The Symmetric–Toeplitz Linear System Problem in Parallel*

Pedro Alonso[1] and Antonio M. Vidal[1]

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

Abstract. Many algorithms exist that exploit the special structure of Toeplitz matrices for solving linear systems. Nevertheless, these algorithms are difficult to parallelize due to its lower computational cost and the great dependency of the operations involved that produces a great communication cost. The foundation of the parallel algorithm presented in this paper consists of transforming the Toeplitz matrix into a another structured matrix called Cauchy–like. The particular properties of Cauchy–like matrices are exploited in order to obtain two levels of parallelism that makes possible to highly reduce the execution time. The experimental results were obtained in a cluster of PC's.

1 Introduction

In this paper, we present a parallel algorithm for the solution of the linear system

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

where $T \in \mathbb{R}^{n\times n}$ is a symmetric Toeplitz matrix $T = (t_{ij})_{i,j=0}^{n-1} = (t_{|i-j|})_{i,j=0}^{n-1}$ and $b, x \in \mathbb{R}^n$ are the independent and the solution vector, respectively.

It is difficult to obtain efficient parallel versions of *fast* algorithms, because they have a reduced computational cost and they also have many dependencies among fine–grain operations. These dependencies produce many communications, which are a critical factor to obtain efficient parallel algorithms, especially on distributed memory computers. This problem could explain partially the small number of parallel algorithms implemented so far dealing with Toeplitz matrices. For instance, it can be found parallel algorithms to solve Toeplitz systems using systolic arrays [1] or dealing only with positive definite matrices or with symmetric matrices [2]. There also exist parallel algorithms for shared memory computers [3–5] and, more recently, several parallel algorithms for distributed architectures have been proposed [6].

One of our main goals is to offer efficient parallel algorithms for general purpose architectures, especially, clusters of personal computers. Furthermore, the codes are portable because they are based on standard libraries, both sequential,

* Supported by Spanish MCYT and FEDER under Grant TIC 2003-08238-C02-02.

LAPACK [7], and parallel, ScaLAPACK [8]. We are mainly interested in the reduction of parallel runtime because one of the main set of applications requires real time computation of the linear system (1) like digital signal analysis [9].

In the next section, the mathematical background used is summarized. In Sections 3, 4 and 5 the parallel algorithm is described. Finally, the experimental results are shown in the last section.

2 Rank Displacement and Cauchy–like Matrices

It is said that a matrix of order n is *structured* if its *displacement representation* has a lower rank regarding n. The *displacement representation* of a symmetric Toeplitz matrix T (1) can be defined in several ways depending on the form of the displacement matrices. A useful form for our purposes is

$$\nabla_F T = F\,T - T\,F = \mathcal{G}\,\mathcal{H}\,\mathcal{G}^T \ ; \tag{2}$$

where $F = T(e_1)$, called *displacement matrix*, is a $n \times n$ symmetric Toeplitz matrix with the second column of the identity matrix as the first column; $\mathcal{G} \in \mathbb{R}^{n\times 4}$ is the *generator* matrix and $\mathcal{H} \in \mathbb{R}^{4\times 4}$ is a skew–symmetric *signature* matrix. The rank of $\nabla_F T$ is 4, that is, lower than n and independent of n.

It is easy to see that the displacement of T with respect to F is a matrix of a considerably sparsity from which it is not difficult to obtain an analytical form of $\mathcal{G}$ and $\mathcal{H}$.

A symmetric *Cauchy–like* matrix C is a structured matrix that can be defined as the unique solution of the displacement equation

$$\nabla_\Lambda C = \Lambda\,C - C\,\Lambda = \hat{\mathcal{G}}\,\mathcal{H}\,\hat{\mathcal{G}}^T \ , \tag{3}$$

being $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$, where $\mathrm{rank}(\nabla_\Lambda C) \ll n$ and independent of n.

