Real-time adaptive algorithms using a Graphics Processing Unit

Jorge Lorente[1], Miguel Ferrer[1], José A. Belloch[1], Gema Piñero[1], María de Diego[1], Alberto González[1], Antonio M. Vidal[2].

Instituto de Telecomunicaciones y Aplicaciones Multimedia[1] (iTEAM),
Departamento de Sistemas Informáticos y Computación[2] (DSIC),
Universitat Politècnica de València,
8G Building - access D - Camino de Vera s/n - 46022 Valencia (Spain)
Jorge Lorente: jorlogi@iteam.upv.es

Abstract

Graphics Processing Units (GPUs) have been recently used as coprocessors capable of performing tasks that are not necessarily related to graphics processing in order to optimize computing resources. The use of GPUs has being extended to a wide variety of intensive-computation applications among which audio processing is included. However data transactions between the CPU and the GPU and vice versa have questioned the viability of GPUs for applications in which direct and real-time interaction between microphone and loudspeaker is required. One of the audio applications that requires real-time feedback is adaptive channel identification. Particularly, when the partitioned Least Mean Squares (LMS) algorithm is used in the frequency domain, the size of input-data buffers and filters and how they can be managed in order to successfully exploit the GPU resources is an important key in the design process. This paper discusses the design and implementation of all the processing blocks of an adaptive channel identification system on a GPU, proposing a GPU implementation that can be easily adapted to any acoustic scenario, while freeing up CPU resources for other tasks.

Keywords: Graphics Processing Unit, adaptive channel identification, LMS algorithm.

1. Introduction

In recent years the number of scientific contributions and research projects related to the use of Graphics Processing Units (GPUs) as general purpose computers (GPGPU) has significantly increased [1-3]. This phenomenon has occurred in almost all engineering fields that require intensive computing, and Signal Processing is not an exception [4-7]. This fact can be appreciated in the following web site [8], where several applications related to audio signal processing using GPU can be found, such as [9] and [10]. In [9], it is described how a generalized audio multichannel system is mapped on the GPU. This system efficiently exploits the GPU resources when multiple convolutions are executed concurrently. The second application consists in designing a crosstalk canceller on a notebook GPU. Moreover, GPU computing has already been applied to different problems in acoustics. Theodoropoulos et al. in [11] study the implementation of beamforming and wave-field synthesis on GPU. In [12] Lorente et al. discussed about parallel implementations of beamforming design and filtering for microphone array applications. Webb and Bilbo in [13] carried out studies of computer modeling of room acoustics, as well as geometric acoustic modeling like ray-tracing [14].

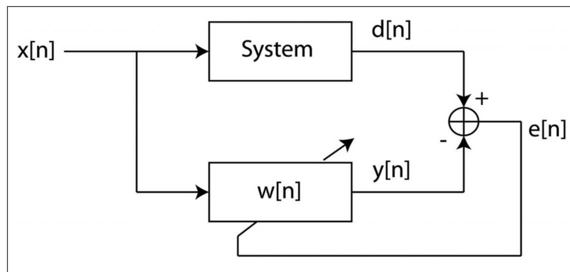
Up to now, the applications where GPU and audio processing have been involved consist in convolving multiple input signals with static filters. However adaptive filter applications in which real-time signal acquisition, data transactions among CPU, GPUs and audio cards, and loudspeaker reproduction could be critical have not yet been tested on GPU. The purpose of this paper is to describe how to use GPUs for real-time adaptive audio applications where the adaptive filter coefficients must vary over time to reach a target that depends on the application. This implies the need of monitoring those signals that are related to the objective. Furthermore, because of the inherent capability of parallelization of the GPUs, not all the adaptive algorithms can run efficiently on these systems. Adaptive algorithms that work sample by

sample are particularly difficult to optimize in GPUs, so in this study we have used block algorithms that introduce the disadvantage of higher latency, but optimize the computational cost of these algorithms in the frequency domain. In order to illustrate how to overcome the difficulties that appear in the use of GPUs in adaptive systems, we have implemented a channel identification system using the block least mean squares (BLMS) algorithm and its computationally efficient version in the frequency domain (FBLMS). Most of the actions described can be extended to any other adaptive algorithm and/or similar application.

The rest of the paper is organized as follows. In section 2 we describe the BLMS and the FBLMS algorithms for a channel identification system. The real-time implementation of this system is presented in section 3. Section 4 offers the implementation carried out on GPU. Section 5 analyzes the performance of the system based on GPU. Finally some concluding remarks are reported in section 6.

2. Algorithms description

Figure 1 shows the general scheme of an adaptive channel identifier, where 'system' represents the channel we want to identify. Signal x[n] is the reference signal, and d[n] is the result of exciting the 'system' with x[n]. Variable w[n] is the adaptive Finite Impulse Response (FIR) filter that must fit the 'system', and y[n] is the adaptive output. Finally, e[n] is the error signal.



■ **Figure 1.** *Adaptive channel identification scheme.*

Considering that the block labeled with 'system' is an electroacoustic system, it is described by an impulse response between two points in space (usually located in a room). In this case, the system is excited with a broadband signal x[n]. The error signal is defined as: e[n]=d[n]-y[n]. The target is the cancellation of the error signal, which would mean that the coefficients of the adaptive filter fit those of the unknown system. Adaptive algorithms minimize the error signal e[n] or some function of this signal, such that w[n] at steady state is a good estimation of the impulse response of the system to identify.

