# GPU BASED IMPLEMENTATION OF MULTICHANNEL ADAPTIVE ROOM EQUALIZATION

*Jorge Lorente, Miguel Ferrer, Maria de Diego, Alberto Gonzalez*

Instituto de Telecomunicaciones y Aplicaciones Multimedia (iTEAM), Universitat Politècnica de València (UPV)

## ABSTRACT

Multichannel adaptive equalization (AE) systems require high computational capacity, which constraints their practical implementation. Graphics Processing Units (GPUs) are well known due to their potential for highly parallel data processing. Although the GPUs seem to be suitable platforms for multichannel scenarios, an efficient use of parallel computation in the adaptive filtering context is not straightforward due to the feedback loops. This paper presents a GPU implementation of a multichannel AE system based on the filtered-x LMS algorithm working over a real-time prototype. Details of the parallelization of the algorithm are given. Experimental results are presented to validate and computationally analyze the real-time performance of the AE GPU implementation. Results show the usefulness of GPUs to develop versatile, scalable and low cost multichannel AE systems.

***Index Terms***— adaptive equalization, filtered-x LMS, graphics processing unit, frequency domain adaptive filters

## 1. INTRODUCTION

Multichannel equalization finds application in audio reproduction systems that create a given acoustic environment such as 3-D audio systems or audio applications in which different sounds are supply to separate listeners in the same listening space. Generally speaking, these systems intend to reproduce some desired signals at certain points of the listening space [1, 2]. The behavior of the acoustic system at a particular listening position (usually monitored by a microphone) is characterized by its impulse response. Thus, room equalization is used to modify the frequency spectrum of the original source before feeding it to the loudspeaker in order to compensate for the loudspeaker and listening room responses. The goal is to make the global impulse response as close as possible to a desired one. Thus, the combined effect of the equalizer and the acoustic path will allow to obtain a good approximation of the desired signal at the microphone.

Traditionally, fixed equalization techniques have been used to compensate these room effects. In these methods the equalization filters are computed once, and usually in a previous stage to the rendering one. However, this task in multichannel scenarios is not straightforward, and efficient computational methods are needed. This has led to the usage of adaptive systems that iteratively compute the filters. Moreover, real systems imply time-varying scenarios and the adaptive equalization filters are able to follow these changes. In the adaptive equalization context, literature contains several interesting algorithms, which usually consider a least mean square (LMS) algorithm, in time or frequency domain. However, most of these works only present results for a single-input multiple-output (SIMO) system, that is, an AE the computes only one adaptive filter for all the microphones signals [3, 4] which seems insufficient to perform equalization in multichannel scenarios with massive audience, where the sound has to be equalized at each listener position. In this massive scenarios, multiple-input multiple-output (MIMO) systems [5] become essential to compensate room effects.

On the other hand, AE MIMO systems which involve high number of long adaptive compensation filters and require high computational capacity, seems suitable to be implemented on GPUs. Using the NVIDIA programming language CUDA [6] the GPUs provides massive parallel computation for general-purpose computing, and therefore are being employed in most of the engineering fields that require intensive computation like signal processing [7]. A general overview of audio signal processing using GPUs is offered in [8]. Moreover, many recent contributions are taking advantage of the GPUs to accelerate acoustic/audio real-time applications like: room acoustics [9], acoustics likelihood computation [10], RIR reshaping [11], wave-field synthesis [12], massive filtering [13] and recursive filtering [14]. Although some authors have suggested the use of GPUs in the adaptive filtering context [15, 16], there are very few publications [17, 18, 19] dealing with the GPU implementation of real-time acoustic applications based on adaptive filtering. This is because the data transactions among GPU, CPU and the audio card needed at each iteration are critical for the real-time performance.

