J. Parallel Distrib. Comput. 74 (2014) 2392–2399

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Block pivoting implementation of a symmetric Toeplitz solver☆

Pedro Alonso [a,*], Manuel F. Dolz [b], Antonio M. Vidal [a]

[a] Dpto. de Sistemas Informáticos y Computación, Universitat Politècnica de València, Cno. Vera s/n, 46022 Valencia, Spain

[b] Department of Informatics, University of Hamburg, Bundesstraße 45a, 20146 Hamburg, Germany

HIGHLIGHTS

- We improve the solution of symmetric Toeplitz linear systems in multicore systems.
- We transform the Toeplitz matrix into a Cauchy-like one to obtain some benefits.
- The problem is partitioned into two half-sized independent problems.
- We use partial local pivoting to improve the accuracy of the solution.
- We propose a special scheme to store data in memory that accelerates the algorithm.

ARTICLE INFO

Article history:
Received 7 December 2011
Received in revised form
18 January 2014
Accepted 6 February 2014
Available online 15 February 2014

Keywords:
Symmetric Toeplitz matrices
Linear systems
Pivoting
Displacement structure
Multicores

ABSTRACT

Toeplitz matrices are characterized by a special structure that can be exploited in order to obtain fast linear system solvers. These solvers are difficult to parallelize due to their low computational cost and their closely coupled data operations. We propose to transform the Toeplitz system matrix into a Cauchy-like matrix since the latter can be divided into two independent matrices of half the size of the system matrix and each one of these smaller arising matrices can be factorized efficiently in multicore computers. We use OpenMP and store data in memory by blocks in consecutive positions yielding a simple and efficient algorithm. In addition, by exploiting the fact that diagonal pivoting does not destroy the special structure of Cauchy-like matrices, we introduce a local diagonal pivoting technique which improves the accuracy of the solution and the stability of the algorithm.

1. Introduction

The linear system of equations that we work with in this paper is defined as

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

where $T \in \mathbb{R}^{n\times n}$ is a real symmetric Toeplitz matrix, and $b, x \in \mathbb{R}^n$ are the independent right-hand side and the solution vectors, respectively. The elements of a symmetric Toeplitz matrix are $T_{i,j} = t_{|i-j|}$, with $t^T = \begin{pmatrix} t_0 & t_1 & \dots & t_{n-1} \end{pmatrix}^T$.

Toeplitz matrices appear in many areas of science and engineering. Signal Processing is one of these fields where Toeplitz matrices can be found in topics like filtering, linear prediction, etc. Signal processing interpretations of Toeplitz matrices can be found, e.g., in [28,22,30]. In particular, Toeplitz matrices appear in the solution of inverse filtering problems and equalization of multichannel acoustic systems where matrices can be very large, according to the filter length [16,17]. In some cases, the number of filters are also large, proportional to the number of sources (loudspeakers) like it is the case in [26,8], where 96 loudspeakers are used to position a sound signal in a 3D space of a room. The least squares problems which arise in that problems conduce to the solution of linear systems with Toeplitz matrices, sometimes nonsymmetric or symmetric indefinite. Efficient solvers, other than the traditional Levinson-type ones [18], represent a good alternative for these cases.

Many (*fast*) algorithms that exploit the special structure of Toeplitz matrices have been developed over recent years [23]. These algorithms reduce the $O(n^3)$ flops required to solve a dense linear system by at least one order of magnitude lower if any type of structure is taken into account. In general, fast algorithms can be classified into Levinson-type algorithms (which perform an

☆ This work was partially supported by the Spanish Ministerio de Ciencia e Innovación (Project TIN2008-06570-C04-02 and TEC2009-13741), Vicerrectorado de Investigación de la Universidad Politécnica de Valencia through PAID-05-10 (ref. 2705), and Generalitat Valenciana through project PROMETEO/2009/2013.

