A Parallel Solution of Hermitian Toeplitz Linear Systems[*,**]

Pedro Alonso[1], Miguel O. Bernabeu[1], and Antonio M. Vidal[1]

Universidad Politécnica de Valencia, cno. Vera s/n, 46022 Valencia, Spain
{palonso, mbernabeu, avidal}@dsic.upv.es

Abstract. A parallel algorithm for solving complex hermitian Toeplitz linear systems is presented. The parallel algorithm exploits the special structure of Toeplitz matrices to obtain the solution in a quadratic asymptotical cost. Our parallel algorithm transfors the Toeplitz matrix into a Cauchy–like matrix. Working on a Cauchy–like system lets to work with real arithmetic. The parallel algorithm for the solution of a Cauchy–like matrix has a low amount of communication cost regarding other parallel algorithms that work directly on the Toeplitz system. We use a message–passing programming model. The experimental tests are obtained in a cluster of personal computers.

1 Introduction

In this work we propose a parallel algorithm for the solution of

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

with $T = (t_{ij}) = (t_{|i-j|})_{0 \le i,j < n} \in \mathbb{C}^{n \times n}$ hermitian and being $b, x \in \mathbb{C}^n$ the independent and the solution vectors, respectively.

There exist the so called *fast* algorithms to obtain the solution to this problem with an order of magnitude lower than the classical algorithms. These algorithms are based on the *displacement rank* property of the *structured* matrices. Nevertheless, there are two main drawbacks. Firstly, if the Toeplitz matrix is not strongly regular, *fast* algorithms can break down or can produce poor results regarding the accuracy of the solution. The other one deals with its parallelization because due to the dependency between operations causes a large number of point–to–point and broadcast–type communications when a message passing model programming is used [1].

The parallel algorithm presented in this paper transforms the Toeplitz matrix into a another type of structured matrix called Cauchy–like. This reduces significantly the execution time with the number of processors thanks to the use of just only one broadcast–type communication per iteration. In addition,

[*] Supported by Spanish MCYT and FEDER under Grant TIC 2003-08238-C02-02.
[**] Funded by Secretaría de Estado de Universidades e Investigación del Ministerio de Educación y Cultura of Spain.

working on Cauchy–like matrices avoids the algorithm to break down although it can still produce inaccuracy results. However, a refinement technique can be incorporated to improve the precision of the solution like it is done in [2] for the real non–symmetric case. Furthermore, we have used a blocking factor that minimizes the execution time overlapping computational and communication operations.

The parallel algorithm constitutes a particular improvement for hermitian matrices. The more general non–hermitian case can be found in [3]. Other related works based on the same idea applied to the real symmetric case was presented in [4].

Standard libraries like LAPACK [5] and ScaLAPACK [6] are used to achieve a more easy and portable implementation. The `fftpack` library [7, 8] is also used together with our own implementation by using the *Chirp-z* factorization [9].

The next two sections includes a brief revision of the mathematical background and the sequential algorithm. Afterward, Sect. 4 and Sect. 5 shows the parallel algorithm and the experimental results, respectively.

2 Rank Displacement and Cauchy–Like Matrices

The concept of *rank displacement* [10] describes the special structure of *structured* matrices. The definition uses the *displacement equation* of a given $n \times n$ matrix. If the rank r of the *displacement equation* is considerably lower than n ($r \ll n$), it is said that the matrix is *structured*.

Given the Toeplitz matrix of (1) its displacement equation can be defined in several ways. A useful form for our purposes is

$$F\,T - T\,F = G\,H\,G^* \ , \tag{2}$$

where $F = Z + Z^T$, being Z the one position down shift matrix, and being $G \in \mathbb{C}^{n\times 4}$ and $H \in \mathbb{C}^{4\times 4}$ the so called *generator pair*.

