Annales UMCS Informatica AI 2 (2004) 263-268

Annales UMCS
Informatica
Lublin-Polonia
Sectio AI

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

Evaluating linear recursive filters on clusters of workstations

Przemysław Stpiczyński*

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

Abstract

The aim of this paper is to show that the recently developed high performance algorithm for solving linear recurrence systems with constant coefficients together with the new BLAS-based algorithm for narrow-banded triangular Toeplitz matrix-vector multiplication allow to evaluate linear recursive filters efficiently, even on clusters of workstations. The results of experiments performed on a cluster of twelve Linux workstations are also presented. The performance of the algorithm is comparable with the performance of two processors of Cray SV-1 for such kind of recursive problems.

1. Introduction

Let us consider the following problem of evaluating linear recursive filters which is very popular in signal processing [1]. For a given sequence of real numbers $x_1,x_2,\ldots,x_n$ called input signals, we have to evaluate an output sequence $y_1,y_2,\ldots,y_n$, satisfying

$$y_k = \sum_{j=1}^{m} b_j y_{k-j} + \sum_{j=0}^{m} a_j x_{k-j} , \tag{1}$$

where $x_k = 0$, $z_k = 0$ for $k \le 0$ and coefficients a_j, b_j are calculated using z-transforms [1]. Unfortunately, the problem has "recursive" nature and simple routines based on (1) do not fully utilize the underlying hardware, i.e. memory hierarchies and multiple processors, and they achieve poor performance. On the other hand, our problem is a typical example of how the speed of the slowest parts of programs influences the overall performance (so called *Amdahl's law* [2]). Thus, it is clear that efficient high performance algorithms for solving our problem should be designed.

It is well known that reducing costs of memory access is essential for achieving good performance of numerical software [3]. Usually, codes based on Level 2 and 3 BLAS (Basic Linear Algebra Subprograms [2]) routines achieve

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

good performance because they allow to reuse the data stored in cache memory [4]. The use of higher levels BLAS routines is the simplest way to utilize the processor's hardware. Moreover, such programs can be easily parallelized. Following this observation we can conclude that our problem can be efficiently solved on modern computer architectures, including clusters of workstations, if we rewrite it in terms of BLAS routines. To do it, let us split (1) into the following two equations:

$$f_k = \sum_{j=0}^{m} a_j x_{k-j} \tag{2}$$

and

$$y_k = f_k + \sum_{j=1}^{m} b_j y_{k-j} \,. \tag{3}$$

In our recent paper [5] we introduced a new algorithm based on Level 2 and 3 BLAS routines for solving (3), which can be efficiently implemented on various shared-memory [6] and distributed-memory [7] parallel computers. The aim of this paper is to present the new high performance, BLAS-based algorithm for evaluating (2), which together with the algorithm for solving (3) allows to evaluate (1) efficiently, even on clusters of workstations with relatively slow connection speed.

2. Algorithm

The algorithm for evaluating (1) comprises two stages. First we compute all coefficients f_k and then we solve the recurrence system (3) using the BLAS-based parallel algorithm [5,7]. For the sake of simplicity, let us introduce the following notation. Let $M \in \mathbb{R}^{m\times n}$. Then $M_{i:j,k:l}$ denotes the submatrix of M formed by intersection of rows i to j and columns k to l. Moreover, let $M_{i:j,*} = M_{i:j,1:n}$, $M_{*,k:l} = M_{1:m,k:l}$ *and* $M_{i:j,k} = M_{i:j,k:k}$, $M_{i,k:l} = M_{i:i,k:l}$.

