PARALLEL IMPLEMENTATIONS OF BEAMFORMING DESIGN AND FILTERING FOR MICROPHONE ARRAY APPLICATIONS

Jorge Lorente[1], *Gema Piñero*[1], *Antonio M. Vidal*[2], *Jose Antonio Belloch*[1], *Alberto Gonzalez*[1]

[1] Institute of Telecommunications and Multimedia Applications (iTEAM)
[2] Interdisciplinary Group of Computer and Communications (INCO2)
Universitat Politècnica de València
{jorlogi, gpinyero}@iteam.upv.es

ABSTRACT

One of the main limitations of microphone array algorithms for audio applications has been their high computational cost in real acoustic environments when real-time signal processing is absolutely required. Regarding audio/speech signal processing, beamforming algorithms have been used for the recovery of acoustic signals from their observations when they are corrupted by noise, reverberation and other interfering signals. In order to reduce their high computational load, frequency-based filtering have been used to achieve a real time application. Our research focuses on the use of different multicore/manycore platforms in order to achieve a real time beamforming application in the time domain. Efficient algorithms has been proposed and tested in several devices and results have shown that GPU implementation of beamforming design and filtering outperforms multicore implementation in computational cost terms. The performance obtained suggests that GPU implementation paves the way for low-cost real-time audio beamforming applications.

1. INTRODUCTION

Since digital signal processors and other devices have substantially increased their computational performance, intensive and complex problems have been addressed and solved in shorter time. This has benefited both the typical real-time applications that are common in signal processing as any other signal processing applications that imply the management of very large data sets.

More recently, specialized (many-core) hardware with hundreds of simple cores are available in the form of cheap, widely-spread NVIDIA and AMD/ATI Graphic Processor Units (GPU) incorporated in any standard graphics card. For example, 512 cores are embedded in the NVIDIA Fermi architecture.

Although programming in many core architectures is not trivial [1], there are many tools to help software developers to adapt their programs to the new architectures [2]. We can cite, for example, the following GPU libraries: CUBLAS and CUFFT: implementations on CUDA of the well-known BLAS computational kernels and FFT algorithms [3], and CULA: implementation of the LAPACK library for GPU [4]. Otherway, there are also CPU libraries that take advantage of parallelization over different CPU cores like Intel MKL library [5].

Some signal processing applications are already taking advantage of these opportunities [6, 7, 8, 9]. Regarding audio/speech signal processing, in [10, 11] the use of GPU is proposed to speech recognition applications achieving speedups up to 5x and 9x respectively; similarly GPU implementation is also proposed for adaptive filtering of Acoustic Echo Canceller in [12]. Moreover, in [13] about 11x speedup is achieved when GPU is used for filtering with beamforming-design filters both in time and frequency. Finally, related to immersive audio technologies, [14] presents multi-core Platforms for Beamforming and Wave Field Synthesis with different speedup performances.

In this paper we discuss the GPU's possibilities to implement a whole beamforming algorithm application (beamforming filter design and beamforming filtering) in real-time. We compare multicore and manycore implementations of beamforming filter design in order to determine that GPU implementation outperforms multicore implementation in all cases. Furthermore, we use GPU implementation for beamforming filter design with a CUDA implementation of the filtering to show that a real-time implementation of a whole beamforming algorithm can be entirely run on a GPU freeing up resources from the CPU.

This paper is organized as follows: section 2 describes the signal model used in the microphone array application. Section 3 introduces an efficient version of the optimum beamforming, the QR-LCMV, and compare it with GSC algorithm. Section 4 explains the implementations of the beamforming filter design and the filtering on the different multicore/manycore platforms. Finally, sections 5 and 6 are devoted to show test results and conclusions respectively.

2. SIGNAL MODEL

Consider the system of Figure 1 where two loudspeakers are emitting two independent signals, $s_1(k)$ and $s_2(k)$, respectively. At the other part of the room, three microphones are recording the mix of the two signals corrupted by noise and room reverberation. The problem is how to recover $s_1(k)$ or $s_2(k)$ by means of the signals recorded at the microphones. The approach taken herein makes use of signal processing algorithms to design the broadband beamformers (filters g_n in Figure 1), once all the room channel responses (h_{nm} in Figure 1) are known. This system can be modeled as a multichannel system with 2 inputs (loudspeakers) and 3 outputs (microphones), and the generalization to a Multiple Input Multiple Output (MIMO) system can be easily addressed [15].