It is said that matrix $C = (c_{ij})_{1\le i,j\le n}$ is a hermitian Cauchy matrix, if for a complex vector $\lambda = (\lambda_i)_1^n$, the matrix

$$\nabla_\lambda C = ((\lambda_i - \lambda_j)c_{ij})_1^n \ , \ (\lambda_i - \lambda_j)c_{ij} = 1 \ , \tag{3}$$

has a very low rank with respect to n. If $(\lambda_i - \lambda_j)c_{ij} \neq 1$, matrix $\nabla_\lambda C$ is said to be a Cauchy–like matrix.

Both Toeplitz and Cauchy–like matrices as defined in (1) and (3), respectively, are structured matrices. Furthermore, there exists a direct relation between both classes of matrices by means of linear transformations. However, it is possible to avoid working in the complex plane in the hermitian case as follows.

Let S be the normalized Discrete Sine Transform (DST-I) [11, 12] matrix

$$S = \sqrt{\frac{2}{n+1}}\left(\sin\frac{ij\pi}{n+1}\right) \ , i,j = 1,\dots,n \ , \tag{4}$$

where $S = S^T$ and $SS^T = S^TS = I$, being I the identity matrix, and let P be a *odd–even* permutation matrix so $P\begin{pmatrix} x_1\, x_2\, x_3\, x_4 \ldots \end{pmatrix} = \begin{pmatrix} x_1\, x_3 \ldots x_2\, x_4 \ldots \end{pmatrix}$. Let $\Re(T)$ and $\Im(T)$ be the real and imaginary part of T, then

$$PS\Re(T)SP^T = \begin{pmatrix} M_1 & 0 \\ 0 & M_2 \end{pmatrix} , \quad \text{and} \quad PS\Im(T)SP^T = \begin{pmatrix} 0 & -M_3^T \\ M_3 & 0 \end{pmatrix} ,$$

where M_1, M_2 and M_3 are all real symmetric Cauchy–like matrices of size $\lceil n/2 \rceil \times \lceil n/2 \rceil$, $\lfloor n/2 \rfloor \times \lfloor n/2 \rfloor$ and $\lceil n/2 \rceil \times \lfloor n/2 \rfloor$, respectively.

Being matrix $\mathcal{D}$ defined as

$$\mathcal{D} = \begin{pmatrix} I_{\lceil n/2 \rceil} & \\ & iI_{\lfloor n/2 \rfloor} \end{pmatrix} ,$$

where $i = \sqrt{-1}$, the following transformation allows to convert a hermitian Toeplitz matrix into a real symmetric Cauchy–like matrix

$$C = \mathcal{D}PSTSP^T\mathcal{D}^* = \begin{pmatrix} M_1 & -M_3^T \\ M_3 & M_2 \end{pmatrix} . \tag{5}$$

Applying the previous transformation to the displacement equation (2) the displacement equation of the real Cauchy–like matrix in (5) is obtained

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

where

$$\hat{G} = \begin{pmatrix} \Re(\mathcal{G})\ \Im(\mathcal{G}) \end{pmatrix} \text{ and } \mathcal{G} = \frac{1}{\sqrt{n+1}}\mathcal{D}PSG_{:,1:2} . \tag{7}$$

Thus, the solution of a hermitian Toeplitz linear system (1) is approached by solving the real symmetric Cauchy–like system

$$C\hat{x} = \hat{b} , \tag{8}$$

where $\hat{b} = \mathcal{D}PSb$ and $\hat{x} = \mathcal{D}PSx$ so the solution of (1) is $x = \mathcal{D}^*P^TS\hat{x}$.

The Cauchy–like linear system (8) is solved performing the factorization $C = LDL^T$ with L lower triangular and D diagonal. This factorization can be done in a “fast” way due to the displacement rank property of structured matrices.

3 Triangular Factorization of Cauchy–Like Matrices

Gohberg, Kailath and Olshevsky [13] proposed an algorithm (GKO) to factorize non–hermitian Cauchy–like matrices. Following the same idea is not hard to derive a fast algorithm for the triangular factorization of real symmetric Cauchy–like matrices that exploits its displacement rank property.