Let us observe that equation (2) can be rewritten in the following matrix-vector form.

$$\begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ \vdots \\ f_{n-1} \\ f_n \end{pmatrix} = \begin{pmatrix} a_0 & & & & & \\ a_1 & \ddots & & & & \\ \vdots & \ddots & \ddots & & & \\ a_m & & \ddots & \ddots & & \\ & \ddots & & \ddots & \ddots & \\ & & a_m & \cdots & a_1 & a_0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ \vdots \\ x_{n-1} \\ x_n \end{pmatrix} \tag{4}$$

Note that the matrix is of the Toeplitz form which means that entries are constant along each diagonal. Now let us choose two positive integers r and s such that $rs \le n$ and for $j=1,\ldots,r$ define vectors

$$\mathbf{x}_j = \left(x_{(j-1)s+1},...,x_{js}\right)^T \text{ and } \mathbf{f}_j = \left(f_{(j-1)s+1},...,f_{js}\right)^T \in \mathbb{R}^s$$

and matrices

$$L = \begin{pmatrix} a_0 & & & & \\ \vdots & \ddots & & & \\ a_m & & \ddots & & \\ & \ddots & & \ddots & \\ & & a_m & \cdots & a_0 \end{pmatrix}, \quad U = \begin{pmatrix} & a_m & \cdots & a_1 \\ & & \ddots & \vdots \\ & & & a_m \\ 0 & & & \end{pmatrix} \in \mathbb{R}^{s\times s}.$$

Then all vectors $\mathbf{f}_j$ satisfy

$$\begin{cases} \mathbf{f}_1 = L\mathbf{x}_1 \\ \mathbf{f}_j = U\mathbf{x}_{j-1} + L\mathbf{x}_j \quad \text{for } j = 2,...,r. \end{cases} \tag{5}$$

Note that we can use (5) to compute $f_1,...,f_{rs}$ and (2) to find $f_{rs+1},...,f_n$. When we define matrices

$$F = \left(\mathbf{f}_1,...,\mathbf{f}_r\right), \quad X = \left(\mathbf{x}_1,...,\mathbf{x}_r\right) \in \mathbb{R}^{s\times r}$$

and

$$G = \begin{pmatrix} a_m & a_{m-1} & \cdots & a_1 \\ 0 & a_m & \cdots & a_2 \\ \vdots & & \ddots & \vdots \\ 0 & \cdots & 0 & a_m \end{pmatrix} \in \mathbb{R}^{m\times m},$$

we can find vectors $\mathbf{f}_j$ as follows. First we perform the operation $F \leftarrow LX$ and next we update first m entries of each vector $\mathbf{f}_2$,…,$\mathbf{f}_r$ using $GX_{s-m+1:s,j}$, j=1,…,r-1, respectively. Thus the algorithm comprises the following two steps:

Step A1:

$$F_{k,*} \leftarrow \left(a_{\min\{k-1,m\}},...,a_1,a_0\right)\cdot X_{\max\{1:m-k\}:k,*}, \quad \text{for } k = 1,...,r. \tag{6}$$

Step A2:

$$F_{1:m,2:r} \leftarrow F_{1:m,2:r} + GX_{s-m+1:s,1:r-1}. \tag{7}$$

Note that **Step A1** can be implemented as a sequence of calls to the Level 2 BLAS operation _GEMV, i.e. matrix-vector multiplication, while **Step A2** as the Level 3 BLAS operation _TRMM, i.e. triangular matrix - general matrix multiplication. The second stage of the algorithm for finding (1) can be done using the *divide and conquer* algorithm for solving (3), which is based on Level 2 and 3 BLAS routines _GEMV, _TRMV and _GEMM} [5].

3. Implementation

The algorithm can be easily implemented on distributed-memory parallel computers and clusters of workstations using MPI [8]. Generally speaking, each processor is responsible for computing a block of output signals y_i. We assume

that each processor holds (or calculates) coefficients $a_0,\ldots,a_m$ and $b_1,\ldots,b_m$. Also, each processor holds r/p columns of the matrix X, where p is the number of available processors. The first stage of the algorithm requires communication, namely each processor (except for the last one) sends m last entries of its last column of the block of X to its "left" neighbour. The details of the second stage of the algorithm can be found in [7].