According to Figure 1, the output of the n-th microphone is given by:

$$x_n(k) = \sum_{m=1}^{M} \sum_{j=1}^{L_h} h_{nm}(j) s_m(k-j) + v_n(k), \qquad (1)$$

where $n = 1, 2, \ldots, N$, being N the number of microphones and M the number of source signals, that is equal to the number of loudspeakers in Figure 1. L_h is the length of the

This work has been supported by Spanish Ministry of Science and Innovation through grant TEC2009-13741, Regional Government Generalitat Valenciana through grant PROMETEO/2009/013 and NVIDIA through CUDA Community program.

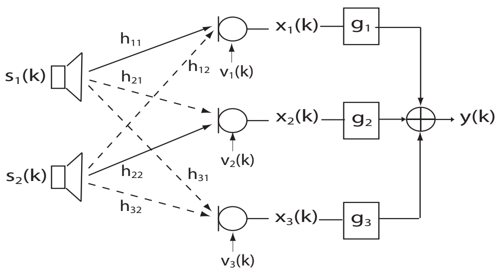

Figure 1: Signal model for $M = 2$ loudspeakers (inputs) and $N = 3$ microphones (outputs).

longest room impulse response of all the acoustic channels h_{nm} and $v_n(k)$ is the noise signal. For the sake of clarity the noise term $v_n(k)$ of (1) will not be considered in the following signal model. In order to improve computation efficiency, equation (1) can be rewritten in vector/matrix form as:

$$x_n(k) = \sum_{m=1}^{M} \mathbf{h}_{nm}^T \mathbf{s}_m(k), \tag{2}$$

where $\mathbf{s}_m(k)$ is the column vector defined as $\mathbf{s}_m(k) = [s_m(k) \quad s_m(k-1) \quad \cdots \quad s_m(k-L_h+1)]^T$, $\mathbf{h}_{nm}$ is the $\mathcal{R}^{L_h \times 1}$ acoustic channel vector from loudspeaker m to microphone n and $()^T$ denotes the transpose of a vector or a matrix.

Considering now the problem of recovering source signals $s_m(k)$ from the recorded observations $x_n(k)$, beamforming filters g_n of Figure 1 have to be designed in such a way that the output signal $y(k)$ is a good estimate of $s_m(k)$, that is, $y(k) = \hat{s}_m(k - \tau)$ with minimum error. Given a maximum length of L_g taps for each of the g_n filters, the broadband beamforming output signal is expressed in a similar form as in (2):

$$y(k) = \sum_{n=1}^{N} \mathbf{g}_n^T \mathbf{x}_n(k), \tag{3}$$

where $\mathbf{g}_n$ is the $\mathcal{R}^{L_g \times 1}$ vector containing the ordered taps of beamforming filters g_n of Figure 2, and $\mathbf{x}_n(k)$ is the column vector defined as $\mathbf{x}_n(k) = [x_n(k) \quad x_n(k-1) \quad \cdots \quad x_n(k - L_g + 1)]^T$.

In order to compute the whole vector $\mathbf{x}_n(k)$ used in (3) in matrix form, equation (2) has to be rewritten in compact form redefining $\mathbf{h}_{nm}$ as Sylvester matrices. See [15] for further details.

3. BEAMFORMING ALGORITHMS

In [15], Benesty et al. present an excellent state-of-the-art of the main algorithms used in audio applications. Due to its better performance, we have focused our study on matrix correlation based algorithms, such as LCMV (Linearly Constrained Minimum Variance) and its unconstrained implementation GSC (Generalized Sidelobe Canceller).

