On Solving Block Toeplitz Systems Using a Block Schur Algorithm[*]

Srikanth Thirumalai, Kyle Gallivan and Paul Van Dooren

April 10, 1995

Abstract

This paper presents a block Schur algorithm to obtain a factorization of a symmetric block Toeplitz matrix. It is inspired by the various block Schur algorithms that have appeared in the literature but which have not considered the influence of performance tradeoffs on implementation choices. We develop a version based on block hyperbolic Householder reflectors by adapting the representation schemes for block Householder reflectors in the literature to the hyperbolic case. The basic algorithm is applicable to symmetric positive definite Toeplitz matrices. Leading evidence is presented that, under certain circumstances, performance gains can be obtained by foregoing some of the Toeplitz structure by using have a block size larger than the actual block size given by the structure of the matrix. This allows the block algorithm to also be used to factor efficiently standard symmetric positive definite Toeplitz matrices. An extension to the algorithm that can be used to solve symmetric Toeplitz systems that are indefinite is also presented. If a singular principal submatrix is encountered during the factorization, the matrix is perturbed and an approximate factorization is obtained. The error introduced into the solution is then reduced to acceptable levels by applying iterative refinement. Typically two steps are sufficient.

1 Introduction

In this paper, computing a factorization of a symmetric block Toeplitz matrix is considered. A matrix $T \in \Re^{mk \times mk}$ is called a block Toeplitz matrix if all blocks $T_{i,j} \in \Re^{m \times m}$ on each diagonal are identical, i.e.:

$$T = \begin{pmatrix} T_{1,1} & T_{1,2} & \dots & T_{1,k-1} & T_{1,k} \\ T_{2,1} & T_{1,1} & T_{1,2} & \dots & T_{1,k-1} \\ T_{3,1} & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ T_{k,1} & T_{k-1,1} & \dots & \dots & T_{1,1} \end{pmatrix} \tag{1}$$

The matrix T above is called a symmetric block Toeplitz if $T_{i,j} = T_{j,i}^T$ for $i, j = 1, \dots, k$. Moreover, T is symmetric.

Given a symmetric positive definite block Toeplitz matrix, an algorithm which computes a lower triangular matrix, $L \in \Re^{mk \times mk}$ such that $A = LL^T$ is derived. The algorithm is a block version of the algorithm described in [5] and [4] and is based on ideas in [8] and [2]. In those papers, algorithms for factoring matrices with small displacement rank using hyperbolic Householder matrices are presented. We present an extension of the work of Cybenko and Berry [4] to symmetric positive definite block Toeplitz matrices based on the use of block hyperbolic Householder matrices to

[*]Available as CSRD Report 1416 and in shortened form in the Proceedings of 1994 ICPP, pp. 274–281.

represent products of hyperbolic Householder reflectors. Block operations are desirable since they are rich in level-3 BLAS operations [6], [7]. The algorithm can also be used to factor symmetric positive-definite Toeplitz matrices by foregoing some of the Toeplitz structure in the matrix and considering it to be a block Toeplitz matrix.

The factorization of symmetric indefinite Toeplitz matrices is handled by a modification to the block algorithm. If the matrix has singular principal submatrices then the matrix is perturbed to obtain an approximate factorization. Iterative refinement is then used to refine the estimated solution. It has been observed that typically two steps of iterative refinement are sufficient to solve the system.

An outline of this paper is as follows. Section 2 contains derivations of the generator matrices for block Toeplitz matrices as defined in [8]. This section also reviews hyperbolic Householder reflectors, introduces some storage and computation efficient ways to construct them and presents the algorithm to factor symmetric positive definite block Toeplitz matrices. Modifications to the algorithm for the indefinite case are also discussed. Section 3 discusses several implementation issues on shared and distributed memory machines Section 4 presents an extension to the Schur algorithm to solve symmetric Toeplitz systems that are indefinite with singular principal minors. The implementation issues for this extended algorithm are identical to those for the symmetric positive definite case that we discuss. Section 5 gives some preliminary results on the Cray-YMP and outlines the work to be done in the future.

2 The classical Schur algorithm

Let T be an $mp \times mp$ symmetric positive definite block Toeplitz matrix with a block size of $m \times m$.

$$T = \begin{bmatrix} \hat{T}_1 & \hat{T}_2 & \dots & \hat{T}_{p-1} & \hat{T}_p \\ \hat{T}_2^T & \hat{T}_1 & \hat{T}_2 & \dots & \hat{T}_{p-1} \\ \vdots & \hat{T}_2^T & \ddots & \ddots & \vdots \\ \hat{T}_{p-1}^T & \vdots & \ddots & \ddots & \vdots \\ \hat{T}_p^T & \hat{T}_{p-1}^T & \dots & \dots & \hat{T}_1 \end{bmatrix}. \tag{2}$$

Let Z be a block right shift matrix defined as:

$$Z = \begin{bmatrix} 0 & I_m & \cdots & 0 & 0 \\ 0 & 0 & I_m & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & I_m \\ 0 & 0 & \cdots & \cdots & 0 \end{bmatrix} \tag{3}$$

The Schur algorithm is based on the fact that the displacement of a block Toeplitz matrix T, defined as $T - Z^T T Z$, has a rank of at most $2m$ [8].

$$T - Z^T T Z = \begin{bmatrix} \hat{T}_1 & \hat{T}_2 & \dots & \hat{T}_{p-1} & \hat{T}_p \\ \hat{T}_2^T & 0 & 0 & \dots & 0 \\ \vdots & 0 & 0 & \ddots & \vdots \\ \hat{T}_{p-1}^T & \vdots & \ddots & \ddots & 0 \\ \hat{T}_p^T & 0 & \dots & 0 & 0 \end{bmatrix}. \tag{4}$$

The derivation of the Schur algorithm to compute the Cholesky factorization of a symmetric positive definite block Toeplitz matrix is outlined below.

Since $\hat{T}_1$ is a symmetric positive definite matrix, we can find its Cholesky factorization $\hat{T}_1 = L_1 L_1^T$, where L_1 is an $m \times m$ lower triangular matrix. Let $T_j = L_1^{-1}\hat{T}_j$. It is easy to see that $T_1 = L_1^T$. We now define two matrices $G_1(T)$ and $G_2(T)$ as follows [8], [2]:

$$G_1(T) = \begin{pmatrix} T_1 & T_2 & T_3 & \dots & T_p \\ 0 & T_1 & T_2 & \dots & T_{p-1} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & T_2 \\ 0 & 0 & \cdots & 0 & T_1 \end{pmatrix} \quad G_2(T) = \begin{pmatrix} 0 & T_2 & T_3 & \dots & T_p \\ 0 & 0 & T_2 & \dots & T_{p-1} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & & \ddots & \ddots & T_2 \\ 0 & 0 & \dots & 0 & 0 \end{pmatrix} \tag{5}$$

from which it follows that

$$T = \begin{bmatrix} G_1^T(T) & G_2^T(T) \end{bmatrix} \begin{bmatrix} I_{mp} & 0 \\ 0 & -I_{mp} \end{bmatrix} \begin{bmatrix} G_1(T) \\ G_2(T) \end{bmatrix} = G^T W_{mp} G \tag{6}$$

where

$$G = \begin{bmatrix} G_1(T) \\ G_2(T) \end{bmatrix} \qquad \text{and} \qquad W_{mp} = \begin{bmatrix} I_{mp} & 0 \\ 0 & -I_{mp} \end{bmatrix}. \tag{7}$$

If we can obtain a transformation matrix U which satisfies the property $U^T W_{mp} U = W_{mp}$ such that $UG = R$, where R is upper triangular, then we have

$$\begin{aligned} T &= G^T W_{mp} G \;=\; G^T U^T W_{mp} U G \\ &= \begin{bmatrix} R^T & 0 \end{bmatrix} \begin{bmatrix} I_{mp} & 0 \\ 0 & -I_{mp} \end{bmatrix} \begin{bmatrix} R \\ 0 \end{bmatrix} \\ &= R^T R, \end{aligned} \tag{8}$$

which gives us the Cholesky factorization of T. The transformation matrix U which satisfies the property $U^T W_{mp} U = W_{mp}$, is called a hyperbolic Householder transformation. The basic properties of hyperbolic Householder reflectors are discussed in the next section. Since the matrix G comprises of two upper triangular block Toeplitz matrix, we show in a later section that considerable computational savings can be obtained by working with a generator matrix defined using the first block rows of G_1 and G_2 as:

$$Gen = \begin{bmatrix} T_1 & T_2 & \cdots & T_{p-1} & T_p \\ 0 & T_2 & \cdots & T_{p-1} & T_p \end{bmatrix}. \tag{9}$$

It can also be seen that the above generator matrix Gen is obtained by a factorization of the displacement of the block Toeplitz matrix into:

$$T - Z^T T Z = Gen^T \begin{bmatrix} I_m & 0 \\ 0 & I_m \end{bmatrix} Gen \tag{10}$$

Notice that when $\hat{T}_1$ is not positive definite we can consider the more general decomposition $T_1 = L_1 \, \Sigma \, L_1^T$, where Σ is some signature matrix with ± 1 on diagonal. This will exist provided $\hat{T}_1$

has nonsingular leading principal submatrices. The blocks T_j are obtained by $T_j = (L_1\Sigma)^{-1}\hat{T}_j$ by Σ and the W_{mp} matrix becomes

$$W_{mp} = \begin{bmatrix} I_p \otimes \Sigma & 0 \\ 0 & -I_p \otimes \Sigma \end{bmatrix} \tag{11}$$

We then again use hyperbolic Householder transformations (now with respect to the new signature matrix W_{mp}) to reduce G to an upper triangular matrix.

3 Hyperbolic Householder transformations

In their paper [4], Cybenko and Berry use hyperbolic Householder transformations to reduce the generator matrix G of a scalar Toeplitz matrix to an upper triangular matrix [4]. We extend their idea to block hyperbolic Householder transformations (required in the block Schur algorithm), using representations very similar to those proposed in [1] and [10].

Let W be a diagonal matrix whose entries are either $+1$ or -1. It is easy to verify that the matrix W satisfies the equalities:

$$W^2 = I \qquad \text{and} \qquad W^T = W. \tag{12}$$

Any matrix U which satisfies the equation $U^TWU = W$, is called a W-unitary matrix. It is easy to see that W-unitary matrices form a multiplicative group, i.e., I is W-unitary, the product of W-unitary matrices is W-unitary, and inverses of W-unitary matrices are W-unitary. The inverse of a W-unitary matrix U is readily obtained as:

$$UWU^T = W \Rightarrow UWU^TW = I \Rightarrow U^{-1} = WU^TW \tag{13}$$

and hence, $\|U\| = \|U^{-1}\|$.

Let x be a column vector such that $x^TWx \neq 0$. A hyperbolic Householder matrix is defined as follows:

$$U_x = W - \frac{2xx^T}{x^TWx}. \tag{14}$$

One easily checks [4] and [9] that U_x is W-unitary, i.e., $U_x^TWU_x = W$. These transformations can be used to map one vector to another as long as they have the same hyperbolic norm, i.e., if $a^TWa = b^TWb$. In our algorithm, we reduce the generator matrix to an upper triangular matrix by successively zeroing elements below the diagonal of columns of the G matrix in (7). Given a column vector u, we would like to find a hyperbolic Householder matrix U_x such that

$$U_xu = -\sigma e_j \tag{15}$$

where e_j is a column vector whose j^{th} element is 1 and other elements are 0 and σ is a constant. We assume here that $e_j^TWe_j = 1$, i.e. the j-th component corresponds to a $+1$ in W. Also the vectors u we consider will have positive hyperbolic norm when the matrix T we decompose, is positive definite. Choosing

$$\sigma = \frac{u_j}{|u_j|}\sqrt{u^TWu} \tag{16}$$

then u and σe_j have the same hyperbolic norm. If we take $x = Wu + \sigma e_j$, we obtain

$$x^TWx = (Wu + \sigma e_j)^TW(Wu + \sigma e_j) = 2(u^TWu + \sigma u^Te_j)$$

and

$$U_x u = Wu - 2\frac{(Wu+\sigma e_j)(Wu+\sigma e_j)^T u}{x^T W x} = Wu - 2\frac{(Wu+\sigma e_j)(u^T W u + \bar{\sigma} e_j^T u)}{2(u^T W u + \bar{\sigma} e_j^T u)} = -\sigma e_j$$

which shows that U_x maps u to $-\sigma e_j$.