The LMS algorithm [15] is one of the most commonly used adaptive algorithms for its high performance, simplicity and robustness. The LMS equation for updating the L filter coefficients is given by:

$$\boldsymbol{w}[n]=\boldsymbol{w}[n\text{-}1]+\mu\boldsymbol{x}_L[n]\mathrm{e}[n] \quad [1]$$

where $\boldsymbol{x}_L[n]$ is a vector with the last L samples of $\boldsymbol{x}[n]$, μ is a positive constant that controls the speed of convergence and $\boldsymbol{w}[n]$ is the L coefficient vector. The filter output is expressed as:

$$y[n]=\boldsymbol{w}^T[n]\boldsymbol{x}_L[n] \quad [2]$$

In equation (1) filter coefficients $\boldsymbol{w}[n]$ are adapted sample by sample. However, because of the type of data management imposed by GPUs, we have to work with blocks of samples, thus, using the block LMS algorithm (BLMS) [16] instead of the LMS. Assuming that each data block is composed by M samples, filter coefficients are adapted according to:

$$\boldsymbol{w}[n]=\boldsymbol{w}[n\text{-}M]+\mu\sum_{i=0}^{M-1}\boldsymbol{x}_L[n\text{-}i]\mathrm{e}[n\text{-}i] \quad [3]$$

and the algorithm generates M output samples (y[n] values) as follows:

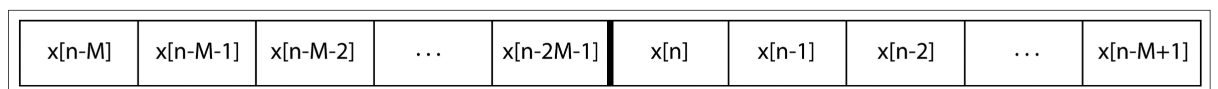
$$y[n\text{-}i]=\boldsymbol{w}^T[n]\boldsymbol{x}_L[n\text{-}i],\quad \mathrm{i}=0,\dots,M\text{-}1 \quad [4]$$

Where equations (3)-(4) perform a total of $(2L+1)M$ multiplications for each block of M samples.

The BLMS algorithm has been efficiently implemented in the frequency domain providing the frequency BLMS (FBLMS) algorithm. Thus, a computational cost reduction is achieved with respect to the BLMS in time domain by efficiently computing the Discrete-time Fourier Transform (DFT) with the Fast Fourier Transform (FFT). To this end, we have used an FFT of $2M$ points and the data blocks of time-signal x[n] have been arranged as follows:

Thus, input signal samples are placed in two blocks of M samples each so the oldest samples are placed in the first block and the most recent ones in the second. Let us call $\mathbf{x}_{2M}[n]$ to this data arrangement of the input signal, equation (2) could be rewritten using the transformed domain as:

$$\boldsymbol{y}_{M[n]} = IFFT\left(FFT\begin{pmatrix}\boldsymbol{w}[n]\\ \boldsymbol{0}\end{pmatrix}FFT(\boldsymbol{x}_{2M}[n])\right) \quad [5]$$



■ **Figure 2.** *Reordering of the data input blocks.*

where the first M elements of the vector on the left-hand side of the equality are discarded, and the vector with the coefficients of the adaptive filter is padded with zeros until reaching a length of $2M$ samples. The coefficients $\boldsymbol{w}[n]$ in time domain are the first M samples of the Inverse FFT (IFFT) of the vector in the frequency domain. Therefore, the coefficients are updated in frequency domain as follows:

$$FFT\begin{bmatrix} w[n] \\ 0 \end{bmatrix} = FFT\begin{bmatrix} w[n-1] \\ 0 \end{bmatrix} + \mu FFT\begin{bmatrix} \phi_M[n] \\ 0 \end{bmatrix} \quad [6]$$

where $\phi_M[n]$ are the first M elements of the IFFT of the estimated cross-correlation vector between the error signal and the vector of the reference signal, calculated as follows:

$$\phi[n] = IFFT\left[FFT\begin{bmatrix} 0 \\ e[n] \end{bmatrix} FFT(\boldsymbol{x}_{2M}[n])^*\right] \quad [7]$$

where symbol * denotes the conjugate and $\boldsymbol{e}[n]$ is a vector with the last M elements of the signal e[n]. In this case, the algorithm, equations (5)-(7), performs $4 \cdot M \cdot (log_2(2 \cdot M) + 1)$ multiplications for each data block of M samples.

One limitation of the above algorithm is that it can identify any acoustic system as long as its impulse response can be approximated by a FIR system with a maximum of M coefficients, which in turn imposes certain constraints when setting the sizes of data blocks (M) on the GPU. One solution to estimate larger response systems is to split up the adaptive filter coefficients into blocks. Thus, the algorithm should work simultaneously with all the partitions of size M that have to be adapted accordingly. Given that the GPU is specifically designed to work in parallel, this solution could efficiently exploit the GPU resources. The resulting implementation is called the partitioned FBLMS (PFBLMS) algorithm.

3. System overview

The detailed implementation of a channel estimator on a GPU is depicted in Figure 3. The CPU controls the data transfer between the audio card and the transducers (microphone and loudspeaker), and between the audio card and the GPU. In our experiment the CPU is an Intel Core i7 (3.07 GHz) with 8 Gb of RAM. The GPU is a Geforce GTX 580 with 2.0 CUDA capability and 16 multiprocessors of 32 cores each (Fermi architecture) [17]. The audio card is a MOTU 24 I/O [18] that uses the ASIO (Audio Stream Input/Output) [19] driver to communicate with the CPU. The ASIO driver provides input/output buffers; the input buffers are connected to the microphones and the output buffers to the loudspeakers. Table 1 summarizes the main GPU characteristics.

This work analyzes the viability of the use of GPUs for real-time adaptive applications.

The audio car allows working at two sampling rates (f_s): 44.1 and 48 kHz. The 44.1 kHz sampling rate has been chosen as it is a quite high rate for the sounds involved. Regarding the block size, and considering the wide range of block sizes that the MOTU audio car offers, we have used values between M=128 and 2048.