Now, we use the normalized Discrete Sine Transformation (DST) $\mathcal{S}$ as defined in [10]. Since $\mathcal{S}$ is symmetric, orthogonal and $\mathcal{S}F\mathcal{S} = \Lambda$ [11, 12], we obtain

$$\mathcal{S}(FT - TF)\mathcal{S} = \mathcal{S}(\mathcal{G}\mathcal{H}\mathcal{G}^T)\mathcal{S} \rightarrow \Lambda C - C\Lambda = \hat{\mathcal{G}}\mathcal{H}\hat{\mathcal{G}}^T \ ,$$

where $C = \mathcal{S}T\mathcal{S}$ and $\hat{\mathcal{G}} = \mathcal{S}\mathcal{G}$. This shows how it can be transformed (2) into (3).

In this paper, we solve the Cauchy–like linear system $C\hat{x} = \hat{b}$, where $\hat{x} = \mathcal{S}x$ and $\hat{b} = \mathcal{S}b$, by performing the triangular decomposition $C = LDL^T$, being L unit lower triangular and D diagonal. The solution of (1) is obtained by computing $Ly = \hat{b}$, $y \leftarrow D^{-1}y$, $L^T\hat{x} = y$ and $x = \mathcal{S}\hat{x}$.

Solving a symmetric Toeplitz linear system by transforming it into a symmetric Cauchy–like system has an interesting advantage due to the symmetric Cauchy–like matrix has an important sparsity. Matrix C has the form (x only denotes non–zero entries),

$$C = \begin{pmatrix} x & 0 & x & 0 & \ldots \\ 0 & x & 0 & x & \ldots \\ x & 0 & x & 0 & \ldots \\ 0 & x & 0 & x & \ldots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix} \ .$$

We define the odd–even permutation matrix P_{oe} as the matrix that, after applied to a vector, groups the odd entries in the first positions and the even entries in the last ones, $P_{oe}\left(x_1\ x_2\ x_3\ x_4\ x_5\ x_6 \ldots\right)^T = \left(x_1\ x_3\ x_5 \ldots x_2\ x_4\ x_6 \ldots\right)^T$. Applying transformation $P_{oe}(.)P_{oe}^T$ to a symmetric Cauchy–like matrix C gives

$$P_{oe}CP_{oe}^T = \begin{pmatrix} C_0 & \\ & C_1 \end{pmatrix} , \tag{4}$$

where C_0 and C_1 are symmetric Cauchy–like matrices of order $\lceil n/2 \rceil$ and $\lfloor n/2 \rfloor$, respectively. In addition, it can be shown that matrices C_0 and C_1 have a displacement rank of 2, as opposed to C that has a displacement rank of 4 [5].

The two submatrices arising in (4) have the displacement representation

$$\Lambda_j C_j - C_j \Lambda_j = G_j H_j G_j^T , \quad i = 0, 1 , \tag{5}$$

where $\begin{pmatrix} \Lambda_0 & \\ & \Lambda_1 \end{pmatrix} = P_{oe}\mathcal{S}\Lambda\mathcal{S}P_{oe}^T$ and $H_0 = H_1 = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$. As it is shown in [13], given vector $u^T = \left(0\ t_2\ t_3 \cdots t_{n-2}\ t_{n-1}\right)^T$ and the first column of the identity matrix e_0, the generators of (5) can be computed as

$$\begin{pmatrix} G_0 \\ G_1 \end{pmatrix} = \sqrt{2} P_{oe}\mathcal{S}\left(u\ e_0\right) . \tag{6}$$

The odd–even permutation matrix is used to decouple the symmetric Cauchy–like matrix arising from a real symmetric Toeplitz matrix into the following two Cauchy–like systems of linear equations

$$C_j \hat{\hat{x}}_j = \hat{\hat{b}}_j , \quad j = 0, 1 , \tag{7}$$

where $\hat{\hat{x}} = \left(\hat{\hat{x}}_0^T\ \hat{\hat{x}}_1^T\right)^T = P_{oe}\mathcal{S}x$ and $\hat{\hat{b}} = \left(\hat{\hat{b}}_0^T\ \hat{\hat{b}}_1^T\right)^T = P_{oe}\mathcal{S}b$.

