Proceedings of the 15th IASTED International Conference
PARALLEL AND DISTRIBUTED COMPUTING AND SYSTEMS
November 3-5, 2003, Marina Del Rey, CA, USA

Parallel design of multichannel inverse filters for audio reproduction*

P. Alonso† J. M. Badía‡ A. González§ A. M. Vidal†

ABSTRACT

In this paper, we present an efficient and portable parallel algorithm for multichannel inverse filter design in sound reproduction systems. The actual matrix used to specify the electroacoustic transmission paths is the matrix of causal finite impulse response filters. An array of finite impulse response filters is designed in a way that best approximates a given time domain response in the least squares sense. We exploit the non–symmetric block–Toeplitz structure of the multichannel system matrix in order to develop a fast parallel algorithm to solve the arising least squares problem. Experimental results, obtained on a cluster of personal computers and on a bi–processor board, show the performance of our approach to the inverse filtering problem.

KEY WORDS

Block–Toeplitz, Multichannel deconvolution, Inverse filters, Audio reproduction, Parallel least squares

1 Introduction

Inverse filtering and equalization of multichannel systems is a field of growing interest. This fact is mainly due to the upcoming applications of multichannel systems such as digital communication (mainly new generation digital mobile communications that incorporates array processing at the base stations) and the modern multichannel audio reproduction systems such as three-dimensional (3-D) audio [1], or active noise control, and the availability of new technology resources which make possible the implementation of more complex signal processing algorithms.

The mathematical model of inverse filtering and equalization multichannel systems are standing for large-scale matrix problems with structure. The major challenge in this area is to design fast and numerically reliable algorithms for large-scale structured linear matrix equations and the least squares matrix problem. For small-size problems, there is often not much else to do except to use one of the already standard methods of solution such as Gaussian elimination or the QR decomposition. However, as the problem size increases it is important to identify special structures in order to reduce the computational burden. A very extensive work have been made in this way and, as a result of, many algorithms have been developed exploiting the special structure.

Several algorithms have been traditionally used to solve systems of equations or the linear least squares problem of Toeplitz–like matrices exploiting its special structure to get a computational cost an order of magnitude lower than other classical algorithms for non–structured matrices. These are the well–known *fast* algorithms. For example, the Levinson–Durbin recursion has been proved to be very useful in several cases [2]. A generalized block processing version of the Levinson–Durbin algorithm [3] was used in [4] and [5], to design inverse filters for cross–talk cancellation. Other algorithm can be found in [6].

Several problems in signal processing like inverse filtering have strong time constraints. This fact leads us to develop parallel algorithms for getting improved performance. But, the lower cost and the data coupling of *fast* algorithms makes it difficult to get efficient parallel algorithms. In this work, we have design an efficient parallel *fast* algorithm to solve the structure matrix problem arising in multichannel systems. Furthermore, our aim is to get good results in general purpose parallel architectures, that is, in clusters of personal computers. We have made a hard effort minimizing the communication cost on these distributed architectures characterized for a large relation between communication and computation speed.

We have used linear algebra subroutines of BLAS and LAPACK [7] for sequential computations optimized for the machine target platform. For parallel processing, linear algebra subroutines in the ScaLAPACK and PBLAS libraries have been used, including BLACS subroutines for sending/receiving arrays [8]. MPI environment has been used for message–passing. These libraries provides portability for our code on very different parallel environments. Also, MPI can be run in shared memory machines. Thus, our algorithm can be used on bi– or tetra–processor boards of isolated workstations. Also, our parallel algorithm is suitable for DSP's or another dedicated architectures.

This paper is set out as follows: firstly, a brief description of multichannel sound reproduction systems is presented. Secondly, we propose a sequential algorithm, and, in section 4, we describe a parallel algorithm based on the sequential one. In the following section, we comment the experimental results obtained on a cluster of personal computers. In the experimental results section we also show a timing analysis of our parallel algorithm running on a bi–processor board for solving a common problem in sound reproduction systems: the *cross talk cancellation* problem. Finally, we include some concluding remarks.

*Supported by Spanish CICYT Project TIC 2000-1683-C03-01-03.

†{palonso avidal}@dsic.upv.es, Dept. Sistemas Informáticos y Computación Universidad Politécnica de Valencia.

‡badia@icc.uji.es, Dept. de Ingeniería y Ciencia de los Computadores Universidad Jaime I.