4 Block Hyperbolic Householder Representations

If we have to perform a sequence of hyperbolic Householder transformations we could block these transformations together and then apply this block to the appropriate matrices. This allows us to use level-3 BLAS primitives rather than level-2 BLAS operations if we applied the transformations sequentially. Storage efficient ways to block regular Householder transformations are derived in [1] and [10]. We extend these methods to hyperbolic Householder transforms.

Suppose $U = U_r U_{r-1} \ldots U_2 U_1$ is a product of r $n \times n$ hyperbolic Householder matrices. The matrix U can be written in two forms corresponding to the VY form and the YTY^T form derived in [1] and [10]. The two forms of the VY representation differ in the types of primitives they use.

Lemma 4.0.1 *Suppose $U^{(k)} = W^k + V_k Y_k^T$ is a product of k $n \times n$ hyperbolic Householder matrices, where V_k and Y_k are $n \times k$ matrices. If $U_{k+1} = W - \frac{2x_{k+1}x_{k+1}^T}{x_{k+1}^T W x_{k+1}}$ and $z_{k+1} = \frac{-2x_{k+1}^T U^{(k)}}{x_{k+1}^T W x_{k+1}}$, then*

$$U^{(k+1)} = U_{k+1} U^{(k)} = W^{k+1} + V_{k+1} Y_{k+1}^T$$

where $V_{k+1} = [WV_k \;\; x_{k+1}]$ and $Y_{k+1} = [Y_k \;\; z_{k+1}^T]$. We call this the first VY form.

Proof. If $r = 1$ then, $U = U_1 = W - 2x_1 x_1^T/(x_1^T W x_1)$ and we assign $V_1 = x_1$ and $Y_1 = -2x_1/x_1^T W x_1$ in order to have the desired form.

$$\begin{aligned} U_{k+1}\, U^{(k)} &= (W - \frac{2x_{k+1}x_{k+1}^T}{x_{k+1}^T W x_{k+1}})(W^k + V_k Y_k^T) \\ &= W^{k+1} + WV_k Y_k^T - \frac{2x_{k+1}x_{k+1}^T U^{(k)}}{x_{k+1}^T W x_{k+1}} \\ &= W^{k+1} + WV_k Y_k^T + x_{k+1} z_{k+1} \\ &= W^{k+1} + [WV_k \;\; x_{k+1}] \begin{bmatrix} Y_k^T \\ z_{k+1} \end{bmatrix} \\ &= W^{k+1} + V_{k+1} Y_{k+1}^T \qquad \square. \end{aligned}$$

By construction we obtain that V_k is an $n \times k$ lower triangular matrix and Y_k is an $n \times k$ matrix with no sparsity. The computation intensive part in this form is the computation of z_{k+1} and this requires two matrix vector products because $x_{k+1}^T U^{(k)} = x_{k+1}^T W^k - (x_{k+1}^T V_k) Y_k^T$. The total number of operations performed at the $(k+1)^{th}$ step is $(5nk - 2.5k^2 + 2n - 1.5k)$. This operation count does not include the cost to compute $-\frac{2}{x_{k+1}^T W x_{k+1}}$.

Lemma 4.0.2 *Suppose $U^{(k)} = W^k + V_k Y_k^T$ is a product of k $n \times n$ hyperbolic Householder matrices, where V_k and Y_k are $n \times k$ matrices. If $U_{k+1} = W - \frac{2x_{k+1}x_{k+1}^T}{x_{k+1}^T W x_{k+1}}$ and $z_{k+1} = \frac{-2x_{k+1}^T W^k}{x_{k+1}^T W x_{k+1}}$, then*

$$U^{(k+1)} = U_{k+1} U^{(k)} = W^{k+1} + V_{k+1} Y_{k+1}^T$$

where $V_{k+1} = [U_{k+1}V_k \ \ x_{k+1}]$ *and* $Y_{k+1} = [Y_k \ \ z^T_{k+1}]$. *We call this the second VY form.*

Proof. If $r = 1$ then, $U = U_1 = W - 2x_1x_1^T/(x_1^TWx_1)$ and we assign $V_1 = x_1$ and $Y_1 = -2x_1/x_1^TWx_1$ in order to have the desired form.

$$\begin{aligned}
U_{k+1}U^{(k)} &= (W - \frac{2x_{k+1}x^T_{k+1}}{x^T_{k+1}Wx_{k+1}})(W^k + V_kY_k^T) \\
&= W^{k+1} + U_{k+1}V_kY_k^T - \frac{2x_{k+1}x^T_{k+1}W^k}{x^T_{k+1}Wx_{k+1}} \\
&= W^{k+1} + U_{k+1}V_kY_k^T + x_{k+1}z_{k+1} \\
&= W^{k+1} + [U_{k+1}V_k \ \ x_{k+1}]\begin{bmatrix} Y_k^T \\ z_{k+1} \end{bmatrix} \\
&= W^{k+1} + V_{k+1}Y^T_{k+1} \qquad \square.
\end{aligned}$$

In this form V_k and Y_k are both $n \times k$ lower triangular matrices. The computation intensive part in this form is the computation of V_{k+1} and this requires one matrix vector product and one rank-1 update because $(W - \frac{2x_{k+1}x^T_{k+1}}{x^T_{k+1}Wx_{k+1}})V_k = W\ V_k - (\frac{-2x_{k+1}}{x^T_{k+1}Wx_{k+1}})x^T_{k+1}V_k$. The total number of operations performed at the $(k+1)^{th}$ step is $(5nk - 4.5k^2 + 2n - 1.5k)$. Notice that this form has more sparsity and requires fewer operations than the first form but it requires a rank-1 update. On machines where a matrix-vector product can be performed more efficiently than a rank-1 update the first form may be better suited even though it requires more operations.

Lemma 4.0.3 *Suppose* $U^{(k)} = W^k + Y_kT_kY_k^TW^{k-1}$ *is a product of* k $n \times n$ *hyperbolic Householder matrices, where* Y_k *is a* $n \times k$ *matrix and* T_k *is a* $k \times k$ *matrix. If* $U_{k+1} = W - \frac{2x_{k+1}x^T_{k+1}}{x^T_{k+1}Wx_{k+1}}$, $a_{k+1} = -\frac{2}{x^T_{k+1}Wx_{k+1}}(x^T_{k+1}Y_kT_k)$ *and* $b_{k+1} = -\frac{2}{x^T_{k+1}Wx_{k+1}}$, *then*

$$U^{(k+1)} = U_{k+1}U^{(k)} = W^{k+1} + Y_{k+1}T_{k+1}Y^T_{k+1}W^k$$

where $Y_{k+1} = \begin{bmatrix} WY_k & x_{k+1} \end{bmatrix}$ *and* $T_{k+1} = \begin{bmatrix} T_k & 0 \\ a_{k+1} & b_{k+1} \end{bmatrix}$.

Proof.
For $k = 1$ it can be seen that $U_1 = W + Y_1T_1Y_1^T$, where $Y_1 = x_1$ and $T_1 = -2/x_1^TWx_1$.

$$\begin{aligned}
U^{(k+1)} &= (W - \frac{2x_{k+1}x^T_{k+1}}{x^T_{k+1}Wx_{k+1}})(W^k + Y_kT_kY_k^TW^{k-1}) \\
&= W^{k+1} + x_{k+1}(-\frac{2}{x^T_{k+1}Wx_{k+1}})(x^T_{k+1}W^k) + (WY_k)T_k(Y_k^TW^{k-1}) \\
&+ \ x_k(-\frac{2}{x^T_{k+1}Wx_{k+1}}x^T_{k+1}Y_kT_k)(Y_k^TW^{k-1}) \\
&= W^{k+1} + x_{k+1}b_{k+1}(x^T_{k+1}W^k) + (WY_k)T_k(Y_k^TW^{k-1}) + x_ka_{k+1}(Y_k^TW^{k-1}) \\
&= W^{k+1} + \begin{bmatrix} WY_k & x_{k+1} \end{bmatrix}\begin{bmatrix} T_k & 0 \\ a_{k+1} & b_{k+1} \end{bmatrix}\begin{bmatrix} Y_k^TW^{k-1} \\ x^T_{k+1}W_k \end{bmatrix} \\
&= W^{k+1} + Y_{k+1}\ T_{k+1}\ Y^T_{k+1}\ W^k \qquad \square.
\end{aligned}$$

In this form Y_k is an $n \times k$ lower triangular matrix and T_k is a $k \times k$ lower triangular matrix. The computation intensive part of this form is the computation of a_{k+1} which requires a matrix-vector update with an $n \times k$ matrix and another with a $k \times k$ matrix. But in both cases the matrices have some sparsity. The number of operations performed at the $(k+1)^th$ step is $(3nk - 2k^2 + n)$. Notice that this form requires less computation than the first two forms and also requires about half the storage for $n \gg k$. The only disadvantage is that the number of operations performed while applying the transformation in this form is more than the other two methods. In some cases on distributed memory machines this form could be useful because the message volume to communicate the transformation is about half the previous two forms.

In Section 5, we discuss the three forms in the context of our algorithm.

5 The Factorization Algorithm

The following algorithm is used to reduce matrix G (7) described in Section 2 to an upper triangular matrix. This algorithm is essentially the same as the one described in [4] except that we are dealing with blocks instead of elements. We describe the algorithm using an example as follows. Let $T = G^T W_{mp} G$ where G and W_{mp} are as shown in (17).

$$G = \left(\begin{array}{c|ccccc}
T_1 & T_2 & T_3 & T_4 & \cdots & T_p \\
0 & T_1 & T_2 & T_3 & \ddots & T_{p-1} \\
0 & 0 & T_1 & T_2 & \ddots & T_{p-2} \\
0 & 0 & 0 & T_1 & \ddots & \vdots \\
\vdots & & & & \ddots & \vdots \\
\hline
0 & \boxed{T_2} & T_3 & T_4 & \cdots & T_p \\
0 & 0 & \boxed{T_2} & T_3 & \ddots & T_{p-1} \\
0 & 0 & 0 & \boxed{T_2} & \ddots & T_{p-2} \\
0 & 0 & 0 & \ddots & \ddots & \vdots \\
\vdots & & & & \ddots & \vdots
\end{array}\right) \quad \text{and } W = \begin{pmatrix} I_{mk} & 0 \\ 0 & -I_{mk} \end{pmatrix} \tag{17}$$

The goal of this algorithm is to reduce G into an upper triangular matrix using block hyperbolic Householder matrices. Since the first column of the generator is already in the right form we only use the generator matrix from the second row down. The first row of the upper submatrix of the generator is the first block row of the triangular factor of the Toeplitz matrix. The first step in this algorithm therefore involves eliminating the first diagonal in the lower half of the generator matrix (the boxed T_2 blocks). If this is done while maintaining the Toeplitz structure of the remaining portion of the matrix (the submatrix from the third row downwards), we can repeat the process on the smaller generator till we triangularize G.