In previous works [18, 19], a single-channel and a multichannel active noise control (ANC) systems based on the frequency partitioned block filtered-x LMS (FPBFxLMS) algorithm were implemented on a GPU. In this paper, a multichannel AE system based on the FPBFxLMS algorithm has been implemented on the GPU. The choice of the algorithm is conditioned by the efficient use of parallel computation. In a first stage, the use of the Frequency-domain Block-based filtered-

This work has been supported by European Union ERDF together with Spanish Government through TEC2012-38142-C04 project, and Generalitat Valenciana through PROMETEO/2009/013 and GVA/2013/134 projects.

**Table 1**. Notation of the description of the algorithms

| | | | |
|---|---|---|---|
| $I$ | Number of input signals | $J$ | Number of secondary sources (actuators) |
| $K$ | Number of error signals (monitoring sensors) | $B$ | Block size |
| $L$ | Length of the adaptive filters | $F$ | $L/B$, number of partitions of the adaptive filters |
| $M$ | Length of the FIR filters that model the acoustic paths | $P$ | $M/B$, number of partitions of the estimated acoustic paths |
| $x_i(t)$ | $i$th input signal at time instant $t$ | $\mathbf{x}i_{Bn}$ | $[x_i(Bn) \quad x_i(Bn-1) \quad \dots \quad x_i(Bn-B+1)]$ |
| $y_j(t)$ | $j$th actuator signal at time instant $t$ | $\mathbf{y}j_{Bn}$ | $[y_j(Bn) \quad y_j(Bn-1) \quad \dots \quad y_j(Bn-B+1)]$ |
| $e_k(t)$ | $k$th algorithm error signal at time instant $t$ | $\mathbf{e}k_{Bn}$ | $[e_k(Bn) \quad e_k(Bn-1) \quad \dots \quad e_k(Bn-B+1)]$ |
| $m_k(t)$ | $k$th microphone signal at time instant $t$ | $\mathbf{m}k_{Bn}$ | $[m_k(Bn) \quad m_k(Bn-1) \quad \dots \quad m_k(Bn-B+1)]$ |
| $d_k(t)$ | $k$th desired signal at time instant $t$ | $\mathbf{d}k_{Bn}$ | $[d_k(Bn) \quad d_k(Bn-1) \quad \dots \quad d_k(Bn-B+1)]$ |
| $\mathbf{h}jk$ | M-length estimation of the acoustic path that links the $j$th secondary source with the $k$th monitoring sensor | | |
| $\mathbf{H}jk^p$ | FFT of size $2B$ of the $p$th partition of the acoustic path $\mathbf{h}jk$ | | |
| $\mathbf{w}ij_n$ | Coefficients of the adaptive filter of length $L$ that links the $i$th input signal with the $j$th secondary source | | |
| $\mathbf{W}ij_n^f$ | FFT of size $2B$ of the $f$th partition of the coefficients of the adaptive filter $\mathbf{w}ij$ during the $n$th block iteration | | |

x LMS [20], is motivated by the following reasons. The parallel resources of a GPU are better exploited working with blocks of samples instead of sample-by-sample, and most of the common audio cards work with block data buffers. In a second stage, since the adaptive filters could be larger than the block size, the adaptive filters are partitioned [21], and therefore the delay is reduced and the parallelization is improved by performing the adaptation of each partition of the filters at the same time. These reasons lead us to the use of the FPBFxLMS algorithm.

This paper is organized as follows: section 2 outlines the FPBFxLMS algorithm. Section 3 describes the GPU implementation of the prototype. Finally, section 4 and 5 are devoted to report the experimental results and conclusions.

## 2. THE FPBFXLMS ALGORITHM

This section focuses on illustrating the FPBFxLMS algorithm. For simplicity, the block diagram of a single channel AE system is depicted in Fig. 1. However, notation in Table 1 will be used to describe the algorithm for a generic multichannel AE system with $I$ input signals, $J$ secondary sources and $K$ monitoring sensors ($I$:$J$:$K$). Furthermore, the subindex and super-index of the following notation denote block iteration and number of partition respectively. The acoustic paths have been previously modeled by FIR filters.