3.1 QR-LCMV Beamforming Algorithm

LCMV algorithm calculates beamforming filters as:

$$\mathbf{g}^{\mathrm{LCMV}} = \hat{\mathbf{R}}_x^{-1} \mathbf{H}_{:m} [\mathbf{H}_{:m}^T \hat{\mathbf{R}}_x^{-1} \mathbf{H}_{:m}]^{-1} \mathbf{u}_m, \tag{4}$$

where $\mathbf{g}^{\mathrm{LCMV}}$ is formed by the concatenation of filters $\mathbf{g}_n$, that is, $\mathbf{g}^{\mathrm{LCMV}} = [\mathbf{g}_1^T \quad \cdots \quad \mathbf{g}_N^T]^T$, matrix $\mathbf{H}_{:m}$ is a partition of the channel impulse matrix that only includes the impulse responses from m-th source to the N microphones [15] used in sylvester matrix form and has dimensions $[NL_g \times Lg + Lh - 1]$. Matrix $\hat{\mathbf{R}}_x$ is the correlation matrix of the recorded signals and $\mathbf{u}_m$ is a vector of zeros except for a one at the proper vector component in order to compensate the room impulse response delay.

Seeking the most efficient LCMV implementation, a method based on QR decomposition of matrix $\mathbf{X}^T$ is presented, being $\mathbf{X} \in \Re^{[NL_g \times K]}$ defined as:

$$\mathbf{X} = \frac{1}{\sqrt{K}} \begin{pmatrix} \mathbf{x}_1(k) & \mathbf{x}_1(k+1) & \cdots & \mathbf{x}_1(k+K-1) \\ \mathbf{x}_2(k) & \mathbf{x}_2(k+1) & \cdots & \mathbf{x}_2(k+K-1) \\ \vdots & \vdots & \cdots & \vdots \\ \mathbf{x}_N(k) & \mathbf{x}_N(k+1) & \cdots & \mathbf{x}_N(k+K-1) \end{pmatrix} \tag{5}$$

where $K > NL_g$ is the number of samples used.

This way, $\mathbf{X}^T = \mathbf{Q} \cdot \mathbf{L}$, where $\mathbf{Q}$ is an orthogonal matrix and $\mathbf{L}$ is an upper triangular matrix which allows a faster resolution in linear algebraic systems. Note that [16] presents a similar method based on a QR decomposition of the microphone observations matrix applied to the MMSE filter design.

Considering QR decomposition of microphone observations, we can redefine $\hat{\mathbf{R}}_x$ as:

$$\hat{\mathbf{R}}_x = \mathbf{X} \cdot \mathbf{X}^T = \mathbf{L}^T \cdot \mathbf{Q}^T \cdot \mathbf{Q} \cdot \mathbf{L} = \mathbf{L}^T \cdot \mathbf{L}. \tag{6}$$

Now, let us denote matrix $\mathbf{W} = \hat{\mathbf{R}}_x^{-1} \mathbf{H}_{:m}$ such that LCMV beamformer filter $\mathbf{g}_m^{\mathrm{LCMV}}$ (4) is expressed as:

$$\mathbf{g}_m^{\mathrm{LCMV}} = \mathbf{W}[\mathbf{H}_{:m}^T \mathbf{W}]^{-1} \mathbf{u}_m. \tag{7}$$

It can be shown that:

$$\mathbf{W} = \hat{\mathbf{R}}_x^{-1} \mathbf{H}_{:m} = (\mathbf{L}^T \mathbf{L})^{-1} \mathbf{H}_{:m} = \mathbf{L}^{-1} \mathbf{Z}, \tag{8}$$

where:

$$\mathbf{Z} = \mathbf{L}^{T^{-1}} \mathbf{H}_{:m}. \tag{9}$$

Let us define now matrix $\mathbf{A}$ as (7)

$$\mathbf{A} = \mathbf{H}_{:m}^T \mathbf{W} = \mathbf{H}_{:m}^T \mathbf{L}^{-1} \mathbf{Z} = \mathbf{Z}^T \mathbf{Z}. \tag{10}$$

Finally defining vector $\mathbf{b}_m = \mathbf{A}^{-1} \mathbf{u}_m$ we can express the LCMV beamformer filter $\mathbf{g}_m^{\mathrm{LCMV}}$ as follows:

$$\mathbf{g}_m^{\mathrm{LCMV}} = \mathbf{L}^{-1} \mathbf{Z} \cdot \mathbf{b}_m. \tag{11}$$