* Corresponding author.
E-mail addresses: palonso@dsic.upv.es (P. Alonso), dolzm@icc.uji.es (M.F. Dolz), avidal@dsic.upv.es (A.M. Vidal).

http://dx.doi.org/10.1016/j.jpdc.2014.02.003
0743-7315/

implicit computation of the inverse of the system matrix), and Schur-type algorithms (which perform a factorization of the system matrix) [15]. Levinson-type algorithms have a memory complexity of $O(n)$ data, but they are very rich in dot products with closely coupled operations. Therefore, a good speed-up in a parallel implementation is difficult to achieve. Schur-type algorithms usually need $O(n^2)$ data in memory and are easier to parallelize; however, they also have closely coupled operations [1].

Fast ($O(n^2)$) and superfast ($O(n \log^2 n)$) algorithms can be unstable or might return inaccurate solutions for indefinite Toeplitz matrices [10]. The work in [14] proposed moving the symmetric Toeplitz matrix to a symmetric Cauchy-like matrix to solve the linear system. This translation allows pivoting techniques to be incorporated in the algorithms since pivoting does not destroy the structure of Cauchy-like matrices, which is contrary to what happens with Toeplitz matrices.

The idea of using Cauchy-like matrices has also been used to develop parallel algorithms. The Cauchy-like matrix obtained in this way has great sparsity that can be exploited to reduce the linear system to two smaller independent linear systems of half the size of the original one, reducing both time and memory to solve the problem. For example, this was used in [33] to propose a shared memory algorithm. It was also used in [9] leading to a multilevel parallel MPI-OpenMP algorithm. In that paper, the Toeplitz matrix was transformed into a Cauchy-like matrix and split into two independent matrices, each one assigned to an MPI process which could be mapped onto different nodes in a network, or onto the same node thus fixing the problem of the lack of compilers that support OpenMP nested parallelism. Then, each one of these matrices arising from the former partition were factorized concurrently to obtain their LDL^T decomposition, but the scalability of this last factorization step was poor for many cores due to the low memory–CPU throughput of the algorithm. Different *out of order* strategies consisting of producer–consumer task queues implemented with `pthreads` were studied in [4] to improve the speed-up of the algorithm. However, it was shown that the sequential order in the computation of the blocks in which the triangular matrix is partitioned is as fast as other more complicated *out of order* approaches.

In this paper, we use a similar approach based on the transformation of linear system (1) into a *Cauchy-like* linear system. The algorithm applied to factorize Cauchy-like matrices makes use of very regular memory access patterns allowing independent blocks of the resulting triangular factor to be computed concurrently. Block versions of $O(n^3)$ algorithms have traditionally been developed to exploit the hierarchical memory levels. In the case of multicore computers, one step beyond this has been proposed to improve results. This step consists of storing all the data belonging to the same block in consecutive memory locations. It was initially proposed and investigated in [19–21] where the layout is referred to as *Square Block Format*. This technique has been successfully applied to level 3 algorithms of BLAS in [11] where the storage format is called *Block Data Layout* (BDL). Based on a hierarchical organization of the data by blocks, new ideas have been proposed as alternatives to the current implementation of LAPACK in different ways [12,34]. A recent contribution has been proposed, in particular, for a similar problem: the LDL^T decomposition of symmetric indefinite dense matrices [6]. In order to improve the efficiency of the parallel triangularization of each one of the two submatrices, we propose using the BDL storage format. The derived algorithm is also easy to implement since it is based on simple OpenMP directives instead of complicated task queues of `pthreads`.

In order to improve accuracy of the solution of symmetric linear systems by matrix decomposition diagonal pivoting (Bunch–Kaufman) is used. In parallel execution pivoting can reduce performance due to the data interchanging. Some variants of diagonal pivoting have been proposed for the factorization of symmetric indefinite matrices that improve performance, thanks to a reduction in the number of matrix column interchanges. This proposition can be found, e.g., in [31], where it is proposed and studied an algorithm variant supported on *lookahead*-type techniques. Also, for Toeplitz matrices, lookahead techniques have widely studied with the aim at improving accuracy, further to ensure stability of Levinson- and Schur-type algorithms which can even break down for well-conditioned matrices [13,7]. But, in general, look-ahead algorithms for Toeplitz matrices are based on heuristics with variable results (depend on the given matrix) and they are difficult to apply in concurrent environments where we try to keep the sough-after performance of the multicore system. Thus, our option consists of using *local diagonal pivoting* in the algorithm. Although diagonal and local pivoting has limitations compared with full or partial pivoting, we show through some examples that the precision of the solution might be improved. Since symmetric Cauchy-like matrices are not destroyed by diagonal pivoting and pivoting is bound to diagonal blocks (*local pivoting*), the execution time is barely affected.