Consider the matrix formed by stacking the second block row of the upper submatrix and the first block row of the lower submatrix as follows

$$G' = \begin{pmatrix} 0 & T_1 & T_2 & T_3 & \cdots & T_{p-1} \\ 0 & T_2 & T_3 & T_4 & \cdots & T_p \end{pmatrix} \tag{18}$$

Let U_1 be a block hyperbolic Householder transformation that eliminates T_2 using T_1. Applying this to $G^{'}$ we get

$$U_1 G' = \begin{pmatrix} 0 & \tilde{T}_1 & \tilde{T}_2 & \tilde{T}_3 & \cdots & \tilde{T}_{p-1} \\ 0 & 0 & \hat{T}_3 & \hat{T}_4 & \cdots & \hat{T}_p \end{pmatrix} \tag{19}$$

The matrix formed by stacking the third row of the upper submatrix and the fourth row of the lower submatrix is just a shifted version of $G^{'}$. Similarly all matrices constructed by stacking the corresponding rows in the two halves of the generator matrix are shifted versions of the $G^{'}$ matrix in (18). Hence, all the work that was needed to zero out the diagonal row of T_2 in the lower submatrix was done in the first step. At this stage, the generator matrix G has a Toeplitz submatrix in its upper half (from the third row onwards) and another Toeplitz submatrix in its lower half as shown below

$$G = \left(\begin{array}{cc|cccc} T_1 & T_2 & T_3 & T_4 & \cdots & T_p \\ 0 & \tilde{T}_1 & \tilde{T}_2 & \tilde{T}_3 & \ddots & \tilde{T}_{p-1} \\ \cline{3-6} 0 & 0 & \tilde{T}_1 & \tilde{T}_2 & \ddots & \tilde{T}_{p-2} \\ 0 & 0 & 0 & \tilde{T}_1 & \ddots & \vdots \\ \vdots & & & & \ddots & \vdots \\ \cline{3-6} 0 & 0 & \boxed{\hat{T}_3} & \hat{T}_4 & \cdots & \hat{T}_p \\ 0 & 0 & 0 & \boxed{\hat{T}_3} & \ddots & \hat{T}_{p-1} \\ 0 & 0 & 0 & 0 & \ddots & \hat{T}_{p-2} \\ 0 & 0 & 0 & \ddots & \ddots & \vdots \\ \vdots & & & & \ddots & \vdots \end{array}\right) \tag{20}$$

The second row of the upper submatrix of G is the second block row of the triangular factor of the Toeplitz matrix. The process is then repeated on the two lower right submatrices of the generator in (20). After $p-2$ steps the generator is completely triangularized.

Notice that in addition to being able to work with only two block rows, we can work with the same two block rows because the reduced generator in the next step has the same lower block row but the upper block row is shifted by one block to the right. Before this shift is made the upper block row must be stored in the right place in the triangular factor of the original Toeplitz matrix. At the first step of the algorithm, this reduced matrix which we refer to as the generator matrix is:

$$Gen = \begin{pmatrix} T_1 & T_2 & T_3 & \ldots & T_p \\ 0 & T_2 & T_3 & \ldots & T_p \end{pmatrix}. \tag{21}$$

Also, we see that in the first step T_1 is upper triangular because by construction $T_1 = L_1^T$. The diagonal elements of T_1 are sequentially used to zero out all the elements in the corresponding column of the lower block (T_2). This implies that at each step of the algorithm the block hyperbolic Householder matrices are computed using vectors that have one non-zero element in their upper half and a non-zero lower half. This means that the V, Y matrices in the first two forms and the Y matrix in the third form have more sparsity than usual. This is discussed in more detail in the next section.

6 Implementation

6.1 Overview

A simple implementation of the algorithm has three phases.

1. Generating the hyperbolic Householder transformation U given the pivot block and the block below it to be eliminated. For example, consider the matrix

$$G' = \begin{pmatrix} 0 & T_1 & T_2 & T_3 & \cdots & T_{p-1} \\ 0 & T_2 & T_3 & T_4 & \cdots & T_p \end{pmatrix}$$

   T_1 is the pivot block and T_2 is the block to be eliminated. The matrix U is a block hyperbolic Householder transformation of size $2m \times 2m$, where m is the dimension of each block.

2. Applying the block transformation U to the portion of the matrix to the right of the pivot block column

$$U\,G' = U \begin{pmatrix} 0 & T_1 & T_2 & T_3 & \cdots & T_{p-1} \\ 0 & T_2 & T_3 & T_4 & \cdots & T_p \end{pmatrix} = \begin{pmatrix} 0 & \tilde{T}_1 & \tilde{T}_2 & \tilde{T}_3 & \cdots & \tilde{T}_{p-1} \\ 0 & 0 & \tilde{T}_3 & \tilde{T}_4 & \cdots & \tilde{T}_p \end{pmatrix}$$

   and copying the upper block row of the generator to the appropriate location in the triangular factor of the Toeplitz matrix. If R_2 is the second block row of the upper triangular factor of T, then

$$R_2 = \begin{pmatrix} 0 & \tilde{T}_1 & \tilde{T}_2 & \tilde{T}_3 & \cdots & \tilde{T}_{p-1} \end{pmatrix} \tag{22}$$

3. Shifting the first row of blocks one block to the right

$$G'_{next} = \begin{pmatrix} 0 & 0 & \tilde{T}_1 & \tilde{T}_2 & \cdots & \tilde{T}_{p-2} \\ 0 & 0 & \tilde{T}_3 & \tilde{T}_4 & \cdots & \tilde{T}_p \end{pmatrix}$$

Depending on the architecture of the machine and parameters such as the problem size and structure of the matrix (block size m) variations of this general implementation are chosen. The next three subsections discuss several implementation issues concerning the three phases.

6.2 Phase 1

In Phase 1, the transformation matrix U is constructed from a sequence of hyperbolic Householder reflectors using either the VY or the YTY^T representation. The sparsity pattern of the pivot block and the block below it to be zeroed out are shown in the following figure.

The block hyperbolic Householder transformation, U needed to zero out the lower block in Figure 1, consists of a series of hyperbolic householder transformations $U_1, U_2, \ldots, U_m$ applied in that order. Each transformation U_k uses the (k,k) diagonal element of the upper block to zero out the elements of the k^{th} column below it. At the k^{th} step, the vector u_k as discussed in Section 2 has the form $(0,\ldots,0,u_{k,k},0,\ldots,0,u_{m+1,k},\ldots,u_{2m,k})$. All the transformations $U_1,\ldots,U_k$ can be blocked using either the two VY forms, the YTY^T form or just combined to form $U^{(k)}$ where

$$U^{(k)} = U_k U_{k-1} \ldots U_1 = W^k + V_k Y_k^T = W^k + Y_k T_k Y_k^T W^{k-1} \tag{23}$$

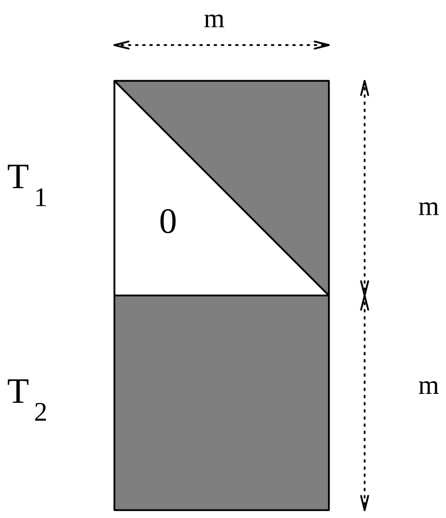

Figure 1: Sparsity pattern of the pivot block and the block below it.

and

$$W = \begin{bmatrix} I_m & 0 \\ 0 & -I_m \end{bmatrix}. \tag{24}$$

The sparsity pattern of $U^{(k)}$ is shown in Figure 2. If the block hperbolic Householder transformation is stored in factored form using the first VY form (requiring 2 matrix vector products), then the sparsity patterns of V and Y are shown in Figure 3. If the second form (requiring 1 matrix vector product and 1 rank-1 update), then the sparsity pattern for V is the same as that of Y in Figure 3 and vice versa. If the YTY^T form is chosen, then the sparsity patterns of the corresponding matrices are shown in Figure 4.

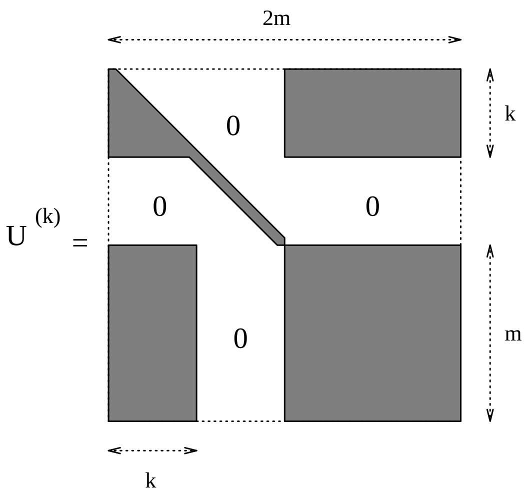

Figure 2: Sparsity pattern of $U^{(k)}$.

For each representation of the block hyperbolic Householder transformation we have different computational costs associated with producing the representation scheme and applying the transformation to the remainder of the generator. We refer to the cost associated with the production

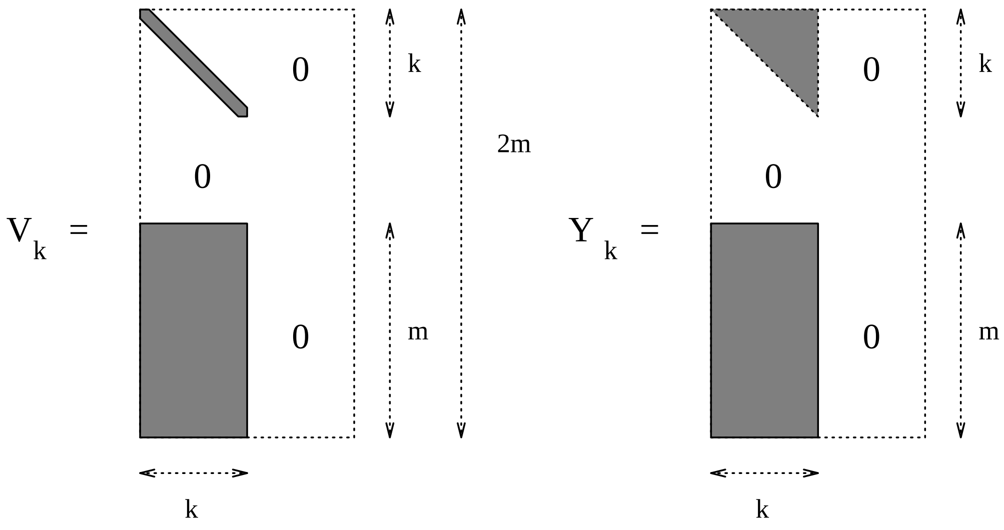

Figure 3: Sparsity pattern of V_k and Y_k where $U^{(k)} = W^k + V_k Y_k^T$.

of the block hyperbolic Householder reflector as the "blocking flops" and the cost associated with the application of the reflector to the remainder of the generator as the "application flops".

Each step in the generation of the V, Y or the Y, T matrices requires some BLAS1 routines such as dotproducts and triads and some BLAS2 routines such as matrix-vector products and rank-1 updates. If the block size m is very large, then on machines with hierarchical memory like the Alliant FX/8 or the Cedar multiprocessor a two level blocking scheme [7] can be used where the hyperbolic Householders are blocked every k steps and the block transformations are applied to the remaining portion of the pivot block or the entire generator matrix. If the block size is small then the generation of V, Y or Y, T can be carried through till the m^{th} step before applying it to the generator matrix.