CUDA Architecture	Fermi
CUDA Capability	2.0
Number of SMs	16
Number of cores per SMs	32
Maximum number of threads per block	1024
Overlap transfer with computations	yes

Table 1. *Characteristics of the GPU (GTX-580M).*

4. Real-Time Implementation

In real-time audio acquisition, once the input-data buffers are filled, they are transferred through the PCI-Express bus to the GPU where all the processing is carried out. Once the execution on GPU ends, audio samples are saved in

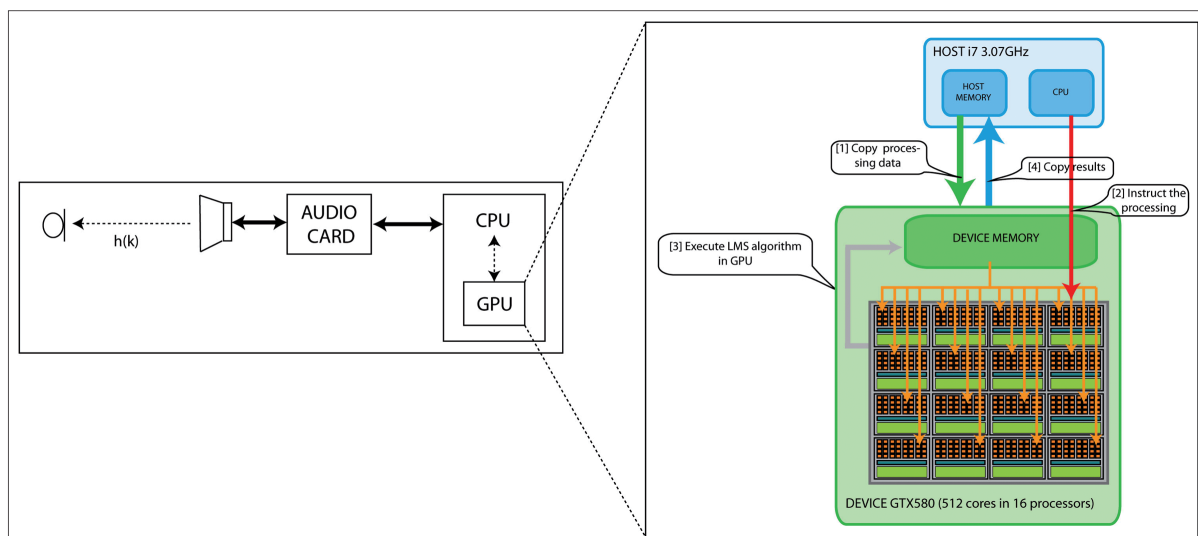


Figure 3. *GPU implementation.*

Adaptive algorithms that work sample by sample are particularly difficult to optimize in GPUs, so in this study we have used block algorithms.

output-data buffers that are subsequently sent back to the CPU in order to be reproduced by the loudspeakers. In adaptive applications this process is repeated at each iteration. The system shown in Figure 3 is composed by one input-data buffer and one output-data buffer.

The designed application will work in real-time if the following condition is satisfied: $t_{proc}<t_{buff}$, where t_{buff} is the time spent to fill the input-data buffer and is equal to M/f_s, and t_{proc} is the execution time measured from the moment the input-data buffer is sent to the GPU to that the output-data buffer comes back to the CPU, including transfer times CPU↔GPU and the buffer processing in the GPU. The times t_{proc} and t_{buff} allow us to calculate two important parameters on audio signal processing: latency and throughput, that will be analyzed in section 6.3.

5. GPU implementation

As commented in section 3 we have implemented a channel identification algorithm by partitioning the adaptive filter. This occurs when the filter size is longer than the input-data buffer or in applications where the latency time requires minimizing the input-data buffer. The main goal of partitioning the filter is to obtain the best performance from the resources of the GPU making use of the "SIMD (Simple Instruction Multiple Data) GPU architecture". Considering a filter size of L, and a block size of M, the filter is uniformly partitioned in $P=L/M$ blocks of M samples. If L is not a multiple of M, the last block is padded with zeros. Figure 4 illustrates the partition of the filter.

Once the filter coefficients are arranged as in Figure 4, the channel estimation is carried out through the PFBLMS algorithm described in Table 2 which is explained below. Additionally, the GPU uses the kernels described in Table

Step 1) The algorithm build an input data matrix $\mathbf{x}$ of size $2MxP$ in order to fit the element-wise multiplication when filtering by $\mathbf{W}$. Matrix $\mathbf{x}$ is formed at each iteration filling one of the P columns with the input-data buffer. As it is shown in Figure 5, column x_1 of size $2M$ is filled with M zeros followed by the M input-data buffer of iteration 1, column x_2 is formed with the M input-data buffer of iteration 1 followed by the M input-data buffer of the iteration 2, and so on. After P iterations all columns are filled. Then, at iteration $P+1$ the column filled in iteration 1 is rewritten using the current input data buffer.

STEP	CHANNEL ESTIMATION PSEUDO CODE
1	Build input data matrix $\mathbf{x}$
2	Perform an element-wise matrix multiplication: $\mathbf{OUT=X{\cdot}W}$, where $\mathbf{X}$=FFT($\mathbf{x}$)
3	Reduce sum of all columns of matrix $\mathbf{OUT}$ to one column
4	Carry out an IFFT of size $2M$ of the result obtained in step 3; discard the first M samples and save the last M samples as output vector $\mathbf{y}$
5	Calculate the M elements of the error signal: $\mathbf{e=d}$-$\boldsymbol{y}$.
6	Perform an FFT of the error signal after filling its first M elements with zeros: $\mathbf{E}$=FFT($\mathbf{0}$ $\mathbf{e}$).
7	Calculate the inverse FFT of the autocorrelation between the error vector and the reference signal: $\phi=FFT(\mathbf{E{\cdot}X^*})$
8	Discard the last M elements of each column of matrix ϕ and fill it with zeros. Then perform an FFT of the resulting matrix to obtain the step variable for updating the filter coefficients: $\mathbf{step}$=FFT($\boldsymbol{\phi}_M$ $\mathbf{0}$).
9	Update the filter coefficients: $\mathbf{W=W+\mu{\cdot}step}$

