PARALLEL IMPLEMENTATIONS OF BEAMFORMING DESIGN AND FILTERING FOR MICROPHONE ARRAY APPLICATIONS

Jorge Lorente¹, Gema Piñero¹, Antonio M. Vidal², Jose Antonio Belloch¹, Alberto Gonzalez¹

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 has 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 have 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 CUFFFT: implementations on CUDA of the well-known BLAS computational kernels and FFT algorithms [3], and CULA: implementation of the LAPACK library for GPU [4]. Otherwise, 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 as a multichannel system with 2 inputs (loudspeakers) and 3 outputs (microphones). 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, s1(k) and s2(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 s1(k) or s2(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 gₙ in Figure 1), once all the room channel responses (hₘₙ 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:

\[ x_n(k) = \sum_{m=1}^{M} L_n \sum_{j=1}^{N} h_{nm}(j)s_m(k-j) + v_n(k), \]  

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. \( L_n \) 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.
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:

$$x_n(k) = \sum_{m=1}^{M} h_{nm}^T s_m(k),$$  \hspace{1cm} (2)

where $s_m(k)$ is the column vector defined as $s_m(k) = [s_m(k) \ldots s_m(k-L_h+1)]^T$, $h_{nm}$ is the $R_n \times 1$ 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 $s_n(k)$, that is, $y(k) = \hat{s}_n(k - \tau)$ 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):

$$y(k) = \sum_{n=1}^{N} g_n^T x_n(k),$$  \hspace{1cm} (3)

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

In order to compute the whole vector $x_n(k)$ used in (3) in matrix form, equation (2) has to be rewritten in compact form redefining $h_{nm}$ 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:

$$g^{\text{LCMV}} = \hat{R}_e^{-1} H_m [H_m^T \hat{R}_e^{-1} H_m]^{-1} u_m,$$  \hspace{1cm} (4)

where $g^{\text{LCMV}}$ is formed by the concatenation of filters $g_n$, that is, $g^{\text{LCMV}} = [g_1^T \ldots g_N^T]^T$, matrix $H_m$ 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 sylvestor matrix form and has dimensions $[NLM_g \times L_g]$. Matrix $\hat{R}_e$ is the correlation matrix of the recorded signals and $u_m$ 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 $X^T$ is presented, being $X \in \mathbb{R}^{[NL_g \times K]}$ defined as:

$$X = \frac{1}{\sqrt{K}} \begin{pmatrix} x_1(k) & x_1(k+1) & \cdots & x_1(k+K-1) \\ x_2(k) & x_2(k+1) & \cdots & x_2(k+K-1) \\ \vdots & \vdots & \ddots & \vdots \\ x_N(k) & x_N(k+1) & \cdots & x_N(k+K-1) \end{pmatrix},$$  \hspace{1cm} (5)

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

This way, $X^T = Q \cdot L$, where $Q$ is an orthogonal matrix and $L$ 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 $\hat{R}_e$ as:

$$\hat{R}_e = X \cdot X^T = L^T \cdot Q^T \cdot Q \cdot L = L^T \cdot L.$$  \hspace{1cm} (6)

Now, let us denote matrix $W = \hat{R}_e^{-1} H_m$ such that LCMV beamformer filter $g^{\text{LCMV}}_m$ (4) is expressed as:

$$g^{\text{LCMV}}_m = W [H_m^T W]^{-1} u_m.$$  \hspace{1cm} (7)

It can be shown that:

$$W = \hat{R}_e^{-1} H_m = (L^T L)^{-1} H_m = L^{-1} Z,$$  \hspace{1cm} (8)

where:

$$Z = L^{-1} H_m.$$  \hspace{1cm} (9)

Let us define now matrix $A$ as (7)

$$A = H_m^T W = H_m^T L^{-1} Z = Z^T Z.$$  \hspace{1cm} (10)

Finally defining vector $b_m = A^{-1} u_m$ we can express the LCMV beamformer filter $g^{\text{LCMV}}_m$ as follows:

$$g^{\text{LCMV}}_m = L^{-1} Z \cdot b_m.$$  \hspace{1cm} (11)

Using this method the main operations involve only matrix $Z$ and vector $u_m$. Once $Z$ is computed, matrix $A$ and vector $u_m$ can be calculated and used to get beamforming filters $g^{\text{LCMV}}_m$. 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 $g$ 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 $H_m$ can't be zero. Then, the GSC method can be formulated as:

$$g_m = f_m - B_m w_m,$$  \hspace{1cm} (12)
The number of floating-point operations needed for each implementation of multichannel filtering is here considered. The algorithm we present is focused on the overlap-save technique to carry out a multichannel convolution.

