Parallel Algorithms for the Solution of Toeplitz Systems of Linear Equations*

Pedro Alonso[1], José M. Badía[2], and Antonio M. Vidal[1]

[1] Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, cno. Vera, s/n, 46022 Valencia, Spain
{palonso,avidal}@dsic.upv.es

[2] Departamento de Ingeniería y Ciencia de los Computadores, Universidad Jaume I, Campus de Riu Sec, 12071 Castellón de la Plana, Spain
badia@inf.uji.es

Abstract. In this paper we present two parallel algorithms to solve non-symmetric Toeplitz systems of linear equations. The first algorithm performs a modified QR factorization of the matrix by using the generalized Schur algorithm. The second one is based on the transformation of the Toeplitz matrix into a Cauchy-like matrix in order to reduce the communication cost. Both sequential methods have small computational cost. This fact makes it difficult to implement efficient parallel algorithms. We have tested the efficiency and stability of the algorithms on a cluster of personal computers. The results show the speed-up reaches the number of processors in many cases and both algorithms offer an accurate solution of the linear system. Besides, we have used public domain computation and communication libraries in order to get portable codes.

1 Introduction

In this paper we present two new parallel algorithms based on "fast" sequential methods for solving Toeplitz linear systems:

$$Tx = b\ , \tag{1}$$

where $T \in \mathrm{R}^{n\times n}$ is a Toeplitz matrix of the form $T = (t_{ij}) = (t_{i-j})$ for $i, j = 0, \ldots, n-1$, $b \in \mathrm{R}^n$, and $x \in \mathrm{R}^n$ is the solution vector.

Fast algorithms for solving Toeplitz linear systems are based on the displacement rank property of this kind of matrix. There is a group of algorithms called Schur-type that offers more possibilities to implement parallel versions.

Regarding the accuracy of the results, almost all the algorithms that solve Toeplitz systems produce poor results except with strongly regular matrices, that is, matrices with all their leading submatrices well conditioned. Several methods are proposed to improve the solution, including *look-ahead* or refinement techniques [6,3].

* This work was supported by the Spanish project CICYT TIC 2000-1683-C03-03.

Our aim in this work is to offer stable and efficient algorithms for general purpose architectures. Our codes are portable because we extensively use standard libraries like LAPACK [4], ScaLAPACK [5] and BIHAR [10].

The first parallel algorithm presented in this paper solves (1) by means of a modified QR decomposition of T proposed in [3] that improves the accuracy of the solution.

Our second parallel algorithm makes a LU factorization of a Cauchy-like matrix resulting from applying fast trigonometric transformations to the Toeplitz matrix. We exploit Cauchy-like matrices in order to reduce the communication cost avoiding a lot of communications present in the classical fast algorithms.

In the next two sections both parallel algorithms are described. Section 4 includes the experimental analysis and comparison of the algorithms in a cluster of personal computers. Finally, some conclusions are presented.

2 QR Factorization of T

The concept of *displacement structure* was first introduced in [8] to describe the special structure of Toeplitz and Toeplitz-like matrices. Given a symmetric matrix $M \in \mathrm{R}^{n\times n}$, and a lower triangular matrix $F \in \mathrm{R}^{n\times n}$, we call *displacement* of M with respect to F to the matrix $\nabla_F M$ defined as:

$$\nabla_F M = M - FMF^T = GJG^T \; . \tag{2}$$

We say that matrix M has *displacement structure* with respect to F, if the rank r of $\nabla_F M$, is considerably lower than n [9]. Matrix $G \in \mathrm{R}^{n\times r}$ is called *generator* and $J = (I_p \oplus -I_q)$, $r = p + q$, is the *signature* matrix, where p the number of positive eigenvalues of $\nabla_F M$ and q the number of negative eigenvalues.