Given the following partitions of matrices C and Λ (6),

$$C = \begin{pmatrix} d & l^T \\ l & C_1 \end{pmatrix}, \Lambda = \begin{pmatrix} \lambda_1 & 0 \\ 0 & \Lambda_1 \end{pmatrix} ,$$

where $\left(d\ l^T\right)^T$ is a one dimensional array, and being $X = \begin{pmatrix} 1 & 0 \\ l/d & I_{n-1} \end{pmatrix}$, if the transformation $X^{-1}(.)X^{-T}$ is applied to (6), we have

$$(X^{-1}\Lambda X)(X^{-1}CX^{-T}) - (X^{-1}CX^{-T})(X^T\Lambda X^{-T}) =$$

$$\begin{pmatrix} \lambda_1 & 0 \\ \frac{\Lambda_1 l - \lambda_1 l}{d} & \Lambda_1 \end{pmatrix} \begin{pmatrix} d & 0 \\ 0 & C_{sc} \end{pmatrix} - \begin{pmatrix} d & 0 \\ 0 & C_{sc} \end{pmatrix} \begin{pmatrix} \lambda_1 & \frac{l^T\Lambda_1 - \lambda_1 l^T}{d} \\ 0 & \Lambda_1 \end{pmatrix} = (X^{-1}\hat{G})H(X^{-1}\hat{G})^T .$$

Equating element $(2,2)$ of the previous expression it is obtained

$$\Lambda_1\ C_{sc} - C_{sc}\ \Lambda_1 = \hat{G}_1\ H\ \hat{G}_1^T ,$$

that is, the displacement equation of the Schur complement C_{sc} of C with respect to its first element d. Matrix $\hat{G}_1 = X^{-1}\hat{G}$ is the *generator* of C_{sc}. This property about that the Schur complements of a structured matrix are also structured is used to derive a triangular factorization fast algorithm.

The computation of the first column of C and the computation of $\hat{G}_1$ defines the first step of the algorithm. The LDL^T factorization of C is obtained repeating this process n iterations on the successive arising Schur complements (Alg. 1).

Algorithm 1 (LDL^T factorization of a real symmetric Cauchy–like matrix): *Given $\hat{G}$, H, Λ (6) and diagonal entries of C, this algorithm returns the unit lower triangular factor L and the diagonal matrix D such that $C = LDL^T$.*

```
for $j = 1, \ldots, n$
    $D_{j,j} = C_{j,j}$
    for $i = j+1, \ldots, n$
        1. $L_{i,j} = (\hat{G}_{i,:}\ H\ \hat{G}_{j,:}^T)/(D_{j,j}\ (\lambda_i - \lambda_j))$
        2. $C_{i,i} \leftarrow C_{i,i} - D_{j,j}L_{i,j}^2$
        3. $\hat{G}_{i,:} \leftarrow \hat{G}_{i,:} - L_{i,j}\hat{G}_{j,:}$
    end for
end for
```

All elements of each Schur complement of C can be computed by solving the Lyapunov equation shown in step 1 of Alg. 1 except diagonal entries. That is why these elements must be computed a priori before calling Alg. 1. There exist several fast algorithms to do that like the one in [14]. However, we have developed a new one that lets to obtain diagonal entries of the Cauchy–like matrix arising from $C = ST_LS$ where T_L is lower triangular Toeplitz. This is useful to obtain diagonal entries of different symmetric Cauchy–like matrices of the form $C = SMS$ where M can be either symmetric or non–symmetric or even the product of two Toeplitz matrices. Furthermore, the algorithm uses a minimal amount of memory avoiding to use of some vectors arising in the description of the algorithm by other authors.