It is obvious that the performance of the algorithm depends on the choice of the parameters s and r. The second stage is more complicated thus it is clear that the parameters should be chosen to minimize the cost of that stage. To predict the behaviour of the algorithm for solving (3), we have observed [7] that its complexity can be expressed in terms of Bulk Synchronous Parallel Architecture (BSP for short [9]). In this model, a parallel program consists of a number of supersteps. Each superstep comprises local computations, global data exchange and the barrier synchronization (see Figure 1).

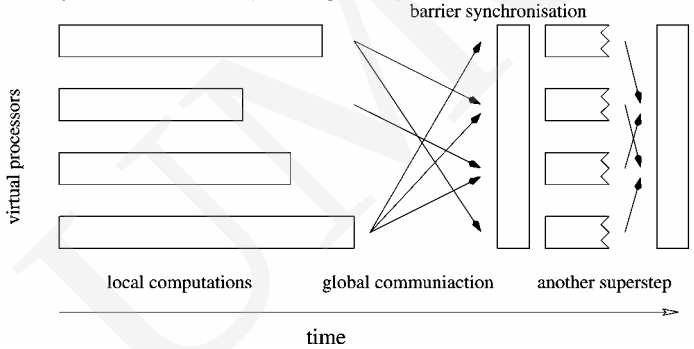

Fig. 1. BSP architecture

The BSP model is characterized by the following parameters: p, the number of available processors, g, the time (in flop time units) it takes to communicate (send or receive) a data element and l, the time (in flop time units) it takes all processors to synchronize. The complexity of a superstep is defined as

$$w_{\max} + gh_{\max} + l$$

where $w_{\max}$ is the maximum number of flops performed, and $h_{\max}$ is the number of messages sent or received by any one processor during this superstep.

To find the optimal value of s, we minimize the BSP cost of the "most time consuming" steps of the algorithm, namely

$$2m\left(s-\frac{m+1}{2}\right)\left(\frac{n}{sp}+1\right)+3m^2\frac{n}{s}+mg\left(p-1\right)+lp \tag{8}$$

and because the value of s should be integer, we get

$$s^*=\left\lfloor\sqrt{\frac{n\left(3mp-m-1\right)}{2p}}\right\rfloor. \tag{9}$$

Thus we can conclude that for the algorithm, the optimal choice of the value of the parameter s does not depend on the BSP parameters. So we can expect that the choice (9) will be good for a wide variety of BSP architectures and the formula (9) can be included in the source code of the algorithm.

4. Results of experiments

The algorithm has been tested on a cluster of twelve 667 MHz Pentium III workstations running under Linux operating system for various problem sizes (n, m) and the number of processors (p). The results of experiments can be summarized as follows.