Using this method the main operations involve only matrix $\mathbf{Z}$ and vector $\mathbf{u}_m$. Once $\mathbf{Z}$ is computed, matrix $\mathbf{A}$ and vector $\mathbf{u}_m$ can be calculated and used to get beamforming filters $\mathbf{g}_m^{\mathrm{LCMV}}$. Moreover, calculation of (9) and (11) involve the solution of linear equations where the inverse of a matrix doesn't need to be computed if efficient left matrix division is used. Therefore, (4) can be computed avoiding costly matrix inversions through a QR decomposition and two left matrix divisions

3.2 GSC Algorithm

The GSC and LCMV beamformers are essentially the same, although GSC transforms the LCMV algorithm from a constrained problem to an unconstrained one dividing the filter vector $\mathbf{g}$ into two components operating on orthogonal subspaces.

In order to avoid the malfunctioning of the GSC, we assume that $L_g > (L_h - 1)(N - 1)$ so that the nullspace of $\mathbf{H}_{:m}^T$ can't be zero. Then, the GSC method can be formulated as:

$$\mathbf{g}_m = \mathbf{f}_m - \mathbf{B}_m \mathbf{w}_m, \tag{12}$$

	LCMV	**GSC filters adaptation**	**GSC room actualization**
M-V Multiplication	OL_3^2	$O(L_1 L_{null}) \simeq OL_1$	$O(L_1 L_3) \simeq OL_1$
M-M Multiplication	$O(L_3^2 L_1) \simeq OL_3^3$	$O(L_1^2 L_2 + L_2^2 L_{null} + L_{null}^2 L_1) \simeq O(L_1^2 L_2)$	$O(L_3^2 L_2)$
M-V Linear Equation System	$O(\frac{1}{3} L_3^2 + 2L_3^2 + L_1^2)$	$O(\frac{1}{3} L_1^2 L_{null} + L_1 L_{null}) \simeq O(\frac{1}{3} L_3^3 + L_1)$	$O(\frac{1}{3} L_3^3 + 2L_3^2)$
M-M Linear Equation System	triangular matrix, so: OL_1^2	0	0
QR decomposition	$O\frac{2}{3} L_1^2 (3L_1 - L_2)$	0	$O\frac{2}{3} L_1^2 (3L_1 - L_3)$
Generate matrix Q	0	0	$O\frac{2}{3} L_1^2 (3L_1 - L_3)$
Floating-point operations	$7.3155 * 10^9$	$4.2195 * 10^9$	$1.4346 * 10^{10}$

Table 1: Floating-point operations of LCMV and GSC algorithms.

where

$$\mathbf{f}_m = \mathbf{H}_{:m}[\mathbf{H}_{:m}^T \mathbf{H}_{:m}]^{-1} \mathbf{u}_m \tag{13}$$

is the minimum norm solution of $\mathbf{H}_{:m}^T \mathbf{f}_m = \mathbf{u}_m$ and $\mathbf{B}_m$ is the blocking matrix that spans the nullspace of $\mathbf{H}_{:m}^T$ so that $\mathbf{H}_{:m}^T \mathbf{B}_m = \mathbf{0}$.

Vector $\mathbf{w}_m$ is obtained from the following unconstrained optimization problem:

$$\min_{\mathbf{w}_m} (\mathbf{f}_m - \mathbf{B}_m \mathbf{w}_m)^T \mathbf{R}_x (\mathbf{f}_m - \mathbf{B}_m \mathbf{w}_m) \tag{14}$$

and its solution is given as:

$$\mathbf{w}_m = [\mathbf{B}_m^T \mathbf{R}_x \mathbf{B}_m]^{-1} \mathbf{B}_m^T \mathbf{R}_x \mathbf{f}_m. \tag{15}$$

3.3 Computational comparison between QR-LCMV and GSC

QR-LCMV and GSC have to update the block of microphone observations $\mathbf{X}$, in order to work in real-time. For the purpose of comparing computational cost of boths algorithms an analysis of the floating-point operations of each block is necessary.