Let us consider the case where the individual hyperbolic Householder transformations till the k^{th} step are combined to produce $U^{(k)}$. Computing x_j and $-2/(x_j^T W x_j)$ requires $(3m + 8)$ flops. Forming U_1 requires $(m^2 + 7m + 11)$ flops. At the j^{th} step $(j = 2, \ldots, m)$, the flops needed to compute $U^{(j)}$ from U_j and $U^{(j-1)}$ are $(4m^2 + 4mj + 2m + j + 10)$. Hence, the total flops to compute $U^k = U_k \ldots U_1$ are

$$\begin{aligned} Total flops &= m^2 + 7m + 11 + \sum_{j=2}^{k}(4m^2 + 4mj + 2m + j + 10) \\ &= 4m^2k + 2mk^2 - 3m^2 + 4mk + 0.5k^2 + m + 10.5k \\ &= 6m^3 + 1.5m^2 + 11.5m \qquad \text{For } k = m. \end{aligned} \tag{25}$$

If the VY representation of the block hyperbolic Householder reflector is chosen, then for the first form, the cost of computing V_j and Y_j from U_j and $(W^{j-1} + V_{j-1}Y_{j-1}^T)$ is $(4mj + j^2 + m + 9 - j + \lceil j/2 \rceil - 1)$ flops. For $j = 1$, the cost of computing V_1 and Y_1 is $(4m + 9)$ flops. The total cost of computing V_k and Y_k using the first form is

$$Total flops \approx 4m + 9 + 0.5mk + \sum_{j=2}^{k}(4mj + j^2 + m + 9 - j/2)$$

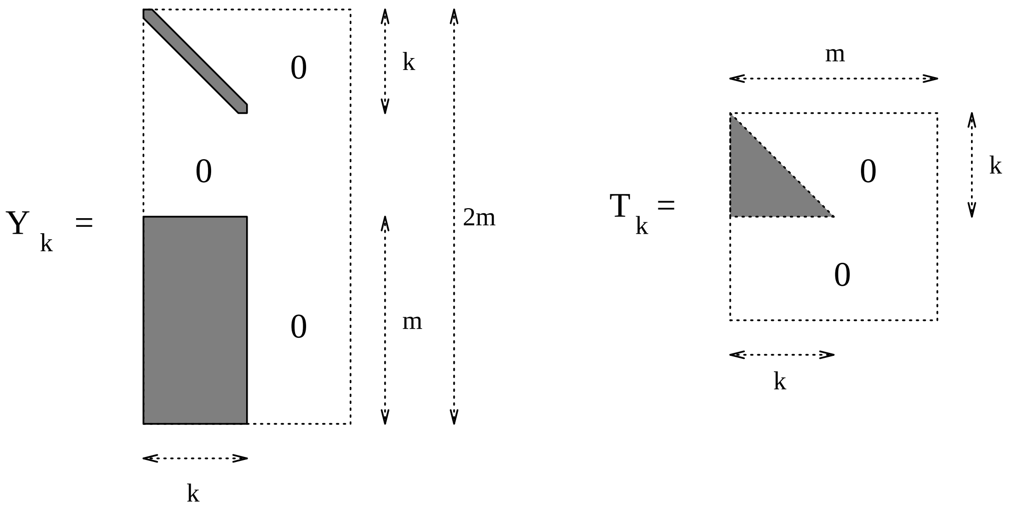

Figure 4: Sparsity pattern of Y_k and T_k where $U^{(k)} = W^k + Y_k T_k Y_k^T W^{k-1}$.

$$\begin{aligned} &\approx 2mk^2 + 0.333k^3 + 3.5mk + 0.25k^2 - m + 9k \\ &\approx 2.333m^3 + 3.75m^2 + 8m \qquad \text{For } k = m. \end{aligned} \tag{26}$$

If the VY form computed using 1 matrix vector product and 1 Rank-1 update is chosen, then the cost to compute V_1 and Y_1 would be $(4m+9)$ and the cost to compute V_j and Y_j from U_j and $(W^{j-1} + V_{j-1}Y_{j-1}^T)$ is $(4mj + j + m + 8)$ flops. The total cost of computing V_k and Y_k using the first form is

$$\begin{aligned} Total flops &= 4m + 9 + \sum_{j=2}^{k}(4mj + j + 0.5m + 8) \\ &= 2mk^2 + 2.5mk + 0.5k^2 - 0.5m + 8.5k \\ &= 2m^3 + 3m^2 + 8m \qquad \text{For } k = m. \end{aligned} \tag{27}$$

If the YTY^T representation is chosen, the cost to compute Y_j and T_j from Y_{j-1} and T_{j-1} and x_j is $(2mj + 2m + 9 + j^2 - j + \lceil j/2 \rceil - 1)$ flops. The total cost of computing Y_k and T_k from $x_{1]}, \ldots, x_k$ is

$$\begin{aligned} Total flops &= 3m + 8 + \sum_{j=2}^{k}(2mj + 2m + 9 + j^2 - j + \lceil j/2 \rceil - 1) + 0.5mk \\ &\approx mk^2 + 0.333k^3 + 3.5mk + 0.25k^2 + 9k - m - 1 \\ &\approx 1.333m^3 + 3.75m^2 + 8m - 1 \end{aligned} \tag{28}$$

From the above calculations of total flops to compute a blocking representation, it can be seen that the YTY^T form is the least expensive. This advantage over the other representations may be offset by the total flops required to apply the block hyperbolic Householder reflector in this form to the rest of the generator. Also, it can be seen that the naive blocking scheme of combining all the reflectors $U_1, \ldots, U_k$ is far more expensive than any of the other block representations. The two VY forms, albeit more expensive, may be used because the cost of applying the transformation to the rest of the generator in this form is slightly less expensive.

It is also possible to apply the hyperbolic Householder transformations sequentially to the generator matrix (without blocking them into U, V, Y or Y, T). This avoids the extra computation involved in blocking the transformations. Application of these transformations to the generator matrix involves BLAS2 operations such as matrix-vector products. On machines that have a memory hierarchy such as the Alliant FX/8 and the Cedar multiprocessor, in some cases, it is usually beneficial to block the transformations so that application of these block transformations involves BLAS3 operations.

6.3 Phase 2

Having outlined the various schemes to block hyperbolic Householder reflectors, we discuss the cost of applying these block reflectors to the rest of the generator. Consider the generator matrix at the j^{th} step of the algorithm. If the block Toeplitz matrix has r blocks to begin with, then at the j^{th} step, the remainder of the generator has a size of $2m \times (r-j-1)m$. Let $p = (r-j-1)$. If we choose a two level blocking strategy till k where $1 \leq k \leq m$, then applying a transformation of the type shown in Figure 2, to the generator requires

$$\begin{aligned} Total flops &= 2m^3p + 4m^2pk + mpk^2 + mpk \\ &= 7m^3p + m^2p \end{aligned} \tag{29}$$

If the first VY form is chosen, then the cost of applying the block reflector to a $2m \times mp$ generator would be

$$\begin{aligned} Total flops &= 4m^2pk + mpk^2 + m^2p + 3mpk && \text{If k is odd.} \\ &= 4m^2pk + mpk^2 + 3mpk && \text{If k is even.} \\ &= 5m^3p + 4m^2p && \text{If } k = m \text{ and m is odd.} \\ &= 5m^3p + 3m^2p && \text{If } k = m \text{ and m is even.} \end{aligned} \tag{30}$$

In the second VY form, the Y matrix has the same sparsity pattern as the V matrix of the first form and vice versa. The cost of applying the block reflector in this form to the generator requires

$$\begin{aligned} Total flops &= 4m^2pk + mpk^2 + m^2p + 2mpk && \text{If k is odd.} \\ &= 4m^2pk + mpk^2 + 2mpk && \text{If k is even.} \\ &= 5m^3p + 3m^2p && \text{If } k = m \text{ and m is odd.} \\ &= 5m^3p + 2m^2p && \text{If } k = m \text{ and m is even.} \end{aligned} \tag{31}$$

The cost of applying the block reflector in the YTY^T form to the rest of the generator is

$$\begin{aligned} Total flops &= 4m^2pk + mpk^2 + m^2p + 4mpk \\ &= 5m^3p + 5m^2p \end{aligned} \tag{32}$$

From the above calculations it can be seen that the second VY representation is the best for most values of k. In terms of operation counts the U matrix is almost always costlier than the VY representation. For some values of m using U_m instead of V_m, Y_m might be preferred since the U matrix is bigger than either V_m or Y_m and the operations might be performed at a higher rate. On some distributed memory machines, if the cost of communicating the block reflector to the other processors is so high that it offsets the cost of applying the transformation to the generator, we may choose the YTY^T representation.

6.4 Phase 3

Phase 3 of each step just involves shifting the upper row of blocks in the generator matrix one block to the right. On shared memory machines, this phase could be avoided if we apply the transformation matrices to the right portions of the matrix. This in-place implementation also requires the U matrix to be split up into 4 quadrants and the V, Y matrices to be split into two $m \times m$ matrices. This not only avoids the shift of the upper block row of the generator matrix but also allows the sparsity of the transformation matrices to the exploited. In our experiments on the Cray Y-MP we use this in-place implementation of the algorithm.

On distributed memory machines where portions of the generator are assigned to processors, the shift operation might include passing the local portions of the generator to a neighboring processor. In the next subsection we suggest three different data distribution schemes for such machines. These three schemes have different amounts of data movement during the shift operation. The cost of communicating with a neighboring processor is an important parameter in deciding how to map the Schur algorithm to a linear array of processors.

6.5 Performance Tradeoffs

Of course, the choices described above can change as the architectural parameters, cost functions, and the efficiency of the set of computational primitives available vary. For example, the YTY^T representation of U requires half the storage of the other methods, so on distributed memory machines with the columns of the generator matrix distributed in a block-cyclic manner or on any machine with a significant hierarchy of memory access costs, it would probably be preferred due to the reduced communication costs and decreased number of operations performed in a less parallel portion of the algorithm.

The most important tradeoff consideration is the choice of the block size. Fortunately, we have found that this issue lends itself to a relatively straightforward analysis similar to that described in [7] when details of the architectures influence on the design details of the BLAS primitives are available. If this is not the case the analysis can be modified to use an empirical characterization of the primitives performance. (This approach was taken when we analyzed the effect of block size choice on our Cray Y-MP implementations.) The key issue that must be decided is whether or not the block size used by the block Schur algorithm, m_s, should be the same as the block size that defines the block Toeplitz structure of the matrix, m. In the description of the algorithm above no distinction was made without loss of information. However, if the architecture heavily favors BLAS3 primitives compared to BLAS1 and BLAS2 primitives or if BLAS3 primitives applied to matrices with larger dimensions have sufficient performance advantage over those applied to matrices with smaller dimensions[1], it may be advantageous to treat a block Toeplitz matrix with block size m as if it had a block size m_s where $m < m_s$. This effectively ignores some of the Toeplitz structure of the original matrix in an attempt to improve performance. The cost is an approximately linear increase in the number of operations, i.e., $\approx 4m_s n^2$ vs. $\approx 4mn^2$, where n is the size of the block Toeplitz matrix. Of course, if m is large enough then the performance of the primitives is sufficient so that $m = m_s$ can be used. In fact, in such a case where n is not overwhelmingly larger than m it may be necessary to take $m_s < m$ in order to allow overlap of the production of U with the update of the remainder of the generator matrix.

[1]Such a situation may not only occur due to architectural considerations. Arguments with extreme shapes can often cause unanticipated performance trends due to a lack of optimization in a particular implementation of a computational primitive library. This was encountered on the Cray Y-MP with the BLAS3 primitives.