Table 2. *PFBLMS algorithm implemented on GPU.*

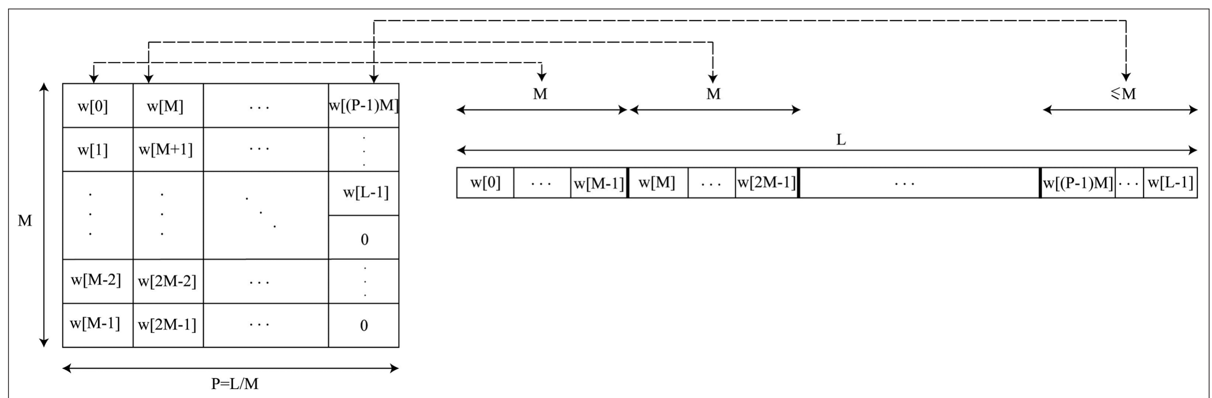


Figure 4. *Partition of the adaptive filter.*

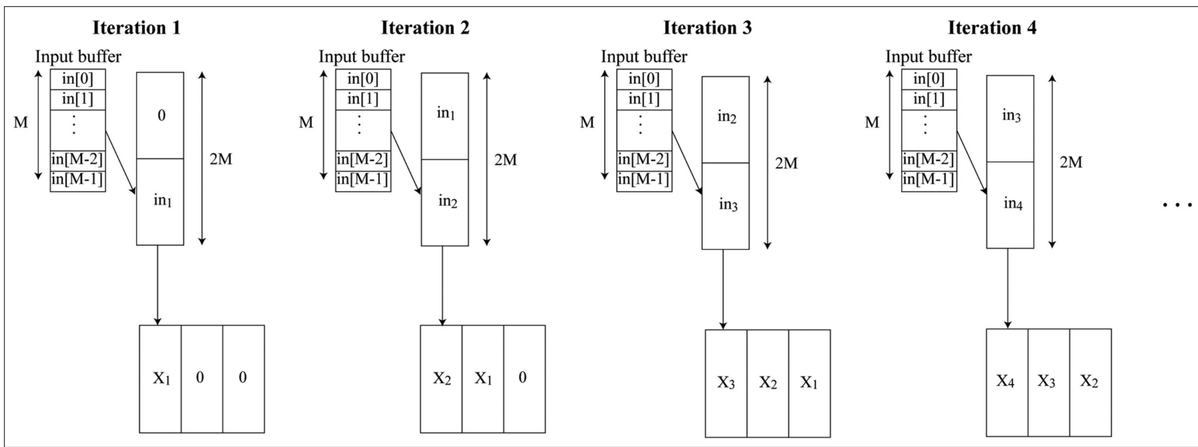


Figure 5. *Example of input-data matrix when P=3.*

Step 2) Once the matrix **x** is arranged, it must be filtered. For this purpose, an FFT of size *2M* of each column of matrix **x** is performed; then, an element-wise multiplication in frequency domain between matrix **X** (obtained as the FFT of the corresponding columns of matrix **x**) and filter coefficients **W** is carried out. **W** is a *2MxP* matrix whose columns are the FFT of size *2M* of the *P* blocks of *M* time-coefficients padded with *M* zeros. The FFT of the NVIDIA CUFFT library [20] has been used since this library allows to carry out multiple one-dimensional FFTS simultaneously. When the *P* FFTS have been carried out, the element-wise multiplication is performed launching a CUDA kernel (*kernel 1* in Table 3).

Figure 6 illustrates two schemes of the complex element-wise matrix multiplication. Scheme A shows the natural implementation of the algorithm where the input-data matrix is ordered in such a way that elements of *i-th* column of matrix **X** are element-wise multiplied by elements of *i-th* column of matrix **W**. This scheme has the disadvantage that all columns except one of the matrix **X** are moved in each iteration, so there are *2 M* (*P-1*) ele-