The adaptive filter output is calculated as follows

$$\mathbf{Y}j_n = \sum_{i=1}^{I} \sum_{f=1}^{F} \mathbf{W}ij_n^f \circ \mathbf{X}i_{n-f+1}, \tag{1}$$

where $\mathbf{X}i_n = \mathrm{FFT}[\mathbf{x}i_{Bn-1} \quad \mathbf{x}i_{Bn}]$, and $\circ$ denotes the element-wise product of two vectors. The adaptive filter output $\mathbf{y}j_{B_n}$ are the last $B$ samples of $\mathrm{IFFT}\{\mathbf{Y}j_n\}$.

The algorithm error signal is calculated by subtracting the desired signal to the microphone signal. Therefore if the algorithm error signal tends to zero, the microphone signal converges to the desired signal. The operations are the following:

$$\mathbf{e}k_{Bn} = \mathbf{m}k_{Bn} - \mathbf{d}k_{Bn}. \tag{2}$$

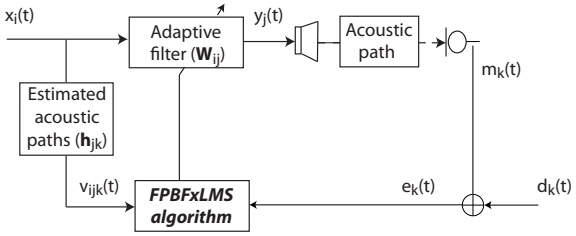


**Fig. 1**. Block diagram of a single channel AE system based on the FPBFxLMS algorithm.

$$\mathbf{E}k_n = \mathrm{FFT}[\mathbf{0}_B \quad \mathbf{e}k_{Bn}]. \tag{3}$$

The update of the coefficients of each partition of the *ij*th adaptive filters is calculated in frequency domain as follows

$$\mathbf{W}ij_n^f = \mathbf{W}ij_{n-1}^f + \mu \sum_{k=1}^{K} \mathrm{FFT}\{[\boldsymbol{\phi}ijk^f \quad \mathbf{0}_B]\}, \tag{4}$$

being $\mu$ the step-size parameter, and the vector $\boldsymbol{\phi}ijk^f$ corresponds to the first B samples of the 2B-IFFT of the corresponding partition $\widetilde{\boldsymbol{\mu}}ijk^f$

$$\mathrm{IFFT}\{\widetilde{\boldsymbol{\mu}}ijk^f\} = [\boldsymbol{\phi}ijk^f \quad \bar{\boldsymbol{\phi}}ijk^f]. \tag{5}$$

Finally, vector $\widetilde{\boldsymbol{\mu}}ijk^f$ is obtained by calculating the correlations between the input signals filtered through the estimated secondary paths $\mathbf{V}ijk_n$, and the error signals $\mathbf{E}k_n$. To this end, the following operations are performed:

$$\mathbf{V}ijk_n = \sum_{p=1}^{P} \mathbf{H}jk^p \circ \mathbf{X}i_{n-p+1}, \tag{6}$$

$$\widetilde{\boldsymbol{\mu}}ijk^f = \mathbf{E}k_n \circ \mathbf{V}ijk^{f*}. \tag{7}$$

## 3. GPU IMPLEMENTATION OF THE PROTOTYPE

The prototype uses a CPU, a GPU and an audio card. The GPU used is a GeForce GTX 580 with Fermi architecture.