§agonzal@dcom.upv.es. Dept Comunicaciones, Universidad Politécnica de Valencia.

2 The Multichannel Sound Reproduction System

Figure 1 shows a block diagram of a multichannel sound reproduction system. The configuration showed in Figure 1 is typical in multichannel sound reproduction systems, where inverse filters are usually calculated using the least squares method in the time domain [9]. The reproduction system renders K input signals into the listening space. A block labeled $\mathbf{H}$ filters these input signals prior to feeding the electroacoustic system. As illustrated in Figure 1, each input signal is filtered through an array of filters whose impulse responses are denoted by $h_{i,j}[n]$, where i indicates which source is reproducing the filter output and j which input signal is filtered by this filter. A different set of filters is chosen depending on the desired application. Cross–talk cancellation, inverse filtering, equalization and virtual source positioning represent some examples of these applications. Regardless of the selection of the desired application, physical and computational boundaries will constrain the filter design. Therefore, efficient and practical computational methods are needed to carry out this design task.

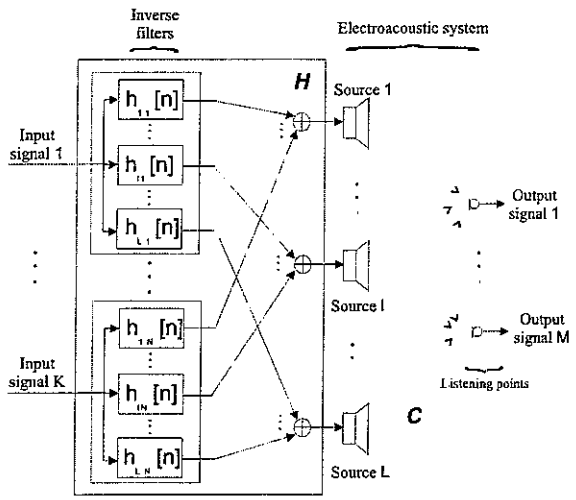

Figure 1. Multichannel sound reproduction system.

The inverse filtering problem in practical multichannel audio reproduction systems basically consists of building a matrix of digital finite duration filters (a different filter vector for each signal to be rendered), whose convolutions with the signal transmission channels best approximate a desired response.

The filter matrix is calculated using the electroacoustic system (or transmission channels) actual responses, which were previously measured, and the desired signals at the listening or reproduction points. Electroacoustic system responses are also usually modeled as finite duration filters. From a mathematical point of view, the multichannel inverse filtering problem is usually resolved using a linear set of equations that results from the linear convolutions of the filter matrix and the electroacoustic system responses. However, it may not be possible in practice to obtain a squared linear set of equations, hence a least squared error solution is preferred.

The electroacoustic system responses will be modeled as FIR filters of n_c coefficients, and $c_{i,j}(k)$ will be the $(k+1)$th coefficient of the channel between the jth source (loudspeaker) and the ith listening point (microphone). Every inverse filter will have n_h coefficients and be denoted by $\mathbf{h}_{i,j} = [h_{i,j}(0), \ldots, h_{i,j}(n_h-1)]^T$; this filter processes the jth input signal to feed the ith loudspeaker. By applying the superposition principle, each input signal can be separately considered. Thus, each input signal has its corresponding desired signal at every listening point. These desired signals will be vectors of (n_c+n_h-1) coefficients, $\mathbf{a}_{i,j} = [a_{i,j}(0), \ldots, a_{i,j}(n_c+n_h-2)]^T$, where subscript i corresponds to the listening point and subscript j corresponds to the input signal. Each input signal is filtered through a different set of filters. Therefore, the set of filters that processes a given input signal can be independently calculated without any loss of generality. A single input signal is usually assumed throughout this paper.