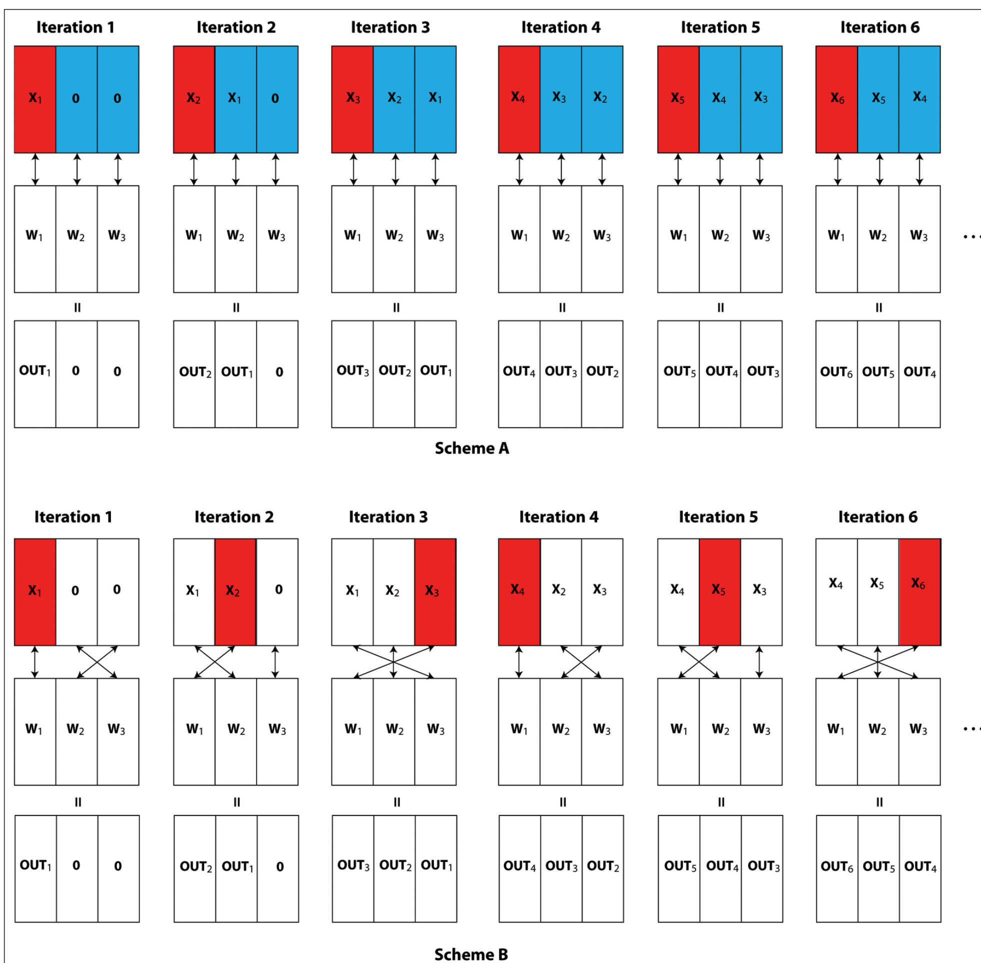
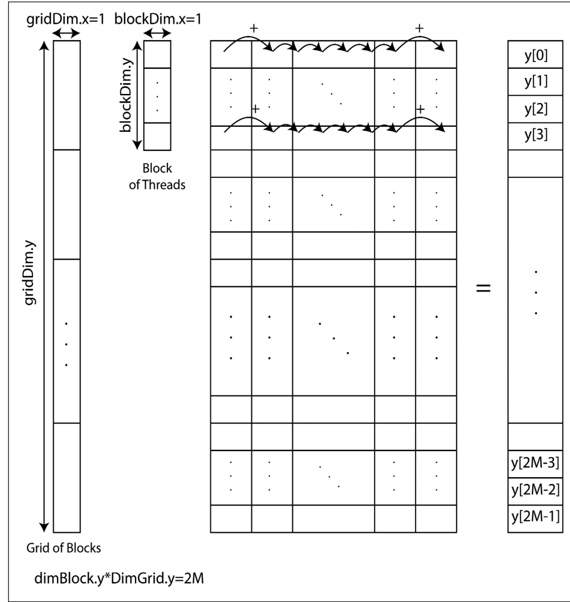


Figure 6. *Two different schemes of element-wise matrix multiplication when P=3.*

ments copied in GPU memory at each iteration. The copied data are represented in blue, and the current input data are represented in red. On the other hand, scheme B of Figure 6 shows an optimized procedure to perform the multiplication, where no ordering is done and GPU memory transactions are avoided. This is achieved by redefining the thread access to GPU memory.

Step 3) Once the multiplication is carried out, a sum of all the components of the partitioned output is needed, reducing all columns to a single one. See Figure 7 and *kernel 2* in Table 3.



■ **Figure 7.** *Representation of the CUDA operation that carries out the reduction sum of each row.*

Step 4) Next step consists in performing a one-dimensional IFFT of size $2M$ of the vector obtained in step 3, and saving only the last M samples which correspond to the output signal of the adaptive filter. As in step 2, the IFFT is carried out using the CUFFT library.

Step 5) Vector $\mathbf{y}$ is used to calculate the error vector of size M: $\mathbf{e}[n]=\mathbf{d}[n]-\mathbf{y_M}[n]$. *Kernel 3* implements this subtraction.

Step 6) An FFT of size $2M$ of the error vector previously padded with zeros is performed resulting in vector $\mathbf{E}$ ($\mathbf{E}$=FFT($[\mathbf{0}^T\ \mathbf{e}^T]^T$)).

Step 7) Next step computes the Inverse FFT of the correlation between the error vector and the reference signal: $\boldsymbol{\phi}[n]$=IFFT($\mathbf{E}\cdot\boldsymbol{X}^*$). *Kernel 4* is implemented to perform the $2M$ element-wise multiplication of the vector $\mathbf{E}$ with matrix $\mathbf{X}^*$.

Step 8) The last M elements of each column of matrix $\boldsymbol{\phi}$ are discarded and filled with zeros. Then, an FFT of the resulting matrix is performed: $\mathbf{step}=\text{FFT}\begin{bmatrix}\boldsymbol{\phi}_M\\ \mathbf{0}\end{bmatrix}$.

Step 9) Finally kernel 5 in Table 3 is implemented to update the filters ($\mathbf{W}=\mathbf{W}+\mu\cdot\mathbf{step}$).

6. Results

In this section we analyze the performance of the real-time implementation from three points of view:

- Algorithm behavior.
- Implementation aspects: latency and throughput times.
- Analysis of the maximum number of channels that could be identified by our implementation in a multichannel application.

6.1 Algorithm behavior

Figure 8 shows the estimated channel and the residual error obtained when a block size of M=128 and a channel length of L=4096 are used. The residual error shows the speed of convergence of the algorithm (note that it can be adjusted varying the step parameter μ). Since the electroacoustic channel to be estimated is composed by the

Kernel 1	This kernel launches $M{\cdot}P$ threads divided into a grid of P blocks of M threads each block. The kernel uses each thread for processing each sample, so each thread will make a complex multiplication between an element of matrix $\mathbf{X}$ and its corresponding component of matrix $\mathbf{W}$.
Kernel 2	This kernel uses one-dimensional blocks and will be configured by $2M$ threads in total. Each thread carries out P sums. The result is a column vector of $2M$ elements, each element containing the reduction sum of each row.
Kernel 3	This kernel is launched with M threads divided in one-dimensional blocks. Each thread performs a subtraction between a complex vale of vector $\mathbf{d}$ and other from vector $\mathbf{y}$.
Kernel 4	It is launched as *kernel 1*, using the access to memory positions of matrix $\mathbf{X}$ explained in scheme B. The difference is that in this case $\mathbf{E}$ is a vector, so each column of matrix $\mathbf{X}$ is element-wise multiplied by the same vector $\mathbf{E}$. Also note that before the element-wise multiplication, the kernel carries out a conjugation of the elements of matrix $\mathbf{X}$.
Kernel 5	It has the same thread configuration as *kernel 1*, but each thread performs a sum instead of a multiplication.