Each one of both linear systems are of half the size and half the displacement rank so this yields substantial saving over the non–symmetric forms of the displacement equation. Furthermore, it can be exploited in parallel by solving each of the two independent sub–systems into two different processors.

3 The Parallel Algorithm

For the parallel solution we used a two dimensional mesh of $p/2 \times 2$ processors as shown in Fig. 1, where each one of the p processors is denoted by the corresponding row and column index.

We used the ScaLAPACK tools in order to manage data distribution over this logical configuration of the processors. Once the symmetric Toeplitz system has been converted into a symmetric Cauchy–like one, the two subsystems arisen (7) will be solved independently on each "logical column" of the two–dimensional processors mesh. This is what the external loop ($j = 0, 1$) of Algorithm 1 represents, that is, iteration 0 and 1 are concurrently executed by processors column 0 and 1, respectively.

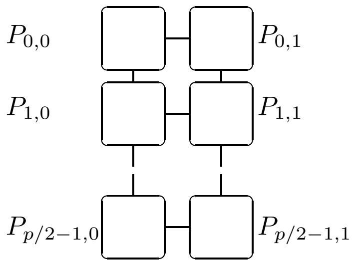

Fig. 1. 2D mesh of processors

Algorithm 1 (Parallel Algorithm for the solution of a symmetric–Toeplitz system with Cauchy–like transformation). *Given $T \in \mathbb{R}^{n\times n}$ a symmetric Toeplitz matrix and $b \in \mathbb{R}^n$ an independent vector, this algorithm returns the solution vector $x \in \mathbb{R}^n$ of the linear system $Tx = b$. For each $P_{i,j}$,*

1. *for $j = 0, 1,$ do*
    - *for $i = 0, \ldots, p/2-1,$ do*
        - *1.1. "Previous computations".*
        - *1.2. $C_j = L_j D_j L_j^T$ (4).*
        - *1.3. Solution of $L_j D_j L_j^T \hat{\hat{x}}_j = \hat{\hat{b}}_j$ (7).*
    - *end for.*

    end for.
2. *$P_{0,0}$ computes $x = \mathcal{S} P_{oe}^T \begin{pmatrix} \hat{\hat{x}}_0^T & \hat{\hat{x}}_1^T \end{pmatrix}^T$.*

Steps 1.1 and 1.2 are explained in the next two sections, respectively. Step 1.3 is performed by ScaLAPACK and PBLAS parallel subroutines. This last step can be repeated several times for iterative refinement. Finally, processor $P_{0,0}$ gathers the partial solutions of the two independent Cauchy–like linear systems and computes the solution of (1).

4 Parallel Triangularization of Symmetric Cauchy–like Matrices

The workload of the step 1.2 of Algorithm 1 falls in the operation with the $n \times 2$ entries of the generators G_0 and G_1 (6). The logical column of processors $P_{i,j}$, $i = 0, \ldots, p/2-1$, performs the triangular decomposition of the corresponding matrix $C_j = L_j D_j L_j^T$, $j = 0, 1$, where L_j is unit lower triangular and D_j is diagonal. The parallel algorithm exploits the fact that operations performed by each column of processors can be carried out independently on each row of G_j.

Let

$$\begin{pmatrix} \Lambda_0 & \\ & \Lambda_1 \end{pmatrix} C - C \begin{pmatrix} \Lambda_0 & \\ & \Lambda_1 \end{pmatrix} = \begin{pmatrix} G_0 \\ G_1 \end{pmatrix} H \begin{pmatrix} G_0 \\ G_1 \end{pmatrix}^T , \tag{8}$$

be the displacement representation of a given symmetric Cauchy–like matrix $C = \begin{pmatrix} C_{00} & C_{01} \\ C_{10} & C_{11} \end{pmatrix}$, then the Schur complement C_{sc} of C_{11} regarding C_{00} is also structured for any partition of C [14] so

$$\Lambda_1 \, C_{sc} - C_{sc} \, \Lambda_1 = G_1' \, H G_1'^T \ . \tag{9}$$

The parallel algorithm uses a sequential algorithm that, given the generator, the displacement matrix and the diagonal of C in equation (8) as entries, returns $G_1'^T$, the diagonal of C_{sc} of equation (9) and the factorization $C_{00} = L_{00} D_{00} L_{00}^T$.