The Generalized Schur Algorithm (GSA) uses the generator pair (G, J) to factorize matrices with the previous structure in $O(rn)$ operations. Applying GSA to the appropriate matrix we can obtain different factorizations (QR, LU, etc.) of a Toeplitz matrix T. In this paper we use the following matrices

$$M = \begin{pmatrix} T^T T & T^T \\ T & 0 \end{pmatrix} \; , \; F = \begin{pmatrix} Z & 0 \\ 0 & Z \end{pmatrix} \; , \tag{3}$$

where $Z = (z_{ij})_{i,j=1,\ldots,n}$ is the down shift matrix, being $z_{ij} = 1$ if $i + 1 = j$ and 0 otherwise. The QR factorization of a Toeplitz matrix can be obtained by applying n steps of the algorithm GSA to the generator pair (G, J),

$$M = \begin{pmatrix} T^T T & T^T \\ T & 0 \end{pmatrix} = \begin{pmatrix} \mathrm{R}^T \\ Q \end{pmatrix} (I) \begin{pmatrix} R \; Q^T \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 0 & -I \end{pmatrix} \; , \tag{4}$$

such as $T^T T = \mathrm{R}^T R$ where R is upper triangular, $T = QR$ and $QQ^T - I = 0$. The displacement rank of M (4) is 5, and the generator can be found in [3].

However, it is not possible to guarantee the stability of the GSA algorithm and the perfect orthogonality of factor Q. To solve this problem, we have incorporated two basic modifications suggested in [3] to our algorithm. First, $2n$ steps of

$$\begin{array}{r|cccccc|cccccccccccc|c}
 & \multicolumn{6}{c|}{G} & \multicolumn{12}{c|}{L} & \\
\hline
 & 0&0&0&0&0&0 & r&0&0&0&0&0&0&0&0&0&0&0 & \\
 & 0&0&0&0&0&0 & r&r&0&0&0&0&0&0&0&0&0&0 & P_0 \\
\hline
 & 0&0&0&0&0&0 & r&r&r&0&0&0&0&0&0&0&0&0 & \\
 & 0&0&0&0&0&0 & r&r&r&r&0&0&0&0&0&0&0&0 & P_1 \\
\hline
\text{step } 5\rightarrow & g&0&0&0&0&0 & r&r&r&r&\bar{r}&0&0&0&0&0&0&0 & \\
 & g&g&g&g&g&g & r&r&r&r&\bar{r}&r'&0&0&0&0&0&0 & P_2 \\
\hline
 & g&g&g&g&g&g & q&q&q&q&\bar{q}&q'&\delta'&0&0&0&0&0 & \\
 & g&g&g&g&g&g & q&q&q&q&\bar{q}&q'&\delta'&\delta'&0&0&0&0 & P_0 \\
\hline
 & g&g&g&g&g&g & q&q&q&q&\bar{q}&q'&\delta'&\delta'&\delta'&0&0&0 & \\
 & g&g&g&g&g&g & q&q&q&q&\bar{q}&q'&\delta'&\delta'&\delta'&\delta'&0&0 & P_1 \\
\hline
 & g&g&g&g&g&g & q&q&q&q&\bar{q}&q'&\delta'&\delta'&\delta'&\delta'&\delta'&0 & \\
 & g&g&g&g&g&g & q&q&q&q&\bar{q}&q'&\delta'&\delta'&\delta'&\delta'&\delta'&\delta' & P_2 \\
\hline
\end{array}$$

Fig. 1. Example of row block cyclic distribution with $\nu = 2$ of a generator matrix $G \in \mathrm{R}^{18\times 6}$ (case ill-conditioned matrix T), and the triangular matrix $L \in \mathrm{R}^{12\times 12}$ in a mesh of 3×1 processors. The figure shows the PAQR at the middle of the step 5. Entries g are generator entries while r, q and δ denote entries of R^T, Q and Δ respectively. Entries $\bar{r}$ and $\bar{q}$ denote the colummn of L computed at step 5 while entries with $'$ denote values that will be computed in the following steps from 6 to 12.

the GSA algorithm are applied to produce the following triangular factorization:

$$L\hat{L}^T = \begin{pmatrix} R^T & 0 \\ Q & \Delta \end{pmatrix} \begin{pmatrix} R & Q^T \\ 0 & -\Delta^T \end{pmatrix} , \tag{5}$$

so that $(\Delta^{-1}Q)$ is orthogonal and the Toeplitz system (1) can then be solved using, $x = R^{-1}(Q^T\Delta^{-T})\Delta^{-1}b$.

Secondly, if matrix T is ill-conditioned, that is, if $\kappa(T) > 1/\sqrt{\epsilon}$, being ϵ the machine precision, then the algorithm GSA can fail. To avoid this problem a modified matrix M with a displacement rank of 6 is factorized in order to guarantee the backward stability. For a deeper discussion see [3].