The computational cost is analyzed in reference to the matrix size of $\mathbf{X}^{[L_1 \times L_2]}$, $\mathbf{H}_{:m}^{[L_1 \times L3]}$ and $\mathbf{B}^{[L_1 \times L_{null}]}$ where $L_1 = NL_g$, $L_2 > NL_g + 1$, $L_3 = L_h + L_g - 1$ and $L_{null} = L_1 - L_3$. In order to avoid ill conditioned correlation matrix, we have take $L_2 = \frac{5}{4} NL_g$. Considering that L_h is an invariant value fixed by the dimensions of the room, N is an invariant value fixed by the dimensions of the microphone array and L_g is a variable value always bigger than $\frac{L_h}{2}$ [15], L_{null} increases as L_g increases.

As it can be noted in section 3.2, GSC algorithm divides the filter design calculation in a fixed part (calculation of $\mathbf{f}_m$ and $\mathbf{B}_m$, only dependent on room dimensions) and an adaptive part (calculation of $\mathbf{w}_m$, dependent on input samples). For this reason, GSC is expected to have less floating-point operations for invariant room responses.

The number of floating-point operations needed for each new block of microphone observations, is shown in the first two columns of Table 1. Column three of Table 1 show number of extra operations needed because of a room actualization. In Table 1, M-V and M-M refers to matrix-vector and matrix matrix operations respectively. Note that for our case with $N = 3$ microphones and the minimum $L_g = \frac{L_h}{2}$ value, $L_1 = 3\frac{L_h}{2}$ and $L_3 = L_h + Lg - 1 = L_h + \frac{L_h}{2} - 1 = 3\frac{L_h}{2} - 1$, so $L_{null} = L_1 - L_3 = 1$ and all the computational cost when L_{null} is involved is neglected compared to dimensions L_1, L_2 and L_3. Last row shows an example of floating-point operations computed in our experiment with $N = 3$ microphones, $L_h = 1000$ and $L_g = \frac{L_h}{2}$.

4. PARALEL IMPLEMENTATIONS

4.1 Beamforming filter design

4.1.1 GPU Implementation

Both QR-LCMV and GSC Algorithms have been implemented on GPU (NVIDIA GeForce GTX285) using Lapack and BLAS CUDA libraries (CULA and CUBLAS respectively). The platform used to develop the algorithms is Microsoft Visual Studio 2008. In figure 2 a basic scheme of the GPU implementation of the algorithms is depicted.

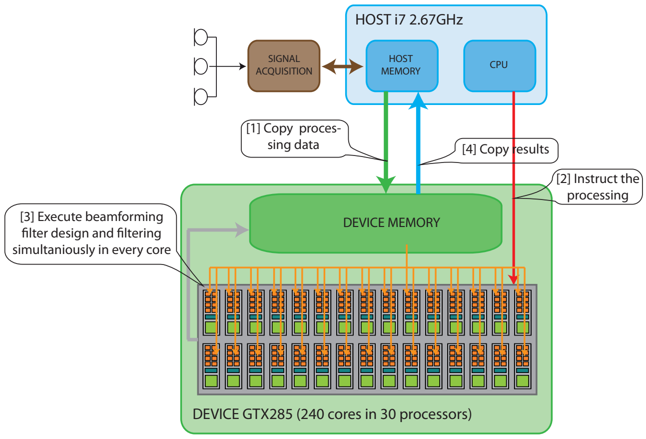

Figure 2: GPU implementation.

4.1.2 Multicore Implementation

A multicore implementation of beamforming algorithms has been also implemented on a CPU (Intel Core i7 2.67Ghz with 4Gb of RAM) using Intel MKL library which includes Lapack and BLAS threaded functions. In our implementation we have parallelized the beamforming filter design in 4 CPU cores setting MKL_NUM_THREADS=4.

4.2 Filtering

Once the filters have been calculated we can recover the desired signal filtering microphone observations with beamforming filters $\mathbf{g}_m$ (11) and (12).

Dealing with multichannel filtering takes a large computational cost and considering our goal of a total GPU implementation to free up resources from the CPU, only a GPU implementation of multichannel filtering is here considered. The algorithm we present is focused on the overlap-save technique to carry out a multichannel convolution.