Let m_j denotes the size of C_j (4), $j = 0, 1$, respectively. Each generator G_j (5) is partitioned in m_j/ν blocks of size $\nu \times 2$ for a given integer ν, $1 \leq \nu \leq (m_j/p)$, and cyclically distributed onto the respective column of processors $P_{k,j}$, for $k = 0, \ldots, p/2 - 1$, in such a way that block $G_{i,j}$, $i = 0, \ldots, m_j/\nu - 1$, belongs to processor $P_{\bmod(i,p),j}$. For simplicity in the exposition we will assume in the next that $(m_j \bmod \nu) = 0$. The unit lower triangular factor L_j obtained by the algorithm is partitioned in a two dimensional array of $(m_j/\nu) \times (m_j/\nu)$ square blocks of order ν, where each square block $L^j_{i,l}$, for $l = 0, \ldots, i$, belongs to processor $P_{\bmod(i,p),j}$ as the generators blocks. The diagonal matrix D_j is stored in the diagonal entries of L_j since all diagonal entries of L_j are implicitly one. In Fig. 2 it can be seen an example of distribution of both G_j and L_j in the logical column $j = 0, 1$ formed of three processors.

$P_{0,j}$	$G_{0,j}$	$L^j_{0,0}$
$P_{1,j}$	$G_{1,j}$	$L^j_{1,0}$ $L^j_{1,1}$
$P_{2,j}$	$G_{2,j}$	$L^j_{2,0}$ $L^j_{2,1}$ $L^j_{2,2}$
$P_{0,j}$	$G_{3,j}$	$L^j_{3,0}$ $L^j_{3,1}$ $L^j_{3,2}$ $L^j_{3,3}$
$P_{1,j}$	$G_{4,j}$	$L^j_{4,0}$ $L^j_{4,1}$ $L^j_{4,2}$ $L^j_{4,3}$ $L^j_{4,4}$
$\vdots$	$\vdots$	$\vdots$ $\vdots$ $\vdots$ $\vdots$ $\vdots$ $\ddots$

Fig. 2. Example of data distribution in a mesh of 3×2 processors

In each iteration k, $k = 0, \ldots, (m_j/\nu - 1)$, the processor containing block $G_{k,j}$ computes $L^j_{k,k}$ by means of the sequential algorithm described at the beginning of this section and broadcasts the suitable information to the rest of the processors of the column that compute $L^j_{i,k}$ and update the $G_{i,j}$, for $i > k$.

5 Previous Computations

Step 1.1 of Algorithm 1, called "previous computations", involves four different tasks, denoted as Task00, Task10, Task01 and Task11, respectively, in which is divided the computation of G_0 and G_1 (6), the computation of the displacement matrices Λ_0 and Λ_1 (5), the computation of the diagonal of C [5] and the computations of $\hat{\hat{b}}_0$ and $\hat{\hat{b}}_1$ (7).

If there are only two processors ($p = 2$), $P_{0,0}$ computes Task$i$0 while $P_{0,1}$ computes Task$i$1, for $i = 0, 1$. For $p \geq 4$, processor $P_{i,j}$ computes Taskij. After the computation of the tasks, all processors perform two communication steps: 1. A multicast of the results obtained by the tasks within each column; 2. A data interchange between pairs of processors in each row of the processors mesh.

Several a DST's are carried out in step 1.1 of Algorithm 1. There exist algorithms related with the DFT that involves $O(n \log n)$ operations to perform the DST. But the performance of these algorithms highly depends on the size of the greatest prime number of the primes decomposition of $(n+1)$. In our algorithms, we used a method to avoid this dependency by applying several power of 2 of order DFT's using the *Chirp-z* factorization described in [10].