The computation of generator G involves matrix-vector products that can be performed in parallel without communications. The generator is distributed cyclically by blocks of ν ($n \bmod \nu = 0$) rows in a one-dimensional mesh of $p \times 1$ processors denoted as P_k, $k = 0, \ldots, p-1$ (see Fig. 1). We use a one-dimensional topology because the generator has only 5 or 6 columns and the operations are applied in parallel on different groups of rows.

The parallel algorithm that we call PAQR (Parallel Algorithm for the QR decomposition of T) proceeds as follows.

1. Compute generator G.
2. Compute QR factorization, for $i = 1, \ldots, n$,
   a) Processor P_k owning row i of G (g_i) chooses a J-unitary transformation Θ_i ($\Theta_i J \Theta_i = J$) such as $g_i\Theta_i = \begin{pmatrix} x\, 0 \ldots 0 \end{pmatrix}$ iff $g_i^T J g_i > 0$ or $g_i\Theta_i = \begin{pmatrix} 0 \ldots 0\, x \end{pmatrix}$ otherwise, and broadcasts Θ_i.
   b) The rest of processors update their rows $j = i+1, \ldots, 2n$ of G, $G \leftarrow G\Theta_i$. Column i of L is the first (or the last) column of G.

c) Update g, $g \leftarrow Fg$, being g the first (or the last) column of G.

3. Compute $x = R^{-1}(Q^T \Delta^{-T})\Delta^{-1}b$ by calling some routines of PBLAS.

Step 2c implies a shift one position down the i-th to n entries of g on the one hand, and the $n+1$ to $2n$ entries of g on the other hand. This last operation of the iteration i requires a point-to-point communication between adjacent processors, and involves a great amount of the total time of the parallel algorithm. In order to reduce the communication cost, all elements to be sent from a given processor P_k to processor $P_{\mathrm{mod}(k+1,p)}$ are packed in a unique message.

3 LU Factorization of T

We can avoid the shifting operation on each iteration (step 2c) on the PAQR if we transform the Toeplitz matrix into a Cauchy-like matrix. This greatly reduces the communication cost of the parallel algorithm.

We say that C is a Cauchy-like matrix if it is the unique solution of the displacement equation

$$\Omega\, C - C\, \Lambda = G\, H^T \ , \tag{6}$$

where $\Omega = \mathrm{diag}(\omega_1, \ldots, \omega_n)$, $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$, and we call matrices G and H of size $n \times r$ *generators.*

Given a Toeplitz matrix T, its displacement equation can be expressed as

$$Z_{00}\, T - T\, Z_{11}^T = \hat{G}\, \hat{H}^T \ . \tag{7}$$

In the displacement equation (7), $Z_{\varepsilon\psi} = Z + Z^T + \varepsilon e_1 e_1^T + \psi e_n e_n^T$ where e_1 and e_n are the first and last columns of the identity matrix respectively and $\varepsilon, \psi \in \{0, 1\}$. An explicit form for $\hat{G}$ and $\hat{H}^T$ can be found in [1].

There exist real discrete trigonometric transformations associated with the FFT that diagonalize the matrix $Z_{\varepsilon\psi}$ in $O(n\log(n))$ operations. The discrete sine transformation S_{00} (DST-I) diagonalizes Z_{00}, while the discrete cosine transformation S_{11} (DCT-II) diagonalizes Z_{11} [1,7]. Using the mentioned transformations, we can convert the displacement equation (7) into the displacement equation (6).

Given a Cauchy-like matrix C (6), its LU factorization can be obtained where L is a unit lower triangular matrix and U is an upper triangular matrix. The algorithm proceeds as follows. In the first step, the first column of C is computed by solving

$$\Omega C_{:,j} - C_{:,j}\Lambda = GH_{j,:}^T \ .$$

Let us partition C and to define matrices X and Y as

$$C = \begin{pmatrix} d & u \\ l & C_1 \end{pmatrix} \ , \quad X = \begin{pmatrix} 1 & 0 \\ ld^{-1} & I \end{pmatrix} \ , \quad Y = \begin{pmatrix} 1 & d^{-1} \\ 0 & I \end{pmatrix} \ ,$$

then C can be factorized as

$$C = X \begin{pmatrix} d & 0 \\ 0 & C_s \end{pmatrix} Y \ ,$$

where C_s is the Schur complement of C regarding its first element d. Further, let Ω and Λ be conformally partitioned $\Omega = \begin{pmatrix} \Omega_1 \oplus \Omega_2 \end{pmatrix}$ and $\Lambda = \begin{pmatrix} \Lambda_1 \oplus \Lambda_2 \end{pmatrix}$. Applying the transformation $X^{-1}(.)Y^{-1}$ to (6) we obtain the following equation