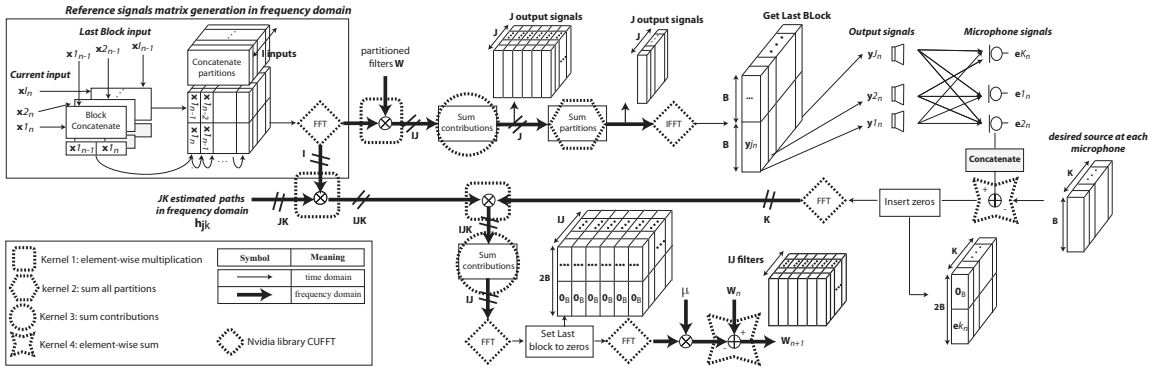


**Fig. 2**. Block diagram of the GPU implementation of the FPBFxLMS algorithm.

The CPU is an Intel Core i7 (3.07 GHz) and the audio card is a MOTU 24I/O. The MOTU audio uses the ASIO (Audio Stream Input/Output) driver to communicate with the CPU. The ASIO driver provides input/output buffers that are used to collect/send the current microphone and loudspeaker signals. The input buffers are linked to the microphones and the output buffers to the loudspeakers. The operation of the prototype consists of three tasks that are executed at each iteration:

1. Collect the $K$ input-data buffers of size B from the sensors, and transfer them through the PCI-Express bus to the GPU.
2. Carry out the corresponding algorithm on the GPU.
3. Save the output audio samples into the $J$ output-data buffers and send them back to the CPU in order to be rendered through the loudspeakers.

Two important parameters given by the MOTU are the sampling rate ($f_s$) and the block size ($B$), which describes the number of transferred discrete-time samples per iteration. The MOTU offers three sampling rates: $44.1$, $44.8$ and $96$ kHz and $B$ sizes between: $B = 256$ and $2048$. The lower rate ($f_s = 44.1$ kHz) has been chosen, however it is a fairly high rate for the sounds involved. The choice of the size of $B$, is critical for the real-time performance of the system, since the application can work in real time if the following condition is satisfied: $t_{proc} < t_{buff}$, where $t_{buff} = B/f_s$ is the time spent to fill up the input-data buffer, and $t_{proc}$ is the execution delay. This includes transfer delays between CPU and GPU and the data processing delay on the GPU.

Figure 2 shows the GPU implementation of the algorithm. The NVIDIA FFT library (CUFFT) [6] has been used for carrying out simultaneously multiple FFTs. The algorithm uses 4 optimized GPU kernels:

1. Kernel 1 performs an element-wise multiplication of two matrices. This kernel launches a three-dimensional grid of three-dimensional blocks of threads. The blocks are dimensioned with (x,y,z)[1] threads, and grid with (F/x, 2B/y, IJ/z) blocks. The kernel uses each thread for processing each sample, thus each thread will perform a complex multiplication between elements of each matrix.
2. Kernel 2 reduce the $F$ columns of each plane to a single one. This kernel uses a three-dimensional grid of blocks, where the dimension of the blocks and grid are (1, y, z) and (1, 2B/y, m/z), respectively. The kernel launches $2B \cdot m$ threads in total. Each thread carries out $F$ sums. The result is a $2B \times m$ matrix where each element of the $2B$-dimension columns contain the reduction sum of each row. Note that $m$ is the number of planes of $2B \times F$ elements of the matrix involved.
3. Kernel 3 performs a sum of $m$ planes with the same sub-index. As an example, the matrix $\widetilde{\boldsymbol{\mu}}$ of $IJK$ planes results in a matrix of $IJ$ planes after performing this kernel. In this case, a sum of $K$ planes ($m = K$) with the same $ij$ sub-indexes is performed. This Kernel launches $2B \cdot P$ threads divided into a grid of $P$ blocks, where each block has $2B$ threads. Each thread performs the sum of $m$ elements.
4. kernel 4 has the same thread configuration as kernel 1, but each thread performs a sum instead of a multiplication.