A reproduction system with L loudspeakers and M microphones can be expressed as

$$[\mathbf{e}]_{(n_c+n_h-1)M\times 1} = [\mathbf{C}]_{(n_c+n_h-1)M\times n_h L}\,[\mathbf{h}]_{n_h L\times 1} - [\mathbf{a}]_{(n_c+n_h-1)M\times 1}, \quad (1)$$

where $\mathbf{h}^T = [\mathbf{h}_{1,1}^T, \ldots, \mathbf{h}_{I,1}^T]$, $\mathbf{a}^T = [\mathbf{a}_{1,1}^T, \ldots, \mathbf{a}_{M,1}^T]$, $\mathbf{e}^T = [\mathbf{e}_{1,1}^T, \ldots, \mathbf{e}_{M,1}^T]$, and matrix $\mathbf{C}$ is composed by $M \times L$ blocks $\mathbf{C}_{i,j}$ of the form

$$\mathbf{C}_{i,j} = \begin{bmatrix} c_{i,j}(0) & 0 & 0 \\ c_{i,j}(1) & c_{i,j}(0) & \\ \vdots & c_{i,j}(1) & \\ c_{i,j}(n_c-1) & \vdots & c_{i,j}(0) \\ 0 & c_{i,j}(n_c-1) & c_{i,j}(1) \\ & & \vdots \\ 0 & & c_{i,j}(n_c-1) \end{bmatrix}$$

Expression (1) represents a system of equations where matrix $\mathbf{C}$ is a nonsquare matrix composed by nonsquare Toeplitz blocks. A least squares error criteria is commonly used to calculate the array of inverse filters. Thus, the array of inverse filters in (1), $\mathbf{h}$, is chosen to minimize the squared error,

$$\min_{\mathbf{h}}\{\|\mathbf{e}\|_2^2\} = \min_{\mathbf{h}}\{\|\mathbf{Ch}-\mathbf{a}\|_2^2\}. \quad (2)$$

3 The Sequential Algorithm

In this work, we solve the least squares linear problem (2) by means of the associated normal equations:

$$\mathbf{C}^T\mathbf{Ch} = \mathbf{C}^T\mathbf{a}$$

The above normal equations are solved by computing the triangular decomposition of the product $\mathbf{C}^T\mathbf{C}$ in order to get the following seminormal equations

$$\mathbf{L}^T\mathbf{L}\mathbf{h} = \mathbf{C}^T\mathbf{a}, \tag{3}$$

where $\mathbf{L}$ is a lower triangular factor.

In our exposition, we will use the following representation for the *displacement* of matrix $\mathbf{C}^T\mathbf{C}$ with regard to the *displacement matrix* $\mathbf{F}$,

$$\nabla_F = \mathbf{C}^T\mathbf{C} - \mathbf{F}\mathbf{C}^T\mathbf{C}\mathbf{F}^T = \mathbf{G}\mathbf{J}\mathbf{G}^T, \tag{4}$$

where $\mathbf{F} = \mathbf{Z}_1 \oplus \ldots \oplus \mathbf{Z}_L$, and $\mathbf{Z}_k \in \mathrm{R}^{n_h \times n_h}$, $k = 1, \ldots, L$, is the *down shift* matrix,

$$\mathbf{Z}_k = \begin{bmatrix} 0 & & & \\ 1 & 0 & & \\ & \ddots & \ddots & \\ & & 1 & 0 \end{bmatrix}.$$

The rank of ∇_F is $2L$. Since ∇_F has a considerably lower rank than $\mathbf{C}^T\mathbf{C}$, the matrix $\mathbf{C}^T\mathbf{C}$ is said to have *displacement structure* with respect to the displacement matrix $\mathbf{F}$. Matrix $\mathbf{G} \in \mathrm{R}^{n_h L \times 2L}$ is called the *generator*, and $\mathbf{J} \in \mathrm{R}^{2L \times 2L}$ is a *signature* matrix.

The Generalized Schur Algorithm (GSA) is a *fast* process that exploits the *displacement structure* of a matrix in order to perform a given triangular decomposition. In this paper, we use the GSA to obtain the Cholesky factor of the product $\mathbf{C}^T\mathbf{C}$. The sequential algorithm to solve (1) can be divided in the three main steps shown in Algorithm 3.1.