The block Schur algorithm has also been implemented on the Cray T3D which is a distributed memory machine. We consider the machine to be a linear array of processors and distribute the generator over the processors in a cyclic fashion. We have studied three different data distribution schemes on the T3D. In the first scheme each processor is assigned one block of the matrix in a cyclic fashion. If the number of blocks in the block Toeplitz matrix is much larger than the number of processors, it may be better to assign a few contiguous blocks to one processor and proceed in a cyclic fashion. This is the second data distribution scheme. If the number of blocks is small and if the block size m is large another data distribution scheme may be chosen where each block is split up among a few contiguous processors to increase parallelism. A performance analysis of the various data distribution schemes is underway to decide the optimal scheme given the problem and machine parameters such as the block size, the number of blocks, the number of processors and the rates of computation and communication.

7 Performance analsis on high performance architectures

In the previous section, various schemes to block the hyperbolic Householder transformations were suggested. The choice of a particular blocking scheme would depend on the architecture of the machine on which the algorithm is implemented. A detailed performance analysis of this algorithm on various high performance architectures is required to make optimal implementation choices. Future work in this area includes performance analysis of the block Schur algorithm on scalar machines, shared memory multiprocessors such as the Cedar and the Cray C90 and distributed memory multiprocessors such as the Cray T3D. In this section we present preliminary implementations results of the block Schur algorithm for symmetric positive definite block Toeplitz matrices on the Cray T3D.

7.1 Implementation on distributed memory multiprocessors

On parallel machines where the memory is physically distributed across all the processors, distributing the data across the processors such that there is minimal data movement across processors is crucial. This is referred to as the data distribution problem. While trying to reduce data movement, care should be taken not to severely reduce the parallelism in the implementation. On most machines this results in a tradeoff between data communication and parallelism.

For the Schur algorithm, in addition to choosing the right blocking scheme to block the hyperbolic Householder transforms, it is important to lay out the generator across the processors in such a way that data movement during the algorithm is reduced without severely affecting the parallelism. In this subsection we discuss three data distribution schemes and discuss their usefulness given various problem and block sizes.

Consider a block Toeplitz matrix T of size $mp \times mp$ with a block size of $m \times m$. The generator for this matrix is of size $2m \times mp$ and can be considered to have p block columns that corresponds to the block structure of the Toeplitz matrix. We refer to size of the block Toeplitz matrix, $N = mp$, as the problem size. The data distrubution problem in the implementation of the Schur algorithm deals with the way in which the generator is distributed across the processors. Let us consider a distributed memory multiprocessor with NP processors to be a linear array of processors. The three ways of distributing this generator are

Version 1: Each block is assigned to a processor in a cyclic manner.

Version 2: A group of b adjacent blocks is assigned to a processor.

version 3: Each block is divided among $spread = 1/b$ adjacent processors.

Figure 5 shows the three data distribution schemes on a 4 processor machine with processing elements (PEs) P_0, P_1, P_2 and P_3. In versions 1 and 2 at any step of the Schur algorithm the pivot block column of the generator wholly resides in one processor. The pivot block column of the generator is used to compute a block hyperbolic Householder transform which is then communicated to the other processors through a broadcast operation. Depending on the cost of a broadcast operation and the preferred primitives on one processor of the machine an optimal blocking scheme is chosen. In the following subsections we examine the three data distribution schemes more closely wth respect to the communication and computation tradeoffs. In all the three versions we assume a compute/communicate paradigm with explicit barrier synchronization between each phase of the Schur algorithm.

7.1.1 Version 1

In this version each block column of the generator resides completely within a processor. The block hyperbolic Householder transformation is computed by the processor that has the pivot block. This transformation is then broadcast to all processors that have the rest of the generator. If p_{active} is the number of block of the rest of the generator (at the i^{th} step of the Schur algorithm the rest of the generator has $p - i$ blocks), then $k_active = \lceil p_{acive}/NP \rceil$ is the number of blocks on each processor. At each step of the Schur algorithm, $O(m^2)$ data is broadcast, $O(k_{active}m^3)$ computation is done and $O(k_{active}m^2)$ data is shifted to the right neighbor.

For moderate block sizes this is the preferred data distribution scheme. If m is very small, then distributing the generator this way would result in a poor compuatation to communication ratio. This can be bettered by using the second data distribution scheme.

7.1.2 Version 2

In this version b adjacent blocks reside within a processor. The parallelism in the algorithm drops by a factor of b and the processors run out of work faster than in version 1. It can also be seen that in this version the amount of data sent to the rigth neighbor during the shift operation also reduces by a factor of b. resulting in an improved computation to communication ratio. This indicates that there is a tradeoff between parallelism and communication. This implies that for a combination of block size m, problem size N and machine size NP, there exists an optimal number of adjacent blocks b that should be assigned to each processor. A detailed performance analysis of this data distribution scheme is necesary to obtain this optimal number. For small block sizes, typically $m \leq 4$, this version provides best results.

7.1.3 Version 3

In this version a block is sperad out among $1/b$ processors. This implies that at each step of the Schur algorithm a set $1/b$ processors have a fraction of the pivot block equal to bm columns. If the block size is large compared to the number of of blocks in the Toeplitz matrix, then each block may be distributed among $1/b$ processors to increase the parallelism in the problem. The communication time in this version is more than version 1 because the number of broadcasts increases by a factor of $1/b$. For some block sizes, the increased parallelism resulting from splitting the blocks could offset

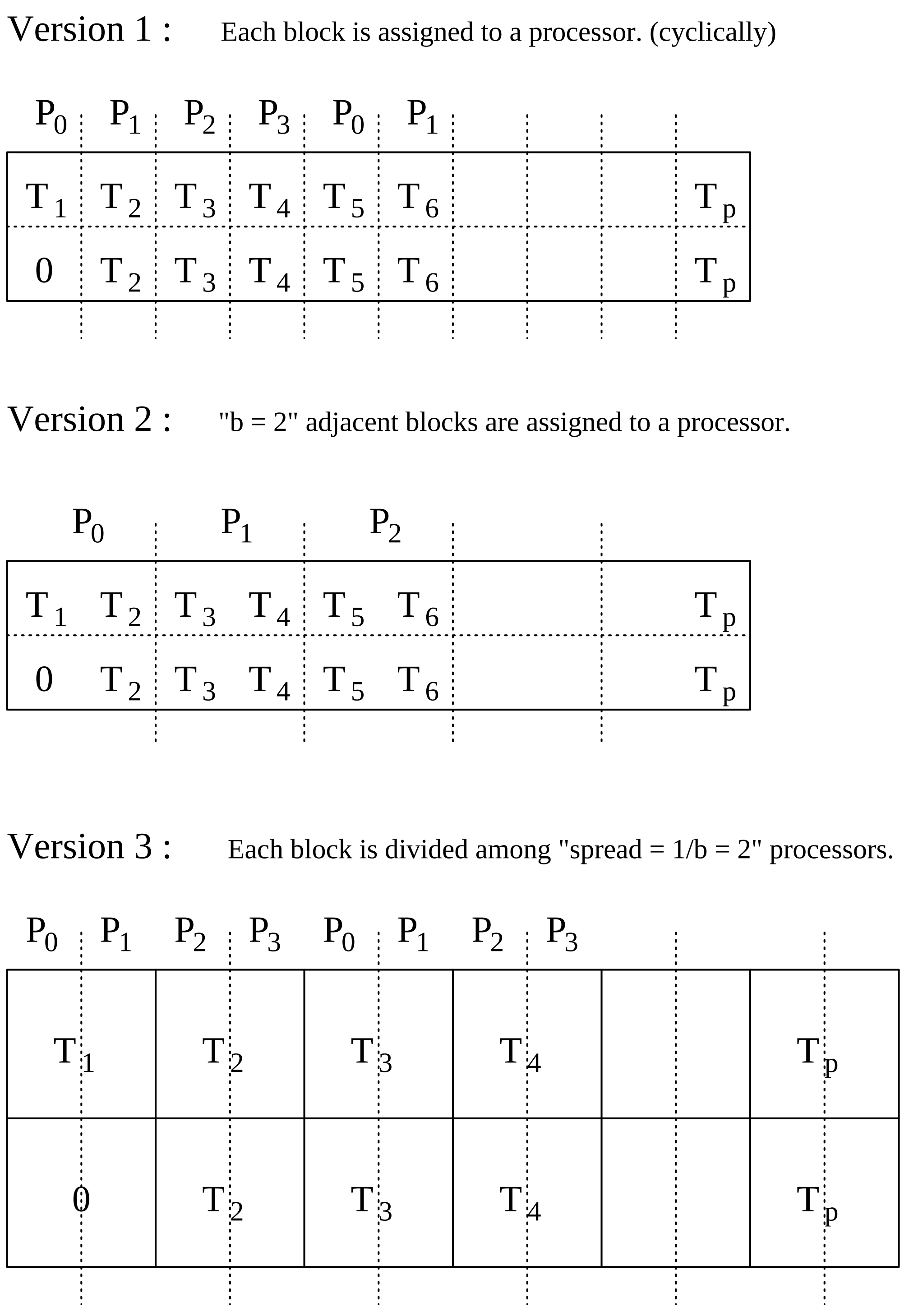
Version 1 : Each block is assigned to a processor. (cyclically)
P0 P1 P2 P3 P0 P1
T1 T2 T3 T4 T5 T6 Tp
0 T2 T3 T4 T5 T6 Tp
Version 2 : "b = 2" adjacent blocks are assigned to a processor.
P0 P1 P2
T1 T2 T3 T4 T5 T6 Tp
0 T2 T3 T4 T5 T6 Tp
Version 3 : Each block is divided among "spread = 1/b = 2" processors.
P0 P1 P2 P3 P0 P1 P2 P3
T1 T2 T3 T4 Tp
0 T2 T3 T4 Tp

Figure 5: Data distribution schemes for distributed memory machines

the increase in communication time. This results again in a tradeoff between commnunication and parallelism. For a combination of block size m, problem size N and machine size NP, there exists an optimal number of processors over which each block of the generator should be spread. This can be obtained by a detailed performance analysis.

7.1.4 Overview of the Cray T3D architecture

The Cray T3D is a massively parallel processor in which the processors are connected in the form of a 3D Torus. Each processor is therefore connected to 6 neighbors and has a 300 MB/s data transfer rate to each neighbor. The processing element in the T3D is the DEC chip 21064 (Alpha) with the following features - 150 Mflops peak, 150 MHz clock, dual issue superscalar, 64 bit integer and floating point and 8 KBytes of direct mapped, write through data cache with a cache line of 4 words (1 word = 8 bytes).

The communication library used in the experiments was the *Shmem library* which is based on low latency ($1\mu s$) *puts* and *gets* from remote memory. The *shmem_put* routine uses no buffers. It writes directly to/from one processors memory to the others without any interference by the other processor. The broadcast was done using the *shmem_broadcast* routine.

7.1.5 Experiment 1

Consider a 4096×4096 point Toeplitz matrix ($m = 1$). Let $NP = 16$, the time to factor the matrix in seconds using version 1 and 2 are showm in Figure 6

In this example $N \gg NP$. Therefore, increasing b initially does not affect the parallelism. But the communication cost gets significantly reduced.This causes a sharp initial fall in the time to factor. The best time is obtained at $b = 16$. When b increases to 32 and 64, the reduction in parallelism outweighs the reduced communication and execution times start increasing. If the shift operation on the T3D were slower, then the optimal b would be greater than 16 whereas if the shift operation were quicker, we would not have seen a significant reduction in execution times with increasing b.