[1] x,y,z are selected according to the restrictions of the CUDA architecture of the device. In our case, Fermi device [6]

## 4. RESULTS

Some experiments are presented here to study the performance of the GPU based AE system. The experiments have been conducted in live by using the prototype described in section 3. On the one hand, the algorithm behavior has been evaluated. On the other hand, the computational limits of the GPU implementation have been studied.

Regarding the algorithm behavior, the experiment consists on evaluating the system distance index ($D$) [22], which finds the distance between the real case and the ideal case. Regarding a system configuration with one input signal ($I = 1$), $D$ is defined for the $n$th iteration at each sensor $k$ as $Dk_n = 20\log_{10}(|\sum_{j=1}^{J}(\mathbf{w}1j_n * \mathbf{h}jk) - \mathbf{d}_k|_2/|\mathbf{d_k}|_2)$, being $\mathbf{d}_k$ the $K$ desired system vectors which ideally corresponds to delayed delta functions, and $|\cdot|_2$ the 2-norm.

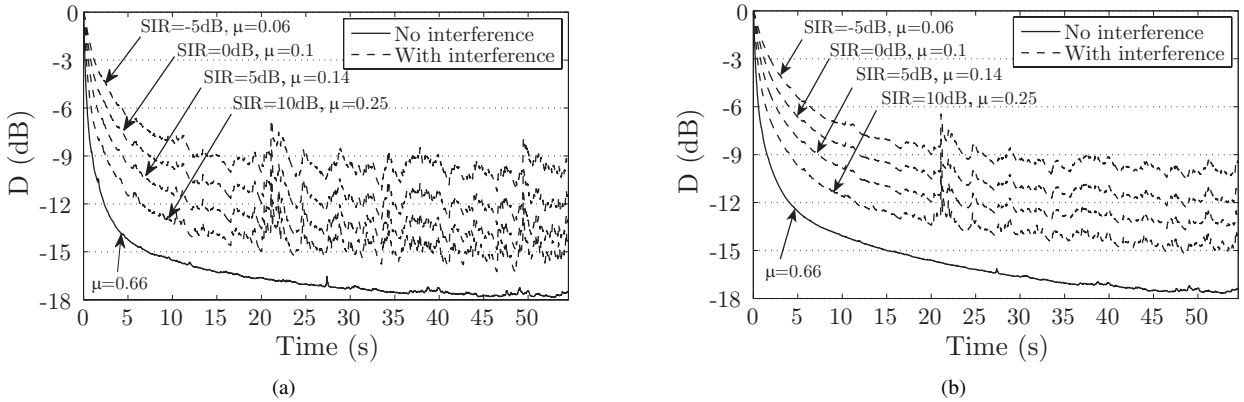


**Fig. 3**. $Dk_n$ evolution for the 1:2:2 configuration at microphone 1 (a) and microphone 2 (b).

**Table 2**. Maximum $I$:$J$:$K$ system for different size of $B$

| | **B=256** | **B=512** | **B=1024** | **B=2048** |
|---|---|---|---|---|
| $t_{proc_{max}}$ | 5.8 ms | 11.6 ms | 23.2 ms | 46.4 ms |
| $I$:$J$:$K$ | 1:13:13 | 1:18:18 | 1:24:24 | 1:36:36 |
| Channels | 169 | 324 | 576 | 1,089 |

Figure 3 shows the evolution of the D parameter at both microphones using a 1:2:2 configuration, a block size of $B = 512$ and adjusting the $\mu$ parameter to the maximum value that assures stability. The D parameter has been measured with a voice interference signal at the microphones for different signal to interference ratios (SIR). If the interferences power increases, the SIR ratio decrease and the maximum $\mu$ parameter that assures stability also decreases [23]. As a result, the convergence speed is reduced. Although low ratios of SIR are used, good results in terms of the $D$ parameter are obtained, which means that the algorithm is robust even with low SIR levels. However, the best performance is achieved when there is no interference affecting to the operation of the prototype, and results get worst by reducing the SIR ratio.