Algorithm 3.1 *Given matrix* $\mathbf{C}$ *and vector* $\mathbf{a}$, *this algorithm returns a vector* $\mathbf{h}$ *that minimizes (1)*

1. *Compute the generator* $\mathbf{G}$ *of (4)*
2. *Compute the triangular factor* $\mathbf{L}$ *(3) with the GSA*
3. *Solve the seminormal equations (3)*

At the first step, the generator matrix $\mathbf{G}$ have to be computed. This can be accomplished as follows. Firstly, we define a permutation matrix $\mathbf{P}$ such that

$$\mathbf{C}\mathbf{P} = \hat{\mathbf{C}} = \begin{bmatrix} \hat{\mathbf{c}}_1 & \hat{\mathbf{c}}_2 & \ldots & \hat{\mathbf{c}}_{n_h L} \end{bmatrix},$$

where column $\hat{\mathbf{c}}_j$, $j = 1, \ldots, n_h L$, is the ith column of $\mathbf{C}$ with the following connection between indexes j and i,

$$i = [(j-1) \bmod L]n_h + (j-1) \operatorname{div} L + 1.$$

Then, given matrix $\mathbf{H} = \mathbf{C}^T\mathbf{C}\mathbf{P}_{1:L}$, where $\mathbf{P}_{1:L}$ is the matrix formed by the first L columns of $\mathbf{P}$, a matrix $\hat{\mathbf{H}}$ is computed as

$$\hat{\mathbf{H}} = \mathbf{P}^T\mathbf{H}\mathbf{R}^{-1} = \begin{bmatrix} \mathbf{X} \\ \mathbf{Y} \end{bmatrix} \tag{5}$$

Matrices $\mathbf{X} \in \mathrm{R}^{L \times L}$ and $\mathbf{Y} \in \mathrm{R}^{n_h(L-1) \times L}$ define a partition of $\hat{\mathbf{H}}$, where matrix $\mathbf{X}$ is lower triangular. The matrix $\mathbf{R}$ is the triangular factor of the QR decomposition of the matrix $\mathbf{U} \in \mathrm{R}^{n_c M \times L}$, which is composed by the first n_c coefficients of the first column of each one of the $M \times L$ blocks $(\mathbf{C}_{ij})$ of matrix $\mathbf{C}$ (1),

$$\mathbf{U} = \begin{bmatrix} c_{11}(0) & c_{12}(0) & \ldots & c_{1L}(0) \\ c_{11}(1) & c_{12}(1) & \ldots & c_{1L}(1) \\ \vdots & \vdots & & \vdots \\ c_{11}(n_c-1) & c_{12}(n_c-1) & \ldots & c_{1L}(n_c-1) \\ c_{21}(0) & c_{22}(0) & \ldots & c_{2L}(0) \\ c_{21}(1) & c_{22}(1) & \ldots & c_{2L}(1) \\ \vdots & \vdots & & \vdots \\ c_{21}(n_c-1) & c_{22}(n_c-1) & \ldots & c_{2L}(n_c-1) \\ \vdots & \vdots & & \vdots \\ c_{M1}(0) & c_{M2}(0) & \ldots & c_{ML}(0) \\ c_{M1}(1) & c_{M2}(1) & \ldots & c_{ML}(1) \\ \vdots & \vdots & & \vdots \\ c_{M1}(n_c-1) & c_{M2}(n_c-1) & \ldots & c_{ML}(n_c-1) \end{bmatrix}$$

Then, the generator $\mathbf{G}$ and signature matrix $\mathbf{J}$ have the following form

$$\mathbf{G} = \mathbf{P}\begin{bmatrix} \hat{\mathbf{H}} & \begin{bmatrix} 0 \\ \mathbf{Y} \end{bmatrix} \end{bmatrix},$$

and

$$\mathbf{J} = \begin{bmatrix} I_L & 0 \\ 0 & -I_L \end{bmatrix},$$

where I_L is the identity matrix of order L.

At the step 2 of Algorithm 3.1, the GSA is used to obtain the lower triangular factor $\mathbf{L}$. The GSA is described in Algorithm 3.2. The step 3 of Algorithm 3.1 involves a block–Toeplitz matrix–vector product and the solution of two triangular systems that can be carried out by using the standard routines of the BLAS library.

Algorithm 3.2 (Generalized Schur Algorithm (GSA)) *Given the generator* $\mathbf{G}$ *of the displacement equation (4) and the displacement matrix* $\mathbf{F}$, *this algorithm returns the lower triangular factor* $\mathbf{L}$ *such that* $\mathbf{C}^T\mathbf{C} = \mathbf{L}\mathbf{L}^T$.

for $i = 1, \ldots, n_h L$

Let be $\vec{g}_i$ *the* i*th row of* $\mathbf{G}$, g *the first column of* $\mathbf{G}$ *and* l_i *the* i*th column of* $\mathbf{L}$