The filtering implementation is based on CUFFT CUDA library whose concurrent copy and execution property allows transferring data from the CPU to the GPU and vice versa at the same time that computations are performing. This allows to obtain the best performance of the algorithm and at the same time to exploit the parallelism of the CUDA architecture. A Matrix-Signal will be configured with the blocks of the signals $\mathbf{x}_n$. The filters $\mathbf{g}_m$ will be sent only once to the GPU. As long as the matrix signal is filled, this matrix will be sent to the GPU and, a new matrix will begin to be filled. In the same sense, as long as operations end at the GPU, the result matrix is sent back to the CPU. Therefore a pipeline configuration can be achieved easily. See reference

				Block processing time (ms)		Number of consecutive blocks processed for real-time	
	$\mathbf{L_g}$	$\mathbf{L_b}$	$\mathbf{L_b/f_s}$**(ms)**	**LCMV time**	**GSC time**	**LCMV**	**GSC**
$\mathbf{f_s = 44100}$	500	3375	76.5	467.4	124.8	7	2
	550	3713	84.2	548.3	182.1	7	3
	600	4050	91.8	656.2	246.1	8	4
$\mathbf{L_h = 1000}$	650	4388	99.5	750.2	326.7	8	4
	700	4725	107.1	852.5	429.1	9	5
	750	5063	114.8	968.8	543.8	9	5

Table 2: Processing time of LCMV and GSC algorithms for the different cases analyzed.

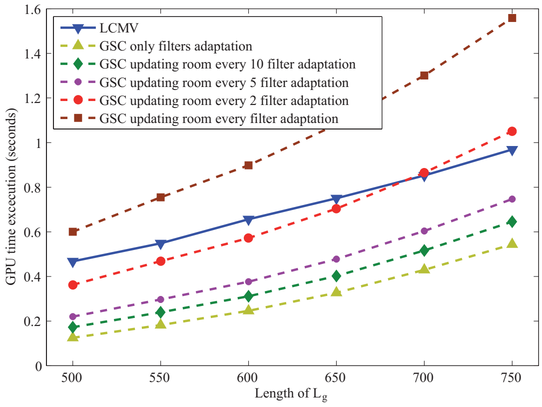

Figure 3: LCMV and GSC execution time varying the frequency of room actualization in GSC algorithm.

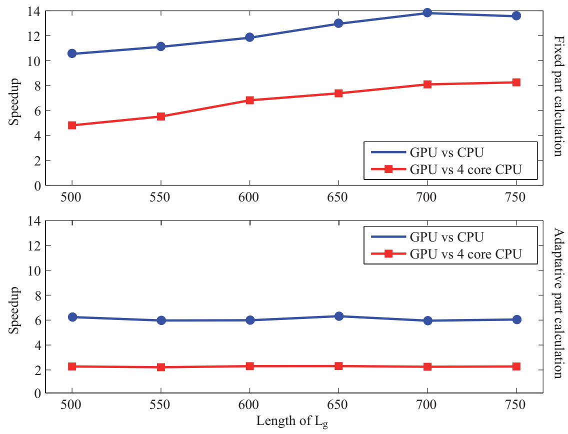

Figure 4: a) GSC execution time only for the fixed part. b) GSC execution time only for the adaptive part.

[17, 18] for further details.

5. TESTING RESULTS

As discussed in section 3.2, GSC algorithm calculation is divided in a fixed part and an adaptive part. The fixed part only needs to be calculated when a room updating is needed. In some applications like immersive-audio technologies where sources are supposed to move over the scenario, the room impulse responses need to be estimated more frequently that in those applications where the sources and microphones are fixed.

In these sense, LCMV and GSC execution time have been compared in figure 3. It is easy to see that when a frequent room updating is not necessary, GSC algorithm outperforms LCMV. For this reason in the next section, only GSC implementation is tested in different platforms.

5.1 GSC CPU-GPU comparison

In this section we compare different implementations of GSC algorithm tested in a room in which baseline impulse responses have length $L_h = 1000$ for different lengths of beamforming filters L_g.

Results are shown in figure 4, where we can see that in both GSC parts (fixed and adaptive) GPU outperform CPU implementation, reaching about 13x speedup in the best case. It is also significant that the speedup is better in fixed part than in adaptive part, and the reason is that fixed operations (for example QR decomposition) have greater computational load, and therefore GPU can take better advantage of the parallelization.