Focusing on the GPU results, the computational limits have been analyzed varying the size of B. The size of B affects both the algorithm behavior and the computing results. On the one hand, as $B$ decreases, the FPBFxLMS converges faster [19], but on the other hand, the real-time condition limits the processing time to $t_{proc} < B/f_s$, so if $B$ decreases there is less time for processing, and therefore less channels can be processed. Therefore, the ideal value of $B$ must be chosen depending on the needs of the applications.

Table 2 shows the maximum number of loudspeakers and microphones that the GPU implementation can handle in real-time for AE systems with one input signal ($I$=1), the same number of loudspeakers and microphones ($J$=$K$) and varying the size of $B$. Using the hardware configuration of section 3, results show that in the best case, the system can handle in real time more than 1.000 channels, defining a channel as a pair loudspeaker-sensor. However, more channels could be processed by using a lower sampling rate or using a newer card with more computing capacity. It is important to note that for this analysis, a previous analysis of the distribution of threads in a block and blocks in a grid is necessary for each specific case in order to achieve a good performance [18]. This previous analysis consists in testing the processing delay of the algorithm for each specific $I$:$J$:$K$ case changing the dimensions of both blocks of threads and grids of blocks to find the fastest configuration for each different case.

## 5. CONCLUSIONS

This work analyzes the suitability of GPUs for real-time implementation of multichannel adaptive systems, specifically for AE systems based on the FxLMS algorithm. To fit the hardware/GPU requirements, the algorithms has been implemented in frequency domain, working with blocks of data and partitioning the adaptive filters. As a result, a prototype of multichannel AE application has been successfully implemented in GPU using CUDA language and taking benefit of the parallelization of the multiple channels involved. Some CUDA programming aspects like the GPU data transfers or the number and distribution of the threads in a block and blocks in a grid has been analyzed in order to guarantee an efficient implementation.

Results show good performance of the AE prototype even when there are interfering signals with low SIR levels. Finally, in order to obtain a massive multichannel equalization system suitable for a massive audience through the use of a high number of loudspeakers/sensors, the computing limits of the AE system has been studied. It has been demonstrated that the GPU is a meaningful and versatile solution for massive multichannel AE systems with even more than 1,000 channels processed in real time. Moreover, it is important to note that more channels could be processed using a different audio card with lower frequency sampling, decimating or using newer GPUs with more computational capacity.

## 6. REFERENCES


[1] C. Kyriakakis, P. Tsakalides, and T. Holman, "Surrounded by sound," *IEEE Signal Process. Mag.*, vol. 16, no. 1, pp. 55–66, 1999.

[2] P. A. Nelson, F. Orduna-Bustamante, and H. Hamada, "Inverse filter design and equalization zones in multichannel sound reproduction," *IEEE Trans. Speech Audio Process.*, vol. 3, no. 3, pp. 185–192, 1995.

[3] S. Cecchi, A. Primavera, F. Piazza, and A. Carini, "An adaptive multiple position room response equalizer," in *Proc. of Eur. Signal Process. Conf (EUSIPCO)*, Barcelona, Spain, 2011, pp. 1274–1278.

[4] S. J. Elliott and P. A. Nelson, "Multiple-point equalization in a room using adaptive digital filters," *Journal of the Audio Eng. Soc.*, vol. 37, no. 11, pp. 899–907, 1989.

[5] L. Fuster, M. de Diego, M. Ferrer, A. Gonzalez, and G. Pinero, "A biased multichannel adaptive algorithm for room equalization," in *Proc. of Eur. Signal Process. Conf (EUSIPCO)*, Bucharest, 2012, pp. 1344–1348.

[6] "NVIDIA programming guide," *online at: http://developer.download.nvidia.com/*.

[7] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips, "GPU computing," *Proc. of IEEE*, vol. 96, pp. 879–899, 2008.

[8] L. Savioja, V. Valimki, and J. O. Smith, "Audio signal processing using graphics processing units," *Journal of the Audio Eng. Soc.*, vol. 59, pp. 3–19, 2011.