6 Experimental Results

All experimental analysed were carried out in a cluster with 20 nodes connected by a SCI network with a topology of a 4×5 torus. Each node is a two–processor board with two Intel Xeon at 2 GHz. and 1 Gb. of RAM memory per node.

The first analysis concerns the block size ν. At the light of the experiments, it can be concluded that there exist a wide range of values for that minimize the execution time. But, none of these values must be selected close to 1 (this implies too many communications) or close to n/p (this implies not enough concurrency between communications and computations). The best value obtained by experimental tuning is a fixed size of $\nu = 20$ that is only hardware dependent.

The second experimental analysis deals with the weight of each step of Algorithm 1 on its total cost. Table 1 shows the time spent on each step. It can be observed that the time spent in Step 1.1 grows with the problem size. The *Chirp-z* factorization used to perform the DST makes the cost of this step independent of the size of the prime numbers in which $(n + 1)$ is decomposed. The weight of this first step is $\approx 25\%$ of the total computational cost of the algorithm. The most costly step is the second one in which is performed the factorization of one of the two Cauchy–like matrices (C_0 or C_1). The weight of this step is $\approx 60\%$ of the total cost of the algorithm. The third step involves $\approx 15\%$ of the total time.

As it was explained in Section 5, the first step is divided in four tasks, each of one is carried out concurrently so it can be obtained a reduction in time in this step using up to 4 processors (Table 2).

Table 1. Execution time in seconds of Algorithm 1 in one processor

n	Step 1.1	Step 1.2	Step 1.3	total
4000	0.11	0.24	0.05	0.39
6000	0.25	0.51	0.12	0.87
8000	0.35	0.88	0.20	1.42
10000	0.60	1.35	0.35	2.28
12000	0.75	1.93	0.47	3.13
14000	0.95	2.59	0.63	4.14
16000	1.17	3.36	0.81	5.28
18000	1.71	4.21	1.09	6.96
20000	1.96	5.18	1.30	8.37

Table 2. Execution time in seconds and efficiency of Step 1.1

	1 processor	2 processors		4 processors	
n	time	time	efficiency	time	efficiency
4000	0.11	0.06	92%	0.04	69%
6000	0.25	0.14	89%	0.10	63%
8000	0.35	0.19	92%	0.13	67%
10000	0.60	0.33	91%	0.22	68%
12000	0.75	0.41	91%	0.27	69%
14000	0.95	0.53	90%	0.35	68%
16000	1.17	0.66	89%	0.43	68%
18000	1.71	0.91	94%	0.60	71%
20000	1.96	1.07	92%	0.70	70%

In Table 3 it can be seen the execution time and the efficiency of Step 1.2. The important effort performed in the parallelization of the triangularization process gives a good efficiency even with 10 processors. The low time obtained with the most costly step using several processors lets to obtain a low total time. This result, although it cannot be as efficient as it was desirable, it can be very useful in applications with real time constraints like digital signal processing.

We note that the efficiency obtained with 2 processors is quite good mainly due to the triangular decomposition of the two independent Cauchy–like matrices over two independent processors.

Table 3. Execution time in seconds and efficiency of Step 1.2

	1 proc.	2 procs.		4 procs.		6 procs.		8 procs.		10 procs.	
n	time	time	effi.	time	effi.	time	effi.	time	effi.	time	effi.
4000	0.24	0.13	92%	0.09	67%	0.07	57%	0.06	50%	0.05	48%
6000	0.51	0.27	94%	0.16	80%	0.13	65%	0.11	58%	0.10	51%
8000	0.88	0.48	92%	0.28	79%	0.20	73%	0.18	61%	0.16	55%
10000	1.35	0.73	92%	0.39	87%	0.30	75%	0.25	68%	0.23	59%
12000	1.93	1.03	94%	0.54	89%	0.41	78%	0.34	71%	0.30	64%
14000	2.59	1.39	93%	0.76	85%	0.53	81%	0.45	72%	0.39	66%
16000	3.36	1.80	93%	0.91	92%	0.67	84%	0.56	75%	0.49	69%
18000	4.21	2.25	94%	1.21	87%	0.83	85%	0.69	76%	0.60	70%
20000	5.18	2.71	96%	1.48	88%	1.00	86%	0.83	78%	0.71	73%

Finally, we analyze the execution time of the parallel algorithm. In Fig. 3, it can be seen that the time decreases with the increment of the number of processors. This reduction in time is more significant if the problem size increases. The algorithm also reduces its execution time with more than four processors although Step 1.1 does not exploit more processors in parallel than this quantity. We emphasize the reduction in time regarding the scalability of the parallel algorithm by its utility in applications with real time constraints.