1. *Compute a transformation* Θ_i *so that*
$$\vec{g}_i\Theta_i = \begin{bmatrix} g_{i,1} & g_{i,2} & \ldots & g_{i,2L} \end{bmatrix}\Theta_i = \begin{bmatrix} g'_{i,1} & 0 & \ldots & 0 \end{bmatrix}.$$
2. *Update generator,* $\mathbf{G} \leftarrow \mathbf{G}\Theta_i$.
3. *Update factor* $\mathbf{L}$, $l_i \leftarrow g$.
4. *Shift the first column of* $\mathbf{G}$, $\mathbf{G} \leftarrow \begin{bmatrix} \mathbf{F}g & \bar{\mathbf{G}} \end{bmatrix}$, *where* $\bar{\mathbf{G}}$ *denotes the last* $2L-1$ *columns of* $\mathbf{G}$

Transformation Θ_i, computed at the first step of Algorithm 3.2, is a $\mathbf{J}$–unitary transformation, that is, $\Theta_i^T\mathbf{J}\Theta_i = \mathbf{J}$. We have used the method proposed in [10] for computing and applying Θ_i in steps 2 and 3 respectively of Algorithm 3.2. In [10, 11] can be found a proof of the above algorithm.

4 The Parallel Algorithm

For the parallel algorithm, we have used the ScaLAPACK model to distribute and manage data on the different processors. The software tools included in ScaLAPACK library let us to map the concurrent processes in a two dimensional logical grid of $p \times q$ processors where each processor is denoted by $P_{i,j}$. In our case, the choice of a one dimensional grid with a "column" of p processors ($q = 1$) is the most adequate to solve the problem. Then, processors in the logical column will simply be denoted by P_k, $k = 0, \ldots, p-1$.

The generator $\mathbf{G}$ is partitioned in L blocks,

$$\mathbf{G} = \begin{bmatrix} \mathbf{G}_1 \\ \mathbf{G}_2 \\ \vdots \\ \mathbf{G}_L \end{bmatrix}, \qquad (6)$$

where $\mathbf{G}_k \in \mathrm{R}^{n_h \times 2L}$. These blocks are cyclically distributed over the "logical column" of p processors so that block $\mathbf{G}_k$ belongs to processor $P_{(k-1) \bmod p}$.

Let be the following partition of factor $\mathbf{L}$ in $L \times L$ square blocks of size $n_h \times n_h$,

$$\mathbf{L} = \begin{bmatrix} \mathbf{L}_{1,1} & & & \\ \mathbf{L}_{2,1} & \mathbf{L}_{2,2} & & \\ \vdots & \vdots & \ddots & \\ \mathbf{L}_{L,1} & \mathbf{L}_{L,2} & \cdots & \mathbf{L}_{L,L} \end{bmatrix}, \qquad (7)$$

where $\mathbf{L}_{i,i}$, $i = 1, \ldots, L$, are lower triangular. The parallel algorithm computes factor $\mathbf{L}$ distributed in such a way so each block $\mathbf{L}_{i,j=1,\ldots,i}$ belongs to processor $P_{(i-1) \bmod p}$.

The parallel algorithm can be divided into the same three steps as the sequential Algorithm 3.1. At the first step of the parallel algorithm, the generator is computed. Computing the generator $\mathbf{G}$ is a concurrent process performed without communication cost. This procedure involves, as a basic computational kernels, L matrix–vector products, a QR factorization and the solution of a triangular system. The part of the result of the matrix–vector products belonging to a processor is computed only by that processor. The computation of the factor $\mathbf{R}$ (5) is performed by calling the appropriate LAPACK subroutine and it is repeated by all processors. This choice increases the overhead with respect to the sequential algorithm, but this overhead is smaller than the overhead caused by the broadcast of $\mathbf{R}$ once it has been computed by one processor. Afterwards, each processor solves the triangular system $\mathbf{H}\mathbf{R}^{-1}$ in order to obtain factor $\hat{\mathbf{H}}$, and so, the generator $\mathbf{G}$ distributed as it have been shown in (6).

In the second step of the parallel algorithm, the triangular factorization of $\mathbf{C}^T\mathbf{C}$ is performed by means of the GSA. Algorithm 4.1 describes the Parallel GSA.

