Annales UMCS Informatica AI 1 (2003) 81-87

Annales UMCS
Informatica
Lublin-Polonia
Sectio AI

http://www.annales.umcs.lublin.pl/

Fast solver for Toeplitz bidiagonal systems of linear equations

Przemysław Stpiczyński*

Department of Computer Science, Marie Curie-Skłodowska University,
Pl. M. Curie-Skłodowskiej 1, 20-031 Lublin, Poland

Abstract

We present a new efficient parallel algorithm for solving the first order linear recurrence systems with constant coefficients which is equivalent to the problem of solving Toeplitz bidiagonal systems of linear equations. The algorithm is formulated in the terms of level 1 and 2 BLAS (Basic Linear Algebra Subprograms) routines AXPY and GER. We also discuss its platform-independent implementation with OpenMP and finally present the results of experiments performed on a dual processor Pentium III computer running under Linux operating system with Altas as an efficient implementation of BLAS. The sequential version of the algorithm is up to 2.5 times faster than a simple sequential algorithm.

1. Introduction

Let us consider the problem of solving the following system of linear equations

$$\begin{pmatrix} 1 & & & 0 \\ -c & \mathbf{O} & & \\ & \mathbf{O} & \mathbf{O} & \\ 0 & & -c & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \mathbb{M} \\ x_n \end{pmatrix} = \begin{pmatrix} a_1 \\ a_2 \\ \mathbb{M} \\ a_n \end{pmatrix}. \tag{1}$$

The matrix of the system is a bidiagonal Toeplitz matrix wich means that entries are constant along each diagonal. The problem of solving (1) is equivalent to the problem of solving the following first order linear recurrence system with the constant coefficients

$$x_k = \begin{cases} a_1 & \text{for } k = 1 \\ a_k + cx_{k-1} & \text{for } k = 2,...,n \end{cases}. \tag{2}$$

The problem (1) or alternatively (2) arises in several fields of scientific computing [1-3]. For example, the well known Horner's scheme [4] can be expressed in terms of (2). The equation (2) is also a critical part of some

* *E-mail address:* przem@hektor.umcs.lublin.pl

numerical algorithms [5-7]. Unfortunately, optimizing compilers are not able to generate machine code which would fully utilize the underlying hardware, thus due to the Amdahl's law [8], such part of an algorithm can cause that the overall performance of a program will not be satisfactory. So it is very important to find an efficient method for solving the problem.

Different algorithms for the solution of the problem (2) have been designed for parallel [9-14] and vector [15-18] computers. However these algorithms like *cyclic reduction, Wang's method* and *recursive doubling* [12] lead to a substantial increase in the number of floating-point operations [8], what makes them unattractive in a classical serial systems (just like Intel Pentium) or parallel computers with a limited number of processors.

The aim of this paper is to present a new efficient algorithm for solving (1) based on a recently developed efficient algorithm for solving *m*-th order linear recurrence systems with constant coefficients [17, 19]. The algorithm is formulated in terms of level 1 & 2 BLAS (Basic Linear Algebra Subprograms) routines AXPY and GER [20, 8] and when an optimized version of BLAS is used (for example Atlas [21], then the algorithm is up to 2.5 times faster than a simple algorithm based on (2), even on one processor. Moreover, it can be easily parallelized on shared-memory parallel computers using OpenMP [5].

2. *Divide* and *conquer* approach

First let us note that our problem (2) is a special case of the more general problem of solving *m*-th order linear recurrence system with the constant coefficients for *n* equations [13, 22,]

$$x_k = \begin{cases} 0 & \text{for } k \le 0 \\ a_k + \sum_{j=1}^{m} c_j x_{k-j} & \text{for } 1 \le k \le n, \end{cases} \tag{3}$$

which can be efficiently solved on different parallel and vector computers using the recently developed *divide and conquer* algorithm [14, 17, 19, 21]. Now let us briefly describe the divide and conquer algorithm for solving the first order linear recurrence systems with the constant coefficients (m=1). For the sake of simplicity, let us assume that there exist integers r and s such that $rs = n$. However, this assumption can be easily omitted: after we find x_{rs}, we apply (2) to find $x_{rs+1},...,x_n$.