P_0	$\hat{G}_0$	$L_{0,0}$
P_1	$\hat{G}_1$	$L_{1,0}$ $L_{1,1}$
P_2	$\hat{G}_2$	$L_{2,0}$ $L_{2,1}$ $L_{2,2}$
P_0	$\hat{G}_3$	$L_{3,0}$ $L_{3,1}$ $L_{3,2}$ $L_{3,3}$
P_1	$\hat{G}_4$	$L_{4,0}$ $L_{4,1}$ $L_{4,2}$ $L_{4,3}$ $L_{4,4}$
$\vdots$	$\vdots$	$\vdots$ $\vdots$ $\vdots$ $\vdots$ $\vdots$ $\ddots$

Fig. 1. Example of data distribution with 3 processors

4 The Parallel Algorithm

The central part of the parallel algorithm falls in the factorization of C. The generator $\hat{G}$ is partitioned in n/η blocks of size $\eta \times 4$ and distributed using the BLACS model. Under this model a *logical* unidimensional mesh with $p \times 1$ processors is built. Blocks of $\hat{G}$ are cyclically distributed among the p processors such that block $\hat{G}_k$, $k = 0, \ldots, n/\eta - 1$, belongs to processor $P_{j \bmod p}$. The ith row of $\hat{G}$ belongs to block $\hat{G}_{i/\eta}$. Matrix Λ and the diagonal of C are both partitioned and distributed in the same way. Alg. 2 is the parallel version of Alg. 1. An example of distribution of $\hat{G}$ and L with 3 processors can be seen in Fig. 1.

Algorithm 2 (Parallel LDL^T factorization of a symmetric Cauchy–like matrix): *Given $\hat{G}$, H, Λ (6) and the diagonal of C cyclically distributed for a given block size $\eta \geq 1$, this algorithm returns the unit lower triangular factor L and the diagonal matrix D such that $C = LDL^T$ both partitioned in blocks of η rows cyclically distributed as the argument matrices.*

Each processor P_j, $j = 0, \ldots, p-1$, performs:

```
for $k = 0, \ldots, n/\eta - 1$
    Let $C_{sc}^k = \begin{pmatrix} C_1 & C_3^T \\ C_3 & C_2 \end{pmatrix}$, $C_1 \in \mathbb{R}^{\eta \times \eta}$, be the Schur complement of $C$
    with respect to the leading submatrix of order $k\eta$.
    if ($\hat{G}_k \in P_j$)
        Compute $C_1 = L_{kk} D_k L_{kk}^T$ by using Alg. 1 and broadcast $\hat{G}_k$ and $\Lambda_k$.
    else Receive $\hat{G}_k$ and $\Lambda_k$.
    end if
    for $i = k+1, \ldots, n/\eta - 1$
        if ($\hat{G}_i \in P_j$)
            Compute $L_{ik}$ and update $\hat{G}_i$ and the diagonal entries of $C_{ii}$.
        end if
    end for
end for
```

Updating blocks $\hat{G}_i$ and diagonal entries of C_{ii}, $k < i < n/\eta$, can be easily derive from the operations described in Alg. 1 if only η iterations are applied. In each step of Alg. 2 the following factorization is obtained

$$C_{sc}^k = \begin{pmatrix} L_{k,k} \\ L_{k+1:n/\eta-1,k} \end{pmatrix} \begin{pmatrix} D_{k,k} & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} L_{k,k}^T & L_{k+1:n/\eta-1,k}^T \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 0 & C_{sc}^{k+1} \end{pmatrix} ,$$

being C_{sc}^{k+1} implicitly known throughout the generator.

The main advantage of Alg. 2 is that it performs only one broadcast per iteration on the contrary that other parallel algorithms for the solution of the same problem. The size and the number of messages are both a function of η. Different values of η changes the overlapping between communications and computations and also the weight of both factors in the total cost of the algorithm. The optimum value of η depends on the machine and is experimentally tuned.

The complete parallel algorithm is summarized in Alg. 3.