Algorithm 4.1 (Parallel Generalized Schur Algorithm) *Given the generator $\mathbf{G}$ of the displacement equation (4) distributed as it is described in (6), this algorithm returns the lower triangular factor $\mathbf{L}$ distributed as it is described in (7), so that $\mathbf{C}^T\mathbf{C} = \mathbf{L}\mathbf{L}^T$. Each processor P_k, $k = 0, \ldots, p-1$, performs*

for $i = 1, \ldots, n_h L$
  Let be $\vec{g}_i$ *the* i*th row of* $\mathbf{G}$
  if $\vec{g}_i \in P_k$
    1. Compute a transformation Θ_i *so that*
    $\vec{g}_i\Theta_i = \begin{bmatrix} g_{i,1} & g_{i,2} & \cdots & g_{i,2L} \end{bmatrix}\Theta_i = \begin{bmatrix} g'_{i,1} & 0 & \cdots & 0 \end{bmatrix}$
    2. $l_{i,i} \leftarrow g'_{i,1}$
    3 Broadcast Θ_i *to the rest of processors.*
  else
    1. Receive Θ_i
  end if
  for $j = i+1, \ldots, n_h L$
    Let be $\vec{g}_j$ *the* j*th row of* $\mathbf{G}$
    if $\vec{g}_j \in P_k$
      1 $\vec{g}_j \leftarrow \vec{g}_j\Theta_i$
      2 $l_{j,i} \leftarrow g'_{j,1}$
      if $\vec{g}_{j-1} \in P_k$
        $g_{j,1} \leftarrow l_{j-1,i}$
      else
        $g_{j,1} \leftarrow 0$
      end if
    end if
  end for
end for

In Figure 2 we can see two adjacent blocks of the generator on processors P_k and P_{k+1} respectively. The first row of the second block is represented by j ($j = ((i + n_h - 1)/n_h)n_h + 1$).

	G			
	...	...	...	...
P_k	$g_{i,1}$	$g_{i,2}$	...	$g_{i,2L}$
	$g_{i+1,1}$	$g_{i+1,2}$	...	$g_{i+1,2L}$
	$g_{i+2,1}$	$g_{i+2,2}$	...	$g_{i+2,2L}$
	...	...	...	...
P_{k+1}	$g_{j,1}$	$g_{j,2}$	...	$g_{j,2L}$
	$g_{j+1,1}$	$g_{j+1,2}$	...	$g_{j+1,2L}$
	$g_{j+2,1}$	$g_{j+2,2}$	...	$g_{j+2,2L}$
	...	...	...	...

Figure 2. Two adjacent blocks of $\mathbf{G}$ in processors P_k and P_{k+1} respectively before iteration i.

At the ith iteration, processor having the ith row computes Θ_i and broadcasts it to the rest of processors. Figure 3 represents the generator $\mathbf{G}$ and the ith column of $\mathbf{L}$ ($l_{:,i}$) after the inner loop of the ith iteration of Algorithm 4.1 have been executed.

In Figure 3 symbol $'$ represents modified entries by the product $\mathbf{G}\Theta_i$. At each step of the inner loop of Algo-

	G				$l_{:,i}$
	...	...	...	...	0
	0	0		0	$g'_{i,1}$
P_k	$g'_{i,1}$	$g'_{i+1,2}$	...	$g'_{i+1,2L}$	$g'_{i+1,1}$
	$g'_{i+1,1}$	$g'_{i+2,2}$	...	$g'_{i+2,2L}$	$g'_{i+2,1}$
	...	...	...	...	...
	0	$g'_{j,2}$	...	$g'_{j,2L}$	$g'_{j,1}$
P_{k+1}	$g'_{j,1}$	$g'_{j+1,2}$		$g'_{j+1,2L}$	$g'_{j+1,1}$
	$g'_{j+1,1}$	$g'_{j+2,2}$	...	$g'_{j+2,2L}$	$g'_{j+2,1}$
	...	...	...	...	...

Figure 3. This figure represents Figure 2 with the ith column of $\mathbf{L}$ after the iteration ith have been carried out.

rithm 4.1, the transformation Θ_i is applied to each row $> i$, the first entry of such row is an entry of the ith column of $\mathbf{L}$, and such first entry is shifted one position down within the block. Due to the form of the displacement matrix $\mathbf{F}$ and the distribution used for $\mathbf{G}$, no data is need to send to the adjacent processor in the shift process.