7.1.6 Experiment 2

Consider a 4096×4096 block Toeplitz matrix with $m = 8$. Let $NP = 64$.The time to factor the matrix using all three versions of the data distribution schemes are shown in Figure 7. For $b > 1$, version 2 is used . For $b < 1$, we use version 3 and for $b = 1$ version 1 is used. It can be seen that for moderate block sizes if the parallelism is adequate (i.e. $N \gg NP$), then version 1 provides the fastest factorization scheme. On the T3D experiments show that for block sizes around 8 version 1 is the fastest implementation. If the communication time in broadcasts and shifts increases then the range of block sizes for which version 1 provides the best factorization times increases.

7.1.7 Experiment 3

Consider a 4096×4096 block Toeplitz matrix with $m = 32$. Let $NP = 64$. The time to factor the matrix using versions 1 and 3 of the data distribution schemes are shown in Figure 8. For $b < 1$, we use version 3 and for $b = 1$ version 1 is used. The parallelism in the problem is not very high if we use version 1. It can be seen that increasing the parallelism by increasing the number of processors over which to distribute each block results in improved performance. The optimal number of processors over which to distribute each block in this example is 8. Further

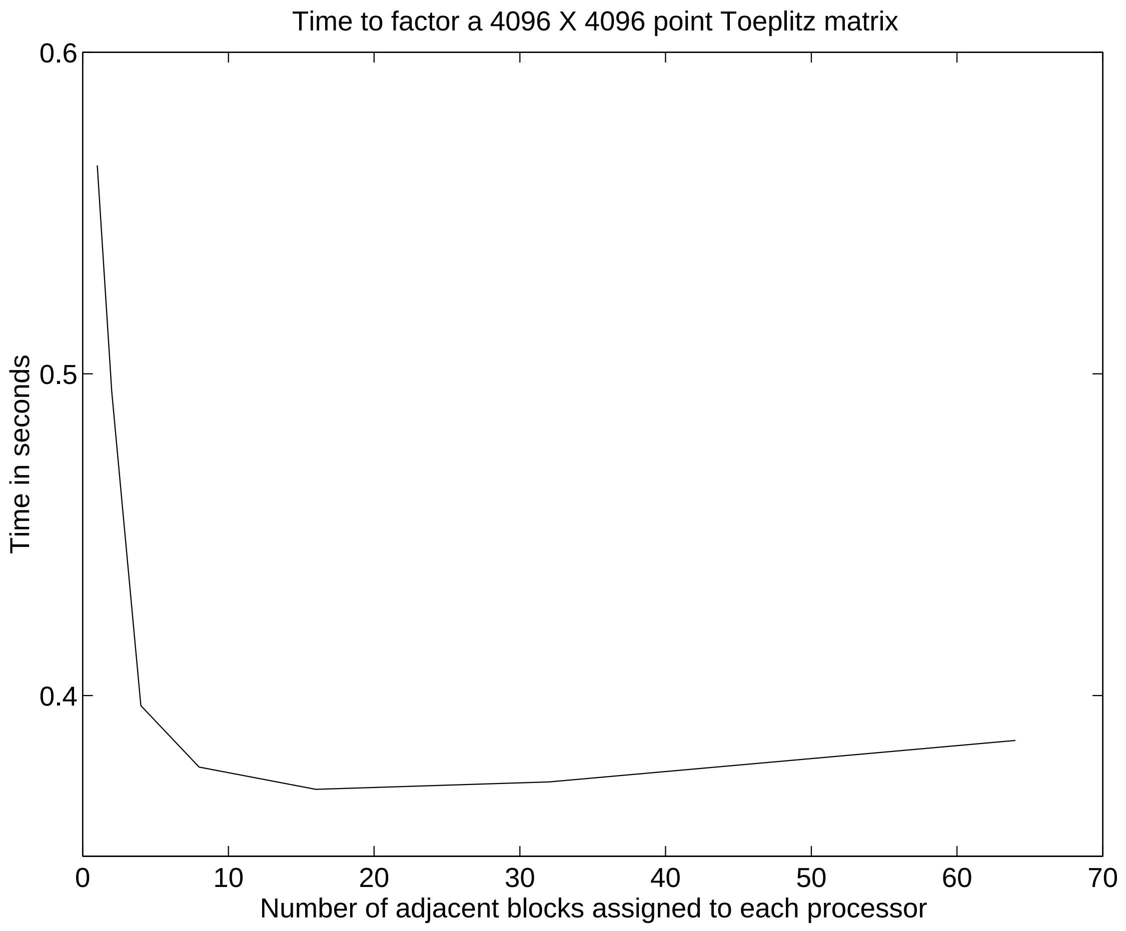

Figure 6: Time to factor a 4096×4096 point Toeplitz matrix on a 16 processor T3D with varying b (number of adjacent blocks assigned to each processor)

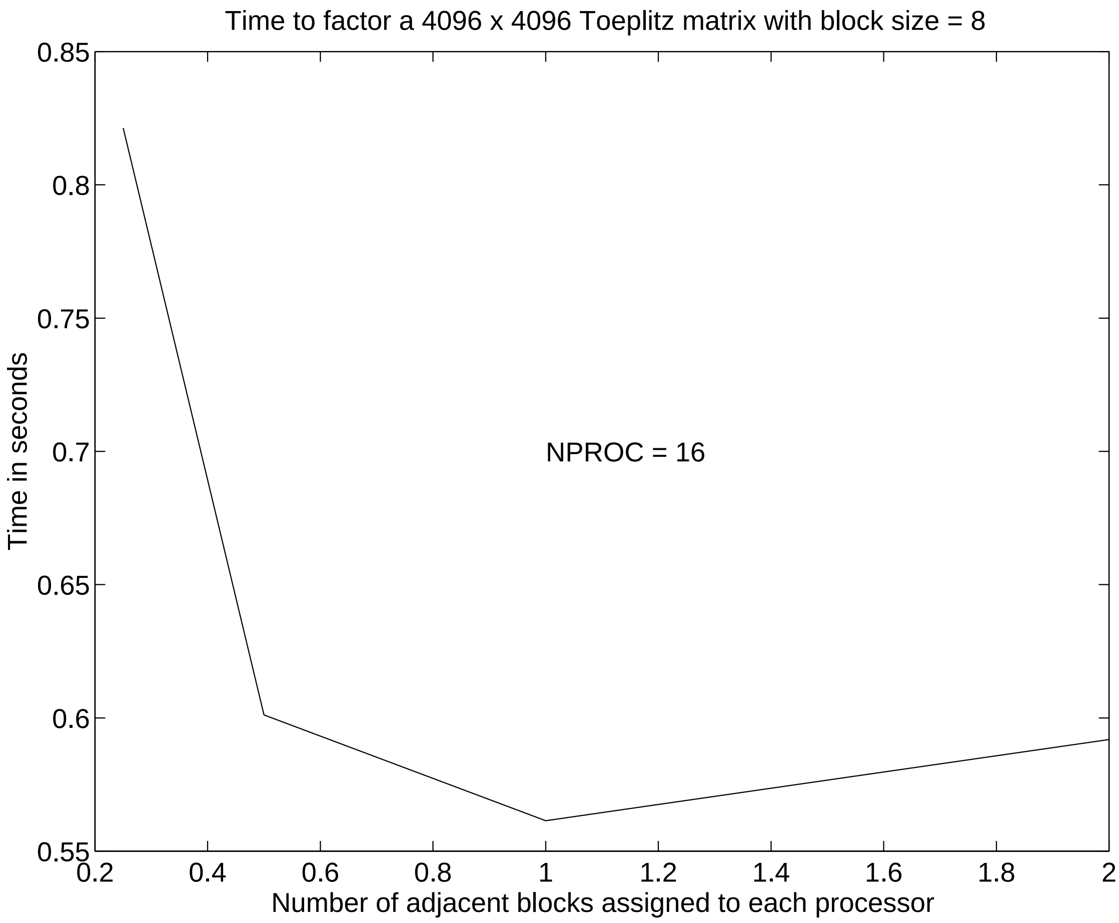

Figure 7: Time to factor a 4096×4096 block Toeplitz matrix with block size $m = 8$ on a 16 processor T3D with varying b (number of adjacent blocks assigned to each processor)

increase in the number of processors over which to spread each block results in higher broadcast costs and offsets the increased parallelism. If the cost of broadcast on the T3D were to reduce then the optimal number of processors over which to distribute a block to increase parallelism would increase.

The above experiments serve to demonstrate the ranges of block sizes over which the three data distribution schemes are useful. Based on the problem size, block size and number of processors one can empirically determine the optimal data distribution scheme. Another interesting observation on the T3D was that for a fixed problem size and fixed number of processors, the time to factor the block Toeplitz matrix having a larger block size was lesser than that of a smaller block size. This was seen for block sizes of 2 and 4 (Figure 9). Since the complexity of the Schur algorithm increases linearly with increasing block size, the performance of the algorithm would have to be improved by more than a factor of 2 for the effect to be observed. A likely explanation for this behavior on the T3D is the following. The amount of data broadcast at each step (the V and Y) matrices are very small for both block sizes 2 and 4. So the time spent in broadcasts is almost identical. The cache line of the T3D is 4 words long. Hence the application of the hyperbolic

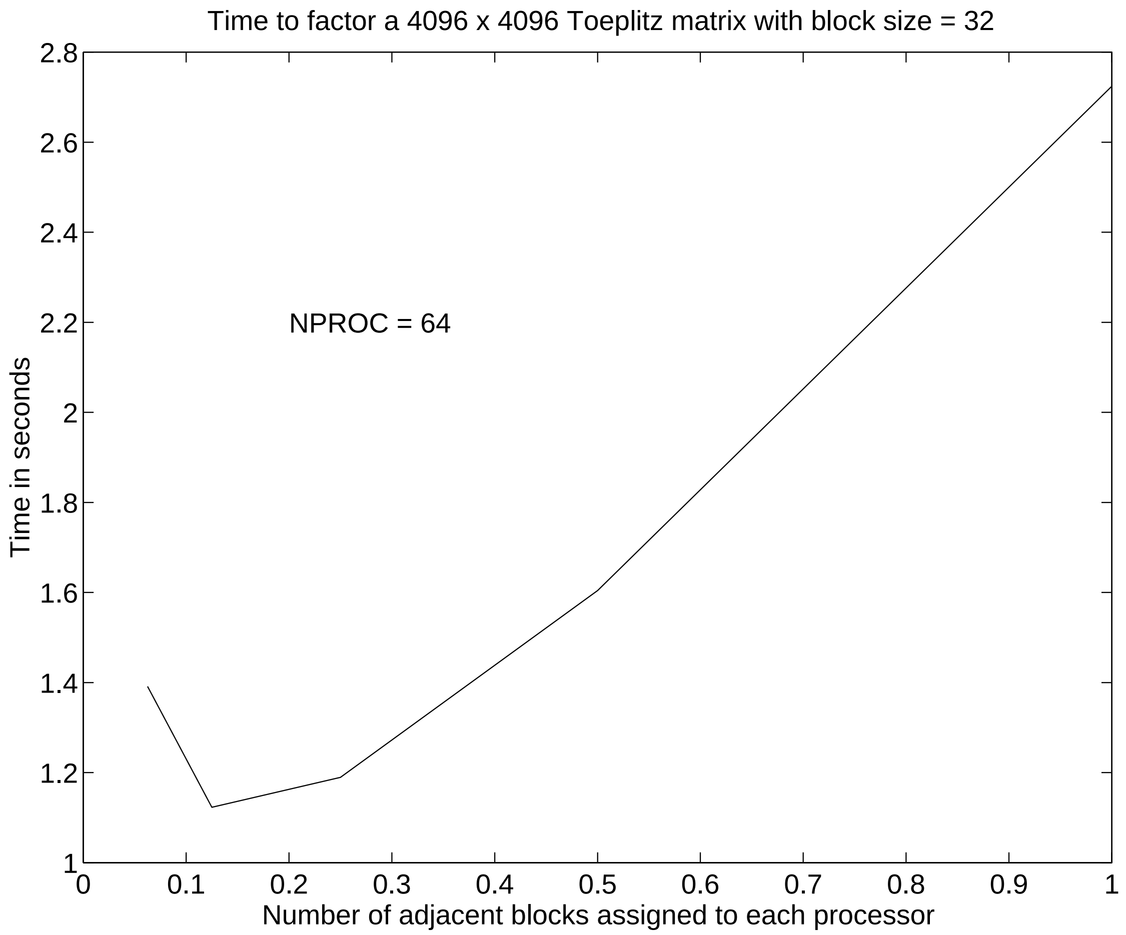

Figure 8: Time to factor a 4096×4096 block Toeplitz matrix with block size $m = 32$ on a 64 processor T3D with varying b (number of adjacent blocks assigned to each processor)