The recurrence equation (2) can be rewritten as the following block system of linear equations

$$\begin{pmatrix} L & & & \\ U & L & & \\ & \mathbf{O} & \mathbf{O} & \\ & & U & L \end{pmatrix} \begin{pmatrix} \mathrm{x}_1 \\ \mathrm{x}_2 \\ \mathrm{M} \\ \mathrm{x}_r \end{pmatrix} = \begin{pmatrix} \mathrm{a}_1 \\ \mathrm{a}_2 \\ \mathrm{M} \\ \mathrm{a}_r \end{pmatrix}, \tag{4}$$

where $\mathrm{x}_j = \left(x_{(j-1)s+1}, ..., x_{js}\right)^T$, $\mathrm{a}_j = \left(a_{(j-1)s+1}, ..., a_{js}\right)^T \in \mathbf{R}^s$ and the matrices are given by

$$\begin{pmatrix} 1 & & & 0 \\ -c & \mathbf{O} & & \\ & \mathbf{O} & \mathbf{O} & \\ 0 & & -c & 1 \end{pmatrix}, \; U = \begin{pmatrix} & -c \\ 0 & \end{pmatrix} \in \mathbf{R}^{s\times s} \tag{5}$$

The system (4) corresponds to the following recurrence system:

$$\begin{cases} \mathrm{x}_1 = L^{-1}\mathrm{a}_1 \\ \mathrm{x}_j = L^{-1}\mathrm{a}_j - L^{-1}U\mathrm{x}_{j-1} \quad \text{for } j = 2,...,r. \end{cases} \tag{6}$$

When we consider the structure of the matrix $U = -c\mathrm{e}_1\mathrm{e}_s^{\mathrm{T}}$, where e_k denotes k-th unit vector of $\mathbf{R}^s$ and set $\mathrm{z}_j = L^{-1}\mathrm{a}_j$, then we get

$$\begin{cases} \mathrm{x}_1 = \mathrm{z}_1 \\ \mathrm{x}_j = \mathrm{z}_j + a_j\mathrm{y} \quad \text{for } j = 2,...,r, \end{cases} \tag{7}$$

where $\mathrm{y}_j = L^{-1}\mathrm{e}_1$ and $a_j = cx_{(j-1)s}$ for $j = 2,...,r$.

The *divide and conquer* algorithm [13, 14] proceeds as follows. First we find (in parallel) all vectors z_k and y, then we find (sequentially) all coefficients a_j and numbers $x_{(j-1)s}$, $j = 2,...,r$. Finally (again in parallel) we calculate $s-1$ first entries of x_j, $j = 2,...,r$. Experimental results show that the algorithm achieves reasonable performance for a bigger number of processors [23].

3. BLAS-based algorithm

In our earlier work [17] we have shown that the first step of the algorithm reduces to the problem of solving the following system of linear equations

$$LZ = F, \tag{8}$$

where L is given by (5) and

$$Z = \left(\mathrm{z}_1, ..., \mathrm{z}_r, \mathrm{y}\right)^T, \; F = \left(\mathrm{a}_1, ..., \mathrm{a}_r, \mathrm{e}_1\right)^T \in \mathbf{R}^{s\times(r+1)}.$$

The solution Z can be found row by row using the following vector-recurrence formula

$$Z_{k,*} = \begin{cases} F_{1,*} & \text{for } k = 1 \\ F_{k,*} + cZ_{k-1,*} & \text{for } k = 2,...,s. \end{cases} \tag{9}$$

where $Z_{k,*}$ and $F_{k,*}$ $ denote the *k*-th row of *Z* and *F* respectively. Thus each row of *Z* can be computed using one call to the level 1 BLAS operation AXPY of the form $\mathrm{x} \leftarrow \mathrm{x} + a\,\mathrm{y}$. Analogously, the last step of the algorithm can be expressed in terms of AXPY, which gives us a very fast algorithm for vector computers [17]. Unfortunately, in the case of scalar processors (just like Pentium III), the performance of the algorithm is comparable with the performance of a simple algorithm based on (2).