5.2 Achieving real-time processing.

Our work aims to achieve both beamforming filter design and real-time filtering, thus working by blocks of microphone observations becomes a condition to achieve a real-time audio application. However, an unavoidable but not critical initial delay, which is exactly the processing time of the first block, appears by working by blocks. Note that the processing time of a block includes the filter design and the filtering.

For each block of microphone observations, a matrix correlation needs to be estimated so that the adaptation of filters fit the microphone observations. Thus, the minimum block size (L_b) is set by the minimum taps of the microphone observations needed to make a correlation matrix of full rank. L_b depends directly of L_h in the way that $L_b \geq 2NL_g - 1$ and $L_g \geq \frac{L_h}{2}$ [15]. To obtain a well conditioned correlation matrix we use $L_b = \frac{9}{4}NL_g$.

Working by blocks and pretending a real time approach, the processing time of a block must be smaller than the reproduction time of the previous block. Thereby, working with f_s Hz of sample frequency, a block must be processed in at most $\frac{L_b}{f_s}$ seconds.

In table 2 significative parameters for different lengths of L_g are specified. As we can see, in all cases block filter calculation needs more execution time than block reproduction ($\frac{L_b}{f_s}$ seconds), so the system wouldn't work in real time. Using the same correlation matrix for blocks of x-times the minimum block size L_b, $\frac{L_b}{f_s}$ and consequently the processing block time grows, and real time can be achieved.

As mentioned in previous section, the filtering is done in CUDA and the three channels are filtered in less than 0.009 seconds [18]. Table 2 shows that the number of times that L_b must grow to achieve real time.

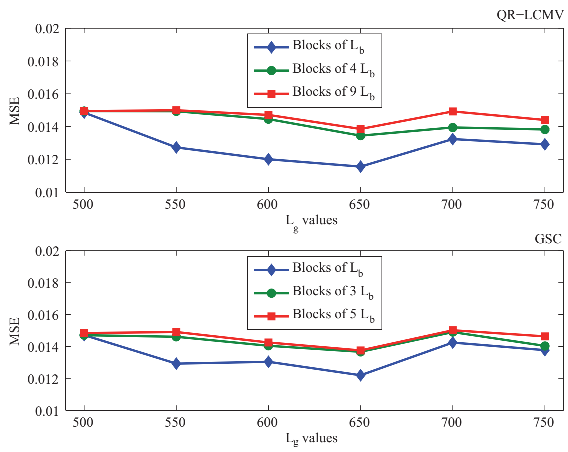

Figure 5: Comparison of MSE working with different block size for QR-LCMV and GSC algorithms.

5.3 Discussion of the introduced error working with sample correlation matrix.

The Mean Squared Error (MSE) has been calculated to quantify the difference between the original signal and the recovered signal with different estimates of the correlation matrix, the so-called sample correlation matrix given in equation (6). The MSE of an estimator $\hat{\mathbf{s}}$ with respect to the estimated parameter $\mathbf{s}$ is defined as $MSE(\hat{\mathbf{s}}) = E[(\hat{\mathbf{s}} - \mathbf{s})^2]$.

Performance of MSE in temporal domain of both algorithms is analyzed in figure 5 to quantify the error. Note that for LCMV algorithm, the application wouldn't work in real-time with blocks of length L_b. On the other hand, blocks of length $8L_g$ would let the application work in real-time for $L_g \leq 700$ and finally, with length $9L_g$ real-time performance is achieved in all cases (see table 2). It can be seen that for all the different values of L_g the error does not increase significantly between different sizes of blocks.The same reasoning can be applied to GSC algorithm.

6. CONCLUSIONS

Different efficient multicore (CPU) and manycore (GPU) implementations of a microphone-array beamforming application have been analyzed in order to achieve a real time beamforming application. The GPU architecture has turned out to be the best option due to its greater speedup performance and its possibility to free up resources from CPU.

It has to be noted that the whole application has been developed in GPU: beamforming filter design, using CULA and CUBLAS libraries, and also the corresponding filtering using CUDA programming.

