			
1*					*					*		
11										*	*	
12											*	*

Figure 5: Inexact factors and partial transform

	1	3	5	7	9	2	4	6	8	10	11	12
1	*					1						*
3		*				*	1					
5			*				*	1				
7				*				*	1			
9					*				*	1		
2	-					*						1
4		-				1	*					
6			-				1	*				
8				-				1	*			
10					-				1	*		
11										*	*	
12											*	*

Figure 6: Elimination in partial transform.

```
/* Initialise A1(i)=i, i=1,2,...,n.  At each recursive
depth d=1,2,...,k the next wrap-around transformation
is defined on the current state of the pointer vector A1.
Let p=p(d), q=q(d).  A2 is a work vector.  */
DO i=1,q {
    DO j=1,p {
        A2(O(d)+(i-1)*p+j)=A1(O(d)+i+(j-1)*q);
        }
    }
DO i=1,p*q {
    A1(O(d)+i)=A2(O(d)+i);
    }
```

Organising the elimination so that the inner loops are over `p` at each stage makes for straightforward vectorization provided the arrays are stored so that the variable changing most rapidly corresponds to the loop index. For example, in FORTRAN, the diagonal submatrices A_i should be stored as `A(i,*,*)`, $i = 1, 2, \cdots, n$, as the first index changes most rapidly. Not every sequence satisfying (1) and (2) gives constant stride memory access in the vectorized sections of the solution process. Non-constant stride access can only happen in the second or later stages of the recursion. For example, let $r_1 = n = 65$, $p_1 = 6$ and $q_1 = 10$ hence $n_1 = 60$ by condition (1). In this case, data access is with stride $q_1 = 10$ in the first elimination stage. Conditions (2) are satisfied with $p_2 = 5$, $q_2 = 2$, $r_2 = 11$ and $n_2 = 10$ but the second stage now requires the first four data accesses with stride 20 while the last data access is stride 2. The reason for this nonuniformity is to be found in the first step where the reduced system is defined such that the first 6 elements are taken with stride 10 where the last 5 elements have stride 1.

In general, the stride for the first data elements in stage i is $q_1 \cdots q_i$. However, towards the end of the array, the stride can be less if factors are not exact when it may be determined by the stride at an earlier stage. This can result in non-constant stride. However, if the end of the pointer array is not accessed up to and including stage i one gets constant stride. This is exactly the case when the subarray containing the modifications in stage i (which is of length $p_i q_1 \cdots q_i$) is a subarray of the block which was modified in the previous step, or, in formulas

$$p_i q_1 \cdots q_i \leq p_{i-1} q_1 \cdots q_{i-1}$$

from which one gets

$$p_i q_i \leq p_{i-1}.$$

In particular, a useful sufficient condition for constant stride is provided by

$$p_0 = n,\ p_i = \left\lfloor \frac{p_{i-1}}{q_i} \right\rfloor,\ i = 1, 2, \cdots, k. \tag{3}$$

Here the stride s_d at recursive depth d is given by

$$s_d = \prod_{j=1}^{d} q_j. \tag{4}$$

This is least when $d = 1$ and $q_1 = 2$ when the stride is 2. However, the stride increases rapidly. Consider, for example, the cyclic reduction orderings in the case $n = 16$. The initialisation is

$$\texttt{A1} \leftarrow (\texttt{1}, \texttt{2}, \texttt{3}, \texttt{4}, \texttt{5}, \texttt{6}, \texttt{7}, \texttt{8}, \texttt{9}, \texttt{10}, \texttt{11}, \texttt{12}, \texttt{13}, \texttt{14}, \texttt{15}, \texttt{16}).$$

The successive pointer arrays are

$$\begin{aligned}
\texttt{p(1)} &= \texttt{8},\ \texttt{q(1)} = \texttt{2},\ \texttt{O(1)} = \texttt{0}, \\
\texttt{A1} &\leftarrow (\texttt{1}, \texttt{3}, \texttt{5}, \texttt{7}, \texttt{9}, \texttt{11}, \texttt{13}, \texttt{15};\ \texttt{2}, \texttt{4}, \texttt{6}, \texttt{8}, \texttt{10}, \texttt{12}, \texttt{14}, \texttt{16}); \\
\texttt{p(2)} &= \texttt{4},\ \texttt{q(2)} = \texttt{2},\ \texttt{O(2)} = \texttt{8}, \\
\texttt{A1} &\leftarrow (\texttt{1}, \texttt{3}, \texttt{5}, \texttt{7}, \texttt{9}, \texttt{11}, \texttt{13}, \texttt{15};\ \texttt{2}, \texttt{6}, \texttt{10}, \texttt{14};\ \texttt{4}, \texttt{8}, \texttt{12}, \texttt{16}); \\
\texttt{p(3)} &= \texttt{2},\ \texttt{q(3)} = \texttt{2},\ \texttt{O(3)} = \texttt{12}, \\
\texttt{A1} &\leftarrow (\texttt{1}, \texttt{3}, \texttt{5}, \texttt{7}, \texttt{9}, \texttt{11}, \texttt{13}, \texttt{15};\ \texttt{2}, \texttt{6}, \texttt{10}, \texttt{14};\ \texttt{4}, \texttt{12};\ \texttt{8}, \texttt{16}).
\end{aligned}$$