The main idea of our new algorithm is to speed up the last step of the algorithm by using the routine GER from the level 2 BLAS. This routine is pretty much faster than the corresponding sequence of calls to the routine AXPY because it reduces the number of memory references in comparison with the number of arithmetic operations [8]. During the second step we collect the computed values of all coefficients a_j and compose the vector

$$\mathrm{u} = \left(a_1,...,a_r\right)^T \in \mathbf{R}^{\mathrm{r-1}}. \tag{10}$$

In the third (final) step, we compute the remaining $s-1$ entries of all vectors x_j, $j = 2,...,r$. It can be done by one call to the level 2 BLAS routine GER of the form $A \leftarrow A + \mathrm{x}\mathrm{y}^T$. Let x_j', z_j' and y_j' denote $s-1$ first entries of vectors x_j, z_j and y respectively. Then from (7) we get

$$\mathrm{x}_j' = \mathrm{z}_j' + a_j \mathrm{y}'.$$

Now using (10) we conclude that the matrix $\left(\mathrm{x}_2',...,\mathrm{x}_r'\right) \in \mathbf{R}^{(s-1)\times(r-1)}$ satisfies the following

$$\left(\mathrm{x}_2',...,\mathrm{x}_r'\right) = \left(\mathrm{z}_2',...,\mathrm{z}_r'\right) + \mathrm{y}'\mathrm{u}^T. \tag{11}$$

This algorithm can be easily parallelized on shared-memory machines. Namely, if we partition the matrices *Z* and *F* into blocks of columns, then (8) can be rewritten in the following form

$$L\left(Z_1,...,Z_p\right) = \left(F_1,...,F_p\right) \tag{12}$$

and each block Z_j can by calculated using (9). Thus, when *p* processors are available, each processor will be responsible for computing one block Z_j. Similarly we can parallelize (11).

4. Performance analysis and results of experiments

Now let us study some basic facts on the complexity of the considered algorithms. It is clear that the simple algorithm based on (2) required $2n-2$ floating point operations ("flops").

Proposition 1 The number of floating-point operations required by the sequential BLAS-based algorithm is

$$T_{BLAS,1}(n,r,s) = 2rs - 2r + 2s - 4 + 2n. \tag{13}$$

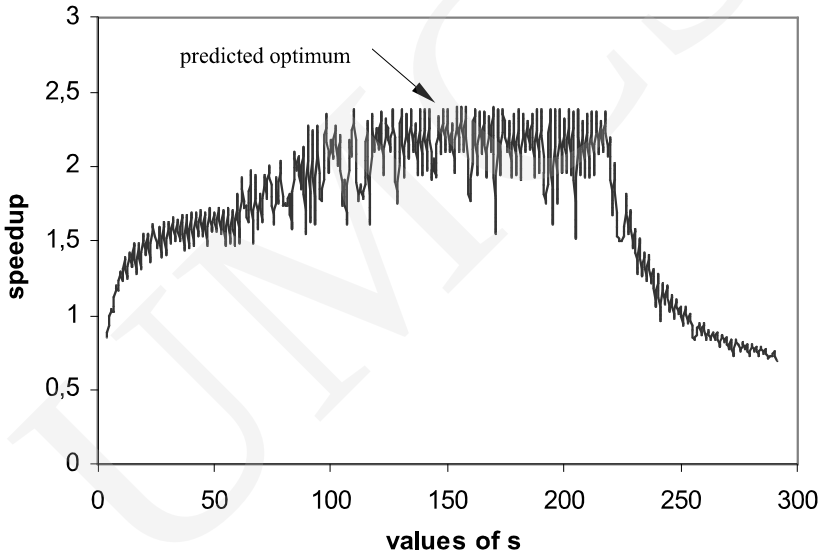

Fig. 1. Performance (in MFlops) od the sequential BLAS-based algorithm for various *s* and *n*=42500