4.2 Filtering

Once the filters have been calculated we can recover the desired signal filtering microphone observations with beamforming filters $g_m$ (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 dimensions of the microphone array and a suitable processor configuration can be achieved easily. See reference 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.

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.

Figure 2: GPU implementation.

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

<table>
<thead>
<tr>
<th>Multiplication</th>
<th>LCMV</th>
<th>GSC filters adaptation</th>
<th>GSC room actualization</th>
</tr>
</thead>
<tbody>
<tr>
<td>M-V Multiplication</td>
<td>$O(L_1^2)$</td>
<td>$O(L_1L_{null})$</td>
<td>$O(L_1L_3)$</td>
</tr>
<tr>
<td>M-M Multiplication</td>
<td>$O(L_1L_2)$</td>
<td>$O(L_1L_{null}L_3)$</td>
<td>$O(L_1L_3)$</td>
</tr>
<tr>
<td>M-M Linear Equation System</td>
<td>$O(L_1^2 + L_1L_3)$</td>
<td>$O(L_1L_{null}L_3)$</td>
<td>$O(L_1L_3)$</td>
</tr>
<tr>
<td>QR decomposition</td>
<td>$O(2L_1^2$</td>
<td>0</td>
<td>$O(4L_1^2 + L_3)$</td>
</tr>
<tr>
<td>Generate matrix $Q$</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Floating-point operations

$3.355 \times 10^5$ 4.2195 $\times 10^5$ 1.4346 $\times 10^5$

where

$$f_m = H_m [H_m^T H_m]^{-1} u_m$$  \hspace{1cm} (13)

is the minimum norm solution of $H_m^T f_m = u_m$ and $B_m$ is the blocking matrix that spans the nullspace of $H_m$ so that $H_m^T B_m = 0$.

Vector $w_m$ is obtained from the following unconstrained optimization problem:

$$\min_{w_m} (f_m - B_m w_m)^T R_s (f_m - B_m w_m)$$  \hspace{1cm} (14)

and its solution is given as:

$$w_m = [B_m^T R_s B_m]^{-1} B_m^T R_s f_m.$$  \hspace{1cm} (15)

3.3 Computational comparison between QR-LCMV and GSC

QR-LCMV and GSC have to update the block of microphone observations $X$, in order to work in real-time. For the purpose of comparing computational cost of both 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 $X$ and $H_m$ where $H_m = [H_1 | H_2 | \ldots | H_L]$, and $B = [B_1 \ldots B_{L_{null}}]$ where $L_1 = N_{Lo}, L_2 = N_{L} + 1, L_3 = L_2 + L_2 - 1$ and $L_{null} = L_3 - L_1$. In order to avoid ill conditioned correlation matrix, we have taken $L_2 = 2N_{Lo}$. Considering that $L_0$ 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 $L_3$ is a variable value always bigger than $\frac{L_2}{2}$, $L_{null}$ increases as $L_0$ increases.

As it can be noted in section 3.2, GSC algorithm divides the filter design calculation in a fixed part (calculation of $f_m$ and $B_m$, only dependent on room dimensions) and an adaptative part (calculation of $w_m$, 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 $L_0 = \frac{L_2}{2}$ value, $L_1 = 3\frac{L_2}{2}$ and $L_3 = L_2 + L_2 - 1 = L_2 + \frac{L_2}{2} - 1 = 3\frac{L_2}{2} - 1$, 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, $L_0 = 1000$ and $L_0 = \frac{L_2}{2}$.
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 $L_b = 1000$ 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 $L_b \geq 2NL_h - 1$ and $L_h \geq \frac{f_s}{2h} \geq 1000$ [15]. To obtain a well conditioned correlation matrix we use $L_b = \frac{4}{5}NL_h$.

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 $\frac{f_s}{2}$ 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{f_s}{2} seconds), so the system wouldn’t work in real time. Using the same correlation matrix for blocks of x-times the minimum block size $L_b$, $\frac{f_s}{2}$ 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 GPU outperform CPU is "4 core CPU" in the fixed part and "4 core CPU" in the adaptive part.
The MSE of an estimator $\theta$ with respect to the estimated parameter $s$ is defined as $MSE(\theta) = E[(\theta - s)^2]$.

Performance of MSE in temporal domain of both algorithms is analyzed in figure 5 to quantify the error. Note that for LCMV algorithm, the approach wouldn’t work in real-time with blocks of length $L_b$. On the other hand, blocks of length $8L_b$ would let the application work in real-time for $L_b \leq 700$ and finally, with length $9L_b$ real-time performance is achieved in all cases (see table 2). It can be seen that for all the different values of $L_b$ 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 multicores 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