In this case the stride is $s_d = 2^d,\ 1 \leq d \leq k - 1$. The corresponding power of 3 stride orderings are

$$\begin{aligned}
\texttt{p(1)} &= \texttt{5},\ \texttt{q(1)} = \texttt{3},\ \texttt{O(1)} = \texttt{0}, \\
\texttt{A1} &\leftarrow (\texttt{1}, \texttt{4}, \texttt{7}, \texttt{10}, \texttt{13};\ \texttt{2}, \texttt{5}, \texttt{8}, \texttt{11}, \texttt{14};\ \texttt{3}, \texttt{6}, \texttt{9}, \texttt{12}, \texttt{15}, \texttt{16}); \\
\texttt{p(2)} &= \texttt{2},\ \texttt{q(2)} = \texttt{3},\ \texttt{O(2)} = \texttt{10}, \\
\texttt{A1} &\leftarrow (\texttt{1}, \texttt{4}, \texttt{7}, \texttt{10}, \texttt{13};\ \texttt{2}, \texttt{5}, \texttt{8}, \texttt{11}, \texttt{14};\ \texttt{3}, \texttt{12};\ \texttt{6}, \texttt{15};\ \texttt{9}, \texttt{16}).
\end{aligned}$$

Here the factors are not exact, and the final subblock does not have stride given by (4).

In the backsubstitution step the recursively defined mappings must be reversed. The implications of this at recursive level d are as follows: the first $(q_d - 1)$ blocks are addressed correctly, but the last which starts at $O_{d+1} + 1$ has been transformed and must be unwrapped. The permutation can also be obtained with a code fragment very similar to the one previously mentioned and is omitted here.

3 Choice of partitioning factors

The question addressed in this section is that of determining a good sequence of partitioning factors $\{p_i,q_i\}$ in order to obtain good vectorization performance. This is treated here under rather restrictive assumptions, but the results provide a reasonable guide in practice. The assumptions made are the following:

1. There is a given partitioning sequence

$$(p_1, q_1), (p_2, q_2), \cdots, (p_k, q_k)$$

where $p_i q_i = p_{i-1},\ i = 1, 2, \cdots, k,\ p_0 = n$. Thus exact factors are assumed so that the q_i can be expressed in terms of the block sizes p_i;

2. The cost of the vector computation is the sum of the vectorized computations at each stage plus the cost of the final sequential component, and can be modelled by

$$C(p_1, p_2, \cdots, p_k) = \sum_{i=1}^{k} \{\frac{p_{i-1}}{p_i} - 1\}\{S + p_i R\} + \gamma p_k$$

where S is the startup cost, R is the asymptotic rate for the computation, and each combines both factorization and backsubstitution components, and γp_k is the cost of the sequential computation for the solution of the final block;

3. The unknowns p_i can be treated as continuous for the purpose of minimizing C.

The necessary conditions satisfied by the optimal factors are

$$0 = \frac{\partial C}{\partial p_i} = \begin{cases} \left(-\frac{p_{i-1}}{p_i^2} + \frac{1}{p_{i+1}}\right) S, & i = 1, 2, \cdots, k-1, \\ -\frac{p_{k-1}}{p_k^2} S - R + \gamma, & i = k. \end{cases} \tag{5}$$

Setting

$$p_{k+1} = \frac{S}{(\gamma - R)}.$$

gives

$$p_i = \sqrt{p_{i-1} p_{i+1}},\ i = 1, 2, \cdots, k.$$

Taking logs to base 2, and setting $\nu_i = \log_2 p_i$ this becomes the difference equation

$$\nu_{i-1} - 2\nu_i + \nu_{i+1} = 0,\ i = 1, 2, \cdots, k,$$

together with boundary conditions

$$\nu_0 = \log_2 n,\ \nu_{k+1} = \log_2 \frac{S}{\gamma - R}.$$

Thus $\nu_i = \alpha + i\beta,\ i = 0, 1, \cdots, k+1$ where

$$\begin{aligned} \alpha &= \log_2 n, \\ \beta &= \frac{1}{k+1} \log_2 \left\{ \frac{S}{n(\gamma - R)} \right\}, \end{aligned}$$

whence

$$q_i = 2^{-\beta},\ p_i = n2^{i\beta}.$$

The first interesting conclusion is that the optimal q_i is independent of i, although the actual value of this constant does depend on k. This gives $p_1 = O(n^{1/2})$, $q_1 = O(n^{1/2})$ when $k = 1$. Typically k would be chosen so that $p_k = \theta \frac{S}{R} = \theta n_{1/2},\ \theta < 1$, is that fraction of $n_{1/2}$ that gives no vector processing gain.Computations carried out with n relatively large with respect to $n_{1/2}$ then indicate that $q_i = q = 2$ is at least a near optimal choice. This suggests that a cyclic reduction style strategy should be used. It is stressed that this can be implemented without n being a power of 2. For example, the following are valid partitioning factors for $n = 523, k = 4, q_i = 2$:

$$p_1 = 260,\ p_2 = 130,\ p_3 = 65,\ p_4 = 32.$$

The final block to be solved sequentially has dimension 36. The corresponding factors when $k = 3, q_i = 3$ are

$$p_1 = 174,\ p_2 = 58,\ p_3 = 19.$$

In this case the final block has dimension 21.