Algorithm 3 (Parallel solution of a hermitian Toeplitz linear system): *Given a hermitian Toeplitz matrix* $T \in \mathbb{C}^{n\times n}$ *and a right hand side vector* $b \in \mathbb{C}^n$*, this algorithm returns the solution vector* x *of the linear system* $Tx = b$*.*
On each processor P_j*, for* $j = 0, \ldots, p-1$*:*

1. *Every processor computes matrices* $\hat{G}$ *(7),* Λ *(6) and vector* $\hat{b}$ *(8), and distribute them cyclically by blocks of size* $\eta \times 4$*.*
2. *Apply Alg. 2 to obtain* $C = LDL^T$*.*
3. *Solve* $LDL^T\hat{x} = \hat{b}$ *in parallel by using PBLAS routines.*
   P_0 *gathers* $\hat{x}$ *from the rest of processors and computes* $x = \mathcal{D}^* P^T S\hat{x}$*.*

Another improvement used is the efficient computation of the DST (4). The time used by the `fftpack` routine to apply a DST to a vector highly depends on the value of the prime numbers in which $n+1$ is factorised. This routine is faster as these prime numbers are small. We have implement a DST routine based on the *Chirp–z* factorization used in the computation of DFT's whose computational cost is independent of the size of these prime numbers (Table 1).

5 Experimental Results

The experimental results have been obtained in a cluster of 10 two–processor boards with two Intel Xeon at 2 GHz. and with 1 Gb. of RAM. The

Table 1. Excution time of $\hat{G}$ with and without using the *Chirp–z* factorization

n	6998	7498	7998	8498	8998	9998	10498	10998
p. d.	$3 \cdot 2333$	7499	$19 \cdot 421$	$3 \cdot 2833$	8999	$3^2 \cdot 11 \cdot 101$	10499	$17 \cdot 647$
`fftpack`	0.64	2.93	0.33	0.96	4.25	0.45	5.76	0.64
Chirp–z	0.36	0.38	0.41	0.60	0.64	0.72	0.77	0.81

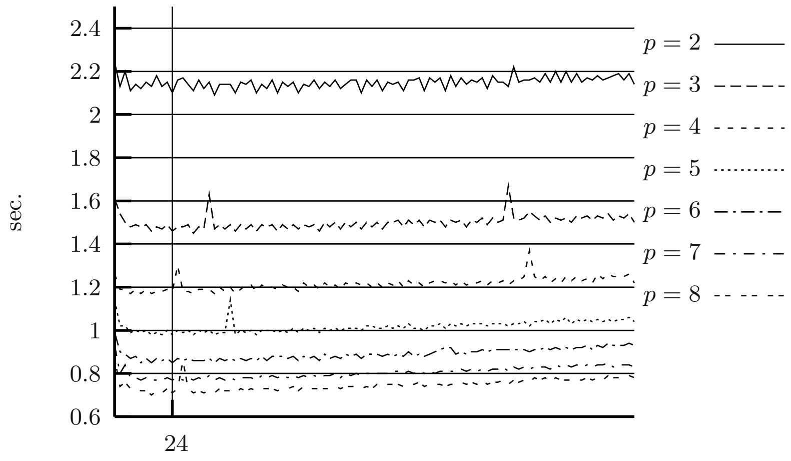

Fig. 2. Time versus different values of η and number of processors for $n = 12000$

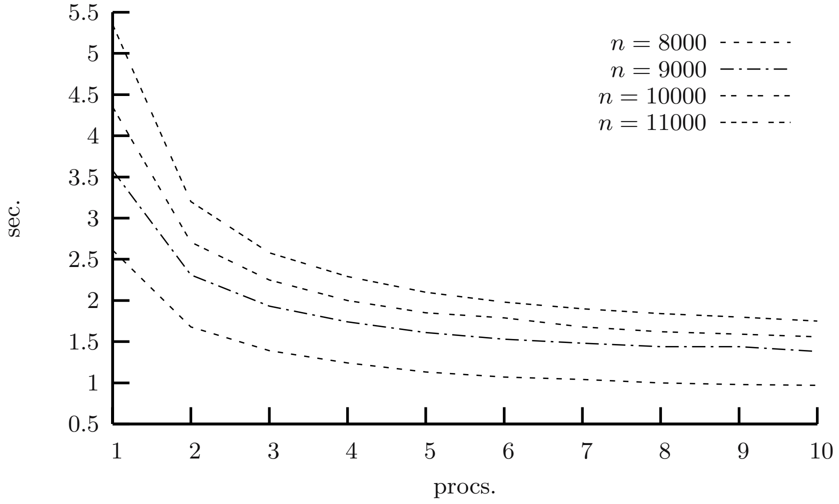

Fig. 3. Time for different size problems and different number of processors.

interconnection network is a SCI with a 2D torus topology. In the experiments, each MPI process is mapped onto only one processor of each node.