■ **Table 3.** *GPU kernel descriptions.*

channel between loudspeaker and microphone and the audio card characteristics, although the estimated channel should be independent of the buffer size, the audio card introduces a delay that depends on the buffer size. More specifically this delay is equal to the buffer *size+22* samples. So at the end, the result of the estimated channel is

It can be stated that GPUs are suitable for audio adaptive systems working as co-processors, and that they are capable of managing multiple channels without overloading the CPU.

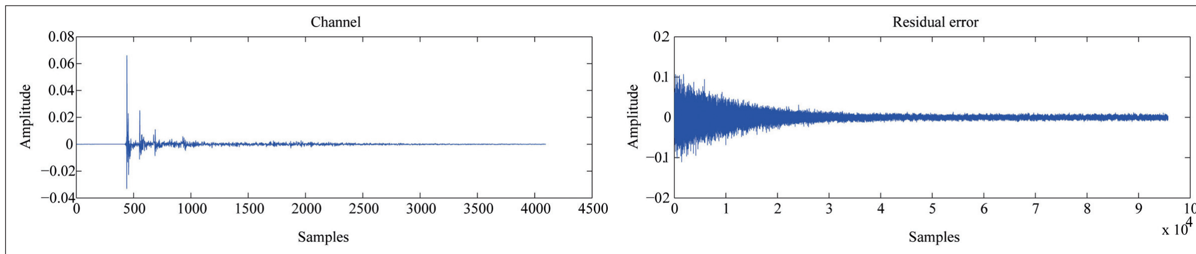


Figure 8. *Estimated channel and residual error for M=128 and L=4096.*

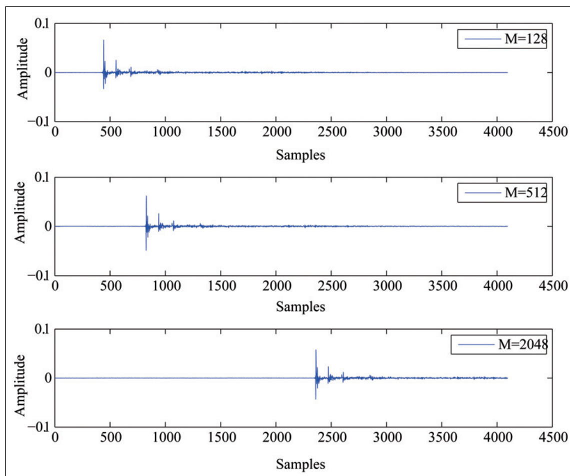


Figure 9. *Estimations of the same channel for different input-data buffer size.*

the channel between the loudspeaker and microphone plus the delay introduced by the audio card, which makes the result be variable with M. Figure 9 shows the same channel estimated with different buffer sizes.

6.2. Implementation aspects

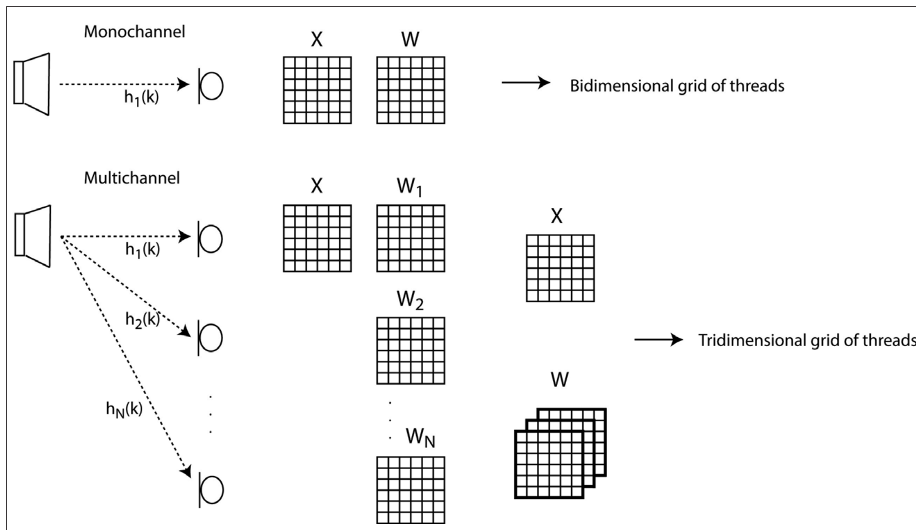
Table 4 shows the latency time and throughput for different thread configuration and different sizes of the input-data buffer. The best thread configurations are highlighted for each buffer size M. Latency time is the time from which the processing starts until an output response is given and is calculated as $t_{proc}+t_{buff}$. The throughput is defined as the number of input samples processed per second (*throughput=M/Latency*). Table 4 shows that the throughput does not vary significantly and this is because tproc is much shorter than tbuff. The maximum throughput is achieved when the size of the input-data buffer is 512 and 256 threads per block are used, achieving a peak value over 45.000 samples per second.