[9] C. J. Webb and S. Bilbo, "Computing room acoustics with CUDA-3D FDTD schemes with boundary losses and viscosity," in *Proc. of IEEE Int. Conf. on Acoustics, Speech and Signal Process. (ICASSP)*, 2011, pp. 317–320.

[10] J. Vanek, J. Trmal, J. V. Psutka, and J. Psutka, "Optimized acoustic likelihoods computation for NVIDIA and ATI/AMD graphics processors," *IEEE Trans. Audio, Speech, Language Process.*, vol. 20, no. 6, pp. 1818–1828, 2012.

[11] R. Mazur, J. O. Jungmann, and A. Mertins, "On CUDA implementation of a multichannel room impulse response reshaping algorithm based on p-norm optimization," in *Proc. of IEEE Workshop on Appl. Signal Process. Audio Acoust. (WASPAA)*, New Paltz, New York, U.S.A, 2011, pp. 305–308.

[12] D. Theodoropoulos, G. Kuzmanov, and G. Gaydadjiev, "Multi-core platforms for beamforming and wave field synthesis," *IEEE Trans. Multimedia*, vol. 13, no. 2, pp. 235–245, 2011.

[13] 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, pp. 449–457, 2011.

[14] D. H. Lee and W. Sung, "GPU based implementation of recursive digital filtering algorithms," in *Proc. of IEEE Int. Conf. on Acoustics, Speech and Signal Process. (ICASSP)*, Vancouver, Canada, 2013, pp. 2684–2687.

[15] W. Bozejko, A. Dobrucki, and M. Walczynski, "Parallelizing of digital signal processing with using GPU," in *Proc. of Conf. Signal Process. Algorithms, Architectures, Arrangements, and App.*, 2010, pp. 29–33.

[16] A. Hirano and K. Nakayama, "Implementation of large-scale FIR adaptive filters on nVIDIA GeForce graphics processing unit," in *Proc. of Int. Conf. on Intelligent Signal Process. and Comm. Systems*, 2010, pp. 1–4.

[17] M. Schneider, F. Schuh, and W. Kellerman, "The generalized frequency-domain adaptive filtering algorithm implemented on a GPU for large-scale multichannel acoustic echo canceller," in *Proc. of ITG conf. on Speech Comm.*, Braunschweig, Germany, 2012, pp. 1–4.

[18] J. Lorente, J. A. Belloch, M. Ferrer, A. Gonzalez, J. A. Belloch, M. de Diego, G. Piñero, and A. Vidal, "Multichannel active noise control system using a GPU accelerator," in *Proc. of Internoise*, New York, USA, 2012.

[19] J. Lorente, M. Ferrer, M. de Diego, J. A. Bellocho, and A. Gonzalez, "GPU implementation of a frequency-domain modified filtered-x LMS algorithm for multichannel local active noise control," in *Proc. of Int. Conf. of Audio Eng. Soc.*, 2013.

[20] Q. Shen and A. S. Spanias, "Time and frequency domain x block LMS algorithms for single channel active noise control," in *Proc. of Int. Congr. Recent Developments in Air-and Structure-Borne Sound Vibration*, Auburn, Alabama, USA, 1992, pp. 353–360.

[21] J. M. Páez Borrallo and M. Garcia Otero, "On the implementation of a partitioned block frequency domain adaptive filter (PBFDAF) for long acoustic echo cancellation," *Signal Processing*, vol. 27, no. 3, 1992.

[22] S. Goetze, M. Kallinger, A. Mertins, and K. D. Kammeyer, "A decoupled filtered-x LMS algorithm for listening-room compensation," in *Proc. of Int. Workshop on Acoust. Echo and Noise Control (IWAENC)*, Seattle, USA, 2008.

[23] D. Slock, "On the convergence behavior of the LMS and the normalized LMS algorithms," *IEEE Trans. Signal Process.*, vol. 41, no. 9, pp. 2811–2825, 1993.