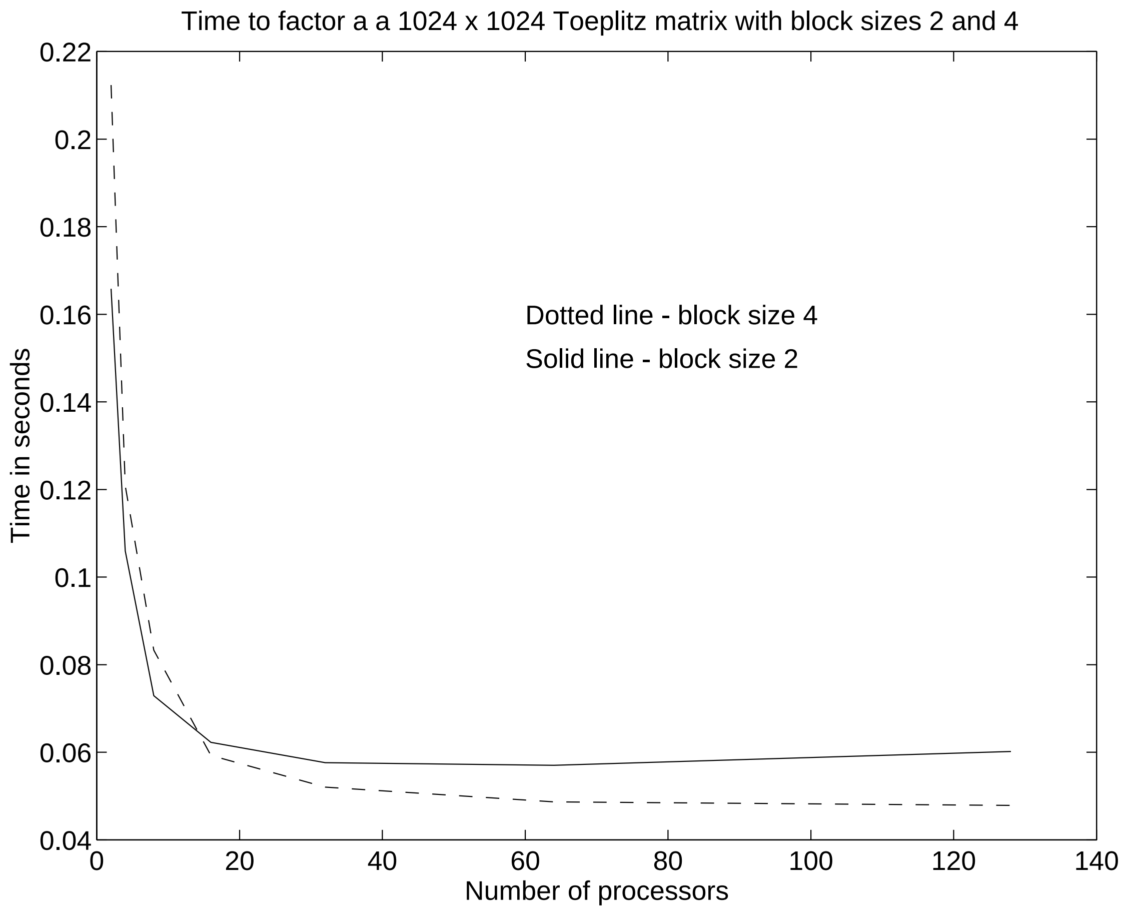

Figure 9: Time to factor a 1024×1024 block Toeplitz matrix with block sizes 2 and 4

transformation to the generator is more efficient for a block size of 4 than 2. The increase in the time spent in the application of V and Y in the case where $m = 4$ is,therefore, not twice that in the case where $m = 2$. Since the number of steps in the Schur algorithm reduce by a factor of two for a block size of 4 over a block size of 2, the number of times a synchronization primitive is invoked in the algorithm during the shift operation reduces by a factor of 2. Since the time spent in synchronizing a small number of processors is insignificant, we see that for small number of processors $m = 4$ takes longer than $m = 2$. But when the number of processors increases, the time spent in synchronization becomes significant and we see that $m = 4$ is faster than $m = 2$.

8 Iterative refinement

The previous sections presented an algorithm to find the factorization of symmetric positive definite Toeplitz matrices. When the matrix is symmetric but indefinite (without any singular principal minors) then a similar blocked version of the algorithm of Cybenko and Berry can be implemented. In the indefinite case, the algorithm involves interchanging rows such that the pivot element always lies along the diagonal row of the pivot block. This is the extra work that needs to be done in the indefinite case. If the number of times the interchanging is done is small, then the performance of the algorithm will be similar to the positive definite case.

If the Toeplitz matrix has singular principal minors then the factorization process cannot be continued. To solve such systems we propose an extension to the algorithm. We introduce a perturbation in the pivot element (the element that is used to zero out all the other elements) of the column of generator matrix whose hyperbolic norm is zero. This perturbation allows us to continue the factorization process but introduces numerical instability in the algorithm. One way to circumvent the possible numerical instability of the Schur algorithm is to use iterative refinement on the system of equations. The perturbation technique has been used in [3] for the Levinson algorithm. They use the approximate factorization as a preconditioner in the conjugate-gradient algorithm. The iterative refinement technique we propose requires significantly lesser work than the preconditioned conjugate-gradient algorithm per iteration.

8.1 Extension to the Schur algorithm

Let us consider the system of equations $Tx = b$, where T is an indefinite symmetric Toeplitz or Toeplitz matrix with singular principal submatrices. Using the perturbation technique we obtain an approximate factorization

$$T + \delta T = LDL^T \tag{33}$$

We solve the system of equations to get x_1

$$LDL^T x_1 = b \tag{34}$$

and then compute the residual r_1

$$r_1 = -Tx_1 + b \tag{35}$$

Using the correction term Δx_1 obtained from

$$LDL^T \Delta x_1 = r_1 \tag{36}$$

we improve the estimated solution by

$$x_2 = x_1 + \Delta x_1 \tag{37}$$

The algorithm then becomes,

```
Construct $LDL^T = T + \delta T$ using the extended Schur algorithm.
Solve $LDL^T x_1 = b$, and set $r_1 = -Tx_1 + b$.
for $i = 1, \ldots$
    Solve $LDL^T \Delta x_i = r_i$
    if $\|\Delta x_i\| < tol\,\|x_i\|$ then stop
    else
        $x_{i+1} = x_i + \Delta x_i$
        $r_{i+1} = -Tx_{i+1} + b$
    endif
endfor
```

From the error analysis of [11] we know that the computed quantities $\overline{x}_i$, $\Delta\overline{x}_i$ and $\overline{r}_i$, satisfy the following identities

$$\overline{r}_i = -T\overline{x}_i + b + \delta\overline{r}_i \quad = \quad r_i + \delta\overline{r}_i \qquad \text{with} \qquad \|\delta\overline{r}_i\| \leq \epsilon_i \|T\|\,\|\overline{x}_i\| \tag{38}$$

$$(LDL^T + \delta T_i)\Delta\overline{x}_i = \overline{r}_i \qquad \text{with} \qquad \|\delta T_i\| \leq \eta_i \|L\|^2 \, \|D\| \tag{39}$$

where ϵ_i, η_i are of the order of the machine precision of the computer. From these equations we obtain

$$(T + \delta T + \delta T_i)\Delta\overline{x}_i = b - T\overline{x}_i + \delta\overline{r}_i \tag{40}$$

and after some rewriting

$$\begin{array}{rrcl} & r_{i+1} & = & b - T(\overline{x}_i + \Delta\overline{x}_i) \;=\; (\delta T + \delta T_i)\Delta\overline{x}_i - \delta\overline{r}_i \\ \text{or also} & r_{i+1} & = & (\delta T + \delta T_i)(T + \delta T + \delta T_i)^{-1}(r_i + \delta\overline{r}_i) - \delta\overline{r}_i \\ & & = & \Delta T_i(T + \Delta T_i)^{-1} r_i - T(T + \Delta T_i)^{-1}\delta\overline{r}_i \end{array}$$

where the terms δT and δT_i which are typically of the same order have been grouped together in ΔT_i. Defining $M_i = \Delta T_i \, T^{-1}$ we have

$$r_{i+1} = M_i(I + M_i)^{-1} r_i - (I + M_i)^{-1}\delta\overline{r}_i \tag{41}$$

If we can now obtain that $\max_i \|\Delta T_i \, T^{-1}\| = \gamma \ll 1$ then the above equation is a difference equation that will converge linearly, with a factor $\beta = \gamma(1-\gamma)$, to a steady state value of

$$\|r_\infty\| \approx \frac{1}{1-\beta}\frac{1}{1-\gamma}\|\delta r_{\max}\| = \frac{1}{1-2\gamma}\|\delta r_{\max}\| \leq \frac{\epsilon_{\max}}{1-2\gamma}\|T\|\|x\| \tag{42}$$

Since our assumption is that γ is small, this final residual is about what one can expect from a stable algorithm. If we obtain that $\gamma = \sqrt[k]{\epsilon}$ then the number of iteration steps to get "convergence" to this result would be k.

8.2 Approximate decomposition

As shown above it is important to bound $\|\delta T \, T^{-1}\|$ in the construction of the factorization. Since LDL^T is only an approximate decomposition of T (but an exact decomposition of $T + \delta T$), we have the freedom to perturb T so as to obtain a better bound for $\delta T \; T^{-1}$. In this subsection we show how to obtain this by selective perturbations introduced in the Schur algorithm. Similar ideas have independently been developed for the Levinson algorithm by Concus and Saylor [3].