$$\Omega_2 \, C_s - C_s \, \Lambda_2 = G_1 \, H_1^T \, , \tag{8}$$

where G_1 is the portion of $X^{-1}G$ from the second row down and H_1 is the portion of $Y^{-1}H$ from the second row down. The first column of L in the LU factorization is $\begin{pmatrix} 1 & d^{-1}l^T \end{pmatrix}^T$ while the first row of U is $\begin{pmatrix} d & u \end{pmatrix}$. The process can now be repeated on the displacement equation (8) of the Schur complement of C with respect to d, C_s, to get the second column of L and row of U.

In the algorithm PALU we have used the same unidimensional mesh topology as in the first one. The generators G and H have been distributed cyclically by blocks of ν rows as it was made with generator G in the previous algorithm PAQR (see Fig. 1). The computed lower triangular factors L and U^T are distributed as the factor L in PAQR.

The parallel algorithm performs a block version of the algorithm described above. Let the following partition for generators G and H and matrix C be,

$$\begin{pmatrix} G_1 \\ G_2 \end{pmatrix} , \quad \begin{pmatrix} H_1 \\ H_2 \end{pmatrix} , \quad C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} L_1 \\ L_2 \end{pmatrix} \begin{pmatrix} U_1 & U_2 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 0 & C_s \end{pmatrix} ,$$

where $G_1, H_1 \in \mathrm{R}^{\nu \times 4}$ and $G_2, H_2 \in \mathrm{R}^{(n-\nu)\times 4}$, $L_1, U_1 \in \mathrm{R}^{\nu \times \nu}$ are lower and upper triangular factors respectively, $L_2, U_2^T \in \mathrm{R}^{(n-\nu)\times\nu}$ and C_s is the Schur complement of C regarding its principal submatrix C_{11}. At the first step of the parallel algorithm, processor P_0 (processor having the firsts blocks G_1 and H_1 of the generators) computes the LU factorization of $C_{11} = L_1 U_1$. Next, processor P_0 broadcasts blocks G_1 and H_1 properly updated. The rest of processors receive blocks G_1 and H_1, compute their blocks of factors L_2 and U_2 and update their blocks of G_2 and H_2. G_2 and H_2 are the generators for a displacement equation of C_s of the form (6). The following steps of the algorithm proceed in the same way.

4 Experimental Results

We have performed the experimental analysis of both algorithms on a cluster of personal computers. Each node of this cluster is a Intel Pentium II-300MHz with 128 Mbytes of memory. The nodes are connected through a Myrinet network [2]. The time required for one of these nodes to perform a flop is approximately 1.55×10^{-3} μs. On the other hand, we model by $\beta + n\tau$ the time required to send a message of size n between two nodes. The latency time of the network is $\beta = 62$ μs, while the time to transfer each double precision real value is 0.021 μs.

Table 1 shows the time spent by both parallel algorithms to solve Toeplitz systems of different matrix sizes using only one processor. We can see how the main part of PAQR is devoted to the modified QR decomposition while the largest time spent in PALU is devoted to the LU factorization of a Cauchy–like

Table 1. Time in seconds (percentage of the total time) of each of the three main steps of both algorithms executed in one processor

		PAQR			PALU		
n	$n+1$	Calc. gen.	Mod. QR	System	Calc. gen.	LU	System
1000	$(7 \cdot 11 \cdot 13)$	0.01 (1%)	0.82 (86%)	0.12 (13%)	0.01 (2%)	0.39 (93%)	0.02 (5%)
1200	(1201)	0.03 (2%)	1.36 (88%)	0.16 (10%)	0.27 (29%)	0.61 (66%)	0.04 (4%)
1400	$(3 \cdot 467)$	0.04 (2%)	1.84 (88%)	0.22 (10%)	0.08 (8%)	0.84 (87%)	0.05 (5%)
1600	(1601)	0.05 (2%)	2.38 (87%)	0.30 (11%)	0.47 (29%)	1.10 (67%)	0.06 (4%)
1800	(1801)	0.07 (2%)	3.04 (87%)	0.38 (11%)	0.60 (29%)	1.41 (67%)	0.08 (4%)
2000	$(3 \cdot 23 \cdot 29)$	0.08 (2%)	3.68 (87%)	0.47 (11%)	0.03 (2%)	1.73 (94%)	0.09 (5%)