6.3. Multichannel performance

Figure 10 shows that a multi-channel extension (1 loudspeaker and N microphones) from the single channel implementation can be easily obtained through data arrangement. The same reference signal is used to calculate the paths between the loudspeaker and the different microphones. In the multi-channel implementation,

Input-data buffer size (M)	Threads per block	t_{buff} (ms) Buffer filling time	t_{proc} (ms) Processing time	Latency (ms) (samples/s)	Throughput
128	128	2.9025	0.4412	3.3437	$3.8289 \cdot 10^4$
256	128	5.8050	0.5412	6.3462	$4.0339 \cdot 10^4$
	256		0.4597	6.2647	$4.0863 \cdot 10^4$
512	128	11.6100	0.4531	12.0631	$4.2445 \cdot 10^4$
	256		0.4307	12.0407	$4.5224 \cdot 10^4$
	512		0.4631	12.0731	$4.2408 \cdot 10^4$
1024	128	23.2200	0.6801	23.9001	$4.2845 \cdot 10^4$
	256		0.5692	23.7892	$4.3044 \cdot 10^4$
	512		0.5162	23.7362	$4.3140 \cdot 10^4$
	1024		0.6145	23.8345	$4.2962 \cdot 10^4$
2048	128	46.4399	0.6019	47.0418	$4.3536 \cdot 10^4$
	256		0.5412	46.9811	$4.3592 \cdot 10^4$
	512		0.6051	47.0450	$4.3533 \cdot 10^4$
	1024		0.7051	47.1450	$4.3440 \cdot 10^4$

Table 4. *Latency time and throughput obtained for different input-data buffer size.*



■ **Figure 10.** *Extension of the GPU channel estimator to a multi-channel system.*

input-data matrix becomes a three-dimensional matrix where each channel occupies a plane of the matrix. These three-dimensional data matrices can be handled by the GPU through CUDA SDK 4.0, which let the user to configure tridimensional grids of threads. Moreover, the same kernels of Table 3 can be used but launched with different grid configuration.

Table 5 shows an estimate of the number of channels that the algorithm could identify for different block sizes. Taking into account that the time t_{proc} is the time used to estimate one channel, and the time t_{buff} is the maximum processing time to achieve real-time condition, the ratio t_{buff}/t_{proc} could be a good estimate of the number of channels that the application could be able to identify. Moreover, considering the parallel properties of the GPU and taking into account that adding more channels to the processing means increasing the size of the matrices, the processing time to identify X channels should be less than X-times the single-channel processing time. Then, this estimation could be even greater.

7. Conclusions

This work analyzes the viability of the use of Graphics Processing Units (GPUs) for real-time adaptive applications. To this end, a single channel identification system has been implemented. A MOTU audio card has been used as the interface between the computer and the microphone/loudspeaker, so the electroacoustic channel involves the audio card acquisition delay plus the actual channel between the loudspeaker and the microphone. A partitioned BLMS algorithm in the frequency domain called PFBLMS algorithm has been implemented in order to exploit the SIMD (Simple Instruction Multiple Data) GPU architecture. For this purpose, each CUDA kernel has been designed seeking the most efficient implementation avoiding memory copies in GPU. Some CUDA characteristics as both the number of threads per block and the distribution of threads within the blocks have been also analyzed.

The evaluation test shows a good performance of the algorithm implemented over GPU in real time. Table 4 has shown that it is important to carry out an analysis of CUDA aspects before configuring a grid of threads, due to the distribution of the threads inside the block and these ones inside the grid. Once CUDA parameters have been set, a multichannel study demonstrates that depending on the input-data buffer sizes the application can identify up to about 85 channels simultaneously. As a result of the good performance offered by the GPU implementation, it can be stated that GPUs are suitable for audio adaptive systems working as co-processors, and that they are capable of managing multiple channels without overloading the CPU.

Finally, this study opens a research line on GPU implementations of computationally complex adaptive algorithms such as the multichannel active noise control algorithms.

Input-data buffer size (M)	t_{buff} (ms) Buffer filling time	t_{proc} (ms) Processing time	Multichannel estimation (t_{buff}/t_{proc})
128	2.9025	0.4412	6
256	5.8050	0.4597	12
512	11.6100	0.4307	26
1024	23.2200	0.5162	44
2048	46.4399	0.5412	85

■ **Table 5.** *Estimation of the number of channels that the system could identify for different input-data buffer sizes.*

Acknowledgements

This work has been financially supported by the Spanish Ministerio de Ciencia e Innovación TEC2009-13741, Universitat Politècnica de València through "Programa de Apoyo a la Investigación y Desarrollo (PAID-05-11)" and Generalitat Valenciana through project PROMETEO/2009/013 and GV/2010/027.

References

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

[2] 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.

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

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

[5] A. Gonzalez, J. A. Belloch, F. J. Martinez, P. Alonso, V. M. Garcia, E. S. Quintana-Orti, A. Remon, A. M. Vidal, "The Impact of the Multi-core Revolution on Signal Processing", Waves, vol. 2, pp. 74-85, 2010.

[6] A. Gonzalez, J. A. Belloch, G. Piñero, J. Lorente, M. Ferrer, S. Roger, C. Roig, F. J. Martinez, M. de Diego, P. Alonso, V. M. Garcia, E. S. Quintana-Orti, A. Remon, A. M. Vidal, "Application of Multi-core and GPU Architectures on Signal Processing: Case Studies", Waves, vol. 2, pp. 86-96, 2010.

[7] V. M. Garcia, A. Gonzalez, C. Gonzalez, F. J. Martinez-Zaldivar, C. Ramiro, S. Roger, A. M. Vidal, "The impact of GPU/Multicore in Signal Processing: a quantitative approach", in Waves, vol. 3, pp. 96-106, 2011.

[8] CUDA ZONE http://www.nvidia.co.uk/object/cuda_apps_flash_new_uk.html#state=home

[9] J.A. Belloch, A. Gonzalez, F. J. Martinez-Zaldivar and A. M. Vidal, "Real-time massive convolution for audio applications on GPU", Journal of Supercomputing, vol. 58, no. 3, pp. 449-457, 2011.

[10] J.A. Belloch, A. Gonzalez, F. J. Martinez-Zaldivar and A. M. Vidal, "A real-time crosstalk canceller on a notebook GPU", in Proc. ICME 2011, Barcelona, July 2011.

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

[12] J. Lorente, G. Piñero, A. M. Vidal, J. A. Belloch, A. Gonzalez, "Parallel Implementations Of Beamforming Design And Filtering For Microphone Array Applications", EUSIPCO 2011, August 2011.

[13] C.J. Webb and S. Bilbo, "Computing room acoustics with CUDA – 3FDTD schemes with boundary losses and viscosity", in Proc. ICASSP 2010, Prague, May 2011.