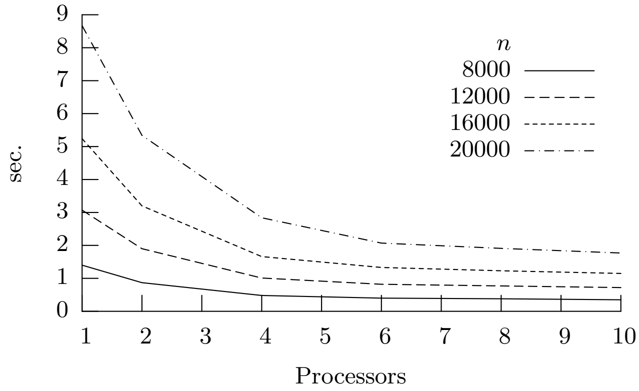

Fig. 3. Time in seconds of the parallel algorithm

References

1. Sweet, D.R.: The use of linear-time systolic algorithms for the solution of toeplitz problems. Technical Report JCU-CS-91/1, Department of Computer Science, James Cook University (1991) Tue, 23 Apr 1996 15:17:55 GMT.
2. Evans, D.J., Oka, G.: Parallel solution of symmetric positive definite Toeplitz systems. Parallel Algorithms and Applications **12** (1998) 297–303
3. Gohberg, I., Koltracht, I., Averbuch, A., Shoham, B.: Timing analysis of a parallel algorithm for Toeplitz matrices on a MIMD parallel machine. Parallel Computing **17** (1991) 563–577
4. Gallivan, K., Thirumalai, S., Dooren, P.V.: On solving block toeplitz systems using a block schur algorithm. In: Proceedings of the 23rd International Conference on Parallel Processing. Volume 3., Boca Raton, FL, USA, CRC Press (1994) 274–281
5. Thirumalai, S.: High performance algorithms to solve Toeplitz and block Toeplitz systems. Ph.d. th., Grad. College of the U. of Illinois at Urbana–Champaign (1996)
6. Alonso, P., Badía, J.M., Vidal, A.M.: Parallel algorithms for the solution of toeplitz systems of linear equations. Lecture Notes in Computer Science **3019** (2004) 969–976
7. Anderson, E., et al.: LAPACK Users' Guide. SIAM, Philadelphia (1995)
8. Blackford, L., et al.: ScaLAPACK Users' Guide. SIAM, Philadelphia (1997)
9. Alonso, P., Badía, J.M., González, A., Vidal, A.M.: Parallel design of multichannel inverse filters for audio reproduction. In: Parallel and Distributed Computing and Systems, IASTED. Volume II., Marina del Rey, CA, USA (2003) 719–724
10. Loan, C.V.: Computational Frameworks for the Fast Fourier Transform. SIAM Press, Philadelphia (1992)
11. Heinig, G.: Inversion of generalized Cauchy matrices and other classes of structured matrices. Linear Algebra and Signal Proc., IMA **Math. Appl. 69** (1994) 95–114
12. Gohberg, I., Kailath, T., Olshevsky, V.: Fast Gaussian elimination with partial pivoting for matrices with displacement structure. Mathematics of Computation **64** (1995) 1557–1576
13. Alonso, P., Vidal, A.M.: An efficient and stable parallel solution for symmetric toeplitz linear systems. TR DSIC-II/2005, DSIC–Univ. Polit. Valencia (2005)
14. Kailath, T., Sayed, A.H.: Displacement structure: Theory and applications. SIAM Review **37** (1995) 297–386