The assumptions in this analysis require that all memory accesses take the same time. In the real world this is unlikely to hold because of memory bank access conflicts which are likely to be worst when the access stride is a power of 2. The situation is easier for odd number strides when access is usually independent of stride at about half the stride 1 rate. This suggests that power of 3 strides, which give C values which are not too far from optimal in our model, provide a simple alternative. A method for the recursive reduction of a tridiagonal matrix which uses power of three stride access is called tricyclic reduction in [1]. It shares the other disadvantages of cyclic reduction when it is required to maintain a tridiagonal structure

4 Implementation

This has been carried out for a single processor of the Fujitsu VPP500 parallel vector computer. Here a vectorized, single processor implementation is appropriate for the case of small m. Initially the algorithm was developed in the full generality described here. Then several stages of optimization were implemented to make better use of the vector pipe. The final code ran at speeds approaching 200 Mflop/s on our test problems. This is about 15% of the VPP500's peak performance. This is quite satisfactory for problems of the inherent low computational density considered (the optimization problems are similar to those encounteres with level one BLAS). The improvements gained over scalar speed are significant, being possibly as much as ten times in the larger problems. The optimization steps are described below.

- Use of explicit indirection doubles memory references and so should be avoided if possible. Also, vectorizing compilers would have trouble optimising explicit indirection references to the pointer array `A1` in the full generality possible here. However, if (3) is satisfied, so that constant stride access is obtained at each recursive level, then the vector loops can be indexed as in the pseudocode fragment. Although this device largely removes indirection from the vector code, and yielded a significant improvement in performance on the VPP, indirection is used in the sequential computation at the bottom recursive level, and here constant stride cannot be assumed in general. At the higher (vectorized) levels the stride must be updated according to (4).

```
/* p=p(d), q=q(d) */
DO i=1,q-1 {
/* addressing of submatrix elements here */
.....................................................
    nn=A1(O(d)+(q-1)*p+1)
    DO j=1,p {
        pk=nn+(j-1)*s(d)
/* vector operations here */
.....................................................
        }
    }
```

- The VPP provides an example where power of 2 strides can cause memory bank conflicts, and the use of power of 3 strides provided a

significant performance improvement of between 25% and 50% in our experiments.

- A further device proved effective. In this the pivotal columns and normalising factors for the current sequence of Householder transformations (vectorizing over p) are copied to temporary storage. This both gains stride 1 access to the copied vectors, and avoids unnecessary divisions.This modification provided a significant further speed-up when used in conjunction with power of three strides giving a further 25%-50% speed-up for the small block (3×3, 4×4) test problems considered in our experiments.

5 Conclusion

Wrap-around partitioning provides an effective means for vectorizing the solutions of systems of linear equations when the matrix has block bidiagonal form and the component submatrices have relatively small dimension. It has the advantage that stable factorization methods which preserve the underlying structure can be used, and this permits the technique to be applied recursively. Further, the recursive implementation does not depend on exact factorization of the order of the current submatrix. Thus it provides a profound generalisation of cyclic reduction which corresponds to one particular special case. It is possible to discuss optimal choice of parameters for the transformation under the standard model of vector computation. This suggests that if n, the block order, is much larger than $n_{1/2}$ then $q_i = 2$ at each stage provides a good strategy provided it avoids memory bank access conflicts. If power of 2 strides are unfavourable, then it is likely that power of 3 strides ($q_i = 3$) would provide a suitable alternative.

6 Acknowledgement

David Singleton of the Australian National University Supercomputer Facility assisted significantly in the development of the vectorizing code. Fujitsu Japan provided the access to the VPP500 under a contract for the development of mathematical software. Helpful comments by the referees have led to improvements in the presentation of the paper.

References

[1] D.Dodson, S.Levin, A tricyclic, tridiagonal equation solver, SIMAX, 13, 1236–1245, 1992.

[2] M.Hegland, On the parallel solution of tridiagonal systems by wrap–around partitioning and incomplete LU factorization, Num. Math., 59, 453–472, 1991.

[3] C.Dun, M.Hegland, M.Osborne, Stable, parallel solution methods for tridiagonal systems of linear equations, CTAC95 Proceedings, to appear.

[4] S.J.Wright, A collection of problems for which Gaussian elimination with partial pivoting is unstable, SISSC, 14, 231–238, 1993.