Proof. After we choose the integers *r* and *s*, we perform a sequence of $s-1$ calls to the operation AXPY. In the second step we find last entries of $r-1$ vectors $x_2,...,x_r$. The third step consists of $2(r-1)r$ flops. Finally we find the numbers $x_{rs+1},...,x_n$ using (2). Thus

$$T_{BLAS,1}(n,r,s) = 2(r+1)(s-1) + 2(r-1) + 2(s-1)r + 2(n-rs).$$

■

The method has been implemented in FORTRAN 77 and OpenMP [24] and tested on a dual processor Pentium III 866MHz computer running under Linux operating system. We have used Atlas [21] as the optimized version of BLAS and Omni OpenMP compiler to express the parallel execution of (12). The algorithm has been tested varying the problem sizes *n* and values of the parameter *s*. To discover an asymptotic behavior of the algorithm we have

decided to run our program not only for small degree systems but also for very large degree. The wall clock time has been measured using the routine omp_get_wtime(). Results of the experiments can be summarized as follows.

1. The BLAS-based algorithm is faster than the simple algorithm based on (2) for n>500. However, the algorithm could be much faster on computer systems where the performance of AXPY and GER is relatively high.
2. The BLAS-based algorithms (both sequential and parallel) achieve the best performance for the value of the parameter $s \approx \sqrt{n/2}$.
3. The sequential BLAS-based algorithm is up to 2.5 times faster than the simple algorithm based (Figure 2). The use of the parallel BLAS-based algorithm is profitable for great problem sizes (n>40000).
4. For smaller values of n, the whole coefficient vector can be stored in the processor cache (Pentium III has 256KB cache). For n>45000 the performance of all algorithms rapidly decreases. This is caused by cache misses.