It has been shown that beamforming algorithm implemented in GPU using CULA and CUBLAS achieve in the best case over 13x of speedup compared to one core CPU. GPU also outperforms multicore parallelization with 4 cores. Note that the speedup could be greater if bigger matrix would be involved in beamforming operations, but that would be a disadvantage to achieve a real time beamforming application.

The CUDA filtering application based on pipeline configuration achieve multichannel filtering needed in our microphone-array system. Furthermore it has been demonstrated that the error of designing beamforming filters using same correlation matrix for 9 blocks instead of calculating on matrix correlation for each block is not critical in the beamforming performance.

REFERENCES

[1] David Patterson, "The Trouble with Multi-Core", *IEEE Spectrum*, Vol. 47, Issue 7, pp. 28–32 and 52–53, Juny 2010.

[2] Trista P. Chen and Yen-Kuang Chen, "Challenges and Opportunities of Obtaining Performance from Multi-Core CPUs and Many-Core GPUs", *ICASSP 2009.*

[3] CUBLAS Library 3.0 and CUFFT Library 1.1, available online: "http://developer.download.nvidia.com"

[4] CULA Library, available online: "http://www.culatools.com"

[5] Intel MKL , available online: "http://software.intel.com/en-us/articles/intel-mkl"

[6] E. Gallo and N. Tsingos, "Efficient 3D audio processing with the GPU", in *ACM Workshop on General Purpose Computing on Graphics Processors*, Los Angeles, USA, vol. 1, pp. C–42, August 2004.

[7] J.D. Owens, M. Houston, D. Luebke, S. Green, J.E. Stone and J.C. Phillips, "GPU computing", *Proc. of the IEEE*, vol. 96, no. 5, pp. 879–899, May 2008.

[8] M.D. McCool, "Signal processing and general-purpose computing and GPUs", *IEEE Signal Processing Magazine*, vol. 24, no. 3, pp. 109–114, May 2007.

[9] Rob V. van Nieuwpoort and John W. Romein, "Building Correlators with Many-Core Hardware", *IEEE Signal Processing Magazine*, Vol. 27, Issue 2, pp. 108–117, March 2010.

[10] Paul R. Dixon, Tasuku Oonishi, Sadaoki Furui, "Fast Computations using Graphics Processors", *ICASSP* 2009.

[11] P. Cardinal, P. Dumouchel, G. Boulianne, and M. Comeau, "GPU accelerated acoustic lilelihood Computations", *ISCA Interspeech*, pp: 964-967, 2008.

[12] Kihiro Hirano and kenji Nakayama, "Implementation of stereophonic acoustic echo canceller on nVIDIA GeForce graphics processing unit", *INSPACS 2009*, January 2009.

[13] Carl-Inge Colombo Nielsen, Ines Hafizovic, "Digital beamforming using a GPU", *ICASSP 2009*, April 2009.

[14] Dimitris Theodoropoulos, Georgi Kuzmanov, Georgi Gaydadjiev, "Multi-core Platforms for Beamforming and Wave Field Synthesis", IEEE Transactions on Multimedia, Issue:99, December 2010.

[15] J. Benesty, J. Chen, Y. Huang and J. Dmochowski, "On microphone-array beamforming from a MIMO acoustic signal processing perspective", *IEEE Trans. on Audio, Speech and Lenguage Processing*, vol. 15, no. 3, pp. 1053–1065, March 2007.

[16] Zen Zhaohua, Zhan Jianhong, Zhao Qian and Liu Hanjun "Research of adaptative beamforming algorithm based on matrix decomposition", Information Engineering and Computer Science (ICIECS), December 2010.

[17] J.A. Belloch, A.M. Vidal, F.J. Martinez-Zaldivar, A. Gonzalez, "Multichannel acoustic signal processing on GPU", *10th International Conference on Computational and Mathematical Methods in Science and Engineering*, Vol 1, pag 181-187,June 2010.

[18] J.A. Belloch, A. Gonzalez, A.M. Vidal, F.J. Martinez-Zaldivar, "Real-time Multichannel Audio Convolution", *GPU Technology Conference (GTC 2010), http://nvidia.com/content/GTC-2010/flvs/2116_GTC2010.mp4*