matrix. The time required to compute the generator in PAQR is almost negligible. However, the study of the generator computation in PALU, that involves the Toeplitz to Cauchy–like translation, shows an interesting behavior. The speed of this process depends on the decomposition in prime factors of $n + 1$ (second column of Table 1). The time spent in this step is little as far as the prime factors are little. The final step of the algorithms, that involves several triangular systems solution and matrix–vector products, are carried out by subroutines of the BLAS library optimized for the target machine and it takes a small percentage of the total time.

One important factor that affects the performance of both parallel algorithms is the block size denoted by ν. The value of ν fixes the number of messages and their sizes, therefore, determines the load-balance between computations and communications. In our experiments with PAQR we have seen that with matrices of a size smaller than $n = 1536$, the best block size is n/p, but, with larger matrices, the best block size depends on the size of the matrix. In the case of the algorithm PALU the best value of ν in our cluster is 31.

Table 2 shows time and speed-up of both parallel algorithms up to 8 processors. Algorithm PAQR always improves its sequential version as the number of processors increases. The speed-up obtained with this parallel algorithm are not very good but always grows with the problem size. The limited performance is due to the influence of the communications forced by the down shift of a column of the generator in each iteration. Another constraint of PAQR is the great amount of memory required but the use of several processors allows us to deal with bigger problems.

On the other hand, it can be seen that PALU obtains better time and speed-up than PAQR. However, the first step is a sequential process. This fact limits the maximum speed-up when the prime factors of $n + 1$ are large and the number of processors increases, as it can be seen if we compare the speed-ups with matrices of sizes 1800 and 2000 in Table 2. For a matrix of size $n = 4095$ we have obtained an efficiency about 65% in 16 processors. Another advantage of the second parallel algorithm is that it can deal with larger matrices than PAQR.

To analyze the stability of the algorithms we have used the matrix $T = T_0 + \xi T_1$ where T_0 is a symmetric Toeplitz matrix called KMS (Kac-Murdock-

Table 2. Time in seconds (speed-up) of the parallel algorithm PAQR

	PAQR				PALU			
$n/p =$	1	2	4	8	1	2	4	8
1000	0.95	0.87 (1.09)	0.73 (1.30)	0.60 (1.58)	0.45	0.25 (1.80)	0.14 (3.21)	0.10 (4.50)
1200	1.54	1.20 (1.28)	0.97 (1.59)	0.82 (1.88)	0.94	0.66 (1.42)	0.45 (2.09)	0.38 (2.47)
1400	2.10	1.59 (1.32)	1.26 (1.67)	0.99 (2.12)	1.01	0.60 (1.68)	0.34 (2.97)	0.23 (4.39)
1600	2.74	2.02 (1.36)	1.56 (1.76)	1.21 (2.26)	1.70	1.14 (1.49)	0.85 (2.00)	0.65 (2.62)
1800	3.40	2.46 (1.38)	1.85 (1.84)	1.46 (2.33)	2.16	1.45 (1.49)	1.08 (2.00)	0.82 (2.63)
2000	4.23	2.96 (1.43)	2.21 (1.91)	1.68 (2.52)	2.01	1.07 (1.88)	0.63 (3.19)	0.30 (6.70)

Table 3. Forward and backward errors of both parallel algorithms

	PAQR		PALU	
n(cols)	Back. error	For. error	Back. error	For. error
1000 (6)	8.47×10^{-14}	6.40×10^{-3}	4.16×10^{-12}	1.07×10^{-5}
1200 (5)	1.20×10^{-15}	1.20×10^{-13}	4.79×10^{-12}	1.83×10^{-11}
1400 (5)	1.10×10^{-15}	8.17×10^{-13}	6.70×10^{-12}	1.68×10^{-11}
1600 (6)	2.79×10^{-13}	5.01×10^{-2}	6.69×10^{-13}	1.05×10^{-11}
1800 (5)	3.90×10^{-15}	1.15×10^{-12}	1.16×10^{-11}	6.72×10^{-11}
2000 (5)	8.73×10^{-16}	5.75×10^{-13}	1.40×10^{-11}	3.12×10^{-11}