It is important to note that, in our algorithm, the only communication cost is composed by $n_h L$ broadcasts. This fact makes more efficient our algorithm than other ones. For example, the Block Schur Algorithm presented in [12] performs a lot of point–to–point communications caused by the displacement process. Furthermore, our method for computing the J–unitary transformations has better numerical properties [13] than the method used in [12].

We have used another technique to reduce even more the communication cost. In Algorithm 4.1, a group of ν J-unitary transformations are computed and packed in each message. Thus, the amount of broadcasts is reduced by the blocking factor ν. This blocking factor is machine dependent. With this modification of the basic process, we minimize the effect of the time spent in sending the messages, maximize the overlapping between computations and communications and achieve better performance within each processor.

In the implementation of Algorithm 4.1, we actually work with $\mathbf{G}^T$ distributed over a "logical row" of processors. Because of the arrays are stored by columns, the algorithm access to the entries of $\mathbf{G}$ stored in adjacent positions of the local memory. However, $\mathbf{L}$ is distributed on a column of processors and its entries are stored with a pattern access of 1 too. We get a great reduction in the execution time with this modification. This fact implies the utilization of two different processor grids: a logical row and a logical column of processors. The usage of ScaLAPACK *contexts* lets to perform this switch between logical grids during the execution time easily.

At the third step of the overall algorithm, the solution of the seminormal equations (3) is obtained in parallel by calling some parallel routines provided by the PBLAS library.

5 Experimental Results

The target platform used for running the experiments is a cluster of personal computers with 12 nodes (IBM xSeries 330 SMP), running under the Linux operating system. Each personal computer has two 866MHz Intel Pentium III with 512 MBytes of memory. The interconnection network is a Gigabit Ethernet. The latency time of the network is 122μsec, and 0.030μsec is the time needed to transmit each double precision scalar of a message. The time for a flop, obtained with an optimized version of the DGEMM routine in BLAS for the product of two general square matrices, is around 1.55×10^{-3} msec.

Figure 4 shows the great reduction achieved in the execution time with the increase of the number of processors for an acoustic system composed by 8 microphones and 8 loudspeakers. In that figure, we have used a $n_c = 300$ coefficients sample for the channels. This is a good result taking into account that with the rise of the number of coefficients of the inverse filters (n_h) we can expect a lower squared error of (2).

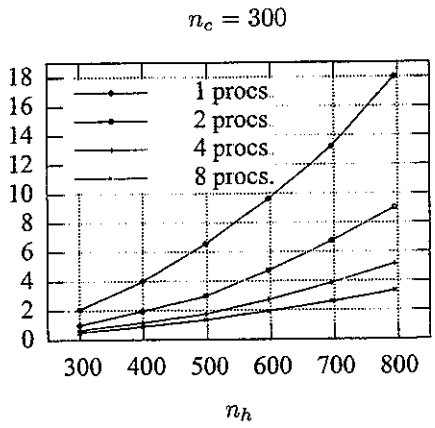

Figure 4. Time in seconds varying the number of coefficients n_h of each inverse filter for $M = 8$ microphones and $L = 8$ loudspeakers with a sample size of $n_c = 300$.

We have obtained similar results with different sizes of the sample (Table 1). For a fixed size of the parameter n_h and varying parameter n_c, the efficiency of the parallel algorithm also increases with the number of processors, but not as much efficiency as we can see in Figure 4. This is because only the computation of the generator and the last matrix-vector product of the parallel algorithm depends on the sample length. Besides, the triangularization process is by far the most costly step, and it does not depend on the number of coefficients of the sample (n_c).

In Figure 5, we show the execution time of the parallel algorithm in a node of the cluster, that is, in a bi–processor board using shared memory. We take advantage of the availability of a shared memory version of the MPI package for running our parallel algorithm without any implementation change. Specifically, we have solved a *cross talk*

Table 1. Efficiency varying the size of n_c for $n_h = 800$

n_c	2 procs.	4 procs.	8 procs.
300	99.9%	87.3%	68.0%
500	97.3%	86.0%	66.8%
700	97.3%	84.3%	66.8%
900	97.3%	86.4%	67.0%

cancellation problem. This problem involves two microphones and two loudspeakers in the acoustic system. The goal of the inverse filters designed for this problem is to cancel the effect of the output signal of the left loudspeaker into the right microphone and the output signal of the right loudspeaker into the left microphone. Matrices used in the test have been built sampling experimental signals obtained in a listening room.