The next section presents an abridged mathematical description of the problem and shows the overall algorithm. Details about the implementation of the proposed algorithm can be found in Section 3. The block pivoting technique incorporated to the algorithm is described in Section 4. Experimental results are shown in Section 5. Some conclusions are presented in Section 6.

2. Mathematical background

The solution of system (1) can be carried out by solving the linear system

$$C\,\tilde{x} = \tilde{b}, \tag{2}$$

where $C \in \mathbb{R}^{n\times n}$ is known as a *Cauchy-like* matrix, and $\tilde{b}, \tilde{x} \in \mathbb{R}^n$. Matrix $C = [a_{ij}]$ is called *Cauchy-like* (also *generalized Cauchy*) if for certain n-tuples of complex numbers $c = (c_i)_0^{n-1}$ and $d = (d_i)_0^{n-1}$ the matrix

$$\nabla(c, d)C = \left[(c_i - d_j)a_{ij}\right]_0^{n-1},$$

has a rank r which is "small" compared with the order of C. In this paper we deal with real symmetric *Cauchy-like* matrices, i.e., the n-tuples are real and $c = d$ [23]. The normalized Discrete Sine Transformation (DST), represented by means of the symmetric and orthogonal matrix $\mathcal{S}$ as defined in [25], allows system (1) to be transformed into system (2) by performing $C = \mathcal{S}T\mathcal{S}$, $\tilde{x} = \mathcal{S}x$, $\tilde{b} = \mathcal{S}b$. Matrices T and C both belong to the class of *structured matrices* [24]. Structured matrices are characterized by having a "low" *displacement rank* r ($r \ll n$), which briefly means that information contained in the full matrix is implicitly contained in only n-size r vectors. This property can be exploited to derive $O(n^2)$ algorithms for their triangular factorization. In the case of *Cauchy-like* matrix C (2), the displacement rank r is 4.

Working in the *Cauchy-like* domain has an additional advantage since entries c_{ij} such that $i + j$ is odd are 0. Let $P \in \mathbb{R}^{n\times n}$ be the odd–even permutation that positions the odd entries of an array to the top and the even entries to the bottom, then the linear system (2) can be divided into the two independent linear systems $C_i\hat{x}_i = \hat{b}_i$, for $i = 1, 2$, with $C_1 \in \mathbb{R}^{n_1\times n_1}$ and $C_2 \in \mathbb{R}^{n_2\times n_2}$, since

$$PCP^T = \begin{pmatrix} C_1 & \\ & C_2 \end{pmatrix}, \qquad P\tilde{x} = \begin{pmatrix} \hat{x}_1 \\ \hat{x}_2 \end{pmatrix} \quad \text{and} \quad P\tilde{b} = \begin{pmatrix} \hat{b}_1 \\ \hat{b}_2 \end{pmatrix}, \tag{3}$$

where $n_1 = \lceil n/2 \rceil$ and $n_2 = \lfloor n/2 \rfloor$.

Matrices C_1 and C_2 are also *Cauchy-like* (though they do not have zero entries as C) and have a displacement rank of 2, i.e.,

$$\begin{bmatrix} \Lambda_1 & \\ & \Lambda_2 \end{bmatrix}\begin{bmatrix} C_1 & \\ & C_2 \end{bmatrix} - \begin{bmatrix} C_1 & \\ & C_2 \end{bmatrix}\begin{bmatrix} \Lambda_1 & \\ & \Lambda_2 \end{bmatrix} = \begin{bmatrix} G_1 \\ G_2 \end{bmatrix} J \begin{bmatrix} G_1^T & G_2^T \end{bmatrix}^T \tag{4}$$