Szegö) whose elements are $t_0 = \epsilon$ and $t_i = t_{-i} = \left(\frac{1}{2}\right)^i$ for $i = 1, 2, \ldots, n-1$, and matrix T_1 is randomly generated. We have chosen $\epsilon = \xi = 10^{-14}$. In this case the leading submatrices of T with sizes $3m+1$, $m = 0, 1, \ldots$ are ill conditioned. Classical Levinson and Schur-type algorithms break down or produce bad results with that matrix because it is not strongly regular. The right-hand side vector b has been chosen in such a way so that the exact solution x is a vector where all elements have a value of one. Now, we have been able to obtain the backward and forward errors,

$$\frac{\|T\tilde{x} - b\|}{\|T\| \cdot \|\tilde{x}\| + \|b\|} \quad \text{and} \quad \frac{\|\tilde{x} - x\|}{\|x\|} \, ,$$

where $\tilde{x}$ is the computed solution.

Table 3 shows both errors with the two parallel algorithms. The first column shows the matrix sizes and also shows the number of columns of the generator in the algorithm PAQR. When $n = 3m+1$ for a given m, $\kappa(T) \approx 10^{14}$. Backward errors of PAQR are good because it produces a corrected QR factorization over the product T^TT which is strongly regular. However, if T is ill conditioned ($n = 1000, 1600$), then $\kappa(T^TT) \gg \kappa(T)$. The Cauchy-like matrix preserves the conditioning of the original Toeplitz matrix. As PAQR works with T^TT while PALU deals with the transformed matrix T, PALU produces smaller forward errors than PAQR with ill-conditioned matrices.

5 Conclusions

Both algorithms presented in this paper parallelize *fast* sequential methods that exploit the displacement structure of Toeplitz matrices. Despite the small computational cost, both parallel algorithms improve their sequential versions. The implemented algorithms are portable because they are based on standard sequential and parallel libraries. They have been tested on a cluster of personal computers, but they can be used on any distributed memory architecture.

Algorithm PAQR involves many communications and has a fine–grain parallelism. This produces small speed–ups in our cluster of personal computers, but the time is reduced with the number of processors. Algorithm PAQR is more backward–stable than PALU in all cases and more accurate for well–conditioned matrices. On the contrary, algorithm PALU avoids a great number of communications and increases the overlapping between computation and communications. Although its efficiency can be affected by the cost of the initial Toeplitz–to–Cauchy transformation, for large matrices with a decomposition of $n + 1$ in small primes we can expect a good efficiency with several processors. Algorithm PALU is more forward–stable with ill–conditioned matrices.

References

1. Pedro Alonso, José M. Badía, and Antonio M. Vidal. Resolución de sistemas lineales de ecuaciones Toeplitz en paralelo por el método de Cauchy. TR DSIC-II/26/2002, DSIC, Universidad Politécnica de Valencia, 2002.
2. N.J. Boden, D. Cohen, R.E. Felderman, A.E. Kulawik, C.L. Seitz, J.N. Seizovic, and W.K. Su. Myrinet. a gigabit-per-second local-area network. *IEEE Micro*, 15:29–36, 1995.
3. S. Chandrasekaran and Ali H. Sayed. A fast stable solver for nonsymmetric Toeplitz and quasi-Toeplitz systems of linear equations. *SIAM Journal on Matrix Analysis and Applications*, 19(1):107–139, January 1998.
4. E. Anderson et al. *LAPACK Users' Guide*. SIAM, Philadelphia, 1995.
5. L.S. Blackford et al. *ScaLAPACK Users' Guide*. SIAM, Philadelphia, 1997.
6. R. W. Freund. A look-ahead Schur-type algorithm for solving general Toeplitz systems. *Zeitschrift für Angewandte Mathe. und Mechanik*, 74:T538–T541, 1994.
7. Georg Heinig and Adam Bojanczyk. Transformation techniques for Toeplitz and Toeplitz-plus-Hankel matrices. I. transformations. *Linear Algebra and its Applications*, 254(1–3):193–226, March 1997.
8. T. Kailath, S.-Y. Kung, and M. Morf. Displacement ranks of a matrix. *Bulletin of the American Mathematical Society*, 1:769–773, 1979.
9. Thomas Kailath and Ali H. Sayed. Displacement structure: Theory and applications. *SIAM Review*, 37(3):297–386, September 1995.
10. P. N. Swarztrauber. FFT algorithms for vector computers. *Parallel Computing*, 1(1):45–63, August 1984.