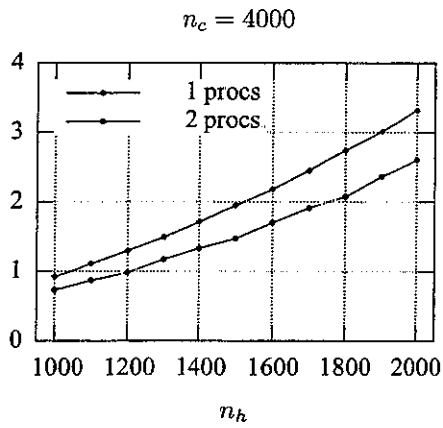

Figure 5. Time in seconds of the parallel algorithm running on a bi–processor board varying the number of coefficients n_h of each inverse filter for the *cross talk* cancellation problem, with a sample length of $n_c = 4000$.

It can be see in Figure 5 that the execution time is reduced using the two processors of the bi–processor board for only two loudspeakers and two microphones. The efficiency obtained with two processors is around 65%.

The relative error of the algorithm for a vector $\mathbf{h}$ in (2) with all its components equal to one has also been measured. The resulting relative errors are around 10^{-14}.

6 Concluding Remarks

The parallel algorithm designed is useful in multiple applications which can be formulated as a multichannel deconvolution problem. The parallel algorithm is efficient and portable, and it computes an accurate solution if it exists for the minimization problem formulated in (2). The main limitation is due to the fact that the maximum number of processors that it can be used is restricted by the number of sources of the multichannel system. However, it can be used with good results on bi– and tetra–processors. Therefore, it can be useful for the upcoming applications like leisure, video–conference or the new generation of digital wireless communications.

References

[1] C. Kyriakakis, P. Tsakalides and T. Holman, Surrounded by sound, *IEEE Signal Processing Magazine*, 16(1), 1999, 55-66.

[2] H. Irisawa, S. Shimada, H. Hokari and S. Hosoya, Study of a fast method to calculate inverse filters, *J. Audio Eng. Soc.*, 46(7/8), 1998, 611-619.

[3] J. J. López and A. González, Two steps Levinson algorithm for time domain multichannel deconvolution, *Electronic Letters*, 36(7), 2000, 686–688.

[4] J. J. López, A. González and F. Orduña–Bustamante, Measurement of cross-talk cancellation and equalization zones in 3-D sound reproduction under real listening conditions, *Proceedings of AES 16th International Conference on Spatial Sound Reproduction*, Rovaniemi, Finland, 1999.

[5] J. J. López, A. González and F. Orduña–Bustamante, Equalization zones for cross talk cancellation as a function of loudspeaker position and room acoustics, *Proceedings of ACTIVE 99*, Ford Lauderdale, Florida, 1999.

[6] Åke Björck, *Numerical Methods for Least Squares Problems* (North-Holland, 1996).

[7] E. Anderson et al., *LAPACK Users' Guide* (SIAM, Philadelphia, 1995).

[8] L. S. Blackford et al. *ScaLAPACK Users' Guide* (SIAM, Philadelphia, 1997).

[9] P. A. Nelson, F. Orduña–Bustamante and H. Hamada, Multichannel signal processing techniques in the reproduction of sound, *J. Audio Eng. Soc.*, 44(11), 1996, 973–989.

[10] S. Chandrasekaran and Ali H. Sayed. A Fast Stable Solver for Nonsymmetric Toeplitz and Quasi-Toeplitz Systems of Linear Equations. *SIAM Journal on Matrix Analysis and Applications*, 19(1), 1998, 107–139.

[11] T. Kailath and A. H. Sayed, editors. *Fast Reliable Algorithms for Matrices with Structure* (SIAM, Philadelphia, PA, 1999).

[12] K. Gallivan, S. Thirumalai and P. Van Dooren, On solving block toeplitz systems using a block schur algorithm, *Proceedings of the 23rd International Conference on Parallel Processing*, Boca Raton, FL, USA, 1994, Volume 3: Algorithms and Applications, 274–281.

[13] A. W. Bojanczyk, R. P. Brent, F. R. de Hoog and D. R. Sweet, On the stability of the Bareiss and related Toeplitz factorization algorithms. *SIAM Journal on Matrix Analysis and Applications*, 16(1), 1995, 40–57.