[14] N. Rober, U. Kaminski, and M. Masuch, "Ray acoustics using computer graphics technology", in Proc. DAFx-07, Bourdeaux, June 2011.

[15] B. Widrow and, S. D. Stearns, Adaptive Signal Processing, Prentice-Hall Signal Processing Series, 1985.

[16] S. Haykin, Adaptive Filter Theory. Prentice-Hall, 4th edition, 2002.

[17] NVIDIA Next Generation: FERMI, online at http://www.nvidia.com/object/fermi_architecture.html

[18] MOTU Audio 24I/O interface, available online at: http://www.motu.com/products/pciaudio/24IO.

[19] Audio Streaming Input Output (ASIO) 2.2, Steinberg Media Technologies GmbH, available online at: http://www.steinberg.net/en/company/developer.html.

[20] NVIDIA Library CUFFT, online at: http://developer.download.nvidia.com/compute/DevZone/docs/html/CUDALibraries/doc/CUFFT_Library.pdf

Curriculum vitae

Jorge Lorente was born in Algemesí, Spain in 1985. He received the Ingeniero Técnico de Telecomunicación degree from the Universidad Politécnica de Valencia, Spain, in 2007 and the MSc. degree in Telecommunication Technologies in 2010. Currently, he is a PhD grant holder from the Universitat Politècnica de València under the FPI program and is pursuing his PhD degree in Electrical Engineering at the Institute of Telecommunications and Multimedia Applications (iTEAM). His research focuses on adaptive algorithms and audio signal processing onto the CUDA environment.

Miguel Ferrer was born in Almería, Spain. He received the Ingeniero de Telecomunicacion degree from the Universidad Politécnica de Valencia (UPV) in 2000, and the Ph.D degree in 2008. In 2000, he spent six months at the Institute of aplicated research of automobile in Tarragona (Spain) where he was involved in research on Active noise control applied into interior noise cars and subjective evaluation by means of psychoacoustics study. In 2001 he began to work in GTAC (Grupo de Tratamiento de Audio y Comunicaciones) that belongs to the Institute of Telecommunications and Multimedia Applications. He is currently working as assitan professor in digital signal processing in communications Department of UPV. His area of interests includes efficient adaptive algorithm and digital audio processing.

José Antonio Belloch was born in Requena, Spain, in 1983. He received the degree in Electrical Engineering from the Universidad Politécnica de Valencia, in 2007 and MSc. Degree Master in Parallel and Distributed Computing in 2010. In 2008, he worked for the Company Getemed (Teltow, Germany) as a software developer through multithreading platforms. His interest in applying parallel programming for Signal processing led him to enroll in a PhD program in 2009 with the Audio and Communications Signal Processing Group (GTAC). Currently, he works towards his PhD degree on multichannel Audio-Signal Processing onto the CUDA environment.

María de Diego was born in Valencia, Spain, in 1970. She received the Telecommunication Engineering degree from the Universidad Politecnica de Valencia (UPV) in 1994, and the Ph.D degree in 2003. Her dissertation was on active noise conformation of enclosed acoustic fields. She is currently working as Associate Professor in digital signal processing and communications. Dr. de Diego has been involved in different research projects including active noise control, fast adaptive filtering algorithms, sound quality evaluation, and 3-D sound reproduction, in the Institute of Telecommunications and Multimedia Applications (iTEAM) of Valencia. She has published more than 40 papers in journals and conferences about signal processing and applied acoustics. Her current research interest include multichannel signal processing and sound quality improvement.

Antonio M. Vidal receives his M.S. degree in Physics from the "Universidad de Valencia", Spain, in 1972, and his Ph.D. degree in Computer Science from the "Universidad Politécnica de Valencia", Spain, in 1990. Since 1992 he has been in the Universidad Politécnica de Valencia, Spain, where he is currently a full professor in the Department of Computer Science. He is the coordinator of the project "High Performance Computing on Current Architectures for Problems of Multiple Signal Processing", currently developed by INCO2 Group and financed by the Generalitat Valenciana, in the frame of PROMETEO Program for research groups of excellence. His main areas of interest include parallel computing with applications in numerical linear algebra and signal processing.

Alberto González was born in Valencia, Spain, in 1968. He received the Ingeniero de Telecomunicacion degree from the Universidad Politecnica de Catalonia, Spain in 1992, and Ph.D degree from de Universidad Politecnica de Valencia (UPV), Spain in 1997. His dissertation was on adaptive filtering for active control applications. From January 1995, he visited the Institute of Sound and Vibration Research, University of Southampton, UK, where he was involved in research on digital signal processing for active control. He is currently heading the Audio and Communications Signal Processing Research Group (www.gtac.upv.es) that belongs to the Institute of Telecommunications and Multimedia Applications (i-TEAM, www.iteam.es). Dr. González serves as Professor in digital signal processing and communications at UPV where he heads the Communications Department (www.dcom.upv.es) since April 2004. He has published more than 70 papers in journals and conferences on signal processing and applied acoustics. His current research interests include fast adaptive filtering algorithms and multichannel signal processing for communications, 3D sound reproduction and MIMO wireless systems.

Gema Piñero was born in Madrid, Spain, in 1965. She received the Ms. in Telecommunication Engineering from the Universidad Politécnica de Madrid in 1990, and the Ph.D. degree from the Universidad Politecnica de Valencia in 1997, where she is currently working as an Associate Professor in digital signal processing. She has been involved in different research projects including array signal processing, active noise control, psychoacoustics and wireless communications in the Audio and Communications Signal Processing (GTAC) group of the Institute of Telecommunications and Multimedia Applications (iTEAM) of Valencia. She has leaded several projects on sound quality evaluation for the automotive and toy industry, and she has also been involved in several projects on 3G wireless communications supported by the Spanish Government and big industries as Telefonica. She has also published more than 60 contributions in journals and conferences. Her current research interests include array and distributed signal processing and its parallel implementation on graphic processing units. During September to November 2011 she was a visiting scholar at Prof. Andrew C. Singer's group at University of Illinois Urbana-Champaign.