At the i^{th} step of the Schur algorithm we apply a block hyperbolic Householder transformation U_i to the generator $G'(i)$ to get $G'(i+1)$ i.e., $U_i G'(i) = G'(i+1)$ The corresponding decomposition for the Toeplitz matrix is

$$T = \begin{bmatrix} G_1^T(i) & G_2^T(i) \end{bmatrix} \hat{U}_i W \hat{U}_i \begin{bmatrix} G_1(i) \\ G_2(i) \end{bmatrix} = \begin{bmatrix} G_1^T(i+1) G_2^T(i+1) \end{bmatrix} W \begin{bmatrix} G_1(i+1) \\ G_2(i+1) \end{bmatrix}, \tag{43}$$

where $\hat{U}_i$ is essentially a block arrangement of identity matrices and U_i blocks. Hence,

$$\|\hat{U}_i\|_2 = \|U_i\|_2 \quad \text{and} \quad \|\hat{U}_i^{-1}\|_2 = \|U_i^{-1}\|_2. \tag{44}$$

If we now perturb the generator matrix $G'(i)$ by a perturbation of norm $\delta\|G(1)\|_2$ then the equivalent perturbation $\|\Delta G(1)\|$ of $G(1)$ is bounded by

$$\|\Delta G(1)\| \leq \delta \|U_1^{-1}\|_2 \cdots \|U_{i-1}^{-1}\|_2 \|G_(1)\|$$

and that of T is proportional to $\delta||U_1^{-1}||_2 \cdots ||U_{i-1}^{-1}||_2||T||$. In other words, the norms of the inverses of the block transformations performed thus far, act as a growth factor in the back-transformations of the perturbation to the original matrix. Another factor that we have to be concerned about is that the transformation U_i for which the δ perturbation was done will have a norm of approximately $1/\delta$ and the norm of the next generator $G(i+1)$ will be increased by that amount. Numerical errors in subsequent steps will thus be proportional to this value and when transforming these back to the original matrix T we find again that we have to keep

$$\epsilon||U_1|| \cdots ||U_{n-1}||$$

bounded. Experience has shown that for each perturbation δ performed at a certain step i, there will be two block transformations of norm approximately $1/\delta$. The total error due to one perturbation is:

$$\frac{||\Delta T||}{||T||} = \delta + \frac{\epsilon}{\delta^2} \tag{45}$$

We choose δ so as to minimize the above expression. The value of δ that minimizes the above expression is $\sqrt[3]{2\epsilon}$ or $\delta \approx \sqrt[3]{\epsilon}$. This gives us:

$$\begin{aligned} \gamma &= ||\Delta T\, T^{-1}|| \\ &\leq ||\Delta T||||T^{-1}|| \\ &\leq \frac{||\Delta T||}{||T||} cond(T) \\ &\approx \delta + \frac{\epsilon}{\delta^2} \text{ (If } T \text{ is well conditioned)} \\ &\approx \sqrt[3]{\epsilon} \text{ (If we set } \delta = \sqrt[3]{\epsilon}) \end{aligned} \tag{46}$$

The subsequent number of steps of iterative refinement would be 3. The above analysis holds true if we perturb the generator matrix just once.

Let us consider the case when we need to perturb twice. Let δ_1 and δ_2 be the two perturbations at steps i and j respectively. The total perturbation to the original Toeplitz matrix can be expected to be of the following order

$$\begin{aligned} ||\delta T|| &= (\ \delta_1\ ||U_1^{-1}|| \ldots ||U_{i-1}^{-1}|| + \delta_2\ ||U_1^{-1}|| \ldots ||U_{j-1}^{-1}||\ )\ ||T|| \\ &\approx (\delta_1 + \frac{\delta_2}{\delta_1^2})||T|| \end{aligned} \tag{47}$$

The numerical error due to the block transformations of norms approximately equal to $1/\delta_1^2$ and $1/\delta_2^2$ is

$$\begin{aligned} Numericalerrors &= \epsilon||U_1|| \ldots ||U_{n-1}||||T|| \\ &= \frac{\epsilon}{\delta_1^2\delta_2^2} \end{aligned} \tag{48}$$

The total error due to the two factors is

$$\frac{||\Delta T||}{||T||} = \delta_1 + \frac{\delta_2}{\delta_1^2} + \frac{\epsilon}{\delta_1^2\delta_2^2} \tag{49}$$

The above expression is minimized by choosing $\delta_1 = \sqrt[9]{\epsilon}$ and $\delta_2 = \sqrt[3]{\epsilon}$. This means that we would require 9 iterations to get to machine precision. It is impossible to know ahead of time how many perturbations one requires to carry on with the Schur algorithm. If, upon performing one perturbation of $\sqrt[3]{\epsilon}$ we see during the Schur algorithm that another perturbation is needed, we would have to backtrack to the first perturbation and change the value of δ_1 from $\sqrt[3]{\epsilon}$ to $\sqrt[9]{\epsilon}$. This is usually very wasteful of computation. Also, if the number of times the generator needs to be perturbed increases, the accuracy is lost very quickly and we might have to look for other ways to handle such cases. From our experiments with Toeplitz matrices, we have observed that even for Toeplitz matrices with several singular minors one perturbation is sufficient. We haven't been able to construct cases requiring more than one perturbation. So, in practice, it might be safe to assume that we need to perturb the generator only once and the above analysis holds good.

We now present an example of a symmetric Toeplitz matrix with a singular principal minor to substantiate our analysis. Consider a symmetric Toeplitz matrix with the first row defined as

$$T(1,1:6) = \begin{pmatrix} 1.0000 & 1.0000 & 0.5297 & 0.6711 & 0.0077 & 0.3834 \end{pmatrix} \tag{50}$$

This matrix has the singular principal minor $\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$. The generator at the second step is

$$G_{(2)} = \begin{pmatrix} 0.0000 & 1.0000 & 1.0000 & 0.5297 & 0.6711 & 0.0077 \\ 0.0000 & 1.0000 & 0.5297 & 0.6711 & 0.0077 & 0.3834 \end{pmatrix}$$

It can be seen that the hyperbolic norm of the pivot column $(1.0000 \quad 1.0000)^T$ is zero and hence we introduce a perturbation of $\sqrt[3]{10^{-16}} \approx 10^{-5}$ so that the perturbed generator becomes

$$G_{(2)}^{perturbed} = \begin{pmatrix} 0.0000 & -1.0000049999875 & -1.0000 & -0.5297 & -0.6711 & -0.0077 \\ 0.0000 & -1.0000000000000 & -0.5297 & -0.6711 & -0.0077 & -0.3834 \end{pmatrix}$$

We then continue the algorithm with this modified generator. The hyperbolic Householder transformation computed using the column $(-1.0000049999875 \quad -1)$ is

$$U_{(2)} = \begin{pmatrix} -100633.458695874 & 100316.727766335 \\ 100316.727766335 & -100001.000001565 \end{pmatrix}$$

which has a norm of $1/\delta$. The transformation $U_{(4)}$ also has a norm of $1/\delta$. The norm of $\delta T.T^{-1}$ for this example is $2.8753e-05$. If we consider $x = (1\ 1\ 1\ 1\ 1\ 1)^T$ then

$$b = \begin{pmatrix} 3.5919 & 4.2085 & 4.7305 & 4.7305 & 4.2085 & 3.5919 \end{pmatrix}^T.$$

We solve for x_1 from $LDL^T x_1 = b$ and we find $||(x - x_1)|| = 3.6375e - 05$. Using our iteration scheme, we find that after one step of iterative refinement, $||(x - x_2)||$ is $6.9982e - 10$ and after the second step of iterative refinement we have $||(x - x_3)|| = 1.5877e - 14$ which is approximately equal to the machine precision.

9 Experimental results and future work

Preliminary versions of the algorithm have been implemented on the Cray Y-MP using four processors. Fig. 10 shows a plot of the performance of the algorithm for different block sizes with increasing problem sizes. It can be seen that for large problem sizes the performance improvement as m_s increases is superlinear implying that using a block size different from the structural block size m may be warranted. As was noted earlier, this is largely due to the less than optimal performance observed for the BLAS3 primitives on the Cray Y-MP when applied to a matrix product involving a relatively small square matrix and a short and wide matrix. The performance trends observed were predictable by a block size analysis based on an empirical characterization of the performance of the BLAS3 primitives on products with the shapes of interest. The performance trends for the algorithm with the modification for indefinite systems are similar when there an excessive number of perturbations are not required.

Performance studies and block size analyses of this algorithm on the Cray T3D are in progress. Three different data distribution schemes varying in the distribution of the generator over the processors are being studied. An analysis of the computation and communication tradeoffs for a given problem size (defined by block size and size of the Toeplitz matrix) and machine size (defined by the number of processors) decides which of the three schemes is best suited.

References

[1] C. Bischof and C. Van Loan. The wy representation for products of householder matrices. *SIAM J. Sci. Stat. Comput.*, 8:s2–s13, 1987.

[2] J. Chun and Thomas Kailath. Generalized displacement structure for block toeplitz, toeplitz block and toeplitz derived matrices. *Informations Systems Lab., Stanford University, CA*, 1988.

[3] Paul Concus and Paul Saylor. A modified direct preconditioner for indefinite symmetric toeplitz systems. *Department of Computer Science, University of Illinois at Urbana Champaign.*

[4] G. Cybenko and M. Berry. Hyperbolic householder algorithms for factoring structured matrices. *SIAM J. Matrix Anal. Appl*, 11:499–520, October 1990.

[5] J.-M. Delosme and I. Ipsen. Parallel solution of symmetric positive definite systems with hyperbolic rotations. *Linear Algebra Appl.*, 77:75–111, 1986.

[6] J. Dongarra, J. DuCroz, I. Duff, and S. Hammarling. A proposal for a set of level 3 basic linear algebra subprograms. *ACM SIGNUM Newsletter*, 22(3):2–14, 1987.

[7] K. A. Gallivan, R. J. Plemmons, and A. H. Sameh. Parallel algorithms for dense linear algebra computations. *SIAM Review*, 32:54–135, 1990.

[8] Thomas Kailath, Sun-Yuan Kung, and Martin Morf. Displacement ranks of matrices and linear equations. *Journal of Mathematical Analysis and Applications*, 68:395–407, 1979.

[9] Charles M. Rader and Allan O. Steinhardt. Hyperbolic householder transforms. *SIAM J. Matrix Anal. Appl.*, 9:269–290, April 1988.

[10] Robert Schreiber and Charles Van Loan. A storage-efficient wy representation for products of householder transformations. *SIAM J. Sci. Stat. Comput.*, 10(1):53–57, January 1989.

[11] J. H. Wilkinson. *The Algebraic Eigenvalue Problem.* Oxford University Press, Oxford, England, 1965.

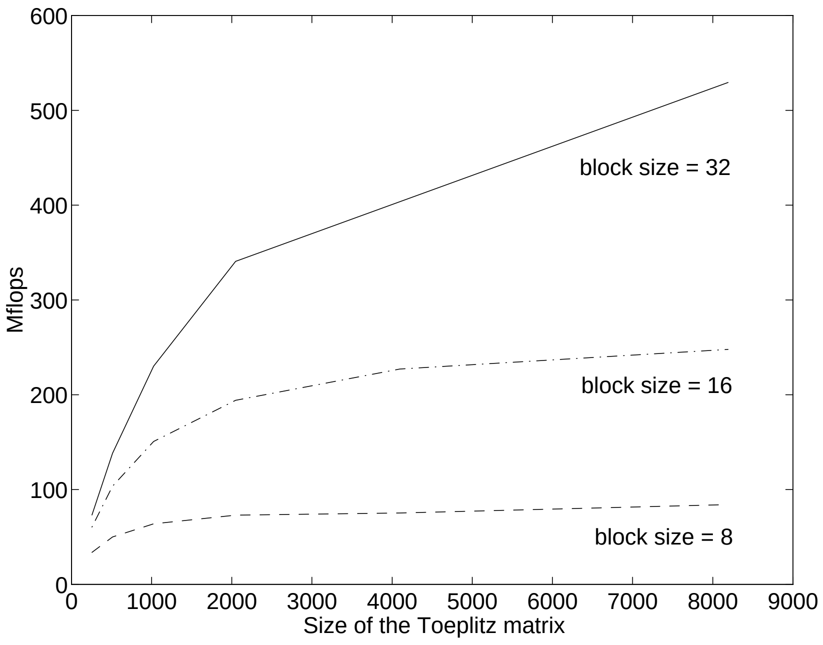

Figure 10: Performance of the block Schur algorithm for factoring symmetric positive definite block Toeplitz matrices with different block sizes on the Cray-YMP.

10 Acknowledgements

We would like to thank Michael Stewart for some useful suggestions on section 4 of this paper. Srikanth Thirumalai and Paul Van Dooren were partially supported by DARPA (Grant COM-NIST-UNIVMINN). Kyle Gallivan was supported by the National Science Foundation under grant number CCR-9120105.