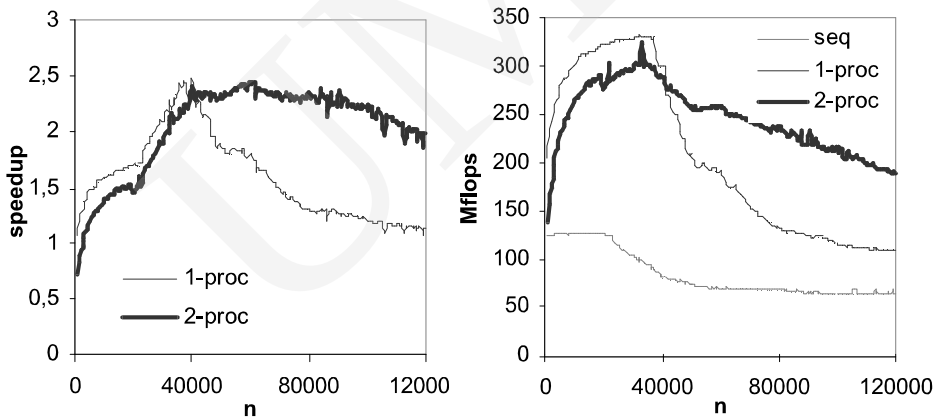

Fig. 2. Performance (in MFlops) of the BLAS-based algorithm for various n

References

[1] Li L., Hu J., Nakamura T., *A simple parallel algorithm for polynomial evaluation*, SIAM J. Sci. Comput., 17 (1996) 260.
[2] Munro I., Paterson M., *Optimal algorithms for parallel polynomial evaluation*, J. Comput. System Sci., 7 (1973) 189.
[3] Swann H., *On solenoidal high-degree polynomial approximations to solutions of the stationary Stokes equations*, Numer. Methods Partial Differ. Equations, 16 (2000) 480.
[4] Stoer J., Bulirsh R., *Introduction to Numerical Analysis*, Springer, New York, (1993).
[5] Abu-Shumays I., *Comparison of methods and algorithms for tridiagonal systems and for vectorization of diffusion computations*, In: Numrich, R., ed., Supercomputer Applications, New York, Plenum Press, (1985).
[6] Guitart J., Ruiz-Moreno S., *Strict calculation of the light statistics at the output of a travelling wave optical amplifier*, Elecronics Letters, 29 (1993) 1589.

[7] Larriba-Pey J.L., Navarro J.J., Jorba A., *Vectorized algorithms for natural cubic spline and B-spline curve fitting*, In: Proceedings of PDP'96 Braga., (1996), 385.
[8] Dongarra J., Duff I., Sorensen D., Van der Vorst H., *Solving Linear Systems on Vector and Shared Memory Computers*, SIAM, Philadelphia, (1991).
[9] Carlson D.A., *Solving linear recurrence systems on mesh-connected computers with multiple global buses*, Journal on Parallel and Distributed Computing, 8 (1990) 89.
[10] Gajski D., *Processor array for computing linear recurrence systems*, In: Preceeding of the International Conference on Parallel Processing., (1978) 246.
[11] Greenberg A., R.E.Lander, Paterson M., Galil Z., *Efficient parallel algorithms for linear recurrence computation*, Inf. Proc. Letters, 15 (1982) 31.
[12] Larriba-Pey J.L., Navarro J.J., Jorba A., Roig, O., *Review of general and Toeplitz vector bidiagonal solvers*, Parallel Computing, 22 (1996) 1091.
[13] Paprzycki M., Stpiczyński P., *Parallel solution of linear recurrence systems*, Z. Angew. Math. Mech., 76 (1996) 5.
[14] Stpiczyński P., *Parallel algorithms for solving linear recurrence systems*, Lecture Notes in Computer Science, 634 (1992) 343.
[15] Axelsson O., Eijkhout V., *A note on the vectorization of scalar recursions*, Parallel Computing, 3 (1986) 73.
[16] Hafner H., Shonauer W., *Investigation of different algorithms for the first order recurrence*, Supercomputer, 40 (1990) 34.
[17] Stpiczyński P., Paprzycki M., *Fully vectorized solver for linear recurrence systems with constant coefficients*, In: Proceedings of VECPAR 2000-4th International Meeting on Vector and Parallel Processing, Porto, June 2000, Facultade de Engerharia do Universidade do Porto, (2000) 541.
[18] Van Der Vorst H.A., Dekker K., *Vectorization of linear recurrence relations*, SIAM J. Sci. Stat. Comput., 16 (1989) 27.
[19] *A new message passing algorithm for solving linear recurrence systems*, Lecture Notes in Computer Science, 2328 (2002) 466.
[20] Anderson E., Bai Z., Bischof C., Demmel J., Dongarra J., Du Croz J., Greenbaum A., Hammarling S., McKenney A., Ostruchov S., Sorensen D., *LAPACK User's Guide*, SIAM, Philadelphia, (1992).
[21] Whaley R.C., Petitet A., Dongarra J.J., *Automated empirical optimizations of software and the ATLAS project*, Parallel Computing, 27 (2001) 3.
[22] Modi, J., *Parallel Algorithms and Matrix Computation*, Oxford University Press, Oxford, (1988).
[23] Paprzycki M., Stpiczyński P., *Solving linear recurrence systems on the Cray Y-MP*, Lecture Notes in Computer Science, 879 (1994) 416.
[24] Chandra R., Dagum L., Kohr D., Maydan D., McDonald J., Menon R., *Parallel Programming in OpenMP*, Morgan Kaufmann Publishers, San Francisco, (2001).