The first test consists of tuning the block size η used in the partition of $\hat{G}$. Fig. 2 shows the execution time of Alg. 2, that is the second step of Alg. 3, so it can be concluded that there exists a range of values for η that can be used to get the best performance. We have chosen a size of $\eta = 24$ independently of the number of processors in our target machine. A more detailed study with other problem sizes shows that this is always the best choice despite of some a very slight variation of time in the different values of η.

Once the optimal value of η is fixed, the next experiment deal with the total cost of the parallel algorithm with different problem sizes. Figure 3 shows a large reduction in time achieved with the increment in the number of processors. This is specially significant because this result means the parallel algorithm can be useful in applications with real time constraints.

6 Conclusions

We have presented a parallel algorithm that exploits the displacement structure of Toeplitz and Cauchy–like matrices as the sequential algorithms do. The algorithm does not break down if the Toeplitz matrix is not strongly regular, uses real arithmetic and reduces the execution time with the use of up to 10 processors. This is a challenge taking into account how difficult it is to improve the performance of this type of fast algorithms due to the operation dependency and the large cost of a message communication regarding the time of a flop.

References

1. Alonso, P., Badía, J.M., Vidal, A.M.: Parallel algorithms for the solution of toeplitz systems of linear equations. LNCS **3019** (2004) 969–976
2. Alonso, P., Badía, J.M., Vidal, A.M.: An efficient and stable parallel solution for non–symmetric Toeplitz linear systems. LNCS **3402** (2005) 685–692
3. Alonso, P., Vidal, A.M.: An efficient parallel solution of complex toeplitz linear systems. Lecture Notes in Computer Science (to appear in 2006)
4. Alonso, P., Vidal, A.M.: The symmetric–toeplitz linear system problem in parallel. LNCS **3514** (2005) 220–228
5. Anderson, E., Bai, Z., Bischof, C., Demmel, J., Dongarra, J., Croz, J.D., Greenbaum, A., Hammarling, S., McKenney, A., Ostrouchov, S., Sorensen, D.: LAPACK Users' Guide. Second edn. SIAM, Philadelphia (1995)
6. Blackford, L., et al.: ScaLAPACK Users' Guide. SIAM, Philadelphia (1997)
7. Swarztrauber, P.: Vectorizing the FFT's. Academic Press, New York (1982)
8. Swarztrauber, P.: FFT algorithms for vector computers. Parallel Computing **1** (1984) 45–63
9. Loan, C.V.: Computational Frameworks for the Fast Fourier Transform. SIAM Press, Philadelphia (1992)
10. Kailath, T., Sayed, A.H.: Displacement structure: Theory and applications. SIAM Review **37** (1995) 297–386
11. Bojanczyk, A.W., Heinig, G.: Transformation techniques for toeplitz and toeplitz-plus-hankel matrices part I. transformations. Technical Report 96-250, Cornell Theory Center (1996)
12. Bojanczyk, A.W., Heinig, G.: Transformation techniques for toeplitz and toeplitz-plus-hankel matrices part II. algorithms. Technical Report 96-251, Cornell Theory Center (1996)
13. Gohberg, I., Kailath, T., Olshevsky, V.: Fast Gaussian elimination with partial pivoting for matrices with displacement structure. Mathematics of Computation **64** (1995) 1557–1576
14. Thirumalai, S.: High performance algorithms to solve Toeplitz and block Toeplitz systems. Ph.d. thesis, Graduate College of the University of Illinois at Urbana-Champaign (1996)