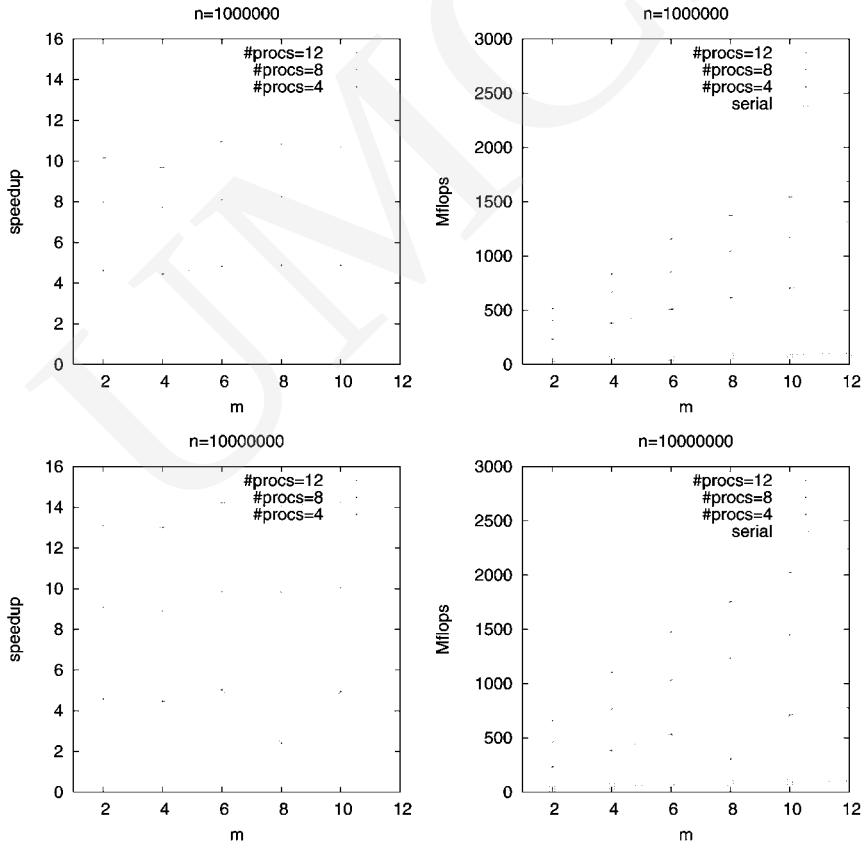

Fig. 2. The performance of the algorithm

1. The algorithm achieves the best performance for the values of $s=0.5s^*$. The performance of the algorithm depends strictly on the performance of the BLAS routines (which is not taken into account in BSP) and for such values of s we achieve "the perfect balance" between the BSP cost and the

efficiency of the BLAS routines. When $s=s^*$, then the matrices are too narrow.

2. The performance of the algorithm (in Mflops) grows when the problem sizes (n, m) and the number of processors grow. However the performance of simple serial algorithm based on (1) also grows when m grows. Thus we can observe that the speedup is almost constant when m grows. For p=12, the speedup of the algorithm is up to 15.
3. When p=12, then the performance of the algorithm (in Mflops) is up to 2200 Mflops, which is comparable with the performance of two processors of Cray SV-1 [6] for solving such a kind of recursive problems.

Acknowledgments

The work has been developed within task WP14 of 6 T11 2003 C/06098 "CLUSTERIX - The national Linux Cluster".

References

[1] Smith S.W., *The Scientist and Engineer's Guide to Digital Signal Processing*, California Technical Publishing, San Diego, CA, (1997).
[2] Dongarra J., Duff I., Sorensen D., Van der Vorst H,. *Solving Linear Systems on Vector and Shared Memory Computers*, SIAM, Philadelphia, (1991).
[3] Dongarra J., Hammarling S., Sorensen D., *Block reduction of matrices to condensed form for eigenvalue computations*, J. Comp. Appl. Math, 27 (1989) 215.
[4] Baker A., Dennis J., Jessup E.R., *Toward memory-efficient linear solvers*, Lecture Notes in Computer Science, 2565 (2003) 315.
[5] Stpiczyński P., *Solving linear recurrence systems using level 2 and 3 BLAS routines*, Lecture Notes in Computer Science, 3019 (2004) 1059.
[6] Stpiczyński P., *Numerical evaluation of linear recurrences on various parallel computers*. In M.Kovacova, editor, Proceedings of Aplimat 2004, 3rd International Conference, Bratislava, Slovakia, (2004) 889.
[7] Stpiczyński P., *Numerical evaluation of linear recurrences on high performance computers and clusters of workstations*, In Proceedings of PARELEC 2004, International Conference on Parallel Computing in Electrical Engineering. IEEE Computer Society Press, (2004).
[8] Pacheco P., *Parallel Programming with MPI*, Morgan Kaufmann, San Francisco, (1996).
[9] Hill J., Donaldson S., Skillicorn D., *Portability of Performance with the BSPlib Communications Library*, In Programming Models for Massively Parallel Computers, (MPPM'97), London, IEEE Computer Society Press, (1997).